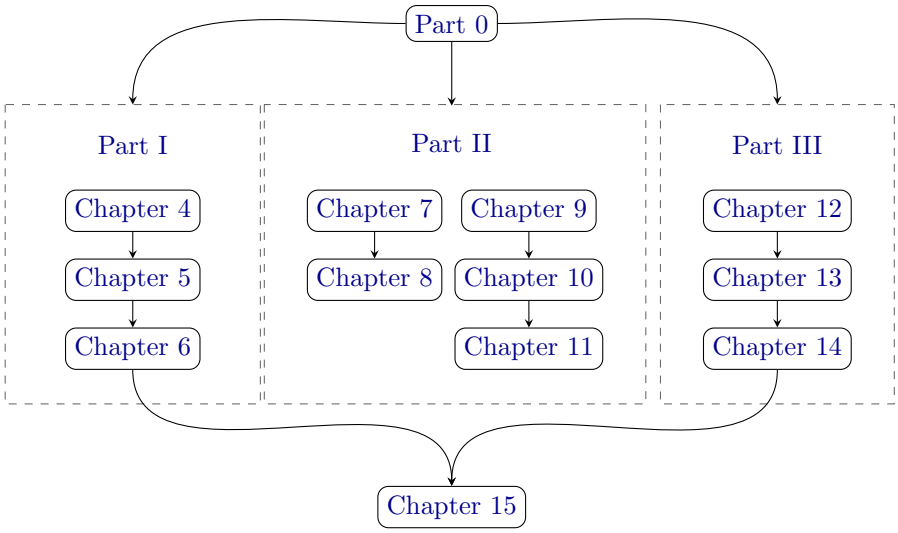
isogenies & isometries

Krijn Reijnders

Contents

Introduction

This thesis will be super cool.



Organizational chart of the components of the thesis.

Part 0

Background

Background

This thesis concerns itself with cryptography. In general, modern cryptography relies on the hardness of mathematical problems to ensure its security and fields within cryptography are often named after the underlying mathematical structures on which such problems are based. *Post-quantum cryptography*, which takes into account an adversary with access to a quantum computer, usually relies on one of five mathematical or cryptographical structures: lattices, codes, isogenies, hash functions and multivariate systems. This thesis concerns itself with code-based and isogeny-based cryptography, and, in particular, has a focus on code-based cryptography using *isometries*. There is some overlap in theory between *isometries* and *isogenies*: both are maps that preserve the essential structure between their domains and codomains, and both such maps should be hard to derive given their domains and codomains. This makes both types of functions suitable for cryptography. Beyond these similarities, there is little overlap in the concrete theory and cryptographical application and we must be careful not to get carried away in their semblance. Nevertheless, we will see throughout this thesis that ideas in one field may inspire the other, and vice versa.

We start with an outline of general constructions in public key cryptography in Chapter 1. We continue with the theoretical background on codes and their isometries (Chapter 2) and curves and their isogenies (Chapter 3).

Chapter 1

General Cryptography

1.1 Public Key Cryptography

We start by outlining the general concepts of public key cryptography that will be used throughout the work such as cryptographic primitives and security notions. A more thorough introduction is the textbook by Galbraith [153], which inspired this section. Although cryptography has many spectacular and 'fancy' applications, we restrict ourselves to three basic constructions in public key cryptography:

1. *non-interactive key exchanges* (NIKEs) which exchange a key between two parties without any interaction between these parties,
2. *key-encapsulation mechanisms* (KEMs), which simply allow two parties to agree on a shared value (a key), but require interaction, and
3. *digital signatures*, which allow anyone with access to some public key to verify that a message or piece of data was signed by the owner of the associated private key.

These three core primitives are crucial building blocks for an extensive list of advanced primitives, protocols and applications, and are used daily in any digital devices connected to other devices. Hence, improving the cryptanalysis or performance of particular NIKEs, KEMs or signatures is essential to ensure safety, security and speed in our day-to-day lives. As we will see later, code-based and isogeny-based KEMs and signatures both face very different challenges in terms of practicality.

1.1.1 Non-interactive key exchange

Informally, a non-interactive key exchange allows two parties A and B to agree on some shared value K, with A only needing to know some public key $\mathsf{pk_B}$ of B and vice versa. Any public key pk, known to all, is associated with a secret key sk, known to only one party. Such a pair $(\mathsf{sk}, \mathsf{pk})$ is generated by an algorithm known as Keygen which takes as input the security parameter λ. Deriving the shared value K requires an algorithm Derive, which returns the same value for both parties. Altogether, we get the following definition.

Definition 1.1. A *non-interactive key exchange (NIKE)* is a collection of two algorithms Keygen and Derive, such that

- Keygen is a probabilistic algorithm that on input 1^λ, with λ the security parameter, outputs a keypair $(\mathsf{sk}, \mathsf{pk})$, and
- Derive is a deterministic algorithm that on input a public key pk and a secret key sk outputs a key K.

A NIKE is *correct* if for every two pairs $(\mathsf{sk_A}, \mathsf{pk_A}) \leftarrow \mathsf{Keygen}(1^\lambda)$ and $(\mathsf{sk_B}, \mathsf{pk_B}) \leftarrow \mathsf{Keygen}(1^\lambda)$ it holds that $\mathsf{Derive}(\mathsf{sk_A}, \mathsf{pk_B}) = \mathsf{Derive}(\mathsf{sk_B}, \mathsf{pk_A})$.

Note that, given the public information pk of some party x, any party y with private key sk' can derive $\mathsf{K} = \mathsf{Derive}(\mathsf{sk}', \mathsf{pk})$ without any interaction with x.

Pre-quantum key exchange. The quintessential example of a NIKE is the *Diffie-Hellman-Merkle key exchange* [128, 210], usually given for finite fields or elliptic curves, which is in general applicable to any cyclic group G. As a scheme parameter, it requires a generator g of order q for G. It relies on the core equality

$$(g^a)^b = g^{ab} = (g^b)^a$$

for positive integers a and b, which allows two parties A and B to compute g^{ab}, where A only needs to know her (secret) value $\mathsf{sk_A} := a$ and the public value $\mathsf{pk_B} := g^b$, and similarly, B can compute g^{ab} given $\mathsf{pk_A} := g^a$ and $\mathsf{sk_B} := b$. In this example, Keygen simply samples a random positive integer $a \in \mathbb{Z}_q$ and returns the pair (a, g^a), whereas Derive computes g^{ab} given g^b and a.

It is easy to see that this cryptographic scheme would be completely broken if one could derive either a from g and g^a or g^{ab} from g, g^a and g^b, as this would allow an adversary without any of the required secret knowledge to derive the secret value K that should only be accessible to A and B. The security

of this scheme therefore relies on two *hardness assumptions*, or equivalently, the difficulty of two mathematical problems, known as the *discrete logarithm problem* and the *computational Diffie-Hellman problem.*

Problem 1.1 (Discrete logarithm problem)**.** Let G be a group. The *discrete logarithm problem* is:

$$\text{Given } g, h \in G, \quad \text{find } a \in \mathbb{N} \text{ such that } h = g^a.$$

For Diffie-Hellman-Merkle key exchange, we usually require a (sub)group G of prime order q, and have to choose this group G and the generator g specifically so that Problem 1.1 is cryptographically hard, i.e. there exist no polynomial time algorithms to find a given g and h.

Problem 1.2 (Computational Diffie-Hellman problem)**.** Let G be a group. The *computational Diffie-Hellman problem* is:

$$\text{Given } g, g^a, g^b \in G, \quad \text{compute } g^{ab}.$$

Sometimes, the decisional variant is required: in this case, one is given g, g^a, g^b and some z, which is either a random value g^c or the value g^{ab}. The goal is then to decide if z is random or $z = g^{ab}$.

If the group G is selected carefully, for example as the group of points on a specific elliptic curve E over a finite field $\mathbb{F}_q$, this problem remains secure for classical adversaries. Unfortunately, Shor [269] shows that an adversary with access to a quantum computer can break both problems in polynomial time.

Post-quantum key exchange. In post-quantum cryptography, we require our problems to remain cryptographically hard, even with access to a quantum computer. As of the moment of writing, there are two classes of post-quantum NIKEs available:

1. CSIDH [85], and generalizations [142], which is the subject of Part II of thesis, and

2. Swoosh [148], a lattice-based key exchange originally considered "folklore", with a first proper attempt given by Kock [189].

Both NIKEs are, at the moment of writing, practically computable, yet far from practical. The performance and security of CSIDH will be analyzed in Part II, Swoosh suffers from large public keys.

1.1.2 Key-encapsulation mechanisms

A key-encapsulation mechanism achieves a similar goal to a NIKE, sharing some key between two parties, but requires interaction. More precisely, in key-encapsulation we again generate a key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Keygen}(1^\lambda)$ for each party, but the public key pk is now used to encapsulate some key K in a ciphertext ct using an algorithm Encaps. The associated sk is required to compute K from ct. Thus, any party that knows pk can get a K from Encaps (and keeps it secret) and knows that only the owner of sk can compute the same K too, using Decaps. Altogether, we get the following definition.

Definition 1.2. A *key-encapsulation mechanism (KEM)* is a collection of three algorithms Keygen, Encaps, and Decaps, such that

- Keygen is a probabilistic algorithm that on input 1^λ, with λ the security parameter, returns a keypair $(\mathsf{sk}, \mathsf{pk})$, and
- Encaps is a probabilistic algorithm that on input a public key pk returns returns a ciphertext ct and a key K, and
- Decaps is a deterministic algorithm that on input a secret key sk and a ciphertext ct returns a key K.

A KEM is *correct* if for every pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Keygen}(1^\lambda)$ and $(\mathsf{ct}, \mathsf{K}) \leftarrow \mathsf{Encaps}(\mathsf{pk})$, it holds that $\mathsf{Decaps}(\mathsf{sk}, \mathsf{ct}) = \mathsf{K}$.

In 2017, to find replacements for pre-quantum key exchange mechanisms, the National Institute of Standards and Technology (NIST) launched a standardization effort for post-quantum KEMs, which resulted in the selection of Kyber [59] in 2022.

Difference with NIKE. Although both NIKEs and KEMs are used to exchange a key K, we can be more flexible with a NIKE as it is non-interactive. Pre-quantum cryptography therefore uses the Diffie-Hellman-Merkle NIKE as a building block for an incredibly wide range of applications, from XXX to XXX. In a post-quantum setting, the limitations of current NIKEs prevent them from being used in most settings. In general, the usage of NIKEs and KEMs in protocols falls into three categories:

1. Some protocols require an interaction by nature, and therefore use a KEM. The usage of a NIKE in such a situation is possible but does not offer any additional benefit. A traditional example is TLS, where the current usage of Diffie-Hellman-Merkle can migrate to post-quantum KEMs [60, 260].

2. Some protocols require a NIKE by nature, which cannot be replaced by a KEM. An example is Signal's X3DH protocol [205], which still needs to work when participants are offline and can therefore not interact.

3. Some protocols can use KEMs but at some additional cost, usually an extra round of communication. In such instances, the comparison between post-quantum NIKEs or KEMs is especially interesting. Chapter 10 discusses this topic in more detail.

1.1.3 Digital signatures

A digital signature allows some party A to compute a signature σ for some piece of data, usually a message msg, using its secret key sk with an algorithm called Sign. Anyone with the associated public key pk can verify this signature, which assures that A computed σ for msg, with an algorithm called Verif. A successful verification is usually interpreted as A authenticating msg and provides integrity of the content of msg. Altogether, we get the following definition.

Definition 1.3. A *signature scheme* is a collection of three algorithms Keygen, Sign, and Verif, such that

- Keygen is a probabilistic algorithm that on input 1^λ, with λ the security parameter, returns a keypair $(\mathsf{sk}, \mathsf{pk})$, and
- Sign is a probabilistic algorithm that on input a message msg and a secret key sk returns a signature σ, and
- Verif is a deterministic algorithm that on input a public key pk, a message msg and a signature σ returns either "valid" or "invalid".

A signature scheme is *correct* if for every pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Keygen}(1^\lambda)$ and signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{msg}, \mathsf{sk})$, it holds that $\mathsf{Verif}(\mathsf{msg}, \sigma, \mathsf{pk})$ returns "valid".

Pre-quantum signatures. The most elegant yet simple and natural example of pre-quantum signatures is the *Schnorr signature scheme* [258], which is based on the same discrete logarithm problem (Problem 1.1) as Diffie-Hellman-Merkle key exchange. This starts from the following *interactive* protocol: Assuming a finite cyclic group G with generator g of order q in which the discrete logarithm problem is hard, set the secret key of A to a random $\mathsf{sk}_\mathsf{A} \leftarrow a \in \mathbb{Z}_q^*$ and the public key as $\mathsf{pk}_\mathsf{A} \leftarrow g^a$. To convince B that she knows sk_A, A takes another random $r \leftarrow \mathbb{Z}_q^*$ and sends $R \leftarrow g^r$ to B (the *commitment*). As a *challenge*, B

sends back a value $c \in \mathbb{Z}_q$. As a *response*, A computes $s \leftarrow r - c \cdot \mathsf{sk}_\mathsf{A} \mod q$ and sends this back to B. The idea is that s can only be computed correctly if one knows sk_A, and so, by confirming that the commitment R equals $g^s \cdot \mathsf{pk}_\mathsf{A}{}^c$, party B is convinced that A really does know sk_A.

To turn the above identification protocol into a non-interactive signature scheme, all users agree on a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$, and we replace the challenge $c \leftarrow \mathbb{Z}_q$ by the hash of R (as a bit string) concatenated with msg to get $c \leftarrow H(R||\,\mathsf{msg})$. The resulting $s \leftarrow r - c \cdot \mathsf{sk}_\mathsf{A} \mod q$ together with c becomes the signature $\sigma \leftarrow (s, c)$. To verify, we recompute $R \leftarrow g^s \cdot \mathsf{pk}_\mathsf{A}{}^c$ and ensure that $H(R||\,\mathsf{msg})$ equals c.

Sigma protocols and the Fiat-Shamir heuristic. The method of the previous example to turn an identification protocol into a signature scheme is a well-known method named the *Fiat-Shamir heuristic* [144] and applies to a broad range of identification protocols. In this thesis, we restrict ourselves to Sigma protocols [115] which are identification protocols based on the commitment-challenge-response structure we saw in the Schnorr identification protocol. Such Sigma protocols assume two parties: a "prover" A and a "verifier" B, with A proving knowledge of some public statement x tied to some secret knowledge w, often called the *witness*. For example, in Scnorr's identification protocol, A convinces B that she knows the witness $w = \mathsf{sk}_\mathsf{A}$ such that $x = \mathsf{pk}_\mathsf{A} = g^{\mathsf{sk}_\mathsf{A}}$. We use the following definition for Sigma protocols.

Definition 1.4. A *Sigma protocol* is a collection of two prover algorithms, the commitment algorithm P_{cmt} and the response algorithm P_{resp}, and a verification algorithm V_{verif}, with the following flow. To prove knowledge of a witness w for the statement x,

1. A computes a commitment $\mathsf{cmt} \leftarrow P_{\mathsf{cmt}}$ and sends it to B,
2. B samples $\mathsf{chal} \leftarrow S$ uniformly random from a predefined set S, and sends chal to A,
3. A computes $\mathsf{resp} \leftarrow P_{\mathsf{resp}}(x, w, \mathsf{cmt}, \mathsf{chal})$ and sends resp to B,
4. B computes $V_{\mathsf{verif}}(x, \mathsf{cmt}, \mathsf{chal}, \mathsf{resp})$ and outputs **pass** or **fail**.

If the *transcript* $(\mathsf{cmt}, \mathsf{chal}, \mathsf{resp})$ verifies x, i.e. $V_{\mathsf{verif}}(x, \mathsf{cmt}, \mathsf{chal}, \mathsf{resp})$ passes, we call the transcript *valid* for x.

A Sigma protocol will only be useful if a prover A can actually convince B, that is, if only A who knows a witness w for x can generate a valid transcript for x, and furthermore that such a transcript can always be verified by B. These are distinct properties of a Sigma protocol, and we list the three most important here. We provide an intuitive explanation, formal definitions are abundant in the literature [**xxx**]

- A Sigma protocol is **complete** if a verifier that actually knows the secret w for x will be able to generate a valid transcript. More precisely, any honestly generated transcript $(\mathsf{cmt}, \mathsf{chal}, \mathsf{resp})$ for x is valid: $V_{\mathsf{verif}}(x, \mathsf{cmt}, \mathsf{chal}, \mathsf{resp})$ passes.
- A Sigma protocol is **honest-verifier zero-knowledge** if the verifier B gains no knowledge on the secret w from the transcript $(\mathsf{cmt}, \mathsf{chal}, \mathsf{resp})$. Zero-knowledge comes in different gradations:
  - the zero-knowledge is *perfect* if it is mathematically impossible to get information from the transcript,
  - the zero-knowledge is *statistical* if the distribution of valid transcripts is indistinguishable from a random transcript, and
  - the zero-knowledge is *computational* if differentiating between a valid transcript and a random transcript is a computationally hard problem.
- A Sigma protocol is **special sound** if, given two transcripts for the same cmt but different chal, it is possible to derive w in polynomial time. This implies that someone without knowledge of w can only convince the verifier with probability $1/|S|$, essentially by guessing the right response resp.

TODO: finish up this section bla bla

1.2 Cryptographic group actions

Cryptographic group actions are a useful framework to unify several different approaches to designing signature schemes[1]. After having defined a single group action, we can design advanced protocols in terms of this group action, whose security reduces to the hardness of recovering the group element that acted. This has led to an abundance of schemes, one of which is the main topic of Part I.

[1] This section is a summary of the results by Borin, Persichetti, Santini Pintore and me [58].

We first introduce the general terminology around group actions, before we summarize several techniques used to amplify the performance of digital signatures based on group actions. We furthermore present a table summarizing possible combinations of these techniques together with their impact on performance.

1.2.1 Properties of cryptographic group actions

Let G be a group and X a set. Denote by $\star$ a group action of G on X, that is, a function

$$\star : G \times X \to X, \quad (g, x) \mapsto g \star x,$$

which behaves well with regard to the group structure: $e \star x = x$ for all $x \in X$, with e the neutral element of G, and $g_1 \star (g_2 \star x) = (g_1 \cdot g_2) \star x$ for all $x \in X$ and all $g_1, g_2 \in G$.

Several properties of group actions are particularly relevant when used in a cryptographic context.

Definition 1.5. Let G act on X. The group action is said to

- *transitive*, if for every $x, y \in X$ there is a $g \in G$ such that $y = g \star x$,
- *faithful*, if no $g \in G$, except the neutral element, acts trivially on X, i.e. $g \star x = x$ for all $x \in X$,
- *free*, if no $g \in G$, except the neutral element, acts trivially on any element of X, i.e. $g \star x = x$ for one $x \in X$ implies $g = e$,
- *regular*, if the group action is both transitive and free, which implies there is a unique $g \in G$ for any two $x, y \in X$, such that $y = g \star x$.

Beyond these mathematical properties, we require several cryptographic properties for a group action to be useful in cryptography.

Definition 1.6. Let G act on X. The group action is *cryptographic* if

- *vectorisation* is hard, that is, given $y = g \star x$ and x, find g
- *parallellisation* is hard, that is, given x, $g_1 \star x$ and $g_2 \star x$, find $(g_1 \cdot g_2) \star x$.

Vectorisation is sometimes called the *Group Action Inverse Problem*, or GAIP, and parallellisation is sometimes called the *computational Group Action Diffie-Hellman Problem*, as it asks to find the bottom right corner of a

Diffie-Hellman protocol. A *cryptographic group action* is secure to use for protocol design, but may still not be very useful. This depends on yet a third set of properties, defined in [7], adapted here to our context.

Definition 1.7. Let G act on X, and let both be finite. The group action is *efficient* if the following procedures have efficient (probabilistic polynomial time) algorithms. For the group G,

- *membership testing*, that is, to decide if some element is a valid element of G,
- *equality testing*, that is, to decide if two elements represent the same element of G,
- *sampling*, that is, to sample (statistically close) to uniformly random from G,
- *multiplication*, that is, to compute $g_1 \cdot g_2$ given any $g_1, g_2 \in G$,
- *inversion*, that is, to compute g^{-1} given any $g \in G$.

For the set X,

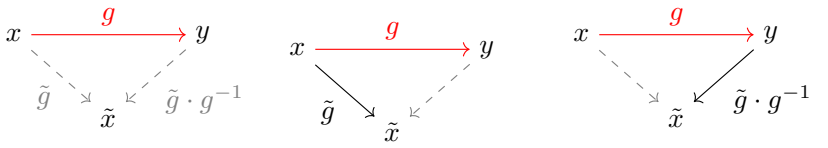
- *membership testing*, that is, to decide if some element is a valid element of X,
- *uniqueness of representation*, that is, to give a unique representation x for any arbitrary $x \in X$.

Furthermore, most crucially, *evaluation* must be efficient, that is, to compute $g \star x$ given any $g \in G$ and any $x \in X$.

Beyond these properties, *commutativity* of the group action is especially nice to have. As we will see in Parts I and II, whereas general cryptographic group actions allow constructions of digital signatures, commutative cryptographic group actions furthermore allow the construction of a NIKE. However, as of the moment of writing, CSIDH is the only known post-quantum commutative cryptographic group action.

1.2.2 Sigma protocols from cryptographic group actions

Given a cryptographic group action, it is easy to define a Sigma protocol and thus a digital signature scheme, after applying the Fiat-Shamir heuristic. The



(a) Commitment generation. (b) $\mathsf{ch} = 0$, we reveal $\tilde{g}$. (c) $\mathsf{ch} = 1$, we reveal $\tilde{g} \cdot g^{-1}$.

Figure 1.1: Simple one-round Sigma protocol based on group actions.

central idea is to prove knowledge of a secret element $g \in G$, by giving as a public key a pair $(x, y) \in X \times X$ with $y = g \star x$. Using a random element $\tilde{g} \in G$, party A can commit to $\tilde{x} = \tilde{g} \star x$ and only A can compute both paths $x \xrightarrow{\tilde{g}} \tilde{x}$ or $y \xrightarrow{\tilde{g} \cdot g^{-1}} \tilde{x}$. Thus, repeating such a *round*, as visualised in Figure 1.1, $t = \lambda$ times, where party B may each time ask for either of these paths, should convince B that A knows g.

For a λ-bit security level, assume we repeat this diagram t times. We denote by $\tilde{g}_i$ the uniformly random group element in round i, and by $\tilde{x}_i$ the commitment in round i. The challenge space of this Sigma protocol then becomes strings $\mathsf{ch} \in \{0,1\}^t$, where each ch_i asks to reveal either $x \to \tilde{x}_i$ or $y \to \tilde{x}_i$. The response then consists of each $\tilde{g}_i$, resp. $\tilde{g}_i \cdot g^{-1}$ for $1 \leq i \leq t$.

One can easily show that this very basic Sigma protocol is complete, honest-verifier zero-knowledge, and 2-special sound, hence, the security reduces to the vectorisation problem with x and $y = g \star x$.

TODO: the remainder of this section becomes the separate SOK paper in the last part of the thesis

Chapter 2

Codes & Isometries

Part I of this thesis concerns itself with code-based cryptography: our fundamental hardness assumption is a presumably difficult problem in coding theory. Contrary to classical code-based cryptography, the underlying hardness assumption on which much of the work in Part I is based is not syndrome decoding, but code equivalence. We first describe codes, and then their isometries, to give an overview of the ingredients required for Part I.

2.1 Codes

This thesis only concerns itself with *linear codes*, e.g. subspaces $\mathcal{C}$ of a vector space V over a finite field $\mathbb{F}_q$, equipped with some metric d. Let n denote the dimension of V, and k the dimension of $\mathcal{C}$, then such codes are called $[n, k]$-codes. Most well-known are *Hamming metric* codes, where $V = \mathbb{F}_q^n$ and $d(v, v')$ simply counts the number of different coefficients $v_i \neq v'_i$.

This thesis, however, focuses on a different class of codes, namely matrix codes: subspaces $\mathcal{C}$ of the vector space $V = \mathbb{F}_q^{n\times m}$ of matrices over a finite field. A more thorough introduction on matrix codes is given by Gorla [159]. The usual choice for measuring distance between matrices over a finite field is the so-called *rank metric*, defined as follows.

Definition 2.1. Let $\mathrm{Rank}(\mathbf{M})$ denote the rank of a matrix $\mathbf{M} \in \mathbb{F}_q^{n\times m}$. The *rank distance* between two $m \times n$ matrices $\mathbf{A}$ and $\mathbf{B}$ over $\mathbb{F}_q$ is defined as

$$d(\mathbf{A}, \mathbf{B}) = \mathrm{Rank}(\mathbf{A} - \mathbf{B}).$$

By symmetry, without loss of generality, in the rest of the works on matrix codes, we assume $n \geqslant m$.

Definition 2.2. A *matrix code* is a subspace $\mathcal{C}$ of $m \times n$ matrices over $\mathbb{F}_q$ endowed with the rank metric. We let k denote the dimension of $\mathcal{C}$ as a subspace of $\mathbb{F}_q^{m\times n}$ and we denote the basis by $\langle \mathbf{C}_1, \ldots, \mathbf{C}_k \rangle$, with $\mathbf{C}_i \in \mathbb{F}_q^{m\times n}$ linearly independent.

TODO: write a bit on vector rank codes versus matrix codes?
TODO: write a bit on the vectorization construction

2.2 Isometries

Definition 2.3. Let V be a linear space equipped with a metric d. An *isometry* is a map $\mu : V \to V$ that preserves the rank,

$$\mu(d(v)) = d(\mu(v)), \quad \text{for all } v \in V.$$

An isometry of a code $\mathcal{C} \subset V$ preserves the rank of the codewords

$$\mu(d(c)) = d(\mu(c)), \quad \text{for all } c \in \mathcal{C}.$$

It is then natural to ask if isometries of codes $\mu : \mathcal{C} \to \mathcal{C}$ extend to the space V they are contained in. For the Hamming metric, the result is satisfactory: The MacWilliams Extension Theorem [201] shows that any isometry μ of a linear code $\mathcal{C}$ can be extended to an isometry $\mu' : V \to V$ such that $\mu'_{|\mathcal{C}} = \mu$.

For the rank metric, this no longer holds. It is not difficult to construct counterexamples of isometries that do not extend, and recent research has explored the question to what extend such a theorem may hold for the rank metric [160]. In light of such obstructions, we have to admit defeat and only consider isometries of the full matrix space $\mathbb{F}_q^{n\times m}$ in this thesis. Luckily, such isometries are easy to categorize.

Lemma 2.4. Let $\mu : \mathbb{F}_q^{n\times m} \to \mathbb{F}_q^{n\times m}$ be an isometry with respect to the rank metric. If $n \neq m$, there exist matrices $\mathbf{A} \in \mathrm{GL}_n(q)$, $\mathbf{B} \in GLmq$ such that

$$\mu(\mathbf{M}) = \mathbf{AMB},$$

for all $\mathbf{M} \in \mathbb{F}_q^{n\times m}$. When $n = m$, we may additionally transpose $\mathbf{M}$, i.e.

$$\mu(\mathbf{M}) = \mathbf{AM}^T\mathbf{B}.$$

Definition 2.5. Two codes $\mathcal{C}$, $\mathcal{D}$ are *equivalent* if there exists an isometry $\mu : \mathcal{C} \to \mathcal{D}$.

In light of Lemma 2.4, we therefore say that two rank-metric codes $\mathcal{C}, \mathcal{D} \subseteq \mathbb{F}_q^{n\times m}$ are equivalent, if there exist two matrices $\mathbf{A} \in \mathrm{GL}_n(q)$, $\mathbf{B} \in \mathrm{GL}_m(q)$ such that

$$\mathbf{C} \mapsto \mathbf{ACB}$$

is an isometry $\mathcal{C} \to \mathcal{D}$. Note that this disregards transpositioning codewords; this has no impact for the cryptographic applications in this thesis. In short, when we write about isometries in the rank metric, we assume isometries in $\mathcal{I}_{n,m}(q) := \mathrm{GL}_n(q) \rtimes \mathrm{GL}_m(q)$. For any scalars $\lambda, \nu \in \mathbb{F}_q^*$, the isometry $\mu = (\mathbf{A}, \mathbf{B})$ and $\mu' = (\lambda\mathbf{A}, \nu\mathbf{B})$ are essentially equal, for all purposes. We can thus think of isometries as elements $(\mathbf{A}, \mathbf{B}) \in \mathrm{PGL}_n(q) \rtimes \mathrm{PGL}_m(q)$. This is conceptually the better interpretation, but often cumbersome. Thus, we stay with the notation $\mu = (\mathbf{A}, \mathbf{B})$ with $\mathbf{A} \in \mathrm{GL}_n(q)$, $\mathbf{B} \in \mathrm{GL}_m(q)$ and refer to the projective interpretation where needed. The set of isometries between two codes $\mathcal{C}$ and $\mathcal{D}$ is denoted $\mathcal{I}(\mathcal{C}, \mathcal{D}) \subseteq \mathcal{I}_{n,m}(q)$.

Automorphisms of matrix codes.

Definition 2.6. An *automorphism* of a code $\mathcal{C}$ is an isometry $\mu : \mathcal{C} \to \mathcal{C}$. The full group of automorphisms is denoted $\mathrm{Aut}^\bullet(\mathcal{C})$. We denote the subgroup of automorphisms for matrix codes derived from $\mu : \mathbb{F}_q^{n\times m} \to \mathbb{F}_q^{n\times m}$ by

$$\mathrm{Aut}(\mathcal{C}) = \{(\mathbf{A}, \mathbf{B}) \in \mathcal{I}_{n,m}(q) \mid \mathbf{A}\mathcal{C}\mathbf{B} = \mathcal{C}\}.$$

We write $\mathrm{Aut}_L(\mathcal{C})$, resp. $\mathrm{Aut}_R(\mathcal{C})$ to denote the subgroups of $\mathrm{Aut}(\mathcal{C})$ where $\mathbf{B} = \lambda\mathbf{I}_m$, resp. $\mathbf{A} = \lambda\mathbf{I}_n$.

Necessarily, $(\lambda\mathbf{I}_n, \nu\mathbf{I}_m) \in \mathrm{Aut}(\mathcal{C})$ for any code $\mathcal{C}$ and any scalars $\lambda, \nu \in \mathbb{F}_q^*$. If the automorphism groups contains no other elements than these, we say the automorphism group is *trivial*[1].

Both $\mathrm{Aut}_L(\mathcal{C})$ and $\mathrm{Aut}_R(\mathcal{C})$ are normal subgroups of $\mathrm{Aut}(\mathcal{C})$. We can thus define the group $\mathrm{Aut}_{LR}(\mathcal{C}) := \mathrm{Aut}(\mathcal{C})/\mathrm{Aut}_L(\mathcal{C}) \times \mathrm{Aut}_R(\mathcal{C})$, which measures the number of distinct non-trivial two-sided automorphisms a code has.

Lemma 2.7. Let $\mathcal{C}$ and $\mathcal{D}$ be equivalent codes. Then, for any $\mu : \mathcal{C} \to \mathcal{D}$,

$$\mu \cdot \mathrm{Aut}(\mathcal{C}) = \mathcal{I}(\mathcal{C}, \mathcal{D}) = \mathrm{Aut}(\mathcal{D}) \cdot \mu.$$

[1] From the projective perspective, this is the trivial subgroup of $\mathrm{PGL}_n(q) \rtimes \mathrm{PGL}_m(q)$.

Proof. As $\mathcal{C}$ and $\mathcal{D}$ are equivalent, there is some $\mu \in \mathcal{I}(\mathcal{C}, \mathcal{D})$. For any $\mu' \in \mathrm{Aut}(\mathcal{C})$, it is clear that $\mu \cdot \mu' \in \mathcal{I}(\mathcal{C}, \mathcal{D})$. For any other $\mu' \in \mathcal{I}(\mathcal{C}, \mathcal{D})$, we get $\mu' = \mu \cdot \mu^{-1} \cdot \mu'$ with $\mu^{-1} \cdot \mu' \in \mathrm{Aut}(\mathcal{C})$. □

As a consquence, for any two codes $\mathcal{C}$ and $\mathcal{D}$, if $\mathcal{I}(\mathcal{C}, \mathcal{D})$ is non-empty, this set is as large as $\mathrm{Aut}(\mathcal{C})$, which is therefore as large as $\mathrm{Aut}(\mathcal{D})$.

Finally, we discuss a central hardness problem for Part I of this thesis, which studies cryptography based on isometries.

Problem 2.1 (Matrix Code Equivalence)**.** Given two k-dimensional matrix codes $\mathcal{C}, \mathcal{D} \subseteq \mathbb{F}_q^{n \times m}$, find, if any, a map $(\mathbf{A}, \mathbf{B}) \in \mathcal{I}_{n,m}(q)$ such that for all $\mathbf{C} \in \mathcal{C}$, it holds that $\mathbf{ACB} \in \mathcal{D}$.

We often refer to Matrix Code Equivalence by MCE. Its hardness has been studied previously [40, 114], and we extend this research in Chapter 4.

Group action from isometries.

The group of isometries $\mathcal{I}_{n,m}(q)$ acts on the set X of k-dimensional matrix codes by simply mapping any code $\mathcal{C} \in X$ to $\mathbf{AC}\mathbf{B} \in X$ for an isometry $(\mathbf{A}, Bb)$. By abuse of notation, we refer to this group action also by MCE, which is justified as vectorisation for this group action of $\mathcal{I}_{n,m}(q)$ on X is precisely MCE.

With respect to the properties given in Section 1.2.1, we find that MCE is neither commutative, nor transitive, nor free. However, MCE is *efficient*, and so it can be used to construct sigma protocols, as is studied in Chapter 5.

The orbits of X under this group action are the classes of equivalent codes, and the stabilizer group of a code $\mathcal{C}$ is the automorphism group $\mathrm{Aut}(\mathcal{C})$. By the orbit-stabilizer theorem, we find the orbit of $\mathcal{C}$ by the action of $\mathcal{I}_{n,m}(q)/\mathrm{Aut}(\mathcal{C})$. For a cryptographic group action, we want the largest orbit and thus a trivial automorphism group. By restricting X to such an orbit, the restriced group action becomes transitive and free. Finding the automorphism group of a code is a difficult problem, and it seems infeasible to determine $\mathrm{Aut}(\mathcal{C})$ in polynomial time, given $\mathcal{C}$. It is however folklore that random codes have trivial automorphism groups, which we study in Chapter 6.

Chapter 3

Curves & Isogenies

Parts II and III of this thesis concern themselves with two subareas of isogeny-based cryptography, i.e. cryptography based on isogenies between abelian varieties. Luckily, most of this thesis concerns itself with isogenies between elliptic curves, which are well-known objects to any cryptographer, and a small part on (isogenies between) hyperelliptic curves of genus 2. This chapter introduces the main players: curves, their isogenies and pairings on curves. We assume basic familiarity with elliptic curves, such as an understanding of Silverman [272] or Galbraith [153], and a basic understanding of quaternion algebras, for which Voight [289] is our main reference.

3.1 Curves

All curves in this thesis are smooth, projective varieties of dimension 1.

Definition 3.1. An *elliptic curve* $(E, \mathcal{O})$ is a smooth projective curve E of genus 1 together with a rational point $\mathcal{O} \in E$. An elliptic curve is defined over k, whenever E is defined over k as a curve, and $\mathcal{O} \in E(k)$.

Elliptic curves are pleasant to work with: we do not have to consider the more abstract algebraic geometry, discussing polarizations, duals and Jacobians, as each elliptic curve comes with a canonical principal polarization, and $E(k)$ is furthermore isomorphic to its Jacobian, ensuring a group structure on this set of points. Even more pleasantly, the fields we are working over never have characteristic 2 or 3, which ensures that every elliptic curve has an isomorphic

curve E in *short Weierstrass form* defined by the affine equation

$$E : y^2 = x^3 + ax + b, \qquad a, b \in k,$$

with the set of points $E(k)$ precisely those points $P = (x, y)$ with $x, y \in k$ such that the equation holds. Such a curve is smooth, i.e. non-singular whenever $\Delta(E) = -16(4a^3 + 27b^2)$ is non-zero. Although E is given in affine form, we assume the reader understands that, properly speaking, we imply the projective closure with $\mathcal{O} = (0 : 1 : 0)$ the *point at infinity* not corresponding to any affine point.

In practice, isogeny-based cryptography uses different curve models too, as the cost of crucial operations highly depends on the choice of model. A popular model was given by Montgomery [218].

Definition 3.2. A *Montgomery curve* E over k of characteristic $\operatorname{char} k > 2$ is an elliptic curve with affine equation

$$E : By^2 = x^3 + Ax^2 + x, \qquad A, B \in k$$

with $\Delta(E) = B(A^2 - 4)$ non-zero.

Whenever we work with Montgomery curves in this thesis, we can choose $B = 1$, and we refer to $A \in k$ as the *Montgomery coefficient* of the curve. This reduces smoothness to $A \neq \pm 2$. Furthermore, in such cases, or whenever it is clear, E_A refers to the Montgomery curve with Montgomery coefficient A. In particular, E_0 refers to the Montgomery curve with the affine equation

$$E_0 : y^2 = x^3 + x.$$

This curve has many 'nice' properties, which make E_0 appear frequently in isogeny-based cryptography as a system parameter.

Curve arithmetic using only the x-coordinate.

Even the very first articles introducing elliptic curves in cryptography [187, 216], Miller and Koblitz independently noted that one could perform most operations required for elliptic curve cryptography using only the x-coordinate of points $P \in E(k)$. In particular for curves in Montgomery form, such x-only approaches are significantly faster. This holds not only for classical ECC, but also for current-day isogeny-based cryptography.

As the x-coordinate of a point is equal for both P and $-P$, mathematically working with x-only arithmetic means an identification of these two. That is,

we work on E with equivalence between P and $-P$. This turns out to be an algebraic variety $\mathcal{K}_E = E/\langle \pm 1 \rangle$ isomorphic to the projective line $\mathbb{P}^1$, named the *Kummer line* of E. The Kummer line $\mathcal{K}_E$ does not inherit the group structure of E due to the identification of P and $-P$. Nevertheless, $\mathcal{K}_E$ is a *pseudo-group*, which means we can still perform scalar multiplication and differential addition. This turns out to be enough for most cryptographic applications, such as a Diffie-Hellman key exchange based on the group $E(k)$, as it only requires the computation of $P \mapsto [n]P$ for $n \in \mathbb{Z}$.

3.1.1 Hyperelliptic curves

Beyond elliptic curves, we require an understanding of hyperelliptic curves. Such objects are well-studied in cryptography for their use in hyperelliptic curve cryptography (HECC) since Koblitz [188]. Hyperelliptic curves are curves of genus $g > 1$ with a ramified double cover of the projective line. As we assume we are working over fields with $\operatorname{char} k \neq 2$, such hyperelliptic curves are isomorphic to curves with a nice affine model.

Lemma 3.3. Any hyperelliptic curve over k, with $\operatorname{char} k \neq 2$, is isomorphic over k to a curve $\mathcal{C}$ given by the affine model

$$\mathcal{C} : y^2 = f(x), \qquad f(x) \in k[x]$$

with $\deg f = 2g + 1$ or $\deg f = 2g + 2$. The curve is smooth if $\gcd(f, f') = 0$, e.g. f has no repeated roots.

The polynomial $f(x)$ may have roots $w_i \in k$, which implies points $(w_i, 0) \in \mathcal{C}(k)$. These points are the *Weierstrass points* of $\mathcal{C}$. In cryptographic applications, we again often require a special curve model suitable for our purposes. In the case of genus 2 hyperelliptic curves, a particularly interesting curve model is given by Rosenhain [255].

Definition 3.4. A hyperelliptic curve $\mathcal{C}$ over k of genus 2 is in *Rosenhain form* if we have

$$\mathcal{C} : y^2 = x(x-1)(x-\lambda)(x-\mu)(x-\nu), \qquad \lambda, \mu, \nu \in k.$$

The coefficients λ, μ, ν are called the *Rosenhain invariants*.

Hyperelliptic curves are isomorphic over k to a curve in Rosenhain form when $f(x)$ splits in $k[x]$, or equivalently, all Weierstrass points are rational. In

this situation, by a Möbius transform κ, one can choose three Weierstrass points and map these to $\mathcal{O}$, $(0,0)$ and $(1,0)$. This determines an isomorphism κ, which determines $(\lambda,0)$, $(\mu,0)$ and $(\nu,0)$ as the images of the other three Weierstrass points. We get a total of 720 possible choices for such a map κ, and therefore 720 possible Rosenhain forms for any hyperelliptic curve.

Unfortunately, the set of points $\mathcal{C}(k)$ no longer have a group structure, as they did in the case of elliptic curves. Fortunately, to every hyperelliptic curve we can associate an abelian variety, the *Jacobian* $\mathcal{J}_\mathcal{C}$ of $\mathcal{C}$, which does carry the structure of the group[1]. Formally, a description of the curve provides us with a canonical principal polarization, and so we can understand $\mathcal{J}_\mathcal{C}$ as the *Picard variety of degree 0* $\mathrm{Pic}^0_k(\mathcal{C})$, that is, the group of degree zero divisors quotiented out by principal divisors. For the purpose of this thesis, these formalities are of lesser importance, and we refer the reader to Cassels and Flynn [75] for an elaborate introduction. We mainly need to understand how elements of $\mathcal{J}_\mathcal{C}$ arise from points on $\mathcal{C}$, how we represent such elements, and how these form a group. We keep our introduction slightly informal, for brevity and clarity.

There is a map $\mathcal{C}^{(g)} \to \mathcal{J}_\mathcal{C}$, which maps tuples of points $P = (P_1, \ldots, P_g) \in \mathcal{C}$ to divisors $D_P = (P_1) + (P_2) + \ldots + (P_g)$, up to symmetry. It is known that any element $D_P \in \mathcal{J}_\mathcal{C}$ can be associated with such a divisor with $0 \leq n \leq g$ points, which is then called *reduced*. A practical representation of such divisors is given by Mumford [223].

Definition 3.5. The *Mumford representation* of an element $D_P = \sum_{i=1}^{n}(P_i)$ with $P_i = (x_i, y_i) \in \mathcal{C}(\overline{k})$ is given by a pair of polynomials $a(x), b(x) \in \overline{k}[x]$, uniquely defined by the properties

$$a(x) = \prod x - x_i, \qquad b(x_i) = y_i$$

with $\deg a = n$ and $\deg b < \deg a$. The representation is denoted as $\langle a(x), b(x) \rangle$.

It can be shown that each reduced divisor $D_P \in \mathcal{J}_\mathcal{C}$ has a unique Mumford representation $\langle a(x), b(x) \rangle$. Furthermore, a divisor is defined over k whenever both $a(x)$ $b(x)$ are defined over k. Note that this does not imply that each $P_i \in \mathcal{C}(k)$! For example, D_P can be rational whenever this divisor contains each Galois conjugate of $P_i \in \mathcal{C}(K)$ for some finite field extension K/k.

An efficient algorithm to add elements D_P and D_Q is given by Cantor [73], which takes the Mumford representations of D_P and D_Q and returns the Mum-

[1] The set of points of an elliptic curve E form a group precisely because E is isomorphic to its Jacobian $\mathcal{J}_E$.

ford representation of $D_P + D_Q$. This algorithm is a higher-dimensional generalization of the chord-tangent construction for the group operation on elliptic curves.

Kummer surfaces.

Similar to the elliptic curve situation, arithmetic on the Jacobian $\mathcal{J}_\mathcal{C}$ using Cantor's algorithm is rather slow and not suitable for cryptography. Where for an elliptic curve E we could simply use the x-coordinate[2] which keeps enough structure to do cryptography, for hyperelliptic curves we can similarly work on a Kummer surface $\mathcal{K}_\mathcal{C} = \mathcal{J}_\mathcal{C}/\langle\pm\rangle$, for which models exists as algebraic varieties in $\mathbb{P}^3$. Several different models of Kummer surfaces are in use in cryptography. A more detailed introduction is given in Chapter 14. One of the most fun challenges in current isogeny-based cryptography is the generalization of techniques we have for Kummer lines to Kummer surfaces, which often drastically increases the complexity. Fortunately, an improved understanding of Kummer surfaces also aids our understanding of Kummer lines.

3.2 Isogenies of Elliptic Curves

As the name suggests, isogenies are the main object of interest in isogeny-based cryptography. Shor's algorithm [269] breaks most cryptography based on the hardness of the discrete logarithm problem for (hyper)elliptic curves. However, so far, no polynomial-time quantum algorithms have been found for hard problems in isogeny-based cryptography. As cryptographic primitives in this domain provide key exchanges and signatures with key or signature sizes far smaller than other areas of post-quantum cryptography, isogeny-based cryptography has seen an explosion of interest in recent years. We formally define an isogeny only between elliptic curves, although they can be defined more generally for abelian varieties and even algebraic groups.

Definition 3.6. An isogeny between elliptic curves E and E' over k is a nonzero group homomorphism

$$\varphi : E \to E'.$$

Any isogeny is a rational map, and we say that φ is defined over k whenever φ can be written as a map $(x, y) \mapsto (f_1(x, y)/f_2(x, y), g_1(x, y)/g_2(x, y))$, where f_i, g_i are polynomials in $k[x]$. Whenever an isogeny exists between two curves E

[2] More abstractly, work on a representation of the Kummer line $\mathcal{K}_E = E/\langle\pm 1\rangle$

and E', they are called *isogenous*. Surprisingly, it is easy to decide if two curves over a finite field are isogenous, due to Tate's theorem [279]: with $k = \mathbb{F}_q$, the curves E and E' are isogenous over $\mathbb{F}_q$ if and only if $\#E(\mathbb{F}_q) = \#E'(\mathbb{F}_q)$. Tate's theorem holds in more generality, and in particular also for Jacobians of hyperelliptic curves. On the other hand, providing such an isogeny $E \to E'$ given only E and E' is presumable hard.

Problem 3.1. Given two isogenous elliptic curves E and E' over $k = \mathbb{F}_q$, find an isogeny $\varphi : E \to E'$.

Any isogeny φ furthermore induces a map $\varphi^* : k(E') \to k(E)$ between the function fields of E and E' by pre-composition, and the degree of φ is defined as the degree of the induced field extension $[k(E) : \varphi^* k(E')]$. The degree is multiplicative: $\deg(\varphi \circ \psi) = \deg \varphi \cdot \deg \psi$.

An isogeny is said to be *separable* if the field extension $\varphi^* k(E')/k(E)$ is separable. In particular, whenever the degree of an isogeny is coprime to $\operatorname{char} k$, the isogeny is separable.

Separable isogenies are pleasant to work with. It can be shown that the degree of a separable isogeny over a finite field is precisely the cardinality of its kernel.

Example 3.1. Let $k = \mathbb{F}_{11}$ and let $E : y^2 = x^3 + x$ and $E' : y^2 = x^3 + 5$. then

$$\varphi : (x, y) \mapsto \left(\frac{x^3 + x^2 + x + 2}{(x-5)^2}, y \cdot \frac{x^3 - 4x^2 + 2}{(x-5)^3} \right)$$

is a separable isogeny $E \to E'$ of degree 3. The kernel of φ are the three points $\mathcal{O}$, $(5, 3)$ and $(5, -3)$.

Example 3.2. For any curve E, the map $[n] : P \mapsto [n]P$ is a seperable isogeny. As $E[n] := \ker[n] \cong \mathbb{Z}/\mathbb{Z}_n \times \mathbb{Z}/\mathbb{Z}_n$, the isogeny $[n]$ is of degree n^2.

Definition 3.7. The isogeny $\pi : (x, y) \mapsto (x^p, y^p)$ is the *Frobenius isogeny*. The Frobenius isogeny is a (purely) inseparable isogeny.

All inseparable isogenies essentially sprout from this single source, as it can be shown that any isogeny φ can be uniquely decomposed as

$$\varphi = \varphi_{\text{sep}} \circ \pi^k,$$

where φ_{sep} is a separable isogeny, up to composition with isomorphisms.

Any isogeny $\varphi : E \to E'$ naturally has a counterpart $\hat{\varphi} : E' \to E$ of the same degree called the *dual* of φ, with the property that $\hat{\varphi} \circ \varphi$ acts as multiplication by $\deg \varphi$ on E, and vice-versa that $\varphi \circ \hat{\varphi}$ acts as multiplication by $\deg \varphi$ on E'. This implies that φ is the dual of $\hat{\varphi}$, and furthermore that the isogeny $[n]$ is self-dual.

The set of isogenies between two elliptic curves E and E', together with the zero map $E \to E', P \mapsto \mathcal{O}$, is denoted $\mathrm{Hom}(E, E')$, with subsets $\mathrm{Hom}_k(E, E')$ for those isogenies defined over k. By pointwise addition $(\varphi + \psi)(P) := \varphi(P) + \psi(P)$ for $\varphi, \psi \in \mathrm{Hom}(E, E')$, we find an abelian group structure.

Computing isogenies.

The mathematical description of isogenies tells us nothing about how to find or compute such isogenies. It turns out that for separable isogenies there is a one-to-one correspondence between finite subgroups $G \leq E$ and separable isogenies $\varphi : E \to E'$, up to post-composition with an isomorphism. We use the notation E/G to denote the codomain of the isogeny φ_G associated to $G \leq E$. Velu's formulas [287] give an explicit method to compute φ given G in $\mathcal{O}(\#G)$. This means we can both compute the codomain E/G, as well as compute $\varphi(Q) \in E/G$ for any $Q \in E$.

Any separable isogeny of degree $n = \prod \ell_i^{e_i}$ with ℓ_i prime can be decomposed into a sequence of isogenies of degrees ℓ_i, which drastically improves the efficiency from $\mathcal{O}(n)$ to $\mathcal{O}(\ell)$ with $\ell = \max_i \ell_i$. Furthermore, the improvements by **velusqrt** provide a square-root speedup over Velu's formulas to compute ℓ-isogenies in $\tilde{\mathcal{O}}(\sqrt{\ell})$.

3.2.1 Endomorphisms and the Endomorphism Ring

Isogenies from E to itself are called *endomorphisms* and they turn out to be more crucial and interesting than one would at first glance surmise. We have seen two examples of endomorphisms already: the multiplication-by-n map $[n] : E \to E$ and the Frobenius $\pi : (x, y) \mapsto (x^p, y^p)$.

Definition 3.8. For an elliptic curve E over k, the set of endomorphisms of E is denoted $\mathrm{End}(E) := \mathrm{Hom}(E, E')$. With addition by $(\varphi+\psi)(P) = \varphi(P)+\psi(P)$ and multiplication $(\varphi \cdot \psi)(P)$ by composition $\varphi \circ \psi(P)$, we obtain the structure of a ring on $\mathrm{End}(E)$, called the *endomorphism ring* of E. The subring of k-rational endomorphisms is denoted $\mathrm{End}_k(E)$, or $\mathrm{End}_q(E)$ when $k = \mathbb{F}_q$.

The maps $[n] \in \mathrm{End}(E)$ for $n \in \mathbb{Z}$ define an embedding of $\mathbb{Z}$ in $\mathrm{End}(E)$, and similarly $\pi \in \mathrm{End}(E)$ then implies a rank-2 subring $\mathbb{Z}[\pi] \subseteq \mathrm{End}(E)$ for any curve E over a field of non-zero characteristic. A more natural way to study the endomorphism ring is using the *endomorphism algebra* $\mathrm{End}^0(E) := \mathrm{End}(E) \otimes_{\mathbb{Z}} \mathbb{Q}$, which is a quaternion algebra. In turn, $\mathrm{End}(E)$ is isomorphic to an order of this quaternion algebra. This neatly characterizes elliptic curves over finite fields.

Theorem 3.9. Let E be an elliptic curve over $k = \mathbb{F}_q$. Then the rank of $\mathrm{End}(E)$ as a $\mathbb{Z}$-module is either

- two, in which case we call E *ordinary*, and $\mathrm{End}(E)$ is isomorphic to a quadratic order in $\mathrm{End}^0(E)$,
- or four, in which case we call E *supersingular*, and $\mathrm{End}(E)$ is isomorphic to a maximal order in $\mathrm{End}^0(E)$.

Supersingular curves lie at the core of isogeny-based cryptography: we only consider supersingular curves in Parts II and III. The main reason is that we have a fine control on the number of points on a supersingular curve. For example, any supersingular curve E over $\mathbb{F}_p$ has $p + 1$ points, hence, as we can choose p freely, we can ensure E has points of order $\ell \mid p + 1$ with ℓ prime. This turns out to be useful in many isogeny-based protocols, due to the correspondence between subgroups and isogenies.

3.2.2 The Action of the Class Group

Assume now a curve E, either ordinary or supersingular, with an endomorphism $\alpha : E \to E$ such that α^2 acts as $[-D] : P \mapsto [-D]P$ for some $D \in \mathbb{N}$. Thus, we may identify a subring of $\mathrm{End}(E)$ with $\mathcal{O} = \mathbb{Z} + \sqrt{-D}\mathbb{Z}$, by $\iota : \sqrt{-D} \mapsto \alpha$. Let K denote a quadratic imaginairy field such that $\mathcal{O}$ is a quadratic order in K, then ι is an $\mathcal{O}$-*orientation* if it can be extended to an embedding $K \to \mathrm{End}^0(E)$ such that $\iota(\mathcal{O}) = \mathrm{End}^0(E) \cap \iota(K)$. In this section, we slightly abuse notation by not explicitly writing the embedding ι.

We quickly get an action of any ideal $\mathfrak{a} \subseteq \mathcal{O}$ on such curves: We define a kernel on E associated to $\mathfrak{a}$ by

$$E[\mathfrak{a}] = \bigcap_{\alpha \in \mathfrak{a}} \ker \alpha,$$

and use Velu's formulas to translate the kernel $E[\mathfrak{a}]$ into an isogeny $\varphi_{\mathfrak{a}} : E \to E/\mathfrak{a}$. As before, this is only efficient if the degree of $\varphi_{\mathfrak{a}}$ is smooth enough and

$E[\mathfrak{a}]$ is either rational or defined over only a small field extension. Whenever $\mathfrak{a}$ is a principal ideal (ω), the kernel $E[\mathfrak{a}]$ is simply $\ker \omega$, and so $\omega = \varphi_{\mathfrak{a}} : E \to E$. However, one can show that any non-principal (fractional) ideal $\mathfrak{a}$ is never an endomorphism, and two fractional ideals $\mathfrak{a}$ and $\mathfrak{b}$ have isomorphic codomains only when $\mathfrak{a} = (\omega)\mathfrak{b}$ for some principal ideal (ω). In other words, the class group $\mathcal{Cl}(\mathcal{O})$ of fractional ideals quotiented out by principal ideals, acts (commutatively) on the set of such curves E over k with $\mathcal{O} \subseteq \mathrm{End}(E)$, which we denote $\mathcal{E}\ell\ell_k(\mathcal{O})$, and Onuki [234] shows that this action $\mathcal{Cl}(\mathcal{O}) \times \mathcal{E}\ell\ell_k(\mathcal{O}) \to \mathcal{E}\ell\ell_k(\mathcal{O})$ is free and transitive.

In theory, assuming vectorization and parallelization is hard for such a group action, we quickly get a Diffie-Hellman-like non-interactive key exchange: given a starting curve E, Alice and Bob both sample their private keys as elements $\mathfrak{a}, \mathfrak{b} \in \mathcal{Cl}(\mathcal{O})$, their public keys are respectively $E/\mathfrak{a}$ and $E/\mathfrak{b}$, and they compute shared secrets by the action of $\mathfrak{a}$ on $E/\mathfrak{b}$ resp. $\mathfrak{b}$ on $E/\mathfrak{a}$ so that both derive $E/\mathfrak{ab}$, summarized in the following diagram.

$$\begin{array}{ccc} E & \xrightarrow{\ \mathfrak{a}\ } & E/\mathfrak{a} \\ \Big\downarrow{\scriptstyle \mathfrak{b}} & & \Big\downarrow{\scriptstyle \mathfrak{b}} \\ E/\mathfrak{b} & \xrightarrow{\ \mathfrak{a}\ } & E/\mathfrak{ab} \end{array}$$

However, the difficulty of using such a group action cryptographically lies in the efficiency of the group action: not only is it difficult to sample random elements of $\mathcal{Cl}(\mathcal{O})$, computing the action of such elements on random curves in $\mathcal{E}\ell\ell_k(\mathcal{O})$ can be terribly or impossibly slow. An approach is to find an order $\mathcal{O}$ in a quadratic imaginairy field K such that many small odd primes ℓ split in $\mathcal{O}$.

CSIDH [85] finds a beautiful solution: by choosing primes of the form

$$p = 4 \cdot \ell_1 \cdot \ldots \cdot \ell_n - 1,$$

where the ℓ_i are small odd primes, then in the order $\mathcal{O} = \mathbb{Z} + \pi\mathbb{Z}$, with $\pi = \sqrt{-p}$ in the quadratic imaginairy field $K = \mathbb{Q}(\sqrt{-p})$, each of these ℓ_i splits as

$$(\ell_i) = \mathfrak{l} \cdot \bar{\mathfrak{l}}, \qquad \mathfrak{l} = (\ell, \pi - 1), \quad \bar{\mathfrak{l}} = (\ell, \pi + 1).$$

Supersingular curves now have $\#E(\mathbb{F}_p) = p + 1$, and the Frobenius endomorphism π on E, defined over $\mathbb{F}_p$, has precisely the property that $\pi^2 = [-p]$. The set $\mathcal{E}\ell\ell_p(\mathcal{O})$ then consists precisely of those curves E with $\mathrm{End}_p(E) \xrightarrow{\sim} \mathbb{Z} + \pi\mathbb{Z}$.

Furthermore, for any such curve E, we can easily compute $E[\mathfrak{l}]$ or $E[\bar{\mathfrak{l}}]$, because by definition

$$E[\mathfrak{l}] = \ker(\ell) \cap \ker(\pi - 1),$$

thus we need both $\pi P = P$ and $\ell P = \mathcal{O}_E$, which means $E[\mathfrak{l}]$ is precisely the set of $\mathbb{F}_p$-rational points of order ℓ on E. As $\ell \mid p+1$ and supersingular curves have $\#E(\mathbb{F}_p) = p + 1$, we can easily find and compute with rational points of order ℓ, and thus compute the action of $\mathfrak{l}$, resp. $\bar{\mathfrak{l}}$ on $E \in \mathcal{E}\ell\ell_p(E)$.

We still face the difficulty of sampling random elements of $\mathcal{C}\ell(\mathcal{O})$. Thus, CSIDH requires the heuristic assumption that $\mathcal{C}\ell(\mathcal{O})$ is generated by these ideals $\mathfrak{l}_1, \ldots, \mathfrak{l}_n$, so that we can sample essentially random elements by

$$\mathfrak{a} = \prod \mathfrak{l}_i^{e_i}, \qquad e_i \in \mathbb{Z},$$

where $\mathfrak{l}_i^{-1}$ denotes $\bar{\mathfrak{l}}_i$, for large enough integers e_i. The action of $\mathfrak{a}$ can then be computed per $\mathfrak{l}_i$, that is, we decompose $\varphi_{\mathfrak{a}} : E \to E/\mathfrak{a}$ into its prime-degree components given by $\mathfrak{l}_i$.

The last problem that needs to be solved to get the efficient NIKE described above, is the efficiency of the membership test: upon receiving a curve E, how are we ensured that $E \in \mathcal{E}\ell\ell_p(\mathcal{O})$ before we act with a secret key? Precisely in the CSIDH situation, this simply requires us to verify that E is supersingular, which is again efficient. The practicality of CSIDH is the main topic of Part II.

3.2.3 The Deuring correspondence

In the previous section, we have seen that $\mathrm{End}(E)$ for supersingular curves is isomorphic to some maximal order in a quaternion algebra $\mathcal{B}_{p,\infty}$. In more generality, the world of supersingular elliptic curves and their isogenies have many properties in common with the world of quaternion algebras and their ideals. The *Deuring correspondence*, after Max Deuring [127], provides us with the general bridge between these two worlds, mainly based on the functorial equivalence given by

$$\begin{array}{rcl} \{\text{ss. curves over } \mathbb{F}_{p^2}\}/\sim & \to & \{\text{max. orders in } \mathcal{B}_{p,\infty}\}/\sim \\ E & \mapsto & \mathrm{End}(E) \end{array}$$

where $\sim$ on both sides implies equivalence up to isomorphism. This thesis does not require an in-depth understanding of quaternion algebras, or the equivalence of categories. However, we sketch the situation informally so that we can

understand SQIsign, an signature scheme whose complex signing procedure relies mainly on this correspondence. Part III of this thesis will focus only on the verification part of SQIsign, which can be understood in terms of elliptic curves and isogenies only. A reader can therefore safely skip the remainder of this section if one is only interested in SQIsign verification.

Similar to the previous section, the Deuring correspondence allows us to see isogenies $\varphi : E \to E'$ as ideals I_φ, but this time of a maximal order instead of a quadratic order. Again, the association relies on the kernel: all endomorphisms ω that vanish on $\ker \varphi$ form a left ideal I_φ of $\mathrm{End}(E)$, and similarly any left ideal I of $\mathrm{End}(E)$ give a subgroup $E[I] \subseteq E$ by the intersection of the kernel of all generators of I, which gives a unique isogeny $\varphi_I : E \to E/I$, up to post-composition with an isomorphism. The *norm* of such an ideal I, the greatest common divisor of the norms of its elements, is then equal to the degree of φ_I.

The codomain associated to such a left ideal I of $\mathcal{O}$ should then be some 'natural' maximal order with I as a right ideal. This natural object is the *right order*[3]

$$\mathcal{O}_r(I) := \{\beta \in \mathcal{B}_{p,\infty} \mid I\beta \subseteq I\},$$

which is isomorphic to the endomorphism ring of the elliptic curve E/I. In such a situation, we say that I is a *connecting ideal* of two maximal orders $\mathcal{O}_1$ and $\mathcal{O}_2$, that is, when I is a left ideal of $\mathcal{O}_1$ whose right order is isomorphic to $\mathcal{O}_2$, or vice versa.

The KLPT algorithm

The main algorithmic building block in SQIsign is a polynomial time algorithm by Kohel, Lauter, Petit, and Tignol [190], the KLPT algorithm. The idea is simple: we may often encounter isogenies $\varphi : E \to E'$ of some generic large degree d. Most likely, d is neither smooth, nor is the d-torsion rational or defined over a small extension field. If we could somehow, given φ construct another isogeny $\varphi' : E \to E'$ whose degree d' is very smooth, such as $d' = \ell^k$ for some small prime ℓ, we could easily compute φ' instead which could alleviate our burdens.

In the world of supersingular elliptic curves, this problem is presumably hard: finding such a φ' implies we find a non-trivial endomorphism $\hat{\varphi}' \circ \varphi$, which is the hard problem all isogeny-based cryptography is based on. However, if we know $\mathrm{End}(E)$, and can therefore use the Deuring correspondence to move this problem to the world of quaternion algebras by $\varphi \mapsto I_\varphi$, our problems are

[3] No pun intended.

magically solved by the KLPT algorithm, as long as the domain of φ is of a special form. We leave out the details of this special form, and simply refer to *special* maximal orders $\mathcal{O}$. Informally, we can state the KLPT algorithm as follows.

Theorem 3.10 (The KLPT algorithm)**.** Let I be a left ideal of a special maximal order $\mathcal{O}$ in $\mathcal{B}_{p,\infty}$. There exists a heuristic algorithm that returns an ideal J, equivalent to I, of norm ℓ^k for some $k \in \mathbb{N}$.

The result is wonderful: instead of working with the isogeny/ideal of ghastly degree/norm d, we can apply KLPT to work with the ideal J of smooth norm ℓ^k. By translating such an ideal back to an isogeny[4], the isogeny $\varphi_J : E \to E'$ can be efficiently computed in polynomial time.

So far, we have ignored two main problems: first, are we ensured the maximal orders we will encounter 'in the wild' are of the special form, and second, how large is k in the output of KLPT? As an answer to the first question, it turns out that the special curve $E_0 : y^2 = x^3 + x$ has an endomorphism ring of precisely the special form we need, and this is all we require for the applications in SQIsign. For the second question, the norm of the output ideal can under some heuristic assumptions be estimated as $p^{7/2}$. In practice, this means that φ_J is a rather long isogeny, but not unpractically long.

The SQIsign signature scheme

SQIsign [89, 121, 122] is a signature scheme that cleverly uses the Deuring correspondence and the KLPT algorithm to construct an identification scheme using a Sigma protocol. Although there are some significant details between subsequent versions of the scheme, the main idea is as follows.

We assume the starting elliptic curve E_0, whose endomorphism ring $\mathcal{O}_0 = \mathrm{End}(E_0)$, a special maximal order, is publicly known. Alice, the prover, samples a secret isogeny $\varphi_A : E_0 \to E_A$, and provides E_A as her public key. By knowing φ_A, she ensures she, and only she, knows $\mathrm{End}(E_A)$. Her goal is to prove this knowledge.

The Sigma protocol then starts with Alice committing to some curve E_1 which is the codomain of another random isogeny $\varphi_{\mathrm{com}} : E_0 \to E_1$. She sends E_1 to Bob, the verifier. Again, only Alice can compute $\mathrm{End}(E_1)$ as she keeps φ_{com} hidden. Bob samples a random isogeny $\varphi_{\mathrm{chall}} : E_1 \to E_2$ and communicates φ_{chall} to Alice, which allows her to compute $\mathrm{End}(E_2)$ using her knowledge

[4] A highly non-trivial and slow task, which we will not discuss in detail in this thesis.

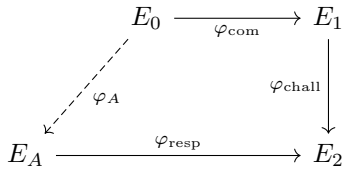


Figure 3.1: The SQIsign protocol as a diagram.

of $\mathrm{End}(E_1)$. Alice thus knows $\mathrm{End}(E_A)$ and $\mathrm{End}(E_2)$, and furthermore can compute the ideals I_{secret}, I_{com}, I_{chall}. Thus, she can compute an ideal I_{resp} as the product $I_{\text{chall}} \cdot I_{\text{com}} \cdot \bar{I}_{\text{secret}}$ to which she can apply KLPT to obtain a smooth equivalent ideal J. She then computes the associated isogeny $\varphi_J : E_A \to E_2$, which is her response $\varphi_{\text{resp}} := \varphi_J$. Bob verifies the response by recomputing $\varphi_{\text{resp}} : E_A \to E_2$ and confirms that the codomain is indeed (isomorphic to) the challenged E_2. The protocol is summarized by the Figure 3.1.

The SQIsign works ingeniously solve many obstacles that are not visible in the above description to allow Alice to sign, e.g. compute φ_{resp}, in practice, and to ensure the security of the scheme. However, with our main focus in Part III being SQIsign verification, we do not go in-depth on the signing procedure. Instead, we focus on the verification procedure.

The ideal J associated with φ_{resp} is an output of the KLPT algorithm and therefore, as noted before, of rather large norm ℓ^k. In practice, we find $\deg \varphi_{\text{resp}} \approx p^{15/4}$. For NIST Level I security, where $\log p \approx 2\lambda = 256$ bits, the current parameters give $\deg \varphi_{\text{resp}} = 2^e$ with $e \approx 1000$. The maximal available rational $2^\bullet$-torsion on supersingular curves is determined by the largest f such that $2^f \mid p+1$. For the given prime for NIST Level I security, this gives $f = 75$, and thus, in verification, we recompute φ_{resp} as a composition of $n := \lceil \frac{e}{f} \rceil$ isogenies $\varphi_i : E^{(i)} \to E^{(i+1)}$ of degree 2^f, such that $\varphi_{\text{resp}} = \varphi_n \circ \ldots \varphi_1$. In Part III, in particular in Chapter 12 and Chapter 13, we analyze the performance of SQIsign verification in detail.

3.3 Moving to higher-dimensions

During the years that spanned the work in this thesis, SIDH/SIKE [**SIDH**, 176] was spectacularly broken by higher-dimensional isogenies [77, 204, 252] using a (generalization of) Kani's lemma [184] combined with Zahrin's trick. From

the ashes of SIDH arose a phoenix: higher-dimensional isogenies allow for an incredible constructive tool that increased the general flexibility for cryptodesign. This has sprouted numerous new schemes and will continue to provide significant improvements in isogeny-based cryptography as this paradigm shift to higher-dimensional isogenies is explored.

For this thesis, we sketch a quick understanding of higher-dimensional isogenies and their applications in variants of SQIsign. Although a detailed understanding is not required, a high-over understanding is useful to grasp the current landscape of isogeny-based cryptography and the precise role Chapters 12 to 14 play in our exploration of this landscape.

3.3.1 Higher-dimensional Isogenies

By higher-dimensional isogenies, we refer to isogenies between Abelian varieties of dimension larger than 1. In practice, we usually encounter 2-dimensional Abelian varieties over some field k, and these are easily classified. Let A be a 2-dimensional Abelian variety, then a) A is (isomorphic to) the Jacobian of some hyperelliptic curve $\mathcal{C}$ of genus 2 over k, or b) A is (isomorphic to) the product of two elliptic curves E and E' over k, or c) A is (isomorphic to) the Weil restriction of an elliptic curve E over K, where K/k is a degree-2 field extension.

We specifically focus on isogenies $f : A \to B$ whose kernel as a subgroup is isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$, and furthermore maximally isotropic[5]. Such isogenies as maps between hyperelliptic curves are precisely the *Richelot isogenies*, which have appeared in hyperelliptic curve cryptography literature. In such cases, we say that f is a $(2, 2)$-isogeny. A concatenation of n such $(2, 2)$-isogenies, modulo some requirements on cyclicity, is then a $(2^n, 2^n)$-isogeny.

Kani's Lemma

Kani's lemma [184] now allows for a very powerful tool: given the right setup, two 1-dimensional isogenies φ and ψ between elliptic curves can be associated with a specific 2-dimensional isogeny Φ. Computing Φ, as a 2-dimensional isogeny[6], will allow us to derive information on φ and ψ that was practically

[5] A technical requirement that we will not explore in more detail.

[6] The reason we only consider $(2^n, 2^n)$-isogeny is precisely that we can only efficiently compute 2-dimensional isogenies of this degree, although recent progress also explores $(3, 3)$-isogenies and beyond [100].

unavailable using only 1-dimensional isogenies. The right setup in this situation is an *isogeny diamond*.

Definition 3.11. Let $\psi : E_1 \to E_2$ and $\varphi : E_1 \to E_3$ be two isogenies of coprime degrees. An *isogeny diamond* is a square of isogenies

$$\begin{array}{ccc} E_1 & \xrightarrow{\varphi} & E_3 \\ \downarrow{\scriptstyle \psi} & & \downarrow{\scriptstyle \psi'} \\ E_2 & \xrightarrow{\varphi'} & E_4 \end{array}$$

such that $\varphi \circ \psi' = \psi \circ \varphi'$, with $\deg \varphi = \deg \varphi'$ and $\deg \psi = \deg \psi'$.

Kani's lemma provides the crucial 2-dimensional isogeny that arises from an isogeny diamond.

Lemma 3.12 (Kani's lemma)**.** Consider an isogeny diamond given by φ, φ', ψ and ψ', and let $N = \deg \varphi + \deg \psi$. then the map

$$\Phi : E_2 \times E_3 \to E_1 \times E_4,$$

given in matrix form as

$$\begin{pmatrix} \hat{\psi} & \hat{\varphi} \\ -\varphi' & \psi \end{pmatrix},$$

is an (N, N)-isogeny of abelian surfaces with kernel

$$\ker \Phi = \{(\varphi(P), \psi(P)) \mid P \in E_1[N]\}.$$

The main idea is now simple: Although φ and ψ individually might be difficult to compute, with the right set-up Φ might be efficient to compute, and can provide the same information.

Embedding isogenies

A practical example of the above idea is given by Robert [253] to *embed an isogeny* φ of dimension 1 into a higher-dimensional isogeny Φ, so that evaluation of Φ allows us to evaluate φ.

Consider as a first example an isogeny $\varphi : E \to E'$, such that $2^n - \deg\varphi = x^2$ for some $n \in \mathbb{N}$ and $x \in \mathbb{Z}$. In such a case, we construct the isogeny diamond

$$\begin{array}{ccc} E & \xrightarrow{\varphi} & E' \\ \downarrow{\scriptstyle [x]} & & \downarrow{\scriptstyle [x]} \\ E & \xrightarrow{\varphi} & E' \end{array}$$

where $[x]$ denotes the simple-to-compute multiplication-by-x map $P \mapsto [x]P$. Kani's lemma now implies that $\Phi : E \times E' \to E \times E'$ is an (N, N)-isogeny with $N = \deg\varphi + \deg[x] = 2^n$, and so

$$\ker\Phi = \{(\varphi(P), [x]P) \mid P \in E[2^n]\}.$$

Thus, if we know $\varphi(P)$ for $P \in E[2^n]$ and we simply compute $[x]P$, we can compute $\varphi(Q)$ for any other $Q \in E$ using Φ, which we can efficiently compute as a $(2^n, 2^n)$-isogeny. Thus, we can 'embed' the 1-dimensional isogeny φ into the 2-dimensional isogeny Φ, if $\deg\varphi$ differs from a power of 2 by precisely a square.

This situation is highly unlikely to happen for any random isogeny φ, as only very few integers differ from a power of 2 by precisely a square. However, by Legendre's four-square theorem, any natural number can be represented as the sum of four squares. This inspired Robert [253] to generalize the result by Kani to a result between 8-dimensional abelian varieties $\Phi : E^4 \times E'^4 \to E^4 \times E'^4$. Then, instead of hoping $\deg\varphi$ precisely fits $2^n - x^2$ for some $x \in \mathbb{Z}$, we find four integers $x_1, x_2, x_3, x_4 \in \mathbb{Z}$ such that $2^n - \deg\varphi = x_1^2 + x_2^2 + x_3^2 + x_4^2$. A similar setup as before then allows us to embed the 1-dimensional isogeny φ into an 8-dimensional isogeny Φ. Using several technical tricks, one can often already achieve a similar result with a 4-dimensional embedding.

3.3.2 Higher-dimensional SQIsign

The technique to embed 1-dimensional isogenies into higher dimensional isogenies is powerful and versatile. For example, whenever the 1-dimensional isogeny φ is of large non-smooth degree but embeddable in a smooth higher-dimensional isogeny Φ, we can simply evaluate φ on points using Φ, which is efficient to compute.

This has led to a revolution in many areas of isogeny-based cryptography, and most notably in SQIsign: instead of having to use KLPT to obtain a smooth

response isogeny (of fairly large degree), we may embed the response isogeny into a smooth higher-dimensional isogeny Φ. Verification is then reduced to the efficient recomputation of Φ. To achieve this requires solving several technical difficulties which was done by Dartois, Leroux, Robert, and Wesolowski [116]. The resulting signature scheme, using 4- or 8-dimensional isogenies is named SQIsignHD and allows significantly faster signing than the original SQIsign. A drawback is slow verification, as computing 4- or 8-dimensional isogenies is much more complex than 1-dimensional isogenies.

Using an algorithmic building block by Nakagawa and Onuki [224], three works independently from each other were able to improve SQIsignHD to achieve verification using 2-dimensional isogenies only. These works, SQIsign2D-West [**SQIsign2D-west**], SQIsign2D-East [225] and SQIPrime [**SQIprime**], achieve both fast signing and verification, ushering in an era of practical higher-dimensional isogeny-based cryptography.

In summary, SQIsign2D is currently the most practical approach to isogeny-based signatures, with relatively fast signing and verification times on par with 1-dimensional SQIsign. However, 1-dimensional SQIsign is still an interesting topic for research, as many questions remain unanswered when comparing the real-world practicality of these two approaches. Chapter 13 deals with some of these questions.

3.4 Pairings

Like dogs to humans, pairings are great companions for all of isogeny-based cryptography. Pairings, bilinear maps between Abelian groups, appear naturally in many versatile use cases due to their interesting properties, in particular in relation to isogenies of abelian varieties. This thesis contains many of these applications of pairings, mostly using the Tate pairing[7] although the Weil pairing is more commonly used in general. Both are usually used in very specific and concrete situations, yet we will introduce these in their general forms, following Bruin [66], Garefalakis [155], and Silverman [271]. This allows us to provide more intuition to the reader in what these pairings are fundamentally doing. More properties, details and computational aspects are widely described in the literature, see [98, 150, 152]. Inspiration is also drawn from [**castryck2023weak**, 254].

[7] More properly named the Tate-Lichtenbaum pairing, or also the Frey-Rück pairing, most authors stick to *Tate pairing*, perhaps simply because it is easiest to write.

3.4.1 The Weil Pairing

Let $\varphi : E \to E'$ be an isogeny with $\operatorname{char} k \nmid \deg \varphi$. To φ, we can associate the generalized φ-Weil pairing

$$e_\varphi : \ker \varphi \times \ker \hat{\varphi} \to \overline{k}^*,$$

which is a bilinear, non-degenerate pairing invariant under the action of $\mathrm{Gal}(\overline{k}/k)$, taking as values m-th roots in $\overline{k}^*$, for m such that $\ker \varphi \subset E[m]$. We denote the cyclic group of m-th roots by μ_m. Several other desirable properties are described in [**castryck2023weak**, 152].

Taking $\varphi = [m] : E \to E$, we recover the more traditional Weil pairing

$$e_m : E[m] \times E[m] \to \mu_m.$$

Computing $e_m(P, Q)$ for two points $P, Q \in E[m]$ can be done using two Miller functions $f_{m,P}, f_{m,Q} \in \overline{k}(E)$, defined by the property that

$$\operatorname{div} f_{m,P} = m(P) - m(\mathcal{O}), \quad \operatorname{div} f_{m,Q} = m(Q) - m(\mathcal{O}).$$

We want to evaluate $f_{m,P}$ on the divisor $(Q) - (\mathcal{O})$, and vice versa. However, as $\mathcal{O}$ is a root of $f_{m,P}$, we must first find a divisor D_Q equivalent to $(Q) - (\mathcal{O})$, with support disjoint from P, OO, and similar for D_P. Then

$$e_m(P, Q) = f_{m,P}(D_Q)/f_{m,Q}(D_P).$$

Computing the Miller functions $f_{m,P}$ and $f_{m,Q}$ can be done using Miller's algorithm [215].

One exemplary use case of the Weil pairing is computing a change of basis matrix for two bases P, Q and P', Q' of $E[m]$. We must have

$$\begin{pmatrix} P' \\ Q' \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} P \\ Q \end{pmatrix}$$

for values $a_i \in \mathbb{Z}/m\mathbb{Z}$. An easy way to compute these values is by

$$a_1 = e_m(P, P'),\ a_2 = e_m(Q, P'),\ a_3 = e_m(P, Q'),\ a_4 = e_m(Q, Q').$$

3.4.2 The Tate Pairing

We focus on the case $k = \mathbb{F}_q$. Let $\varphi : E \to E'$ be an isogeny with $\ker \varphi \subset E[m] \subset E[q-1]$. To φ, we can associate the generalized φ-Tate pairing

$$T_\varphi : \ker \varphi(\mathbb{F}_q) \times \operatorname{coker} \hat{\varphi}(\mathbb{F}_q) \to \mathbb{F}_q^*/\mathbb{F}_q^{*m},$$

which is a bilinear, non-degenerate pairing invariant under the action of $\mathrm{Gal}(\overline{k}/k)$. The values are equivalence classes modulo m-th powers. Sometimes this is enough, but in practice, we often raise the value to the power $(q-1)/m$ to obtain specific m-th roots. We then call the Tate pairing *reduced*, and denote by t_φ the composition of T_φ with the exponentiation $z \mapsto z^{(q-1)/m}$, which is an isomorphism $\mathbb{F}_q^*/\mathbb{F}_q^{*m} \to \mu_m(\mathbb{F}_q)$. Several other desirable properties are described in [**castryck2023weak**, 152].

Computing $t_\varphi(P,Q)$ for $P \in \ker\varphi$, $Q \in E'$ can be done using the Miller function $f_{m,P}$ evaluated on a divisor D_Q equivalent to $(Q)-(\mathcal{O})$, with disjoint support, where again we can compute $f_{m,P}$ using Miller's algorithm. Taking $\varphi = [m]$ recovers the more traditional Tate pairing

$$t_m : E[m] \times E(\mathbb{F}_q)/[m]E(\mathbb{F}_q) \to \mu_m.$$

The case $\varphi = [m]$ also shows an immediate use case of the Tate pairing: as it is non-degenerate, we can identify the points $Q \in [m]E(\mathbb{F}_q)$ precisely as those points where $t_m(P,Q) = 1$ for all $P \in E[m]$. Conversely, this implies that $t_m(P,Q) \neq 1$ for some $P \in E[m]$ and $Q \in E(\mathbb{F}_q)$ immediately implies that $Q \notin [m]E(\mathbb{F}_q)$. Even more specific for $[m] = [2]$, for any Montgomery curve E_A, which has $(0,0) \in E[2]$, this implies that if $t_2((0,0),Q) \neq 1$, then $Q \notin [2]E$. The value $t_2((0,0),Q)$ can be computed to be precisely the quadratic reciprocity of x_Q. For supersingular curves E_A with $2^f \mid p+1$, this thus implies that the order of Q is divisible by 2^f, whenever x_Q is non-square. Thus, a classical result on the image of E under doubling [171, Thm. 4.1] can concisely be described using the Tate pairing. More generally speaking, we can use the φ-Tate pairing together with $\ker\varphi$ to determine the coset in $\operatorname{coker}\hat{\varphi}(\mathbb{F}_q)$ in which a point $Q \in E$ lies.

Part I

Isometries

Intro

Some introduction on isometries testing cref Chapter 6 in Part I.

Chapter 4

Hardness analysis

4.1 Placeholder

The paper will go here.

4.2 Introduction

Given two mathematical objects of the same type, an equivalence problem asks the question whether there exists an equivalence map between these objects – and how to find it – that preserves some important property of the objects. These kind of problems come in different flavors depending on the objects – groups, graphs, curves, codes, quadratic forms, etc. – and quite often the interesting maps are isomorphisms or isometries. Interestingly, equivalence problems are one of the core hard problems underlying the security of many public-key cryptosystems, especially post-quantum ones. Many multivariate and code-based systems employ an equivalence transformation as a hiding technique, and thus intrinsically rely on the assumption that a particular equivalence problem is intractable, for example [**Patarin96**, 54, 74, 124, 130, 206, 231] . In addition, quite remarkably, a hard equivalence problem gives rise to a Sigma protocol and, through the Fiat-Shamir transform, a provably secure digital signature scheme [**FS87**]. This idea has been revisited many times, being the basis of several signature schemes [**Patarin96**, 33, 56, 117, 119, 158]. Two such schemes actually appeared during the writing of this manuscript [134, 277] as a result of NIST's announcement for an additional fourth round on signatures in the post quantum

standardization process [228]. Understanding the hardness of these equivalence problems is an essential task in choosing appropriate parameters that attain a certain security level of these cryptographic schemes. Comment Simona: LESS case....

One of these problems is the Code Equivalence problem, which given two codes (with the Hamming metric), asks for an isometry (equivalence transformation that preserves the metric) that maps one code to the other. It was first studied by Leon [197] who proposed an algorithm that takes advantage of the Hamming weight being invariant under monomial permutations. It was improved very recently by Beullens [49] using collision-based techniques. Sendrier [267] proposed another type of algorithm, the Support Splitting Algorithm (SSA), that is exponential in the dimension of the hull (the intersection of a code and its dual). Interestingly, in low characteristic, random codes have very small hull, rendering the problem easy.

In this work, we focus on the code equivalence problem, but for matrix codes (an $\mathbb{F}_q$-linear subspace of the space of $m \times n$ matrices over $\mathbb{F}_q$) endowed with the rank metric - *Matrix Code Equivalence* (MCE). Evaluating the hardness of this problem is only natural – rank-based cryptography has become serious competition for its Hamming-based counterpart, showing superiority in key sizes for the same security level [13, 14, 39, 209]. MCE, and variations of it, has been introduced by Berger in [**Berger03**], but it was only recently that the first concrete statements about its hardness were shown in two concurrent independent works publicly available as preprints [114, 163][1]. Couvreur et al. [114] showed that MCE is at least as hard as the (Monomial) Code Equivalence problem in the Hamming metric, while for only right equivalence, or when the codes are $\mathbb{F}_{q^m}$-linear, the problem becomes easy. Grochow and Qiao [163] show the same reduction from (Monomial) Code Equivalence to MCE but using a completely different technique of linear algebra coloring gadgets which makes the reduction looser than the one in [114].

4.2.1 Our contributions

In this paper, we investigate the theoretical and practical hardness of the Matrix Code Equivalence (MCE) problem. Our contributions can be summarized as follows:

[1] The two works use different techniques and terminology, and seem to be mutually unaware of the line of work preceding the other. In [163] the MCE problem is referred to as Matrix Space Equivalence problem and 3-Tensor Isomorphism problem.

First, we link in a straightforward manner the MCE problem to hard problems on systems of polynomials by showing that MCE is polynomial-time equivalent to the Bilinear Maps Linear Equivalence (BMLE) problem. We then extend this result by proving that MCE is polynomial-time equivalent to the Quadratic Maps Linear Equivalence (QMLE) problem, under a mild assumption of trivial automorphism groups of the codes in question. While our technique fails to give a proof without this assumption, we consider it to be reasonable for randomly generated codes and for cryptographic purposes. As the QMLE problem is considered to be the hardest equivalence problem for systems of multivariate polynomials, it is essential to understand under which conditions MCE and QMLE reduce to one another. Note that previous work[2] requires much stronger assumptions for related results [38, 147, 163], such as algebraically closed fields or existence of square or third roots. Our reduction to QMLE is tight and gives a tight upper bound on the hardness of MCE. Furthermore, it is very simple, thus establishing connection between code equivalence problems and polynomial equivalence problems that is usable in practice. This is the basis of our contributions on the practical hardness of MCE.

Second, using similar techniques, and under the same assumptions, we show that MCE is polynomial-time equivalent to other code equivalence problems, such as Matrix Sum-Rank Code Equivalence Problem, and at least as hard as the Vector Sum-Rank Code Equivalence Problem. All these connections and our results are visualized in Figure 4.1.

On the practical side, we provide the first two non-trivial algorithms for solving MCE using the connection to QMLE. The first algorithm is a generalization of a known birthday-based algorithm for QMLE [63, 64] for systems of polynomials with the same number of variables as equations. We show that this algorithm extends to different invariance properties and code dimensions, which helps us prove complexity of $q^{\frac{2}{3}(n+m)}$ up to a polynomial factor for MCE for $m \times n$ matrix codes. The algorithm is probabilistic with success probability that can be made arbitrarily close to 1, and can be used for code dimensions up to $2(m+n)$. For larger dimensions, the complexity becomes $q^{(n+m)}$ up to a polynomial factor, but the algorithm is deterministic. The birthday-based algorithm for QMLE [64] assumed existence of a polynomial-time solver for the inhomogeneous variant of QMLE to achieve these complexities. Interestingly, due to the specific instances of the inhomogeneous QMLE arising from the collision search, the problem seems to be much harder than for random instances

[2] We were made aware of this line of work by one of the authors after our results were first presented at WCC 2022.

– a fact previously overlooked in [64]. In contrast, [63] uses a non-polynomial estimate for this solver. We analyse the most recent results regarding such solvers, and show that for parameter sets of cryptographical interest the above complexities hold, even if such solvers do not achieve polynomial time.

Our second algorithm uses the bilinear structure of the polynomials arising from MCE. Because matrix codes show symmetry between the parameters, as given in Lemma 4.14, the complexity of solving MCE using this result and Algorithm 2 becomes $q^{\min\{m,n,k\}}$ up to a polynomial factor. The algorithm is deterministic and does not require a polynomial-time solver for the inhomogeneous QMLE instance, but the weaker assumption that the solver has a complexity of $\mathcal{O}(q^{\min\{m,n,k\}})$ at most. This general result, valid for any m,n, and k, is summarized in our main result Theorem 4.24.

Lastly, to verify the results and performance of these algorithms in practice, we have implemented both and solved randomly generated instances of MCE for different parameter sets. The results of these experiments show that our assumptions are reasonable and the above complexities hold. Our implementations are open source and available at:

https://github.com/mtrimoska/matrix-code-equivalence

4.3 Preliminaries

Let $\mathbb{F}_q$ be the finite field of q elements. $\mathrm{GL}_n(q)$ and $\mathrm{AGL}_n(q)$ denote respectively the general linear group and the general affine group of degree n over $\mathbb{F}_q$.

We use bold letters to denote vectors $\mathbf{a}, \mathbf{c}, \mathbf{x}, \ldots$, and matrices $\mathbf{A}, \mathbf{B}, \ldots$. The entries of a vector $\mathbf{a}$ are denoted by a_i, and we write $\mathbf{a} = (a_1, \ldots, a_n)$ for a (row) vector of dimension n over some field and $\mathbf{a}^\top = (a_1, \ldots, a_n)^\top$ for the respective column vector. Similarly, the entries of a matrix $\mathbf{A}$ are denoted by A_{ij}. A matrix $\mathbf{A}$ is called symmetric if $\mathbf{A}^\top = \mathbf{A}$ and skew-symmetric if $\mathbf{A}^\top = -\mathbf{A}$. The space of matrices over $\mathbb{F}_q$ of size $m \times n$ is denoted $\mathcal{M}_{m,n}(q)$. The set of k-subsets of $\mathcal{M}_{m,n}(q)$ is denoted by $\mathcal{M}_{m,n}^{[k]}(q)$.

Random sampling from a set S is denoted by $a \xleftarrow{\$} S$. We use the notation $\tilde{\mathcal{O}}(f(n))$ to denote $\mathcal{O}(f(n)\log(f(n)))$ whenever we want to omit polynomial factors from the complexity expression. We use the notation $f = \Theta(g)$ whenever f is bounded from below and above by g asymptotically.

For a computational problem P, if we want to emphasize a list of parameters p defining the size of the inputs and the input set S, we will use the notation

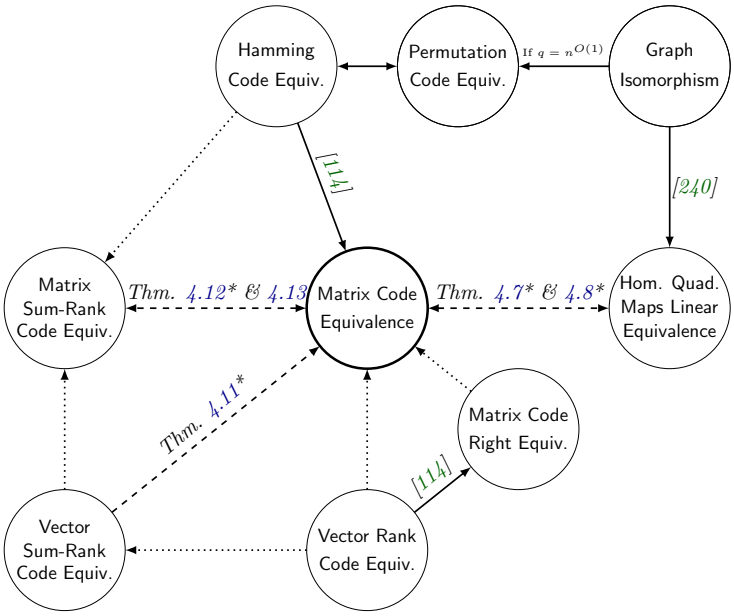


Figure 4.1: Reductions around Matrix Code Equivalence. Dashed arrows are contributions from this work, dotted arrows are trivial reductions. "$\mathsf{A} \longrightarrow \mathsf{B}$" means that "Problem A reduces to Problem B in polynomial time". Results with * assume trivial automorphism groups.

$\mathsf{P}(p, S)$. If these are not relevant, clear from context, or the set S is the entire universe U, we will use only $\mathsf{P}(p)$ or P.

Our results in Section 4.4 use the following standard notion of Turing reduction.

Definition 4.1. Given two computational problems $\mathsf{P}(\mathsf{p}, S)$ and $\mathsf{P}'(\mathsf{p}', S')$, with inputs coming from sets S and S' respectively, we say that $\mathsf{P}(\mathsf{p}, S)$ reduces to $\mathsf{P}'(\mathsf{p}', S')$ if there exists a probabilistic polynomial-time oracle machine $\mathcal{B}$ such that for every oracle $\mathcal{A}$ that solves $\mathsf{P}'(\mathsf{p}', S')$ on all inputs from S', $\mathcal{B}^{\mathcal{A}}$ ($\mathcal{B}$ given access to $\mathcal{A}$) solves $\mathsf{P}(\mathsf{p}, S)$ on all inputs from S.

Note that our reductions are meaningful only as worst-case to worst-case, and therefore in the definition we include the statement that the oracles solve

the problems on all inputs. On the other hand, we do not always require the oracle $\mathcal{A}$ to be able to solve P' on the entire universe U' of inputs in order for $\mathcal{B}^{\mathcal{A}}$ to be able to solve P on the entire universe U of inputs. When this is the case, it will be emphasized through the definition of the input sets S and S'. These restrictions, however, can not be used to show a stronger statement such as worst-case to average-case reduction.

4.3.1 The Matrix Code Equivalence problem.

This section introduces basic notions on matrix codes and their equivalences. A more thorough introduction on matrix codes can be found in [**gorla**]. The usual choice for measuring distance between matrices over a finite field is the so-called *rank metric*, defined as follows.

Definition 4.2. Let $\mathrm{Rank}(\mathbf{M})$ denote the rank of a matrix $\mathbf{M} \in \mathcal{M}_{m,n}(q)$. The *rank distance* between two $m \times n$ matrices $\mathbf{A}$ and $\mathbf{B}$ over $\mathbb{F}_q$ is defined as

$$d(\mathbf{A}, \mathbf{B}) = \mathrm{Rank}(\mathbf{A} - \mathbf{B}).$$

An *isometry* is a map $\mu : \mathcal{M}_{m,n}(q) \to \mathcal{M}_{m,n}(q)$ that preserves the rank, i.e. $\mathrm{Rank}(\mu(\mathbf{M})) = \mathrm{Rank}(\mathbf{M})$ for all $\mathbf{M} \in \mathcal{M}_{m,n}(q)$.

By symmetry, without loss of generality, in the rest of the text we assume $n \geqslant m$.

Definition 4.3. A *matrix code* is a subspace $\mathcal{C}$ of $m \times n$ matrices over $\mathbb{F}_q$ endowed with the rank metric. Let k denote the dimension of $\mathcal{C}$ as a subspace of $\mathbb{F}_q^{m \times n}$ and its basis by $\langle \mathbf{C_1}, \ldots, \mathbf{C_k} \rangle$, with $\mathbf{C_i} \in \mathbb{F}_q^{m \times n}$ linearly independent. Two matrix codes $\mathcal{C}, \mathcal{D} \subset \mathcal{M}_{m,n}(q)$ are said to be *equivalent* if there exists an isometry μ with $\mu(\mathcal{C}) = \mathcal{D}$.

An isometry from $\mathcal{C}$ to $\mathcal{D}$ is always of the form $\mathbf{M} \mapsto \mathbf{AMB}$, $\mathbf{M} \mapsto \mathbf{M}^\top$ or a composition of these two, where $\mathbf{A} \in \mathrm{GL}_m(q)$ and $\mathbf{B} \in \mathrm{GL}_n(q)$ [169, 290]. We restrict our attention to the isometries of the first form and we will say that two matrix codes are equivalent if there exists a map $\mathbf{C} \mapsto \mathbf{ACB}$ from $\mathcal{C}$ to $\mathcal{D}$ where $\mathbf{A} \in \mathrm{GL}_m(q)$ and $\mathbf{B} \in \mathrm{GL}_n(q)$. We will denote this map as a pair $(\mathbf{A}, \mathbf{B})$. When $n = m$, If there exists a map $(\mathbf{A}, \mathbf{A}^\top) : \mathbf{C} \mapsto \mathbf{ACA}^\top$ from $\mathcal{C}$ to $\mathcal{D}$, where $\mathbf{A} \in \mathrm{GL}_m(q)$, we will say that the codes $\mathcal{C}$ and $\mathcal{D}$ are *congruent*. This is a direct generalization of the notion of congruent matrices. An *automorphism* of a code is a map $(\mathbf{A}, \mathbf{B}) : \mathcal{C} \to \mathcal{C}$, i.e. for each $\mathbf{C} \in \mathcal{C}$, we get $\mathbf{ACB} \in \mathcal{C}$. The *automorphism group* of $\mathcal{C}$ contains all the automorphisms of $\mathcal{C}$. If the

automorphism group contains only the maps $(\lambda\mathbf{I}, \nu\mathbf{I})$ for scalars $\lambda, \nu \in \mathbb{F}_q^*$, we say the automorphism group is trivial.

The main focus of this article will be the *Matrix Code Equivalence*(MCE) problem which is formally defined as follows:

Problem 4.1. MCE $(n, m, k, \mathcal{M}_{m,n}^{[k]}(q))$:
Input: Two k-dimensional matrix codes $\mathcal{C}, \mathcal{D} \subset \mathcal{M}_{m,n}(q)$
Question: Find – if any – a map $(\mathbf{A}, \mathbf{B})$, where $\mathbf{A} \in \mathrm{GL}_m(q), \mathbf{B} \in \mathrm{GL}_n(q)$ such that for all $\mathbf{C} \in \mathcal{C}$, it holds that $\mathbf{ACB} \in \mathcal{D}$.

This is the computational version of MCE which, similarly to its counterpart in the Hamming metric [31, 33, 56], seems to be more interesting for cryptographic applications than its decisional variant. We will thus be interested in evaluating the practical hardness only of MCE, and present algorithms only for MCE and not its decisional variant. It is also interesting to consider the following variant of MCE:

Problem 4.2. MCE base$(n, m, k, \mathcal{M}_{m,n}^{[k]}(q))$:
Input: The bases $(\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(k)})$ and $(\mathbf{D}^{(1)}, \ldots, \mathbf{D}^{(k)})$ of two k-dimensional matrix codes $\mathcal{C}, \mathcal{D} \subset \mathcal{M}_{m,n}(q)$
Question: Find – if any – a map $(\mathbf{A}, \mathbf{B})$, where $\mathbf{A} \in \mathrm{GL}_m(q), \mathbf{B} \in \mathrm{GL}_n(q)$ such that for all $\mathbf{C}^{(i)}$, it holds that $\mathbf{AC^{(i)}B} = \mathbf{D}^{(i)}$.

Intuitively, MCE base seems easier than MCE, and as a matter of fact, we will show later that most random instances are solvable in polynomial time. Another variant of the MCE problem is the *Matrix Codes Right Equivalence* problem (MCRE) (left equivalence could be defined similarly):

Problem 4.3. MCRE $(n, m, k, \mathcal{M}_{m,n}^{[k]}(q))$:
Input: Two k-dimensional matrix codes $\mathcal{C}, \mathcal{D} \subset \mathcal{M}_{m,n}(q)$
Question: Find – if any – $\mathbf{B} \in \mathrm{GL}_n(q)$ such that for all $\mathbf{C} \in \mathcal{C}$, it holds that $\mathbf{CB} \in \mathcal{D}$.

It has been shown in [114] that MCE is at least as hard as code equivalence in the Hamming metric, *Hamming Code Equivalence* (HCE), also known as Linear or Monomial Equivalence. Interestingly, the same paper shows that MCRE is actually easy and can always be solved in probabilistic-polynomial time.

For *vector rank codes* $\mathcal{C} \subset \mathbb{F}_{q^m}^n$, isometries are similar to the case of matrix codes. We get the *Vector Rank Code Equivalence* (VRCE) problem.

Problem 4.4. VRCE $(n, m, k, \mathcal{M}_{m,n}^{[k]}(q))$:
Input: Two k-dimensional vector rank codes $\mathcal{C}, \mathcal{D} \subset \mathbb{F}_{q^m}^n$

Question: Find – if any – a matrix $\mathbf{B} \in \mathrm{GL}_n(q)$ such that for all $\mathbf{c} \in \mathcal{C}$, it holds that $\mathbf{cB} \in \mathcal{D}$.

Given a vector rank code $\mathcal{C} \subset \mathbb{F}_{q^m}^n$ and a basis Γ for $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$, each vector $\mathbf{c} \in \mathcal{C}$ can be expanded to a matrix $\Gamma(\mathbf{c}) \in \mathcal{M}_{m,n}(q)$, giving rise to a matrix code $\Gamma(\mathcal{C})$. For any two bases Γ and Γ', an equivalence between two vector rank codes $\mathcal{C}$ and $\mathcal{D}$ implies an equivalence between the matrix codes $\Gamma(\mathcal{C})$ and $\Gamma'(\mathcal{D})$ [**gorla**], so VRCE is trivially a subproblem of MCE. However, using the $\mathbb{F}_{q^m}$-linearity of vector rank codes, VRCE reduces *non-trivially* to MCRE [114].

4.3.2 Systems of quadratic polynomials.

Let $\mathcal{P} = (p_1, p_2, \ldots, p_k) : \mathbb{F}_q^N \to \mathbb{F}_q^k$ be a vectorial function of k quadratic polynomials in N variables $x_1, \ldots, x_N$, where

$$p_s(x_1, \ldots, x_N) = \sum_{1 \leqslant i \leqslant j \leqslant N} \gamma_{ij}^{(s)} x_i x_j + \sum_{i=1}^{N} \beta_i^{(s)} x_i + \alpha^{(s)},$$

with $\gamma_{ij}^{(s)}, \beta_i^{(s)}, \alpha^{(s)} \in \mathbb{F}_q$ for $1 \leqslant s \leqslant k$.

It is common to represent the quadratic homogeneous part of the components of $\mathcal{P}$ using symmetric matrices, but unfortunately, a natural correspondence only exists for finite fields of odd characteristic. For the case of even characteristic, we will adopt a technical representation that is a common workaround in the literature of multivariate cryptography and will still be good for our purposes.

Let $p(x_1, \ldots, x_N) = \sum_{1 \leqslant i \leqslant j \leqslant N} \gamma_{ij} x_i x_j$ be a quadratic form over $\mathbb{F}_q$. Then, for fields of odd characteristic, we can associate to p a symmetric matrix $\mathbf{P} = \overline{\mathbf{P}} + \overline{\mathbf{P}}^\top$, where $\overline{\mathbf{P}}$ is an upper triangular matrix with coefficients $\overline{\mathbf{P}}_{ij} = \gamma_{ij}/2$ for $i \leqslant j$. Clearly, there is a one-to-one correspondence between quadratic forms and symmetric matrices, since for $\mathbf{x} = (x_1, \ldots, x_N)$ it holds that

$$p(x_1, \ldots, x_N) = \mathbf{x}\mathbf{P}\mathbf{x}^\top. \tag{4.1}$$

Now, all operations on quadratic forms naturally transform into operations on matrices since the one-to-one correspondence between quadratic forms and symmetric matrices is in fact an isomorphism. Note that, in matrix form, a change of variables (basis) works as:

$$p(\mathbf{x}\mathbf{S}) = \mathbf{x}\mathbf{S}\mathbf{P}\mathbf{S}^\top\mathbf{x}^\top. \tag{4.2}$$

In what follows, we will interchangeably work with both the quadratic form p and its matrix representation $\mathbf{P}$.

Over fields $\mathbb{F}_q$ of even characteristic, the relation (4.1) does not hold, since for a symmetric matrix $\mathbf{P}$ we have $(\mathbf{P}_{ij} + \mathbf{P}_{ji})x_i x_j = 2\mathbf{P}_{ij}x_i x_j = 0$. The nice correspondence between quadratic forms and symmetric matrices is broken, but we would still like to be able to use some sort of matrix representation for quadratic forms. Thus, in even characteristic we associate to p a symmetric matrix $\mathbf{P} = \overline{\mathbf{P}} + \overline{\mathbf{P}}^\top$, where $\overline{\mathbf{P}}$ is an upper triangular matrix with coefficients $\overline{\mathbf{P}}_{ij} = \gamma_{ij}$ for $i \leqslant j$.

This representation can also be used in odd characteristic when it comes to linear operations and changes of basis, as the correspondence $p \mapsto \mathbf{P}$ is a homomorphism. However, it is not a bijection, since all the quadratic forms in the set $\{ \sum_{1 \leqslant i < j \leqslant N} \gamma_{ij} x_i x_j + \sum_{1 \leqslant i \leqslant N} \gamma_{ii} x_i^2 \mid \gamma_{ii} \in \mathbb{F}_q \}$ map to the same symmetric matrix (note that it has zeros on the diagonal). In practical, cryptographic applications, this typically does not pose a problem, and can be overcome. The same holds for our purpose of solving equivalence problems for systems of quadratic polynomials.

Differential of quadratic functions.

Given a non-zero $\mathbf{a} \in \mathbb{F}_q^N$, an object directly related to the symmetric matrix representation of quadratic forms is the *differential* of $\mathcal{P}$ at $\mathbf{a}$ (see [132, 145]):

$$D_{\mathbf{a}}\mathcal{P} : \mathbb{F}_q^N \to \mathbb{F}_q^k, \quad \mathbf{x} \mapsto \mathcal{P}(\mathbf{x} + \mathbf{a}) - \mathcal{P}(\mathbf{x}) - \mathcal{P}(\mathbf{a}).$$

Note that the differential of a quadratic function is closely related to the bilinear form $\beta(\mathbf{x}, \mathbf{y}) = q(\mathbf{x} + \mathbf{y}) - q(\mathbf{x}) - q(\mathbf{y})$ associated to a quadratic form q. In this work we are especially interested in the kernel of $D_{\mathbf{a}}\mathcal{P}$, as $D_{\mathbf{a}}\mathcal{P}(\mathbf{x}) = 0$ implies $\mathcal{P}(\mathbf{x} + \mathbf{a}) = \mathcal{P}(\mathbf{x}) + \mathcal{P}(\mathbf{a})$, that is, $\mathcal{P}$ acts linearly on the kernel of $D_{\mathbf{a}}\mathcal{P}$.

4.3.3 Isomorphism of polynomials.

The Isomorphism of Polynomials (IP) problem (or Polynomial Equivalence (PE) [140]) was first defined by Patarin in [**Patarin96**] for the purpose of designing a "graph isomorphism"-like identification scheme and a digital signature scheme using the Fiat-Shamir transform [**FS87**]. It is defined as follows.

Problem 4.5. IP $(N, k, \mathbb{F}_q[x_1, \ldots, x_N]^k \times \mathbb{F}_q[x_1, \ldots, x_N]^k)$:
Input: Two k-tuples of multivariate polynomials $\mathcal{F} = (f_1, f_2, \ldots, f_k)$, $\mathcal{P} =$

$(p_1, p_2, \ldots, p_k) \in \mathbb{F}_q[x_1, \ldots, x_N]^k$.
Question: Find – if any – $(\mathbf{S}, \mathbf{s}) \in \mathrm{AGL}_N(q), (\mathbf{T}, \mathbf{t}) \in \mathrm{AGL}_k(q)$ such that

$$\mathcal{P}(\mathbf{x}) = \mathcal{F}(\mathbf{x}\mathbf{S} + \mathbf{s})\mathbf{T} + \mathbf{t}. \tag{4.3}$$

The variant of the problem where $(\mathbf{T}, \mathbf{t})$ is trivial is known as the Isomorphism of Polynomials with one secret (IP $1\mathcal{S}$), whereas if $\mathcal{P}$ and $\mathcal{F}$ are quadratic and both $\mathbf{s}$ and $\mathbf{t}$ are the null vector, the problem is known as Quadratic Maps Linear Equivalence (QMLE) problem.

The decisional version of IP is not $\mathcal{NP}$-complete [240], but it is known that even IP $1\mathcal{S}$ is at least as difficult as the Graph Isomorphism problem [240]. The IP problem has been investigated by several authors, initially for the security of the C^* scheme [240]. In [**Perret05**] it was shown that the IP $1\mathcal{S}$ is polynomially solvable for most of the instances with $k \geq N$, and Bouillaguet et al. [62] gave an algorithm with running time of $\mathcal{O}(N^6)$ for random instances of the IP $1\mathcal{S}$ problem, thus fully breaking Patarin's identification scheme [**Patarin96**]. The authors of [240] gave an algorithm for solving the general IP, called To-and-Fro, that runs in time $\mathcal{O}(q^{2N})$ for $q > 2$ and $\mathcal{O}(q^{3N})$ for $q = 2$. It was noted in [64] that the algorithm is only suited for bijective mappings $\mathcal{F}$ and $\mathcal{P}$. Getting rid of the bijectivity constraint has been explored in [63] with the conclusion that the proposed workarounds either have a non-negligible probability of failure or it is unclear how greatly they affect the complexity of the algorithm.

Regarding QMLE, the linear variant of IP, an empirical argument was given in [140] that random inhomogeneous instances are solvable in $\mathcal{O}(N^9)$ time, but a rigorous proof for this case still remains an open problem. Under this assumption, the same paper provides an algorithm of complexity $\mathcal{O}(N^9 q^N)$ for the homogeneous case which is considered the hardest, that was subsequently improved to $\mathcal{O}(N^9 q^{2N/3})$ in [64]. Both works reduce a homogenous instance to an inhomogenous instance and assume the obtained inhomogeneous instance behaves as a random instance. This, however, is a wrong assumption which questions the claimed complexity of the algorithm.

In this work, we will be interested in the homogeneous variant of QMLE, that we denote hQMLE, as the hardest and most interesting instance of QMLE.

Formally, the hQMLE problem is defined as follows.

Problem 4.6. hQMLE $(N, k, \mathbb{F}_q[x_1, \ldots, x_N]^k \times \mathbb{F}_q[x_1, \ldots, x_N]^k)$:
Input: Two k-tuples of homogeneous multivariate polynomials of degree 2

$$\mathcal{F} = (f_1, f_2, \ldots, f_k),\ \mathcal{P} = (p_1, p_2, \ldots, p_k) \in \mathbb{F}_q[x_1, \ldots, x_N]^k.$$

Question: Find – if any – a map $(\mathbf{S}, \mathbf{T})$ where $\mathbf{S} \in \mathrm{GL}_N(q), \mathbf{T} \in \mathrm{GL}_k(q)$ such that

$$\mathcal{P}(\mathbf{x}) = (\mathcal{F}(\mathbf{x}\mathbf{S}))\mathbf{T}. \tag{4.4}$$

Interestingly, the case of $k = 1$, which we will call Quadratic Form Equivalence (QFE) has been completely solved for more than 80 years already in the works of Witt [280] and Arf [284]. It is known that every quadratic form is equivalent to a unique canonical diagonal (for odd characteristic) or block diagonal (for even characteristic) form which can be obtained in time $\mathcal{O}(N^3)$. Thus, QFE can also be solved in time $\mathcal{O}(N^3)$ by first calculating the transformations to the canonical forms of the two quadratic forms. If the canonical forms are the same, by composition, one can find the equivalence. If the canonical forms are not the same, the two quadratic forms are not equivalent.

In this work, we also consider a variant of QMLE where $\mathcal{F}$ and $\mathcal{P}$ are bilinear forms. We call this problem Bilinear Maps Linear Equivalence (BMLE). In this variant, $\mathcal{F}$ and $\mathcal{P}$ are k-tuples of homogeneous polynomials of degree 2 in two sets of variables $[x_1, \ldots, x_n]$ and $[y_1, \ldots, y_m]$, where each monomial is of the form $x_i y_j$. Formally, the BMLE problem is defined as follows.

Problem 4.7. BMLE $(n, m, k, \mathbb{F}_q[x_1, \ldots, x_n, y_1, \ldots, y_m]^k \times \mathbb{F}_q[x_1, \ldots, x_n, y_1, \ldots, y_m]^k)$:
Input: Two k-tuples of bilinear forms

$$\mathcal{F} = (f_1, f_2, \ldots, f_k),\ \mathcal{P} = (p_1, p_2, \ldots, p_k) \in \mathbb{F}_q[x_1, \ldots, x_n, y_1, \ldots, y_m]^k$$

Question: Find – if any – a triplet $(\mathbf{S}_1, \mathbf{S}_2, \mathbf{T})$ where $\mathbf{S}_1 \in \mathrm{GL}_n(q), \mathbf{S}_2 \in \mathrm{GL}_m(q)$, $\mathbf{T} \in \mathrm{GL}_k(q)$ such that

$$\mathcal{P}(\mathbf{x}, \mathbf{y}) = (\mathcal{F}(\mathbf{x}\mathbf{S}_1, \mathbf{y}\mathbf{S}_2))\mathbf{T}. \tag{4.5}$$

The inhomogenous versions of QMLE and BMLE will be referred to as inhQMLE and inhBMLE respectively. We write inh(Q/B)MLE these two problems. when it does not matter if we are referring to the quadratic or the bilinear version.

4.4 How hard is MCE?

In this section we investigate the relation of the MCE problem to other known problems that we notably split in two groups – equivalence problems for systems of multivariate quadratic polynomials and equivalence problems for codes.

4.4.1 Relations to equivalence problems for qaudratic polynomials

We start with establishing a straightforward link between MCE and polynomial equivalence problems by proving that the MCE and BMLE problems are equivalent.

Theorem 4.4. The MCE problem is at least as hard as the BMLE problem.

Proof. In order to prove our claim, we need to show that an oracle $\mathcal{A}$ solving any instance of the MCE problem can be transformed in polynomial time to an oracle $\mathcal{B}$ solving any instance of the BMLE problem.

Suppose $\mathcal{B}$ is given an instance $\mathcal{I}_{\mathsf{BMLE}}(\mathcal{F}, \mathcal{P})$ of BMLE $(n, m, k, \mathbb{F}_q[\mathbf{x}, \mathbf{y}]^k \times \mathbb{F}_q[\mathbf{x}, \mathbf{y}]^k)$, where $\mathcal{F} = (f_1, f_2, \ldots, f_k)$, $\mathcal{P} = (p_1, p_2, \ldots, p_k) \in \mathbb{F}_q[\mathbf{x}, \mathbf{y}]^k$ are k-tuples of bilinear forms. Without loss of generality, we assume $f_1, f_2, \ldots, f_k$ (respectively $p_1, p_2, \ldots, p_k$) to be linearly independent. $\mathcal{B}$ can efficiently construct an instance of the MCE problem as follows.

$\mathcal{B}$ represents the components f_s and p_s, $s \in \{1, \ldots, k\}$ of the mappings $\mathcal{F}$ and $\mathcal{P}$ as $m \times n$ matrices $\mathbf{F}^{(s)}$ and $\mathbf{P}^{(s)}$, where $\mathbf{F}^{(s)}_{i,j}$ equals the coefficient of $x_i y_j$ in f_s and $\mathbf{P}^{(s)}_{i,j}$ equals the coefficient of $x_i y_j$ in p_s. Taking $(\mathbf{F}^{(1)}, \ldots, \mathbf{F}^{(k)})$ to be a basis of a matrix code $\mathcal{C}$ and $(\mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(k)})$ a basis of a matrix code $\mathcal{D}$, $\mathcal{B}$ obtains an instance $\mathcal{I}_{\mathsf{MCE}}(\mathcal{C}, \mathcal{D})$ of MCE $(n, m, k, \mathcal{M}^{[k]}_{m,n}(q))$.

$\mathcal{B}$ gives the instance $\mathcal{I}_{\mathsf{MCE}}(\mathcal{C}, \mathcal{D})$ as an input to $\mathcal{A}$. $\mathcal{A}$ outputs either a solution $(\mathbf{A}, \mathbf{B})$ to the MCE instance (in the case it was a positive instance) or outputs that there is no solution (in the case it was a negative instance). In the latter case, $\mathcal{B}$ immediately outputs: no solution. In the former case, $\mathcal{B}$ constructs the matrices $\mathbf{R}^{(s)} = \mathbf{A}\mathbf{F}^{(s)}\mathbf{B} \in \mathcal{D}$ and solves the following system of equations in the variables $t_{i,j}$:

$$\sum_{j=1}^{k} t_{j,i} \cdot \mathbf{R}^{(j)} = \mathbf{P}^{(i)}, \forall i \in \{1, \ldots, k\} \tag{4.6}$$

The system has always a solution, since $(\mathbf{R}^{(1)}, \ldots, \mathbf{R}^{(k)})$ is a basis of the code $\mathcal{D}$.

$\mathcal{B}$ sets $\mathbf{T} = (t_{i,j})$, and outputs $(\mathbf{A}, \mathbf{B}^\top, \mathbf{T})$ as the solution to $\mathcal{I}_{\mathsf{BMLE}}(\mathcal{F}, \mathcal{P})$. $\mathcal{B}$ succeeds whenever $\mathcal{A}$ succeeds and the reduction runs in time $\mathcal{O}(k^6)$.

□

Theorem 4.5. BMLE is at least as hard as MCE.

Proof. We proceed similarly as in the other direction – Given an oracle $\mathcal{A}$ solving any instance of BMLE, we can construct in polynomial time an oracle $\mathcal{B}$ with access to $\mathcal{A}$ that can solve any instance of MCE.

Suppose $\mathcal{B}$ is given an instance $\mathcal{I}_{\mathsf{MCE}}(\mathcal{C}, \mathcal{D})$ of MCE $(n, m, k, \mathcal{M}_{m,n}^{[k]}(q))$. $\mathcal{B}$ takes arbitrary bases $(\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(k)})$ and $(\mathbf{D}^{(1)}, \ldots, \mathbf{D}^{(k)})$ of the codes $\mathcal{C}$ and $\mathcal{D}$ respectively. For each of the matrices $\mathbf{C}^{(s)}$, $\mathcal{B}$ constructs the bilinear forms $c_s(\mathbf{x}, \mathbf{y}) = \sum_{1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n} \mathbf{C}_{ij}^{(s)} x_i y_j$ and for the matrices $\mathbf{D}^{(s)}$ the bilinear forms $d_s(\mathbf{x}, \mathbf{y}) = \sum_{1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n} \mathbf{D}_{ij}^{(s)} x_i y_j, \forall s, 1 \leqslant s \leqslant k$. Taking $\mathcal{F} = (c_1, c_2, \ldots, c_k)$ and $\mathcal{P} = (d_1, d_2, \ldots, d_k)$ we obtain an instance $\mathcal{I}_{\mathsf{BMLE}}(\mathcal{F}, \mathcal{P})$ of BMLE $(n, m, k, \mathbb{F}_q[\mathbf{x}, \mathbf{y}]^k \times \mathbb{F}_q[\mathbf{x}, \mathbf{y}]^k)$.

$\mathcal{B}$ queries $\mathcal{A}$ with the instance $\mathcal{I}_{\mathsf{BMLE}}(\mathcal{F}, \mathcal{P})$ and $\mathcal{A}$ outputs a solution $(\mathbf{S}_1, \mathbf{S}_2, \mathbf{T})$ to the BMLE instance, or no solution if there isn't any. In the first case, this immediately gives a solution $(\mathbf{S}_1, \mathbf{S}_2^\top)$ to the MCE instance. In the second case, there is no solution to the MCE instance. □

In order to prove the connection of MCE to the more general problem hQMLE we first need to establish some properties of matrix codes.

Lemma 4.6. Let $\mathcal{C}$ and $\mathcal{D}$ be matrix codes generated by the bases $= (\mathbf{C}_1, \ldots, \mathbf{C}_k)$ and $(\mathbf{D}_1, \ldots, \mathbf{D}_k)$ of (skew-)symmetric matrices, and assume that $\mathcal{C}$ and $\mathcal{D}$ have trivial automorphism groups. Then $\mathcal{C}$ is equivalent to $\mathcal{D}$ if and only if $\mathcal{C}$ is congruent to $\mathcal{D}$.

Proof. Clearly, by definition if $\mathcal{C}$ is congruent to $\mathcal{D}$, then $\mathcal{C}$ is equivalent to $\mathcal{D}$.

For the opposite direction, let $\mathcal{C}$ be equivalent to $\mathcal{D}$. Then there exist non-singular matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{T}$ such that

$$\sum_{i=1}^{k} t_{j,i} \mathbf{D}_i = \mathbf{A} \mathbf{C}_j \mathbf{B}$$

Since $\mathbf{C}_i$ and $\mathbf{D}_i$ are (skew-)symmetric the last rewrites as

$$\sum_{i=1}^{k} t_{j,i} \mathbf{D}_i = \mathbf{B}^\top \mathbf{C}_j \mathbf{A}^\top$$

Combining the two, and since $\mathbf{A}$ and $\mathbf{B}$ are non-singular, we obtain

$$\mathbf{C}_j = \mathbf{A}^{-1} \mathbf{B}^\top \mathbf{C}_j \mathbf{A}^\top \mathbf{B}^{-1}$$

The automorphism group being trivial implies $\mathbf{A} = \lambda \mathbf{B}^\top$ for some $\lambda \in \mathbb{F}_q$ which in turn implies that $\mathcal{C}$ is congruent to $\mathcal{D}$. □

Remark 4.1. The result of Lemma 4.6 has already been known for algebraically closed fields of non-even characteristic [38, 268]. Since finite fields are not algebraically closed, this result is not useful in our context. On the other hand, requiring a trivial automorphism group for the codes is not a huge restriction, and we typically expect the automorphism group to be trivial for randomly chosen matrix codes. Specifically for cryptographic purposes with regards to MCE, one wants the orbit of $\mathcal{C}$ to be maximal under the action of suitable isometries, which happens when the automorphism group of $\mathcal{C}$ is trivial. Similar requirements for trivial or small automorphism groups occur in the Hamming metric, where it is known that without this requirement there might exist weak keys [138, 139].

Theorem 4.7. Let $\mathcal{T}$ denote the subset of $\mathcal{M}_{m,n}^{[k]}(q)$ of k-dimensional matrix codes of symmetric matrices with trivial automorphism groups. Further, let $\mathcal{T}'$ denote the subset of $\mathbb{F}_q[x_1, \ldots, x_N]^k$ of k-tuples of polynomials with trivial automorphism groups.

The MCE $(\mathcal{T})$ problem is at least as hard as the hQMLE $(\mathcal{T}')$ problem

Proof. We perform the reduction in a similar manner as previously.

Suppose $\mathcal{B}$ is given an instance $\mathcal{I}_{\mathsf{hQMLE}}(\mathcal{F}, \mathcal{P})$ of hQMLE $(N, k, \mathcal{T}')$, where $\mathcal{F} = (f_1, f_2, \ldots, f_k)$, $\mathcal{P} = (p_1, p_2, \ldots, p_k) \in [x_1, \ldots, x_N]$ are k-tuples of linearly independent quadratic forms from $\mathcal{T}'$. $\mathcal{B}$ can efficiently construct an instance of the MCE $(N, N, k, \mathcal{T})$ problem as follows.

$\mathcal{B}$ forms the $N \times N$ symmetric matrices $\mathbf{F}^{(s)}$ and $\mathbf{P}^{(s)}$ associated to the components f_s and p_s, $s \in \{1, \ldots, k\}$ of the mappings $\mathcal{F}$ and $\mathcal{P}$. Taking $(\mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(k)})$ to be a basis of a matrix code $\mathcal{D}$ and $(\mathbf{F}^{(1)}, \ldots, \mathbf{F}^{(k)})$ a basis of a matrix code $\mathcal{C}$, $\mathcal{B}$ obtains an instance $\mathcal{I}_{\mathsf{MCE}}(\mathcal{C}, \mathcal{D})$ of MCE. Per assumption, the matrix codes $\mathcal{C}$ and $\mathcal{D}$ have trivial automorphism groups, hence the instance is from MCE $(N, N, k, \mathcal{T})$.

$\mathcal{B}$ queries $\mathcal{A}$ with the instance $\mathcal{I}_{\mathsf{MCE}}(\mathcal{C}, \mathcal{D})$. , $\mathcal{A}$ answers with a solution $(\mathbf{A}, \mathbf{B})$ to the MCE instance if it is positive, and no solution otherwise. In the former case, from Lemma 4.6, since the matrices are symmetric, $\mathbf{A} = \mathbf{B}^\top$. Now, $\mathcal{B}$ applies the change of variables $\mathbf{xA}$ to $\mathcal{F}$ and obtains $\mathcal{R}(\mathbf{x}) = \mathcal{F}(\mathbf{xA})$. It then solves the system

$$\sum_{j=1}^{k} t_{j,s} \cdot r_j = p_s, \forall s \in \{1, \ldots, k\} \tag{4.7}$$

The system has a solution if $\mathcal{I}_{\mathsf{hQMLE}}(\mathcal{F},\mathcal{P})$ is a positive instance. This is always the case in odd characteristic, because there is a one-to-one correspondence between polynomials and their symmetric matrix representation. Over characteristic 2, it may happen that the $\mathcal{I}_{\mathsf{hQMLE}}(\mathcal{F},\mathcal{P})$ is not a positive instance while its symmetric matrix representation $\mathcal{I}_{\mathsf{MCE}}(\mathcal{C},\mathcal{D})$ is. In this case, the system (4.7) does not have a solution and $\mathcal{B}$ outputs no solution.

If the system has a solution, $\mathcal{B}$ sets $\mathbf{T}=(t_{i,j})$, and outputs $(\mathbf{A},\mathbf{T})$ as the solution to $\mathcal{I}_{\mathsf{hQMLE}}(\mathcal{F},\mathcal{P})$. $\mathcal{B}$ succeeds whenever $\mathcal{A}$ succeeds and the reduction takes time $\mathcal{O}(k^6)$.

□

For the following theorem, we define the symmetric matrix representation of a matrix code $\mathcal{C}$ as the code $\{\begin{bmatrix}\mathbf{0} & \mathbf{C}^\top\\ \mathbf{C} & \mathbf{0}\end{bmatrix} \mid \mathbf{C}\in\mathcal{C}\}$.

Theorem 4.8. Let $\mathcal{T}_s$ denote the subset of $\mathcal{M}_{m,n}^{[k]}(q)$ of k-dimensional matrix codes whose symmetric matrix representation has a trivial automorphism group. Similarly, let $\mathcal{T}_s'$ denote the subset of $\mathbb{F}_q[x_1,\ldots,x_N]^k$ of k-tuples of polynomials with trivial automorphism groups.

The hQMLE $(\mathcal{T}_s')$ problem is at least as hard as the MCE $(\mathcal{T}_s)$ problem.

Proof. We show that given any oracle $\mathcal{A}$ that solves the hQMLE $(\mathcal{T}_s')$ problem there exists an oracle $\mathcal{B}$ running in polynomial time that solves the MCE $(\mathcal{T}_s)$.

Suppose $\mathcal{B}$ is given an instance $\mathcal{I}_{\mathsf{MCE}}(\mathcal{C},\mathcal{D})$ of MCE $(n,m,k,\mathcal{T}_s)$. $\mathcal{B}$ can efficiently construct an instance of the hQMLE $(n+m,k,\mathcal{T}_s')$ problem as follows.

$\mathcal{B}$ fixes bases $(\mathbf{D}^{(1)},\ldots,\mathbf{D}^{(k)})$ of the code $\mathcal{D}$ and $(\mathbf{C}^{(1)},\ldots,\mathbf{C}^{(k)})$ of the code $\mathcal{C}$. For each of the matrices $\mathbf{C}^{(s)}$, $\mathcal{B}$ constructs the quadratic forms $c_s(\mathbf{x}) = \sum_{1\leqslant i\leqslant m, m+1\leqslant j\leqslant m+n} \mathbf{C}_{ij}^{(s)} x_i x_j$ and for the matrices $\mathbf{D}^{(s)}$ the quadratic forms $d_s(\mathbf{x}) = \sum_{1\leqslant i\leqslant m, m+1\leqslant j\leqslant m+n} \mathbf{D}_{ij}^{(s)} x_i x_j, \forall s, 1\leqslant s\leqslant k$, where $\mathbf{x}=(x_1,\ldots,x_{m+n})$. Taking $\mathcal{F}=(c_1,c_2,\ldots,c_k)$ and $\mathcal{P}=(d_1,d_2,\ldots,d_k)$ $\mathcal{B}$ obtains an instance $\mathcal{I}_{\mathsf{hQMLE}}(\mathcal{F},\mathcal{P})$ of hQMLE $(n+m,k,\mathcal{T}_s')$.

$\mathcal{B}$ queries $\mathcal{A}$ with the instance $\mathcal{I}_{\mathsf{hQMLE}}(\mathcal{F},\mathcal{P})$ which outputs a solution $(\mathbf{S},\mathbf{T})$ to the hQMLE instance.

We argue that this solution can be transformed to a solution to the MCE instance, if it is a positive instance. The symmetric matrix representation of the codes $\mathcal{C}$ and $\mathcal{D}$ is given by

$$\begin{bmatrix}\mathbf{0} & (\mathbf{D}^{(i)})^\top\\ \mathbf{D}^{(i)} & \mathbf{0}\end{bmatrix} \text{ and } \begin{bmatrix}\mathbf{0} & (\mathbf{C}^{(i)})^\top\\ \mathbf{C}^{(i)} & \mathbf{0}\end{bmatrix}, i\in\{1,\ldots,k\}. \tag{4.8}$$

The solution $(\mathbf{S}, \mathbf{T})$ means

$$\sum \tilde{t}_{i,j} \begin{bmatrix} \mathbf{0} & (\mathbf{D}^{(j)})^\top \\ \mathbf{D}^{(j)} & \mathbf{0} \end{bmatrix} = \mathbf{S} \begin{bmatrix} \mathbf{0} & (\mathbf{C}^{(i)})^\top \\ \mathbf{C}^{(i)} & \mathbf{0} \end{bmatrix} \mathbf{S}^\top, i \in \{1, \ldots, k\}. \tag{4.9}$$

If the given MCE instance is positive, then there exist matrices $\mathbf{A}, \mathbf{B}, \mathbf{L}$ such that $\mathbf{A}\mathbf{C}_i\mathbf{B} = \sum_j l_{i,j}\mathbf{D}_j$. This implies

$$\sum l_{i,j} \begin{bmatrix} \mathbf{0} & (\mathbf{D}^{(j)})^\top \\ \mathbf{D}^{(j)} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{B}^\top & \mathbf{0} \\ \mathbf{0} & \mathbf{A} \end{bmatrix} \begin{bmatrix} \mathbf{0} & (\mathbf{C}^{(i)})^\top \\ \mathbf{C}^{(i)} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}^\top \end{bmatrix}, i \in \{1, \ldots, k\}. \tag{4.10}$$

The last two imply

$$\sum \lambda_{i,j} \begin{bmatrix} \mathbf{0} & (\mathbf{D}^{(j)})^\top \\ \mathbf{D}^{(j)} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{B}^\top & \mathbf{0} \\ \mathbf{0} & \mathbf{A} \end{bmatrix} \mathbf{S}^{-1} \begin{bmatrix} \mathbf{0} & (\mathbf{D}^{(i)})^\top \\ \mathbf{D}^{(i)} & \mathbf{0} \end{bmatrix} \mathbf{S}^{-\top} \begin{bmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}^\top \end{bmatrix}, i \in \{1, \ldots, k\}. \tag{4.11}$$

By assumption, the automorphism group of the $\begin{bmatrix} \mathbf{0} & (\mathbf{D}^{(i)})^\top \\ \mathbf{D}^{(i)} & \mathbf{0} \end{bmatrix}$ matrices is trivial, which means $\mathbf{S}$ necessarily equals $\begin{bmatrix} \mathbf{B}^\top & \mathbf{0} \\ \mathbf{0} & \mathbf{A} \end{bmatrix}$ up to scalar multiplication. For such an $\mathbf{S}$, the MCE solution can immediately be extracted. $\mathcal{B}$ then outputs the extracted solution.

If on the other hand, $\mathbf{S}$ is not of such block-diagonal form, $\mathcal{B}$ outputs no solution, as this implies the instance is not positive.

□

Remark 4.2. Using the above reduction between MCE and hQMLE, we can reduce the MCE base problem to and from a special case of IP known as IP $1\mathcal{S}$. Interestingly, Perret [**Perret05**] shows IP $1\mathcal{S}$ is polynomially solvable for most instances $k \geq N$, and later work [62] gives an algorithm with running time of $\mathcal{O}(N^6)$ for most random instances, although no rigorous proof that bounds the complexity of the problem to polynomial was given. This nevertheless implies that the MCE base problem can practically be solved in polynomial time for most cryptographically interesting parameters.

4.4.2 Relations to equivalence problems for linear codes

In this section, we show that MCE is at the heart of various code equivalence problems. Equivalence problems for different metrics, such as the Hamming

metric or the sum-rank metric, reduce to MCE, making the hardness analysis of MCE the more exciting.

Hamming code equivalence.

Codes $\mathcal{C} \subset \mathbb{F}_q^n$ equipped with the *Hamming metric* have isometries of the form

$$\tau : (c_1, \ldots, c_n) \mapsto (\alpha_1 c_{\pi^{-1}(1)}, \ldots, \alpha_n c_{\pi^{-1}(n)}), \quad \alpha_i \in \mathbb{F}_q^*, \ \pi \in S_n. \tag{4.12}$$

From this, we define *Hamming code equivalence* (HCE) as the problem of finding an isometry between two Hamming codes $\mathcal{C}$ and $\mathcal{D}$.

Problem 4.8. HCE(k, n):
Input: Two k-dimensional Hamming codes $\mathcal{C}, \mathcal{D} \subset \mathbb{F}_q^n$
Question: Find – if any – $\alpha \in \mathbb{F}_q^{*n}, \pi \in S_n$ such that $\alpha\pi(\mathbf{c}) \in \mathcal{D}$ holds for all $\mathbf{c} \in \mathcal{C}$.

The subproblem where α is trivial is called the *monomial equivalence problem*.

It is easy to turn an HCE instance into a MCE instance [114], given the description of isometries in Equation (4.12). First, define $\Phi : \mathbb{F}_q^n \to \mathcal{M}_n(\mathbb{F}_q)$ by

$$\mathbf{x} = (x_1, \ldots, x_n) \mapsto \begin{pmatrix} x_1 & & \\ & \ddots & \\ & & x_n \end{pmatrix}.$$

The map Φ is an isometry from the Hamming metric to the rank metric: codewords with weigh t are mapped to matrices of rank t. From this, we quickly get the reduction: Writing π as a matrix $\mathbf{P} \in \mathrm{GL}_n(q)$, Φ translates a Hamming isometry τ to a rank-metric isometry by

$$\Phi(\tau) : \Phi(\mathbf{x}) \mapsto \mathbf{P}^{-1}\Phi(\mathbf{x})\mathbf{A}\mathbf{P}, \quad \text{where } \mathbf{A} = \begin{pmatrix} \alpha_1 & & \\ & \ddots & \\ & & \alpha_n \end{pmatrix} \in \mathrm{GL}_n(q).$$

A second reduction from HCE to MCE is given later in [114], which concerns the search variant of the problem, and is more explicit. Both reductions, however, do not help with solving HCE in practice: both the permutational ($\mathbf{A}$ is trivial) and the linear variant of code equivalence in the Hamming metric have algorithms [33, 243] that perform much better for an HCE instance τ than the algorithms we propose for solving $\Phi(\tau)$ as an MCE instance.

Sum-rank code equivalence.

The *sum-rank metric* [233] is a metric that is gaining in popularity in coding theory. It is commonly given as a generalization of the vector-rank metric, but one can also define a variant that generalizes matrix-rank metric. We will reduce both vector and matrix sum-rank equivalence problems to MCE. The idea is the same as for HCE, we find the right isometry from sum-rank metric to rank metric to get the reduction.

Definition 4.9. Let n be partitioned as $n = n_1 + \ldots + n_\ell$. Let $\mathbf{v}^{(i)} = (v_1^{(i)}, \ldots, v_{n_i}^{(i)}) \in \mathbb{F}_{q^m}^{n_i}$ and $\mathbf{v} = (\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(\ell)}) \in \mathbb{F}_{q^m}^{n}$. Let Γ be a basis for $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. Then the *vector sum-rank* of $\mathbf{v}$ is defined as

$$\mathrm{SumRank}(\mathbf{v}) := \sum_{i=1}^{\ell} \mathrm{Rank}\, \Gamma(\mathbf{v}^{(i)}).$$

Let m be partitioned as $m = m_1 + \ldots + m_\ell$. Let $\mathbf{V}^{(i)} \in \mathcal{M}_{m_i \times n_i}(\mathbb{F}_q)$ and $\mathbf{V} = (\mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(\ell)})$. Then the *matrix sum-rank* of $\mathbf{V}$ is defined as

$$\mathrm{SumRank}(\mathbf{V}) = \sum_{i=1}^{\ell} \mathrm{Rank}\, \mathbf{V}^{(i)}.$$

The sum-rank generalizes both the Hamming metric and the rank metric: taking $\ell = n$ gives the Hamming metric, whereas $\ell = 1$ gives the rank metric. We define isometries again as maps that preserve the sum-rank. Sum-rank isometries are simple generalisations of rank isometries (see Problem 4.4).

Proposition 4.10 ([9, Thm. 3.7])**.** Isometries with respect to the vector sum-rank metric are given by vector rank isometries $\mu^{(i)} : \mathbf{x}^{(i)} \mapsto \alpha^{(i)}\mathbf{x}^{(i)}\mathbf{B}^{(i)}$ per 'block' with $\alpha^{(i)} \in \mathbb{F}_{q^m}^*$ and $\mathbf{B}^{(i)} \in \mathrm{GL}_{n_i}(q)$, and suitable permutations π of such blocks if $n_i = n_j$ for $i \neq j$, so

$$\mu : (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\ell)}) \mapsto (\alpha^{(1)}\mathbf{x}^{\pi^{-1}(1)}\mathbf{B}^{(1)}, \ldots, \alpha^{(\ell)}\mathbf{x}^{\pi^{-1}(\ell)}\mathbf{B}^{(\ell)})$$

is a general description of a vector sum-rank isometry.

Generalizing to matrix sum-rank codes is achieved by simply replacing $\alpha^{(i)} \in \mathbb{F}_{q^m}^*$ with $\mathbf{A}^{(i)} \in \mathrm{GL}_{m_i}(q)$ [229, Prop. 4.25]. This gives us the *Vector Sum-Rank Code Equivalence* (VSRCE) and *Matrix Sum-Rank Code Equivalence* (MSRCE) problems.

Problem 4.9. VSRCE (n, m, k):
Input: Two k-dimensional vector sum-rank codes $\mathcal{C}, \mathcal{D} \subset \mathbb{F}_{q^m}^n$
Question: Find – if any – $\alpha^{(i)} \in \mathbb{F}_{q^m}^*, \mathbf{B}^{(i)} \in \mathrm{GL}_{n_i}(q)$ and a permuation π such that for all $\mathbf{c} \in \mathcal{C}$, it holds that $\mu(\mathbf{c}) \in \mathcal{D}$.

Problem 4.10. MSRCE(n, m, k):
Input: Two k-dimensional matrix sum-rank codes $\mathcal{C}, \mathcal{D} \subset (\mathcal{M}_{m_i \times n_i}(\mathbb{F}_q))_i$
Question: Find – if any – $\mathbf{A}^{(i)} \in \mathrm{GL}_{m_i}(q), \mathbf{B}^{(i)} \in \mathrm{GL}_{n_i}(q)$ and a permuation π such that for all $\mathbf{C} \in \mathcal{C}$, it holds that $\mu(\mathbf{C}) \in \mathcal{D}$.

In order to give a reduction to MCE, we use the same idea as for HCE. First, we define a 'nice' map $\Psi : \mathbb{F}_q^n \to \mathcal{M}_{\ell \cdot m \times n}(\mathbb{F}_q)$ by

$$\mathbf{x} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\ell)}) \mapsto \begin{pmatrix} \mathrm{Mat}(\mathbf{x}^{(1)}) & & \\ & \ddots & \\ & & \mathrm{Mat}(\mathbf{x}^{(\ell)}) \end{pmatrix}.$$

It is clear that Ψ is an isometry from the vector sum-rank metric to the rank metric, as it preserves the weight. We get the following reduction.

Theorem 4.11. Let $\mathcal{T}$ denote the subset of $\mathcal{M}_{m,n}^{[k]}(q)$ of k-dimensional matrix codes with trivial automorphism groups. Let $\mathcal{T}'$ denote the subset of k-dimensional vector sum-rank codes that are in the preimage $\Psi^{-1}(\mathcal{T})$. Then MCE $(\mathcal{T})$ is at least as hard as VSRCE $(\mathcal{T}')$.

Proof. Suppose $\mathcal{B}$ is given an instance $\mathcal{I}_{\mathsf{VSRCE}}(\mathcal{C}, \mathcal{D})$ of VSRCE $(n, m, k, \mathcal{T}')$, where $\mathcal{C}$ and $\mathcal{D}$ are k-dimensional vector sum-rank codes. $\mathcal{B}$ can efficiently construct an instance of the MCE $(\mathcal{T})$ problem as follows. By writing the permutation π of the 'blocks' by a matrix representation $\mathbf{P}$, $\mathcal{B}$ can translate a vector sum-rank isometry μ into a matrix code isometry $\Psi(\mu)$ by

$$\Psi(\mu) : \Psi(\mathbf{x}) \mapsto \mathbf{P}^{-1}\mathbf{A}\Psi(\mathbf{x})\mathbf{B}\mathbf{P} \quad \text{where } \mathbf{A} = \begin{pmatrix} \alpha^{(1)} & & \\ & \ddots & \\ & & \alpha^{(\ell)} \end{pmatrix}, \mathbf{B} = \begin{pmatrix} \mathbf{B}^{(1)} & & \\ & \ddots & \\ & & \mathbf{B}^{(\ell)} \end{pmatrix}$$

with $\mathbf{A} \in \mathrm{GL}_\ell(q^m), \mathbf{B} \in \mathrm{GL}_n(q)$. Hence, $\Psi(\mu)$ is an instance of MCE $(n, m, k, \mathcal{T})$, with which $\mathcal{B}$ queries $\mathcal{A}$. $\mathcal{A}$ outputs a solution $(\mathbf{A}', \mathbf{B}')$ to this MCE $(\mathcal{T})$ instance. As the automorphism group is trivial, $\mathcal{B}$ computes $\lambda\mathbf{A}' = \mathbf{P}^{-1}\mathbf{A}$ and $\lambda\mathbf{B}' = \mathbf{B}\mathbf{P}$ for $\lambda \in \mathbb{F}_q$, and therefore solves the $\mathcal{I}_{\mathsf{VSRCE}}$ instance. □

From vector sum-rank code equivalence to matrix sum-rank code equivalence is only a small step. Given a partition $m = m_1 + \ldots + m_\ell$, the map we need is only slightly different from Ψ, namely $\tilde{\Psi} : (\mathcal{M}_{m_i \times n_i}(\mathbb{F}_q))_i \to \mathcal{M}_{m \times n}(\mathbb{F}_q)$ by

$$\mathbf{X} = (\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(\ell)}) \mapsto \begin{pmatrix} \mathbf{X}^{(1)} & & \\ & \ddots & \\ & & \mathbf{X}^{(\ell)} \end{pmatrix}.$$

Theorem 4.12. Let $\mathcal{T}$ denote the subset of $\mathcal{M}_{m,n}^{[k]}(q)$ of k-dimensional matrix codes with trivial automorphism groups. Let $\mathcal{T}'$ denote the subset of k-dimensional matrix sum-rank codes that are in the preimage $\tilde{\Psi}^{-1}(\mathcal{T})$. Then $\mathsf{MCE}\ (\mathcal{T})$ is at least as hard as $\mathsf{MSRCE}\ (\mathcal{T}')$.

Proof. This is a simple generalization of Theorem 4.11: Replace $\alpha^{(i)}$ by $\mathbf{A}^{(i)} \in \mathrm{GL}_{m_i}(q)$ so that $\mathbf{A} \in \mathrm{GL}_m(q)$. Then again, for a matrix sum-rank μ we get $\tilde{\Psi}(\mu)$ by $\Psi(\mathbf{x}) \mapsto \mathbf{P}^{-1}\mathbf{A}\Psi(\mathbf{x})\mathbf{B}\mathbf{P}$ as an $\mathsf{MCE}\ (\mathcal{T})$ instance. □

The link between such MCE instances $\Psi(\mu)$ coming from vector sum-rank and $\tilde{\Psi}(\mu)$ coming from matrix sum-rank is given by a representation $\rho : \mathbb{F}_{q^m}^* \to \mathrm{GL}_m(q)$. We map a vector sum-rank instance to a matrix sum-rank instance by $\mathbf{A}^{(i)} = \rho(\alpha^{(i)})$, so that $\mathbf{A} \in \mathrm{GL}_{\ell \cdot m}(q)$.

To show the equivalences between the rank and sum-rank instances, we need to show that an MCE instance is also an MSRCE instance. But this is trivial: the sum-rank metric generalizes the rank metric, thus an MCE instance is an MSRCE instance with $\ell = 1$. Hence, we get the following theorem for free.

Theorem 4.13. MSRCE is at least as hard as MCE.

4.5 Solving Matrix Code Equivalence

In this section, we analyze the complexity of solving an instance of $\mathsf{MCE}\ (n, m, k)$. We start by establishing a useful lemma.

Lemma 4.14. An $\mathsf{MCE}\ (n, m, k)$ instance can in polynomial time be turned into an $\mathsf{MCE}\ (\sigma(n), \sigma(m), \sigma(k))$ instance for any permutation σ on the set $\{n, m, k\}$. Furthermore, they are either both positive, or both negative instances.

Proof. Let $\mathcal{I}_{\mathsf{MCE}}(\mathcal{C}, \mathcal{D})$ be a given $\mathsf{MCE}\ (n, m, k)$ instance. Let $(\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(k)})$ and $(\mathbf{D}^{(1)}, \ldots, \mathbf{D}^{(k)})$ be bases of the codes $\mathcal{C}$ and $\mathcal{D}$ respectively. Without loss

of generality, we will turn this instance into an MCE (m, k, n) instance (the rest can be done analogously). We set $\bar{\mathbf{C}}_{j,t}^{(i)} = \mathbf{C}_{i,j}^{(t)}$, $\bar{\mathbf{D}}_{j,t}^{(i)} = \mathbf{D}_{i,j}^{(t)}$ and we take $(\bar{\mathbf{C}}^{(1)}, \ldots, \bar{\mathbf{C}}^{(n)})$ and $(\bar{\mathbf{D}}^{(1)}, \ldots, \bar{\mathbf{D}}^{(n)})$ to be the bases of the codes $\bar{\mathcal{C}}$ and $\bar{\mathcal{D}}$ respectively. Clearly, $\bar{\mathcal{C}}$ and $\bar{\mathcal{D}}$ are equivalent if and only if $\mathcal{C}$ and $\mathcal{D}$ are equivalent. □

Without loss of generality, and with Lemma 4.14 in mind, we assume $m = \min\{m, n, k\}$.

As a baseline we have a straightforward algorithm that uses a result from [114] that MCRE can be solved in polynomial time. By enumerating either $\mathbf{A}$ or $\mathbf{B}$, we obtain an instance of MCRE. This means the dominating complexity is the enumeration resulting in an overall complexity of $\tilde{\mathcal{O}}(q^{m^2})$ for MCE.

The approach we outline in the section makes use of the reduction of MCE to hQMLE (see Theorem 4.8). This means that we use techniques already applied for solving hQMLE, but generalize and improve them by making use of the specific structure that MCE instances show when viewed as hQMLE instances.

4.5.1 Solving MCE as QMLE

At Eurocrypt 2013, Bouillaguet et al. [64] proposed an algorithm for solving hQMLE using techniques from graph theory. Their main idea was to reduce the homogeneous case to the inhomogeneous case, which they assume is efficiently solvable (e.g. using the heuristic algebraic approach of [140]). Starting from an instance of hQMLE, they build two exponentially-large graphs that correspond to the given maps $\mathcal{F}$ and $\mathcal{P}$ such that, finding an isomorphism between the two graphs is equivalent to finding an isomorphism between the two quadratic maps. Since the graphs are exponentially large, a technique is provided to *walk* through the graphs without constructing them. Walking through the graphs consists of finding adjacent vertices and computing the degree of a vertex, both in polynomial time. The algorithm consists in finding pairs of vertices from the first and the second graph that have the same degree and making queries to an inhomogenous QMLE solver. If the solver finds an isomorphism by which two vertices are related, then the isomorphism between the two graphs, and thus the isomorphism between the two quadratic maps, is found.

4.5.2 First algorithm for solving MCE

The algorithm for solving hQMLE from [64] considers a graph arising from the differential of a given polynomial map – a vertex $\mathbf{a}$ is connected to all the

vertices that vanish at the differential at **a**. It is, however, not entirely clear how the property we choose to construct such graphs impacts the complexity of the algorithm. We revisit the algorithm, and show how it can be generalized, i.e. abstracted from the property used in [64], under certain conditions. In this section we present this generalization – a birthday-based algorithm for finding an isomorphism between two objects when a specific solver exists. In this form, it can be applied to a broader type of equivalence problems, using more general invariants, here implemented as a predicate $\mathbb{P}$.

Let S_1 and S_2 be subsets of a universe U of equal size N. Algorithm 1 finds an equivalence function $\varphi : S_1 \to S_2$. We assume there exists a predicate $\mathbb{P} : U \to \{\top, \bot\}$ that can be computed in polynomial time, and we denote the cost $C_{\mathbb{P}}$. We assume $\mathbb{P}$ is invariant under the equivalence φ, i.e. $\mathbb{P}(x) = \top \leftrightarrow \mathbb{P}(\varphi(x)) = \top$. Let $U_\top = \{x \in U \mid \mathbb{P}(x) = \top\}$, and $d = |U_\top|/|U|$. We will call d the *density* of the predicate $\mathbb{P}$ and we assume the density on S_1 and S_2 is approximately equal to d. We further assume the existence of an algorithm FindFunction, that given $x \in S_1, y \in S_2$ returns φ if $y = \varphi(x)$ and $\bot$ otherwise. We denote the cost of a query to FindFunction by C_{FF}.

Algorithm 1 General Birthday-based Equivalence Finder

```
1:  procedure SampleSet(S, ℙ, ℓ)
2:      L ← ∅
3:      repeat
4:          a ←$ S
5:          if ℙ(a) then L ← L ∪ {a}
6:      until |L| = ℓ
7:      return L

8:  procedure CollisionFind(S1, S2)
9:      L1 ← SampleSet(S1, ℙ, ℓ)
10:     L2 ← SampleSet(S2, ℙ, ℓ)
11:     for all (a, b) ∈ L1 × L2 do
12:         φ ← FindFunction(a, b)
13:         if φ ≠ ⊥ then
14:             return solution φ
15:     return ⊥
```

Lemma 4.15. For a fixed success probability of $1 - 1/e$, Algorithm 1 performs on average $\mathcal{O}(\sqrt{N/d})$ operations in SampleSet, queries FindFunction at most $d \cdot N$ times.

The optimal value for d, up to a polynomial factor, is $d = N^{-1/3} \cdot C_{\mathrm{FF}}^{-2/3}$, for which the total time complexity of the algorithm is $\mathcal{O}(N^{\frac{2}{3}} \cdot C_{\mathrm{FF}}^{\frac{1}{3}})$ and the memory complexity is $\mathcal{O}(N^{\frac{1}{3}} C_{\mathrm{FF}}^{-\frac{1}{3}})$. If FindFunction runs in polynomial time, this reduces to time complexity of $\tilde{\mathcal{O}}(N^{\frac{2}{3}})$ and memory complexity of $\mathcal{O}(N^{\frac{1}{3}})$.

Proof. First note that the expected number of elements in S_1 and S_2 such that $\mathbb{P}(x)$ holds is equal to dN by the definition of the density d. By the birthday paradox, it is enough to take lists of size $\ell = \sqrt{d \cdot N}$, to be sure that with probability of $1 - \frac{1}{e}$ FindFunction returns a solution [286]. With this length of the lists, the number of queries to FindFunction is dN. On the other hand, the number of samples needed to build the list L_1 (resp. L_2) of elements $a \in S_1$ (resp. $b \in S_2$) such that $\mathbb{P}(a)$ (resp. $\mathbb{P}(b)$) holds is ℓ/d, which gives a complexity of $\mathcal{O}(\sqrt{N/d})$ to build these lists L_i.

The total running time is optimal when these two quantities $\sqrt{N/d}$ and $d \cdot N \cdot C_{\mathrm{FF}}$ are equal, which holds when $d = N^{-1/3} \cdot C_{\mathrm{FF}}^{-2/3}$. Such a density gives complexity of $\mathcal{O}(N^{\frac{2}{3}} \cdot C_{\mathrm{FF}}^{\frac{1}{3}})$ for SampleSet and at most $N^{\frac{2}{3}}$ queries to FindFunction. If C_{FF} is polynomial, this gives a total time complexity of $\tilde{\mathcal{O}}(N^{\frac{2}{3}})$. The memory requirements of the algorithm correspond to the size of the lists L_i. This results in a memory complexity of $\mathcal{O}(N^{\frac{1}{3}} C_{\mathrm{FF}}^{-\frac{1}{3}})$, or $\mathcal{O}(N^{\frac{1}{3}})$ if C_{FF} is polynomial. □

Remark 4.3. The success probability in Lemma 4.15 is chosen rather arbitrarily, mostly for practical verification of the algorithm's correctness. It can be increased to any value $1 - 1/c$ for a positive constant c by appropriately building lists that are larger only by a constant factor compared to the case treated in Lemma 4.15. The overall complexity only differs by a constant factor, i.e., does not change asymptotically.

As said earlier, the algorithm presented in [64] is a special case of Algorithm 1. Their algorithm can be seen as an instantiation of Algorithm 1 by defining $G_{\mathcal{F}}$ (resp. $G_{\mathcal{P}}$) to be the linearity graph of $\mathcal{F}$ (resp. $\mathcal{P}$), where a vertex $\mathbf{a}$ is connected to all vertices $\mathbf{x}$ such that $D_{\mathbf{a}}\mathcal{F}(\mathbf{x}) = 0$ (resp. $D_{\mathbf{a}}\mathcal{P}(\mathbf{x}) = 0$), taking the predicate $\mathbb{P}_\kappa(\mathbf{a}) : \dim \ker D_{\mathbf{a}}\mathcal{F} = \kappa$ on the universe $\mathcal{M}_{k,N}(q)$, and taking for FindFunction the assumed polynomial-time solver from [140] for inhQMLE. Finding a collision (α, β) such that $\beta = \alpha S$ makes the instance $\mathcal{P}(\mathbf{x}+\alpha) = \mathcal{F}(\mathbf{x}S+\beta)\mathbf{T}$ an inhomogeneous instance by defining $\mathcal{P}'(\mathbf{x}) = \mathcal{P}(\mathbf{x}+\alpha)$ and $\mathcal{F}'(\mathbf{x}) = \mathcal{F}(\mathbf{x} + \beta)$. Running FindFunction on $\mathcal{P}'$ and $\mathcal{F}'$ then returns $\mathbf{S}$ and $\mathbf{T}$. In this case, Lemma 4.15 gives the precise result from [64, Thm. 1], which we present as a corollary to our Lemma 4.15, for completeness.

Corollary 4.1. Assuming a polynomial-time solver for the inhomogenous case of QMLE, an hQMLE (N, N) instance $\mathcal{I}_{\mathsf{hQMLE}}(\mathcal{F}, \mathcal{P})$ over $\mathbb{F}_q$ can be solved with complexity and number of queries equal to $\tilde{\mathcal{O}}(q^{\frac{2}{3}N})$ with a success probability of $1 - 1/c$ for any $c > 0$ and a memory complexity of $\mathcal{O}(q^{\frac{1}{3}N})$.

Proof. Let $G_{\mathcal{F}}$ (i.e. $G_{\mathcal{P}}$) be the linearity graph of $\mathcal{F}$ (i.e. $\mathcal{P}$), where a vertex $\mathbf{a}$ is connected to all $\mathbf{x}$ such that $D_{\mathbf{a}}\mathcal{F}(\mathbf{x}) = 0$ (i.e. $D_{\mathbf{x}}\mathcal{P}(\mathbf{a}) = 0$). We use the predicate $\mathbb{P}_\kappa(\mathbf{a}) : \dim\ker D_{\mathbf{a}}\mathcal{F} = \kappa$ we have that $\deg(\mathbf{a}) = q^\kappa$. The density of the predicate d_κ in the universe of $N \times N$ matrices is independent of $\mathcal{F}$ and $\mathcal{P}$, and is therefore the same as the density of linear maps with kernel of dimension κ. Thus, d_κ is approximately a monotonic decreasing function in κ, going from 1 to 0. Hence, by Lemma 4.15, there exists some optimal κ for which we get that $d_\kappa \approx |G_{\mathcal{P}}|^{-1/3} = q^{-N/3}$, which gives a total time complexity of $q^{\frac{2}{3}N}$ and a memory complexity of $q^{\frac{1}{3}N}$. □

Remark 4.4. The assumption on a polynomial-time solver in [64] turns out to be too strong: such a solver exists for random instances, however, for inhQMLE instances as obtained in Corollary 4.1 the running time is probably not polynomial [63]. Nevertheless, the algorithm and result are valid, but require a different rebalancing depending on C_{FF}. Section 4.6 analyzes C_{FF} in detail for different instances.

To apply this approach to MCE instances, we need to generalize to the case of N not necessarily equal to k. For an MCE (n, m, k) instance $\mathcal{I}_{\mathsf{MCE}}(\mathcal{C}, \mathcal{D})$, we get an hQMLE $(n+m, k)$ instance $\mathcal{I}_{\mathsf{hQMLE}}(\mathcal{F}, \mathcal{P})$ by Theorem 4.8. We take again the predicate $\mathbb{P}_\kappa(\mathbf{a}) : \dim\ker D_{\mathbf{a}}\mathcal{F} = \kappa$, but this time on the universe $\mathcal{M}_{k,n+m}(q)$, where $D_{\mathbf{a}}\mathcal{F}$ lives. To get a similar result to Corollary 4.1, we need to show two things. **a)**, that this predicate satisfies the assumptions required for Algorithm 1. **b)**, that there is a κ such that the density d_κ of $\mathbb{P}_\kappa$ is optimal as described in Lemma 4.15. If both are satisfied, we get a complexity of $\mathcal{O}(q^{\frac{2}{3}(n+m)}C_{\mathrm{FF}}^{\frac{1}{3}})$, hence $\tilde{\mathcal{O}}(q^{\frac{2}{3}(n+m)})$ when the solver is polynomial, with a success probability of $1-1/c$ for any $c > 0$ for an MCE (n, m, k) instance $\mathcal{I}_{\mathsf{MCE}}(\mathcal{C}, \mathcal{D})$. We start with **a)**.

Lemma 4.16. The predicate $\mathbb{P}_\kappa(D_{\mathbf{a}}\mathcal{F}) : \dim\ker D_{\mathbf{a}}\mathcal{F} = \kappa$ is a suitable predicate for Algorithm 1, as **i)** $\mathbb{P}_\kappa$ can be computed in polynomial time, **ii)** is invariant under equivalence, **iii)** and d_κ does not depend on $\mathcal{F}$.

Proof.

1. The cost $C_{\mathbb{P}_\kappa}$ is the cost of computing $\dim\ker D_{\mathbf{a}}\mathcal{F}$, i.e. computing the kernel of a $k \times (n+m)$ matrix over $\mathbb{F}_q$. This can be done in polynomial time.

2. Let $\mathcal{P}(\mathbf{x}) = \mathcal{F}(\mathbf{xS})\mathbf{T}$ be the equivalence. If $\mathbf{x} \in \ker D_{\mathbf{a}}\mathcal{P}$ then $\mathbf{xS} \in \ker \mathcal{F}_{\mathbf{aS}}$ and vice versa, as $\mathbf{T}$ does not affect the kernel. As $\mathbf{S}$ is invertible, we get a

one-to-one correspondence $\mathbf{x} \mapsto \mathbf{xS}$ between the kernels, so $\mathbb{P}_\kappa(D_{\mathbf{aS}}\mathcal{F}) = \mathbb{P}_\kappa(D_{\mathbf{a}}\mathcal{P})$.

3. For $\mathcal{F}$ coming from an MCE instance, we always have $-\mathbf{a} \in \ker D_{\mathbf{a}}\mathcal{F}$. We want to show that the distribution of the rank of $D_{\mathbf{a}}\mathcal{F}$ follows the ranks of linear maps vanishing at $-\mathbf{a}$. This is given by [132, Thm. 2] for even characteristic and easily adapted to odd characteristic, which shows d_κ is independent of $\mathcal{F}$.

□

We now continue with **b)**: we show that there is a κ such that d_κ is optimal. For now, existence of κ is enough to derive a complexity on MCE. We will explicitely compute κ later, in Section 4.6, when we have a detailed view of C_{FF} for specific parameter sets (k, n, m).

The general density d_κ for the predicate $\mathbb{P}_\kappa$ is given by the following lemma, taking $a = k$ and $b = n + m$ to avoid confusion with regards to n, m and $n + m$.

Lemma 4.17. Define the predicate $\mathbb{P}_\kappa : \dim\ker \mathbf{M} = \kappa$ for $\mathbf{M} \in U = \mathcal{M}_{a,b}(q)$ with $a \geqslant b$. Then the density of the predicate $\mathbb{P}_\kappa$ is $d_\kappa = 1/\Theta(q^{(\kappa^2+\kappa\cdot(a-b))})$.

Proof. There are $|U| = q^{ab}$ matrices in $\mathcal{M}_{a,b}(q)$, out of which

$$\prod_{i=0}^{r-1} \frac{(q^a - q^i)(q^b - q^i)}{q^r - q^i} = \Theta\left(q^{(a+b-r)r}\right)$$

have rank r [194]. We have $\kappa = b - r$ and so $d_\kappa^{-1} = \frac{|U|}{|U_\top|} = \Theta(\frac{q^{ab}}{q^{-(a+b-r)r}}) = \Theta(q^{\kappa^2+\kappa(a-b)})$. Specifically when the matrix is square, $d_\kappa^{-1} = \Theta(q^{\kappa^2})$. □

From Lemma 4.17 we can conclude that for some κ, the density d_κ is optimal. This means we satisfy both **a)** and **b)** and we can apply Lemma 4.15.

In conclusion, we get our first result on the hardness of MCE, which significantly improves straightforward enumeration. This requires that such a κ exists, which happens when $k \leqslant 2(n + m)$, by Lemma 4.17. Note that, in contrast to [64, Thm. 1], we do not assume a polynomial-time solver for the inhomogeneous case of QMLE. Instead, we write this cost as C_{FF} and explore the precise cost in Section 4.6.

Theorem 4.18. An MCE (n, m, k) instance $\mathcal{I}_{\mathsf{MCE}}(\mathcal{C}, \mathcal{D})$ over $\mathbb{F}_q$ with $k \leqslant 2(n + m)$ can be solved using Algorithm 1 with time complexity equal to $\mathcal{O}(q^{\frac{2}{3}(n+m)}$.

$C_{\mathrm{FF}}^{\frac{1}{3}} \cdot (C_{\mathbb{P}_\kappa} + 1)$), memory complexity equal to $\mathcal{O}(q^{\frac{1}{3}(m+n)} C_{\mathrm{FF}}^{-\frac{1}{3}})$ and with success probability of $1 - 1/c$ for any $c > 0$, where C_{FF} denotes the cost of a single query to FindFunction.

We will show in Section 4.6 that, even though C_{FF} is *not* polynomial-time, the complexity of Algorithm 1 is still $\tilde{\mathcal{O}}(q^{\frac{2}{3}(n+m)})$ for some optimal κ.

When $k > 2(n+m)$, we can no longer assume elements with $\dim \ker D_{\mathbf{a}}\mathcal{F} > 1$ exist, as practically all differentials $D_{\mathbf{a}}\mathcal{F}$ will have only the trivial kernel spanned by $-\mathbf{a}$. In such a scenario, we have two alternatives:

- Take a single element $\mathbf{a}$ and run FindFunction on $(\mathbf{a}, \mathbf{b})$ for all $\mathbf{b} \in \mathbb{F}_q^{n+m}$ until we find the isometry. This deterministic process has a time complexity of $\mathcal{O}(q^{(n+m)} \cdot C_{\mathrm{FF}})$. The memory requirements of this algorithm are negligible, since we do not build lists of elements;

- Alternatively, note that in this case $n \leqslant 2(k+m)$. Thus, we can also use the result of Lemma 4.14, and instead of an MCE (n, m, k) instance, we can solve an MCE (k, m, n) instance using Algorithm 1. In this case we end up with a complexity of $\tilde{\mathcal{O}}(q^{\frac{2}{3}(k+m)})$. However, for the given regime of parameters, this is always larger than $\tilde{\mathcal{O}}(q^{(n+m)})$, so the first deterministic approach is better.

4.5.3 Second algorithm

The algorithm that we presented in the previous section does not take advantage of the bilinear structure of an instance of MCE when viewed as hQMLE. In such a case, the differential $D_{(\mathbf{a},\mathbf{b})}\mathcal{F}$ of a k-dimensional bilinear form admits a special structure.

Lemma 4.19. Let $\mathcal{F}(\mathbf{x}, \mathbf{y})$ be a k-dimensional bilinear form with $\mathbf{x} \in \mathbb{F}_q^m$ and $\mathbf{y} \in \mathbb{F}_q^n$. Let $\mathbf{F_a}$ denote the $k \times n$ matrix of the linear map $\mathcal{F}(\mathbf{a}, -) : \mathbb{F}_q^n \to \mathbb{F}_q^k$ for a fixed $\mathbf{a} \in \mathbb{F}_q^m$. Similarly let $\mathbf{F_b}$ denote the $k \times m$ matrix of the linear map $\mathcal{F}(-, \mathbf{b}) : \mathbb{F}_q^m \to \mathbb{F}_q^k$ for a fixed $\mathbf{b} \in \mathbb{F}_q^n$. Then

$$D_{(\mathbf{a},\mathbf{b})}\mathcal{F}(\mathbf{x}, \mathbf{y}) = \begin{pmatrix} \mathbf{F_b} & \mathbf{F_a} \end{pmatrix} \begin{pmatrix} \mathbf{x}^\top \\ \mathbf{y}^\top \end{pmatrix}.$$

Proof. By bilinearity, $D_{(\mathbf{a},\mathbf{b})}\mathcal{F}(\mathbf{x}, \mathbf{y}) := \mathcal{F}(\mathbf{x} + \mathbf{a}, \mathbf{y} + \mathbf{b}) - \mathcal{F}(\mathbf{x}, \mathbf{y}) - \mathcal{F}(\mathbf{a}, \mathbf{b})$ equals $\mathcal{F}(\mathbf{a}, \mathbf{y}) + \mathcal{F}(\mathbf{x}, \mathbf{b}) = \mathbf{F_a}\mathbf{y}^\top + \mathbf{F_b}\mathbf{x}^\top$. □

Similarly for $\mathcal{P}$, we use the notation $\mathbf{P_a}$ and $\mathbf{P_b}$. The equivalence in such a case becomes $\mathcal{P}(\mathbf{x}, \mathbf{y}) = \mathcal{F}(\mathbf{xA}, \mathbf{yB}^\top)\mathbf{T}$, with $\mathbf{A}, \mathbf{B}$ precisely the matrices from the MCE instance. Then, as $\mathcal{F}$ and $\mathcal{P}$ are bilinear, one can see SampleSet in Algorithm 1 as sampling both $\mathbf{a} \in \mathbb{F}_q^n$ and $\mathbf{b} \in \mathbb{F}_q^m$ at the same time as one $(\mathbf{a}, \mathbf{b}) \in \mathbb{F}_q^{n+m}$, until $D_{(\mathbf{a},\mathbf{b})}\mathcal{F}$ has a kernel of dimension κ. However in the bilinear case, $\mathbf{a}$ influences only the matrix $\mathbf{F_a}$, and $\mathbf{b}$ influences only $\mathbf{F_b}$. Hence, we can sample $\mathbf{a} \in \mathbb{F}_q^m$ and $\mathbf{b} \in \mathbb{F}_q^n$ separately. This hints that we can apply ideas from Algorithm 1 to the smaller universes $U_\mathbf{a} = \mathcal{M}_{k,n}(q)$ and $U_\mathbf{b} = \mathcal{M}_{k,m}(q)$, where $\mathbf{F_a}$ and $\mathbf{F_b}$ live. By finding well-chosen predicates in these smaller universes, we hope to find collisions faster.

We first analyse the properties of $\mathbf{F_a}$ and $\mathbf{F_b}$ a bit more. Let $\mathfrak{F}_a$ be the set of elements $\mathbf{a}$ for which $\dim\ker \mathbf{F_a}$ is non-trivial, and $\mathfrak{F}_b$ similarly, i.e.

$$\mathfrak{F}_a = \{\mathbf{a} \in \mathbb{F}_q^m \mid \dim\ker \mathcal{F}(\mathbf{a}, -) > 0\}, \quad \mathfrak{F}_b = \{\mathbf{b} \in \mathbb{F}_q^n \mid \dim\ker \mathcal{F}(-, \mathbf{b}) > 0\}.$$

For $\mathcal{P}$, we define $\mathfrak{P}_a$ and $\mathfrak{P}_b$ similarly. For isomorphic bilinear forms $\mathcal{F}$ and $\mathcal{P}$, these sets have special properties.

Lemma 4.20. Let $(\mathbf{A}, \mathbf{B}, \mathbf{T}) : \mathcal{F} \to \mathcal{P}$ be an isomorphism between two k-tuples of bilinear homogenous quadratic polynomials $\mathcal{F}$ and $\mathcal{P}$, such that $\mathcal{P}(\mathbf{x}, \mathbf{y}) = \mathcal{F}(\mathbf{xA}, \mathbf{yB}^\top)\mathbf{T}$. We have the following properties:

1. Given $\mathbf{a} \in \mathfrak{F}_a$ and any $\mathbf{b} \in \ker \mathbf{F_a}$, we get $\mathcal{F}(\mathbf{a}, \mathbf{b}) = 0$.
2. $\mathfrak{F}_b$ is completely determined by $\mathfrak{F}_a$, as $\mathfrak{F}_b = \bigcup_{\mathbf{a}\in\mathfrak{F}_a} \ker \mathbf{F_a}$.
3. For $\mathbf{a} \in \mathbb{F}_q^n$ and $\mathbf{y} \in \mathbb{F}_q^m$, we have $\mathbf{P_a}(\mathbf{y}) = \mathbf{F_{aA}}(\mathbf{yB}^\top)\mathbf{T}$.
4. For $\mathbf{a} \in \mathbb{F}_q^n$, we get $\ker \mathbf{P_a} = \ker \mathcal{F}_\mathbf{aA} \cdot \mathbf{B}^\top$.
5. The isomorphism $(\mathbf{A}, \mathbf{B}, \mathbf{T})$ induces the bijections

$$\mathfrak{P}_a \to \mathfrak{F}_a : \mathbf{a} \mapsto \mathbf{aA}, \quad \mathfrak{P}_b \to \mathfrak{F}_b : \mathbf{b} \mapsto \mathbf{bB}^\top.$$

Proof.

1. $\mathbf{b} \in \ker \mathbf{F_a}$ is equivalent by definition to $\mathbf{F_a b}^\top = \mathcal{F}(\mathbf{a}, \mathbf{b}) = \mathbf{0}$.
2. This follows directly from 1.: $\mathbf{b} \in \mathfrak{F}_b$ only if there exists an $\mathbf{a} \in \mathfrak{F}_a$ such that $\mathcal{F}(\mathbf{a}, \mathbf{b}) = \mathbf{0}$. But then $\mathbf{b} \in \ker \mathbf{F_a}$ for this specific $\mathbf{a}$.
3. Per definition $\mathbf{P_a}(\mathbf{y}) = \mathcal{P}(\mathbf{a}, \mathbf{y}) = \mathcal{F}(\mathbf{aA}, \mathbf{yB}^\top)\mathbf{T} = \mathbf{F_{aA}}(\mathbf{yB}^\top)\mathbf{T}$.

4. This follows directly from 3.: as $\mathbf{T}$ is invertible, it does not affect the kernels, so $\mathbf{y} \in \ker \mathbf{P_a}$ if and only if $\mathbf{yB}^\top \in \ker \mathbf{F_{aA}}$

5. This follows directly from 4.: Given $\mathbf{a} \in \mathfrak{P}_a$ we get $\mathbf{aA} \in \mathfrak{F}_a$ and vice versa as $\mathbf{A} \in \mathrm{GL}_m(q)$. A similar argument gives $\mathfrak{F}_b \to \mathfrak{P}_b$.

□

Lemma 4.20 shows that $\mathbf{a} \in \mathfrak{F}_a$ and $\mathbf{b} \in \mathfrak{F}_b$ describe all non-trivial roots $(\mathbf{a}, \mathbf{b})$ of a given $\mathcal{F}$. For an instance $(\mathbf{A}, \mathbf{B}, \mathbf{T}) : \mathcal{F} \to \mathcal{P}$, Item 5 shows that non-trivial roots are mapped bijectively by $(\mathbf{A}, \mathbf{B}, \mathbf{T})$. Such non-trivial roots can be used to find collisions more easily between $\mathcal{F}$ and $\mathcal{P}$. However, this requires that instances $\mathcal{F} \to \mathcal{P}$ *have* non-trivial roots. We can get an estimate on the sizes of $\mathfrak{F}_a$, $\mathfrak{F}_b$, $\mathfrak{P}_a$, and $\mathfrak{P}_b$ for given parameters n, m, and k, in the following way.

Lemma 4.21. When $k \geqslant n$, $|\mathfrak{F}_a| = |\mathfrak{P}_a| \approx q^{2n-k-1}$ and $|\mathfrak{F}_b| = |\mathfrak{P}_b| \approx q^{2m-k-1}$.

Proof. By Lemma 4.20, we get $|\mathfrak{F}_a| = |\mathfrak{P}_a|$. Then, using Lemma 4.17, we see that the size of these sets is dominated by elements $\mathbf{a}$ with $\kappa = \dim \ker \mathbf{F_a} = 1$ (a one-dimensional kernel). From the same lemma, the density of $\kappa = \dim \ker \mathbf{F_a} = 1$ elements is $d_1 = q^{-(1+1\cdot(k-n))}$. Hence we expect $d_1 \cdot q^n = \Theta(q^{2n-k-1})$ such elements. A similar argument gives $|\mathfrak{F}_b| = |\mathfrak{P}_b|$ as $\Theta(q^{2m-k-1})$. □

Summarizing, this implies

Corollary 4.2. Assuming $n = m$ as the hardest case, a random MCE (n, m, k) instance $\mathcal{I}_{\mathsf{MCE}}(\mathcal{F}, \mathcal{P})$ over $\mathbb{F}_q$ has an expected value $\mathcal{E}_{n,m,k,q}$ of non-trivial roots

- when $k < 2n$, with $\mathcal{E}_{n,m,k,q} = \Theta(q^{2n-k-1})$,
- when $k = 2n$, with $\mathcal{E}_{n,m,k,q} = \Theta(\frac{1}{q})$,
- when $k > 2n$, with $\mathcal{E}_{n,m,k,q} = \Theta(\frac{1}{q^{k-2n+1}})$.

From these results, we can expect non-trivial roots for an MCE (n, m, k) instance $\mathcal{I}_{\mathsf{MCE}}(\mathcal{F}, \mathcal{P})$ over $\mathbb{F}_q$ with $k \leqslant n + m$. These non-trivial roots can be seen as a suitable predicate on the smaller universes $U_{\mathbf{a}}$ and $U_{\mathbf{b}}$: we search for collisions $(\mathbf{a}, \mathbf{b}) \times (\mathbf{aA}, \mathbf{bB}^\top)$, where $(\mathbf{a}, \mathbf{b})$ is a non-trivial root of $\mathcal{P}$, and $(\mathbf{aA}, \mathbf{bB}^\top)$ of $\mathcal{F}$. Given such a collision, we proceed as in Section 4.5.2.

The following result shows that we always find such a collision if $\mathcal{F}$ and $\mathcal{P}$ have non-zero roots.

Lemma 4.22. Let $m \leqslant n$ and $k \leqslant n+m$. Let $\mathfrak{F}_a$, $\mathfrak{F}_b$ and $\mathfrak{P}_a$, $\mathfrak{P}_b$ describe the non-trivial roots of an MCE (n,m,k) instance $\mathcal{I}_{\mathsf{MCE}}(\mathcal{F},\mathcal{P})$ over $\mathbb{F}_q$. Let $\mathbf{x} = (\mathbf{a},\mathbf{b}) \in \mathfrak{F}_a \times \mathfrak{F}_b$, then looping over $\mathbf{y} \in \mathfrak{P}_a \times \mathfrak{P}_b$ gives a collision $(\mathbf{x},\mathbf{y})$ with certainty.

Proof. This follows quickly from Lemma 4.20. We have $\mathbf{x} = (\mathbf{a},\mathbf{b})$ and two bijections $\mathfrak{F}_a \to \mathfrak{P}_a$ and $\mathfrak{F}_b \to \mathfrak{P}_b$, so $\mathbf{x}$ is mapped to some $\mathbf{y} \in \mathfrak{P}_a \times \mathfrak{P}_b$. As this set is finite, we can loop over it in a finite number of steps until we find the collision. □

Therefore, as soon as we have non-trivial roots, we can use a single one of them to find a collision. This leads to the following pseudo-algorithm:

1. compute $\mathfrak{F}_b$ by computing $\ker \mathbf{F_b}$ for all $\mathbf{b} \in \mathbb{F}_q^m$,
2. if $\mathfrak{F}_b$ is non-empty, compute $\mathfrak{F}_a$ using Lemma 4.20-2. Same for $\mathfrak{P}_a$ and $\mathfrak{P}_a$.
3. sample a single $\mathbf{x} \in \mathfrak{F}_a \times \mathfrak{F}_b$
4. loop over $\mathbf{y} \in \mathfrak{P}_a \times \mathfrak{P}_b$ with $\textsc{FindFunction}(\mathbf{x},\mathbf{y})$ until the solver finds μ.

Corollary 4.3. Let $m \leqslant n$ and $k \leqslant n+m$. The above algorithm terminates successfully and has a total complexity of $\mathcal{O}(q^m \cdot C_{\mathbb{P}_\kappa} + q^{2(n+m-k-1)} \cdot C_{\mathrm{FF}})$, where $C_{\mathbb{P}}$ denotes the cost of computing $\ker \mathbf{F_b}$ and C_{FF} denotes the cost of a single query to $\textsc{FindFunction}$.

Proof. Building $\mathfrak{F}_b$ and $\mathfrak{P}_b$ has a complexity of $\mathcal{O}(q^m \cdot C_{\mathbb{P}_\kappa})$, and these give us $\mathfrak{F}_a$ and $\mathfrak{P}_a$ by Lemma 4.20. Then for every step in the loop we get a query to $\textsc{FindFunction}$. By Lemma 4.21, the size of $\mathfrak{P}_a \times \mathfrak{P}_b$ is at most $\mathcal{O}(q^{2(n+m-k-1)})$. □

We will see later in Section 4.6 that the dominating complexity is $q^m \cdot C_{\mathbb{P}_\kappa}$ as for specific parameters (k,n,m) the number of queries z can be reduced so that $z \cdot C_{\mathrm{FF}} < q^m$. As $C_{\mathbb{P}_\kappa}$ is polynomial, we get a complexity of $\tilde{\mathcal{O}}(q^m)$ for such instances.

For efficiency, one can decrease further the number of queries to $\textsc{FindFunction}$ by applying other, secondary predicates. For example, the sets $\mathfrak{F}_a \times \mathfrak{F}_b$ and $\mathfrak{P}_a \times \mathfrak{P}_b$ can be split into zeros $\mathfrak{F}^{\mathbf{0}} = \{\mathbf{x} \in \mathbb{F}_q^{n+m} | \mathcal{F}(\mathbf{x}) = \mathbf{0}\}$ and non-zeros $\mathfrak{F} = \mathfrak{F}_a \times \mathfrak{F}_b \setminus \mathfrak{F}^{\mathbf{0}}$, which reduces the collision search to each of these sets. Another secondary predicate is to only use elements $\mathbf{a}$ with $\dim \ker \mathbf{F_a} = \kappa$ for some specific value $\kappa > 0$.

Algorithm 2 Bilinear MCE-Solver, assuming $n \geqslant m$.

```
1: procedure SampleZeros(F)
2:     S, S_a, S_b ← ∅
3:     for all b ∈ F_q^m do
4:         if dim ker F_b > 0 then
5:             S_b ← S_b ∪ {b}
6:             S_a ← S_a ∪ ker F_b \ {0}
7:     S ← S_a × S_b
8:     return S
9: procedure CollisionFind(F, P)
10:    𝔉 ← SampleZeros(F)
11:    𝔓 ← SampleZeros(P)
12:    x ←$ 𝔉
13:    for all y ∈ 𝔓 do
14:        μ ← FindFunction(x, y)
15:        if μ ≠ ⊥ then
16:            return solution μ
17:    return ⊥
```

We summarize the MCE solver for instances with roots in Algorithm 2.

Practically, since the algorithm is deterministic, we do not need to build and store the list $\mathfrak{F}$. We only need to find one element from it. However, for iterating through the list $\mathfrak{P}$, S_a and S_b need to be stored. The estimated size of these lists is $q^{n+m-k-1}$.

The next theorem summarises the conditions and cost of Algorithm 2 for solving MCE.

Theorem 4.23. Assuming a solver for the inhomogenous case of QMLE with cost C_{FF}, an MCE (n, m, k) instance over $\mathbb{F}_q$ with $m \leqslant n$ and $k \leqslant n + m$ (in which case roots exist for $\mathcal{F}$ and $\mathcal{P}$ with overwhelming probability) can be solved using Algorithm 2 with $\mathcal{O}\left(q^m \cdot C_{\mathbb{P}_\kappa}\right)$ operations in SampleZeros and z queries to the solver. This amounts to a total time complexity of $\mathcal{O}\left(q^m \cdot C_{\mathbb{P}_\kappa} + z \cdot C_{\mathrm{FF}}\right)$. The memory complexity of the algorithm is $\mathcal{O}(q^{n+m-k-1})$.

We will show in Section 4.6 that, even though C_{FF} is *not* polynomial-time, the dominating factor in this complexity is still $q^m \cdot C_{\mathbb{P}_\kappa}$, where $C_{\mathbb{P}_\kappa}$ is the cost to compute the kernel of an $m \times k$ matrix.

The regime of operation of Theorem 4.23 seems to imply that we can use it only if $k \leqslant n + m$. However, note that if $k > n + m$ then $n \leqslant k + m$. Hence, by Lemma 4.14, we can turn the given MCE (n, m, k) instance into an MCE (k, m, n) instance and solve this instance using Algorithm 2. This results in a complexity of $\tilde{\mathcal{O}}(q^m)$. Recall that we assume $m = \min\{m, n, k\}$, thus, we obtain the following general theorem which is our main result about the practical complexity of solving MCE.

Theorem 4.24. An MCE (n, m, k) instance over $\mathbb{F}_q$ can be solved using Algorithm 2 in time $\tilde{\mathcal{O}}\left(q^{\min\{m,n,k\}}\right)$.

4.6 Filling the gaps in the complexity analysis

The cost $C_{\mathbb{P}}$ is polynomial in all of the cases because it either requires computing the rank of a linear map or sampling a random element from a set. The FindFunction in Algorithms 1 and 2 checks whether a given pair of vectors is a collision, and if so, it returns the solution to the MCE instance. This is done by solving an instance of the inhBMLE that has the same solutions as the input MCE instance. Thus, to estimate the value of C_{FF}, we analyse the complexity of inhBMLE on these instances, by relying on algorithms that have been developed for the inhQMLE case with $N = k$.

4.6.1 Algorithms for inhQMLE

The two algorithms described in this section have been used for tackling the inhQMLE problem within the birthday-based algorithm for hQMLE [63, 64]. Their analysis is thus important to estimate C_{FF}. In Section 4.6.2 we adapt this analysis for the inhBMLE case with arbitrary k and N and we see how this affects Algorithms 1 and 2 for different parameter sets.

The Gröbner bases attack

The algebraic attack on the inhQMLE problem starts by reducing $\mathcal{P}(\mathbf{x})\mathbf{T}^{-1} = \mathcal{F}(\mathbf{xS})$, with $\mathbf{S}$ and $\mathbf{T}$ unknown, to a system of polynomial equations. By rewriting the problem in matrix form we obtain the following constraints

$$\begin{aligned} \sum_{1 \leqslant r \leqslant k} \widetilde{T}_{rs} \mathbf{P}^{(r)} &= \mathbf{S}\mathbf{F}^{(s)}\mathbf{S}^{\top}, \ \ \forall s, 1 \leqslant s \leqslant k, \\ \mathbf{P}^{[1]}\mathbf{T}^{-1} &= \mathbf{S}\mathbf{F}^{[1]}, \\ \mathbf{P}^{[0]}\mathbf{T}^{-1} &= \mathbf{F}^{[0]}, \end{aligned} \tag{4.13}$$

where $\mathbf{F}^{[1]} \in \mathbb{F}_q^{N \times k}$ and $\mathbf{P}^{[1]} \in \mathbb{F}_q^{N \times k}$ describe the degree-1 homogeneous part of an inh(Q/B)MLE instance and $\mathbf{F}^{[0]} \in \mathbb{F}_q^{k}$ and $\mathbf{P}^{[0]} \in \mathbb{F}_q^{k}$ describe the degree-0 part. We will denote the subsystem of equations derived from the degree-d homogeneous part as $\mathcal{S}_d$. The resulting system can be solved using Gröbner basis algorithms and this is referred to as the Gröbner attack [140]. The observation

that $\mathbf{S}$ and $\mathbf{T}$ are common solutions to homogeneous parts of separate degrees of an inhQMLE instance (also proven in [62, Lemma 1]) and the idea that moving $\mathbf{T}$ to the other side of the equality results in a lower degree system where we solve for $\mathbf{T}^{-1}$ originate from this work.

The complexity of Gröbner basis algorithms depends foremost on the *degree of regularity*, which is usually hard to estimate, but it can sometimes be observed through experimental work. Such experiments applied to inhQMLE instances imply that the system is solved at degree three. A degree-three linearized system in n variables is represented by a matrix of size roughly n^3 and thus, Gaussian Elimination on such a system is performed in $\mathcal{O}(n^{3\omega})$ operations, where ω is the linear algebra constant. This reasoning leads to the assumption that there exists a polynomial-time solver for the inhomogeneous case of QMLE. Another empirical observation made in [140] is that the time to construct the system exceeds the time of the Gröbner basis computation. Since the generation of the system is known to be polynomial, this suggests that the Gröbner basis computation is performed in polynomial time as well. However, these experiments are performed on random inhomogeneous instances of the QMLE problem.

In the birthday-based approach for solving QMLE, $\mathbf{F}^{[1]}$, $\mathbf{P}^{[1]}$, $\mathbf{F}^{[0]}$ and $\mathbf{P}^{[0]}$ are obtained from a collision [64]. Specifically, if we have a collision on $\mathbf{x} \in \mathbb{F}_q^N$ and $\mathbf{y} \in \mathbb{F}_q^N$ such that $\mathbf{y} = \mathbf{x}\mathbf{S}$, they are obtained as

$$\mathbf{F}^{[1]} = D_{\mathbf{y}}\mathcal{F}, \qquad \mathbf{P}^{[1]} = D_{\mathbf{x}}\mathcal{P},$$
$$\mathbf{F}^{[0]} = \mathcal{F}(\mathbf{y}), \qquad \mathbf{P}^{[0]} = \mathcal{P}(\mathbf{x}).$$

Instances of inhQMLE derived from a collision are, on average, harder to solve than random inhQMLE instances. Recall that in Algorithm 1 the instances of inhQMLE are chosen such that $\dim\ker D_{\mathbf{y}}\mathcal{F} = \dim\ker D_{\mathbf{x}}\mathcal{P} = \kappa$. Hence, the number of linearly independent equations in $\mathcal{S}_1$ is exactly $k(N-\kappa)$, instead of the expected kN on average. The size of $\mathcal{S}_0$ can also depend on the predicate that we choose for the birthday-based algorithm. For instance, when we use the predicate of searching for a collision between the non-trivial roots of $\mathcal{P}$ and $\mathcal{F}$, we obtain no equations in $\mathcal{S}_0$. Additionally, since $\mathbf{F}^{[1]}$ (i.e. $\mathbf{P}^{[1]}$) and $\mathbf{F}^{[0]}$ (i.e. $\mathbf{P}^{[0]}$) are obtained respectively from computing the differential of and evaluating $\mathcal{F}$ (i.e $\mathcal{P}$) at a given point, $\mathcal{S}_1$ and $\mathcal{S}_0$ are not as independent from $\mathcal{S}_2$ as they would be in the random case. It is difficult to estimate the complexity of solving these instances compared to solving random instances with the same structure. Figure 4.2 shows experiments confirming our intuition that the complexity of collision-derived instances is worse than that of random ones. This implies

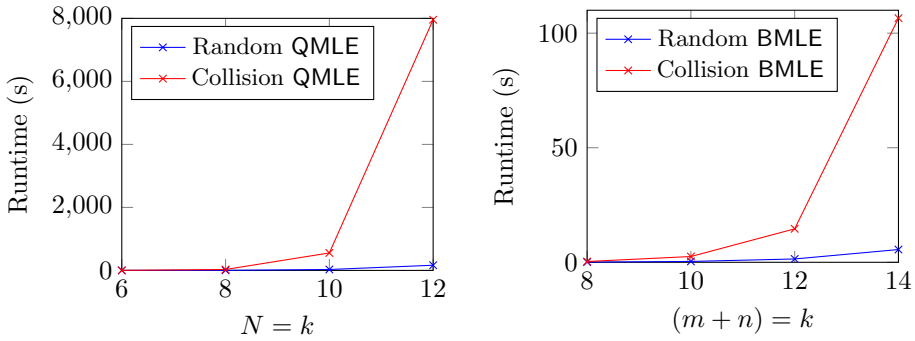


Figure 4.2: Comparison of runtime for solving random and collision-derived inh(Q/B)MLE instances using the Gröbner attack. Results are averaged over 50 runs.

that we can not rely on the experimental observations in [140] to estimate the complexity of these specific instances. We conclude that, in contrast with the literature, we can not assume that C_{FF} is polynomial when the Gröbner attack is used.

The matrix-pencil attack

The matrix-pencil attack was proposed in Bouillaguet's thesis [63] and used for the implementation of the birthday-based attack [64]. This algorithm has a complexity of $\mathcal{O}(N^6)$ with non-negligible probability for random inhQMLE instances where $N = k$. Its complexity for inhQMLE instances derived from a collision attack depends strongly on the parameter κ. We give a general description of the approach. For details on how it relates to the matrix pencil equivalence problem, we refer to [63, Ch. 14].

The first step is to retrieve a basis of the solution space V of the subsystem of linear equations $\mathcal{S}_1$. Let $\ell = \dim V$ and let $(\mathbf{S}^{[1]}, \mathbf{T}^{[1]}), \ldots, (\mathbf{S}^{[\ell]}, \mathbf{T}^{[\ell]})$ be a basis of V. Once the solution space of $\mathcal{S}_1$ is known, in order to find the solution space of the overall system one rewrites $\mathcal{S}_2$ as a system in ℓ variables. Concretely, this is done by replacing $\mathbf{S}$ and $\mathbf{T}$ by $\sum_{i=1}^{\ell} x_i \mathbf{S}^{[i]}$ and $\sum_{i=1}^{\ell} x_i \mathbf{T}^{[i]}$ in Equation (4.13) and then looking for solutions in variables $x_1, \ldots, x_\ell$. This standard approach is also described in [62]. A key idea in the matrix-pencil attack is to use the knowledge of $\mathbf{F}^{[1]}/\mathbf{P}^{[1]}$ and $\mathbf{F}^{[0]}/\mathbf{P}^{[0]}$ to find a (second) collision and double the number of linear equations in $\mathcal{S}_1$. Supposing that

there exists $\mathbf{x}'$ such that $\mathbf{x}'\mathbf{P}^{[1]} = \mathbf{P}^{[0]}$, we infer that there also exists $\mathbf{y}'$ such that $\mathbf{y}'\mathbf{F}^{[1]} = \mathbf{F}^{[0]}$ and that $\mathbf{y}' = \mathbf{x}'\mathbf{S}$. We can thus append the equations obtained from $(D_{\mathbf{x}'}\mathcal{P})\mathbf{T}^{-1} = \mathbf{S}(D_{\mathbf{y}'}\mathcal{F})$ to $\mathcal{S}_1$. After applying this technique, the resulting system is usually highly overdetermined and can be solved through direct linearization. The most favorable case is when $\mathbf{x}'$ and $\mathbf{y}'$ are uniquely identified. However, if $\dim\ker \mathbf{F}^{[1]} = \kappa > 1$, then $\mathbf{x}'$ is chosen arbitrarily and we loop through the q^κ possible values for $\mathbf{y}'$. The complexity of the algorithm is $\mathcal{O}(q^\kappa \ell^2 N^4)$, under the condition that $\ell(\ell+1)/2 \leq |\mathcal{S}_2|$. Another condition for the success of this approach is that $\mathcal{P}(\mathbf{x}) \neq 0$ and there is an $\mathbf{x}$ such that $\mathbf{x}D_{\mathbf{x}}\mathcal{P} = \mathcal{P}(\mathbf{x})$, because this assumption is used to find the second collision. As per the analysis in [63], the probability that the condition for success is met is $1 - 1/q + 1/q^3 + \mathcal{O}(1/q^6)$.

4.6.2 The complexity of inhBMLE

In the following analysis, we use the matrix-pencil algorithm as the inhBMLE solver, as it seems to outperform the Gröbner attack and we have a better understanding of its complexity for these specific instances.

The case $k \leq n + m$

Based on the analysis in Section 4.5.3 for the purpose of usage in Algorithm 2 we can assume without loss of generality that $k \leq n+m$ and $m = \min\{m, n, k\}$.

The complexity of Algorithm 2 is dominated by the SampleZeros function, as long as the complexity of the inhBMLE solver does not surpass $\mathcal{O}(q^m)$. In the matrix-pencil algorithm, we can not use the zero subsets $\mathfrak{F}^{\mathbf{0}}$ and $\mathfrak{P}^{\mathbf{0}}$, as this contradicts its condition for success $\mathcal{P}(\mathbf{x}) \neq 0$. The non-zeros subsets $\mathfrak{F}$ and $\mathfrak{P}$ can be used with a small adjustment to the algorithm: after finding a basis of the solution space of $\mathcal{S}_1$, we rewrite and solve through linearization the system comprised of both $\mathcal{S}_2$ and $\mathcal{S}_0$. Note that $\mathfrak{F}$ and $\mathfrak{P}$ are non-empty only when the instance has at least two roots. Since in Algorithm 2 we do not restrict the value of κ, we will approximate to the one that has the highest probability, which for the case of $k \leq n + m$ is $\kappa = (m + n) - k$. Hence, C_{FF} is approximated to

$$\mathcal{O}(q^{m+n-k} \cdot (m+n)^6).$$

When $k \geq m$, this is always smaller than $\mathcal{O}(q^m)$.

The case $n + m < k < 2(n + m)$

This case is not relevant for Algorithm 2, but it is for Algorithm 1. Since the complexity of the inhBMLE solver contains a non-negligible factor of q^κ, the choice of κ needs to be adapted, so that the running times of SampleSet and CollisionFind are equal. Let $N = n + m$ and let $r = N - k$. The optimal κ is chosen such that

$$q^{\frac{N-(\kappa^2+\kappa r)}{2}} \cdot q^{\kappa^2+\kappa r} \approx q^{N-(\kappa^2+\kappa r)} \cdot q^\kappa.$$

This gives us $\kappa = \frac{k-(n+m+\sqrt{\delta})}{2} + \frac{1}{3}$, with $\delta = (k-(n+m))^2 + \frac{4}{3}(k+\frac{1}{3})$. The complexity of the overall algorithm with this optimal choice for κ is then

$$q^{\frac{n+m}{2} + \frac{k-\sqrt{\delta}}{6} + \frac{1}{9}}.$$

We get that $\sqrt{\delta} \geqslant |k-(n+m)|$ and so for all values of k between $n+m$ and $2(n+m)$, the term $k - \sqrt{\delta}$ is bounded by $n+m$, and hence this gives a bound on the complexity by $\mathcal{O}(q^{\frac{2}{3}(n+m)+\frac{1}{9}})$. The term $\frac{1}{9}$ adds a few bits at most to this complexity, but is negligible for most cryptographic purposes.

The case $k \geq 2(n + m)$

When $k \geq 2(n+m)$, as per Lemma 4.17, the probability that there exist elements with $\dim \ker D_{(\mathbf{a},\mathbf{b})}\mathcal{F} > 1$ is extremely small, which is why we can not define a distinguishing predicate for Algorithm 1 and $\kappa = 1$ with overwhelming probability. In this case, the complexity of the matrix-pencil algorithm is

$$\mathcal{O}(q \cdot (m+n)^6),$$

as with random inhBMLE instances.

4.7 Experimental results

To confirm our theoretical findings, we solved randomly generated positive instances of the MCE problem, using the two approaches presented in this paper. First, we implement the birthday-based Algorithm 1 in three steps. (1) We randomly generate a positive instance $\mathcal{I}_{\mathsf{MCE}}(\mathcal{C},\mathcal{D})$ of MCE (n,m,k) and reduce it to an instance $\mathcal{I}_{\mathsf{hQMLE}}(\mathcal{F},\mathcal{P})$ of hQMLE $(m+n,k)$. (2) We build the two sample sets for a predefined predicate $\mathbb{P}$ and we combine them to create pairs

of potential collisions. (3) For each pair we create an inhQMLE instance and we query an inhQMLE solver until it outputs a solution for the maps $\mathbf{S}$ and $\mathbf{T}$. Our implementation is built on top of the open source birthday-based hQMLE solver from [63], which is implemented in MAGMA [**Magma**].

Table 4.1 shows running times for solving the MCE problem using Algorithm 1. The goal of this first experiments was to confirm that there is a parameter choice where the probability of success of the algorithm surpasses $1 - 1/e$ and that our running times are comparable to the ones given in [64]. These experiments are done with the parameter $q = 2$ and all results are an average of 50 runs.

Table 4.1: Experimental results on solving the MCE problem using Algorithm 1.

$m = n$	k	κ	Sample set size	Runtime (s) SampleSet	Runtime (s) inhQMLE solver	Success probability
10	20	5	2	21	3154	0.70
11	22	5	3	31	2004	0.63
12	24	5	6	76	13873	0.73

The second approach, described in Section 4.5.3 uses the bilinear structure of hQMLE instances derived from MCE instances to have an improved algorithm for building the sample sets and a more precise predicate that results in fewer queries to the inhQMLE solver. The consequence of these two improvements to the runtime can be observed in Table 4.2 where we show experimental results of Algorithm 2 using the non-zeros subsets. Recall that, this approach can be used only when there exist at least two roots of $\mathcal{F}$ and $\mathcal{P}$. Otherwise, the sampled sets contain only the trivial root and the instance is solved using Algorithm 1. Table 4.2 shows results of the case when the sets are non-trivial and the probability of this case for the given parameters is shown in the last column. For efficiency, we take the minimal subset with a common dimension of the kernel of $\mathbf{F_b}$, and when looking for collisions, we are careful to skip pairs $(\mathbf{ab}, \mathbf{a'b'})$ where $\dim\ker \mathbf{F_b} = \dim\ker \mathbf{P_{b'}}$ but $\dim\ker D_{(\mathbf{a},\mathbf{b})}\mathcal{F} \neq \dim\ker D_{(\mathbf{a'},\mathbf{b'})}\mathcal{P}$. In these experiments, $q = 3$ and all results are an average of 50 runs.

Our experiments confirm that Algorithm 2 outperforms Algorithm 1 for solving MCE instances with non-trivial roots.

Acknowledgements. The authors thank Charles Bouillaguet for providing the implementation resulting from [63].

Table 4.2: Experimental results on solving the MCE problem using the non-zeros-subsets variant of Algorithm 2.

$m = n$	k	Sample set size	Runtime (s) SampleZeros	Runtime (s) inhQMLE solver	% instances with two roots
8	15	10.4	0.56	175.34	24
	14	35.56	0.60	236.12	68
9	17	12.00	1.74	396.04	22
	16	37.97	1.72	1020.25	70
10	19	25.6	5.13	2822.32	14
	18	36.72	5.05	1809.09	82

Chapter 5

MEDS

5.1 Placeholder

The paper will go here.

5.2 Introduction

Post-Quantum Cryptography (PQC) comprises all the primitives that are believed to be resistant against attackers equipped with a considerable quantum computing power. Several such schemes have been around for a long time [239, 247], some being in fact almost as old as RSA [206]; however, the area itself was not formalized as a whole until the early 2000s, for instance with the first edition of the PQCrypto conference [230]. The area has seen a dramatic increase in importance and volume of research over the past few years, partially thanks to NIST's interest and the launch of the PQC Standardization process in 2017 [232]. After 4 years and 3 rounds of evaluation, the process has crystallized certain mathematical tools as standard building blocks (e.g. lattices, linear codes, multivariate equations, isogenies etc.). Some algorithms [133, 146, 170] have now been selected for standardization, with an additional one or two to be selected among a restricted set of alternates [4, 8, 12] after another round of evaluation. While having a range of candidates ready for standardization may seem satisfactory, research is still active in designing PQC primitives. In particular, NIST has expressed the desire for a greater diversity among the hardness assumptions behind signature schemes, and announced a partial re-

opening of the standardization process for precisely the purpose of collecting non-lattice-based protocols.

Cryptographic group actions are a popular and powerful instrument for constructing secure and efficient cryptographic protocols. The most well-known is, without a doubt, the action of finite groups on the integers modulo a prime, or the set of points on an elliptic curve, which give rise to the *Discrete Logarithm Problem (DLP)*, i.e. the backbone of public-key cryptography. Recently, proposals for post-quantum cryptographic group actions started to emerge, based on the tools identified above: for instance, isogenies [**CSIDH**], linear codes [56], trilinear forms [**Tang22**] and even lattices [168]. All of these group actions provide very promising solutions for cryptographic schemes, for example signatures [**Tang22**, 33, 117], ring signatures [32, 51] and many others; at the same time, they are very different in nature, with unique positive and negative aspects.

Our Contribution. In this work, we formalize a new cryptographic group action based on the notion of *Matrix Code Equivalence*. This is similar in nature to the *code equivalence* notion at the basis of LESS [33, 56], and in fact belongs to a larger class of isomorphism problems that include, for example, the lattice isomorphism problem, and the well-known isomorphism of polynomials [239]. The hardness of the MCE problem was studied in [114, 248], from which it is possible to conclude that this is a suitable problem for post-quantum cryptography. Indeed, we show that it is possible to use MCE to build a zero-knowledge protocol, and hence a signature scheme, which we name *Matrix Equivalence Digital Signature*, or simply MEDS. For our security analysis, we first study in detail the collision attacks from [248] and then we develop two new attacks. The first attack that we propose uses a nontrivial algebraic modeling inspired from the minors modellings of MinRank in [30, 137]. The second one is an adaptation of Leon's algorithm [**Leon82**] for matrix codes. Based on this analysis, we provide an initial parameter choice, together with several computational optimizations, resulting in a scheme with great flexibility and very competitive data sizes.

A reference implementation for the full scheme is available at

`meds-pqc.org`

Furthermore, we show that this group action allows for the construction of (linkable) ring signatures, with performance results that improve on the existing state of the art [32].

5.3 Preliminaries

Let $\mathcal{K}$ be the finite field of q elements. $\mathrm{GL}_n(q)$ and $\mathrm{AGL}_n(q)$ denote respectively the general linear group and the general affine group of degree n over $\mathcal{K}$. We use bold letters to denote vectors $\mathbf{a}, \mathbf{c}, \mathbf{x}, \ldots$, and matrices $\mathbf{A}, \mathbf{B}, \ldots$. The entries of a vector $\mathbf{a}$ are denoted by a_i, and we write $\mathbf{a} = (a_1, \ldots, a_n)$ for a (row) vector of dimension n over some field. Similarly, the entries of a matrix $\mathbf{A}$ are denoted by a_{ij}. Random sampling from a set S is denoted by $a \xleftarrow{\$} S$. For two matrices $\mathbf{A}$ and $\mathbf{B}$, we denote the Kronecker product by $\mathbf{A} \otimes \mathbf{B}$. Finally, we denote the set of all $m \times n$ matrices over $\mathcal{K}$ by $\mathcal{M}_{m,n}(\mathcal{K})$.

5.3.1 Cryptographic group actions

Definition 5.1. Let X be a set and $(G, \cdot)$ be a group. A group action is a mapping

$$\begin{array}{rccc} \star: & G \times X & \to & X \\ & (g, x) & \mapsto & g \star x \end{array}$$

such that the following conditions hold for all $x \in X$:

- $e \star x = x$, where e is the identity element of G.
- $g_2 \star (g_1 \star x) = (g_2 \cdot g_1) \star x$, for all $g_1, g_2 \in G$.

A group action can have a number of mathematically desirable properties. For example, we say that a group action is:

- *Commutative*: for any $g_1, g_2 \in G$, we have $g_2 \star (g_1 \star x) = g_1 \star (g_2 \star x)$.
- *Transitive*: given $x_1, x_2 \in X$, there is some $g \in G$ such that $g \star x_1 = x_2$.
- *Free*: if $g \star x = x$, then g is the identity.

In particular, a *cryptographic* group action is a group action with some additional properties that are useful for cryptographic applications. To begin with, there are some desirable properties of computational nature. Namely, the following procedures should be efficient:

- *Evaluation*: given x and g, compute $g \star x$.
- *Sampling*: sample uniformly at random from G.
- *Membership testing*: verify that $x \in X$.

Finally, cryptographic group actions should come with security guarantees; for instance, the *vectorization problem* should be hard:

Problem 5.1 (Group Action Vectorization)**. Given:** The pair $x_1, x_2 \in X$. **Goal:** Find, if any, $g \in G$ such that $g \star x_1 = x_2$.

Early constructions using this paradigm are based on the action of finite groups of prime order, for which the vectorization problem is the discrete logarithm problem. Lately, multiple isogeny-based constructions have appeared: see, for instance, the work of Couveignes in [113] and later by Rostovtsev and Stolbunov [256]. A general framework based on group actions was explored in more detail by [**cga**], allowing for the design of several primitives. The holy grail are those cryptographic group actions that possess both the mathematical and cryptographic properties listed above. Currently, CSIDH [**CSIDH**] is the only post-quantum commutative cryptographic group action, although there is an ongoing debate about the efficiency and quantum hardness of its vectorization problem [57]. In Section 5.4, we introduce the group action that is relevant to our work.

5.3.2 Protocols

We give here an explicit characterization of the protocols we will build. The corresponding security definitions are presented only in an informal manner; formal definitions will are included in the full version of this work.[1]

Definition 5.2. A *Sigma protocol* is a three-pass interactive protocol between two parties: a prover $\mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2)$ and a verifier $\mathsf{V} = (\mathsf{V}_1, \mathsf{V}_2)$. The protocol is composed of the following procedures:

I. Keygen: on input some public data (including system parameters), output a public key pk (the instance) and the corresponding secret key sk (the witness). Give sk to the prover; pk is distributed publicly and is available to all parties. For simplicity, we assume that the public data is available as input in all the remaining procedures.

II. Commit: on input the public key pk, P_1 outputs a public commitment cmt and sends it to the verifier.

[1] https://eprint.iacr.org/2022/1559.pdf

III. Challenge: on input the public key pk and the commitment cmt, V_1 samples uniformly at random a challenge chal from the challenge space C and sends it to the prover.

IV. Response: on input the secret key sk, the public key pk, the commitment cmt and the challenge chal, P_2 outputs a response resp and sends it to the verifier.

V. Verify: on input a public key pk, the commitment cmt, the challenge chal, and the response resp, V_2 outputs either 1 (accept) if the *transcript* (cmt, chal, resp) is valid, or 0 (reject) otherwise.

A Sigma protocol is usually required to satisfy the following properties. First, if the statement is true, an honest prover is always able to convince an honest verifier. This property is called *Completeness*. Secondly, a dishonest prover cannot convince an honest verifier other than with a small probability. This is captured by the *Soundness* property, which also bounds such probability, usually known as *soundness error* or, informally, *cheating probability*. Finally, the protocol has to be *Zero-Knowledge*, i.e. anyone observing the transcript (including the verifier) learns nothing other than the fact that the statement is true. definitions will be included in the full version of this work, but are omitted here in the interest of space.

Definition 5.3. A *Digital Signature scheme* is a protocol between 2 parties: a signer S and a verifier V. The protocol is composed of the following procedures:

I. Keygen: on input the public data (including system parameters), output a secret signing key sk for S and the corresponding public verification key pk.

II. Sign: on input a secret key sk and a message msg, output a signature σ.

III. Verify: on input a public key pk, a message msg and a signature σ, V outputs either 1 (accept) if the signature is valid, or 0 (reject) otherwise.

Correctness means that an honest signer is always able to get verified. The usual desired security notion for signature schemes is *Unforgeability*, which guarantees computationally infeasible to forge a valid signature without knowing the secret signing key. definitions to the full version of this work.

Definition 5.4. A *Ring Signature scheme* is a protocol between $r + 1$ parties: potential signers $\mathsf{S}_1, \ldots, \mathsf{S}_r$ (the *ring*) and a verifier V. The protocol is composed of the following procedures:

I. Keygen: on input the public data (including system parameters), output a secret key sk_j for each signer S_j, and the corresponding public key pk_j.

II. Sign: on input a set of public keys $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_r\}$, a secret key sk_{j^*} and a message msg, output a signature σ.

III. Verify: on input a set of public keys $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_r\}$, a message msg and a signature σ, V outputs either 1 (accept) if the signature is valid, or 0 (reject) otherwise.

A crucial feature of a ring signature scheme is that anyone in the ring can produce a valid signature (on behalf on the entire ring), and the verifier can check validity, but cannot learn the identity of the signer. This is achieved by requiring that the scheme satisfies the *Anonymity* property. This captures the idea of protecting the identity of the signer, i.e. it should be impossible for a verifier to tell which secret key was used to sign. Then, *Unforgeability* corresponds to the familiar security notion for signature schemes presented above, extended to include all ring participants. In other words, it should be infeasible to forge a valid signature without knowing at least one of the secret keys in the ring. In addition, *Correctness* is also required, as before.

Linkable ring signature schemes possess additional security features. These protocols are composed of the same three procedures described in Definition 5.4, plus a fourth one, given below:

IV. Link: on input two signatures σ and σ', output either 1 if the signatures were produced with the same secret key, or 0 otherwise.

A linkable ring signature scheme is required to satisfy the following security properties. *Linkability* asks that it is computationally infeasible for an adversary to produce more than r unlinked valid signatures, even if some or all of the r public keys are malformed. *Linkable Anonymity* guarantees that, even if an adversary is able to obtain multiple signatures from the same signer, they are still unable to determine which secret key was used. *Non-Frameability* protects the user's identity by requiring that it is computationally infeasible for an adversary to produce a valid signature linked to one produced by an honest party.

Remark 5.1. Note that the linkability property is usually formulated in an alternative way, that is, if more than r valid signatures are produced, then the Link algorithm will output 1 on at least two of them. It is then easy to see, as

also pointed out in [51, Remark 2.4], that unforgeability is obtained directly by satisfying linkability and non-frameability.

Finally, we recall the definition of *commitment scheme*, which is a tool necessary for our particular construction of ring signatures. This is a non-interactive function $\mathsf{Com} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{2\lambda}$ mapping a message string of arbitrary length to a commitment string of 2λ bits. The first λ bits of input, chosen uniformly at random, guarantee that the input message is hidden in a very strong sense, as captured in the next definition.

Definition 5.5. Given an adversary A, we define the two following quantities:

$$\mathsf{Adv}^{\mathsf{Bind}}(\mathsf{A}) = \mathsf{Pr}\Big[\, \mathsf{Com}(\mathsf{r},\mathbf{x}) = \mathsf{Com}(\mathsf{r}',\mathbf{x}') \mid (\mathsf{r},\mathbf{x},\mathsf{r}',\mathbf{x}') \leftarrow \mathsf{A}(1^\lambda)\Big];$$

$$\mathsf{Adv}^{\mathsf{Hide}}(\mathsf{A},\mathbf{x},\mathbf{x}') = \Big| \underset{\mathsf{r} \leftarrow \{0,1\}^\lambda}{\mathsf{Pr}}\Big[\mathsf{A}(\mathsf{Com}(\mathsf{r},\mathbf{x})) = 1\Big] - \underset{\mathsf{r} \leftarrow \{0,1\}^\lambda}{\mathsf{Pr}}\Big[\mathsf{A}(\mathsf{Com}(\mathsf{r},\mathbf{x}')) = 1\Big]\Big|.$$

We say that Com is *computationally binding* if, for all polynomial-time adversaries A, the quantity $\mathsf{Adv}^{\mathsf{Bind}}(\mathsf{A})$ is negligible in λ. We say that Com is *computationally hiding* if, for all polynomial-time adversaries A and every pair $(\mathbf{x},\mathbf{x}')$, the quantity $\mathsf{Adv}^{\mathsf{Hide}}(\mathsf{A})$ is negligible in λ.

5.4 The Matrix Code Equivalence Problem

A $[m \times n, k]$ *matrix code* is a subspace $\mathcal{C}$ of $\mathcal{M}_{m,n}(\mathcal{K})$. These objects are usually measured with the rank metric, where the *distance* between two matrices $\mathbf{A},\mathbf{B} \in \mathcal{M}_{m,n}(\mathcal{K})$ is defined as $d(\mathbf{A},\mathbf{B}) = \mathrm{Rank}(\mathbf{A}-\mathbf{B})$. We denote the basis of the subspace by $\langle \mathbf{C_1},\ldots,\mathbf{C_k}\rangle$, where the $\mathbf{C_i}$'s are linearly independent elements of $\mathcal{M}_{m,n}(\mathcal{K})$. Due to symmetry, without loss of generality, in the rest of the text we will assume $m \leqslant n$.

For a matrix $\mathbf{A} \in \mathcal{M}_{m,n}(\mathcal{K})$, let Vec be a mapping that sends a matrix $\mathbf{a}$ to the vector $\mathrm{Vec}(\mathbf{A}) \in \mathbb{F}_q^{mn}$ obtained by 'flattening' $\mathbf{A}$, i.e.:

$$\mathrm{Vec}: \ \mathbf{A} = \begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix} \mapsto \mathrm{Vec}(\mathbf{A}) = (a_{1,1},\ldots,a_{1,n},\ldots,a_{m,1},\ldots,a_{m,n}).$$

The inverse operation is denoted by Mat, i.e. $\mathrm{Mat}(\mathrm{Vec}(\mathbf{A})) = \mathbf{A}$. Using the map Vec, an $[m \times n, k]$ matrix code can be thought of as an $\mathbb{F}_q$-subspace of

$\mathbb{F}_q^{mn}$, and thus we can represent it with a generator matrix $\mathbf{G} \in \mathbb{F}_q^{k \times mn}$, in a manner similar to the common representation for linear codes. Indeed, if $\mathcal{C}$ is an $[m \times n, k]$ matrix code over $\mathbb{F}_q$, we denote by $\mathrm{Vec}(\mathcal{C})$ the vectorization of $\mathcal{C}$ i.e.:

$$\mathrm{Vec}(\mathcal{C}) := \{\mathrm{Vec}(\mathbf{A}) \colon \mathbf{A} \in \mathcal{C}\}.$$

In this case, $\mathrm{Vec}(\mathcal{C})$ is a k-dimensional $\mathbb{F}_q$-subspace of $\mathbb{F}_q^{mn}$.

Definition 5.6. Let $\mathcal{C}$ and $\mathcal{D}$ be two $[m \times n, k]$ matrix codes over $\mathbb{F}_q$. We say that $\mathcal{C}$ and $\mathcal{D}$ are *equivalent* if there exist two matrices $\mathbf{A} \in \mathrm{GL}_m(q)$ and $\mathbf{B} \in \mathrm{GL}_n(q)$ such that $\mathcal{D} = \mathbf{A}\mathcal{C}\mathbf{B}$, i.e. for all $\mathbf{C} \in \mathcal{C}$, $\mathbf{ACB} \in \mathcal{D}$.

The equivalence between two matrix codes can be expressed using the Kronecker product of $\mathbf{A}^\top$ and $\mathbf{B}$, which we denote by $\mathbf{A}^\top \otimes \mathbf{B}$.

Lemma 5.7. Let $\mathcal{C}$ and $\mathcal{D}$ be two $[m \times n, k]$ matrix codes over $\mathbb{F}_q$. Suppose that $\mathcal{C}$ and $\mathcal{D}$ are equivalent with $\mathcal{D} = \mathbf{A}\mathcal{C}\mathbf{B}$, with $\mathbf{A} \in \mathrm{GL}_m(q)$ and $\mathbf{B} \in \mathrm{GL}_n(q)$. If $\mathbf{G}$ and $\mathbf{G}'$ are generator matrices for $\mathcal{C}$ and $\mathcal{D}$ respectively, then there exists a $\mathbf{T} \in \mathrm{GL}_k(q)$ such that $\mathbf{G}' = \mathbf{T}\mathbf{G}(\mathbf{A}^\top \otimes \mathbf{B})$.

It is common to write the generator matrices in systematic form (i.e., as a matrix of the shape $(I|M)$); we denote this operation by SF. Following Lemma 5.7, this gives us that $\mathcal{D} = \mathbf{A}\mathcal{C}\mathbf{B}$ if and only if $\mathsf{SF}(\mathbf{G}') = \mathsf{SF}(\mathbf{G}(\mathbf{A}^\top \otimes \mathbf{B}))$.
To simplify notation, we introduce the following operator:

$$\pi_{\mathbf{A},\mathbf{B}}(\mathbf{G}) := \mathbf{G}(\mathbf{A}^\top \otimes \mathbf{B}).$$

We are now ready to describe some hard problems connected to the objects we just introduced. The Matrix Code Equivalence (MCE) problem is formally defined as follows:

Problem 5.2 (Matrix Code Equivalence)**.** $\mathsf{MCE}\ (k, n, m, \mathcal{C}, \mathcal{D})$:
Given: Two k-dimensional matrix codes $\mathcal{C}, \mathcal{D} \subset \mathcal{M}_{m,n}(q)$.
Goal: Determine if there exist $\mathbf{A} \in \mathrm{GL}_m(q)\mathbf{B} \in \mathrm{GL}_n(q)$ such that $\mathcal{D} = \mathbf{A}\mathcal{C}\mathbf{B}$.

The map $(\mathbf{A}, \mathbf{B}) : \mathbf{C} \mapsto \mathbf{ACB}$ is an *isometry* between $\mathcal{C}$ and $\mathcal{D}$, in the sense that it preserves the rank i.e. $\mathrm{Rank}\,\mathbf{C} = \mathrm{Rank}(\mathbf{ACB})$. When $n = m$, such isometries can also be extended by transpositions of codewords, however, we choose to work with this smaller set of isometries for simplicity, at no cost to cryptographic security. Note that, although we defined MCE as a decisional problem, our signature construction relies on the computational version of it.

Remark 5.2. We thank Giuseppe D'Alconzo for the following sharp observation: An MCE instance of dimension k with $m \times n$ matrices over $\mathbb{F}_q$ can be viewed as a 3-tensor problem, which is symmetrical in its arguments k, m and n. This means that it is equivalent to an MCE instance of dimension m with $k \times n$ matrices and to an MCE instance of dimension n with $k \times m$ matrices. Switching to equivalent instances is a matter of changing perspective on the $k \times m \times n$ object over $\mathbb{F}_q$ defined by $A_{ijl} = A_{ij}^{(l)}$. In other words, each basis matrix $m \times n$ defines a slice of a cube, and one can take different slices for equivalent instances.

The following related problem is at the basis of the ring signature construction.

Problem 5.3 (Inverse Matrix Code Equivalence)**.** IMCE $(k, n, m, \mathcal{C}, \mathcal{D}_1, \mathcal{D}_2)$:
Given: Three k-dimensional matrix codes $\mathcal{C}, \mathcal{D}_1, \mathcal{D}_2 \subset \mathcal{M}_{m,n}(q)$.
Goal: Determine if there exist $\mathbf{A} \in \mathrm{GL}_m(q)\mathbf{B} \in \mathrm{GL}_n(q)$ such that $\mathcal{D}_1 = \mathbf{A}\mathcal{C}\mathbf{B}$ and $\mathcal{D}_2 = \mathbf{A}^{-1}\mathcal{C}\mathbf{B}^{-1}$.

Problem 5.3 is closely connected to MCE, in a manner similar to the well-known connection between discrete logarithm and related problems, i.e.:

$$\mathsf{DLOG} \geq \mathsf{DDH} \geq \mathsf{InverseDDH}.$$

Reductions in the opposite direction are not known, as explained for example in [29]. Although this does not provide any further indication about the complexity of such problems, no explicit weaknesses are known either, and the problems are generally considered hard. A more generic overview about hardness of group action-based problems (of which the discrete logarithm is only a particular instantiation) is given in [141], with similar conclusions. For our specific case, we present a discussion about the concrete hardness of IMCE in Section 5.7.

Finally, we present a *multiple-instance* version of MCE, which is at the base of one of the optimizations, using *multiple public keys*, which we will describe in Section 5.6. It is easy to see that this new problem reduces to MCE, as done for instance in [33] for the Hamming case.

Problem 5.4 (Multiple Matrix Code Equivalence)**.** MIMCE $(k, n, m, r, \mathcal{C}, \mathcal{D}_1, \ldots, \mathcal{D}_r)$:
Given: $(r+1)$ k-dimensional matrix codes $\mathcal{C}, \mathcal{D}_1, \ldots, \mathcal{D}_r \subset \mathcal{M}_{m,n}(\mathcal{K})$.
Goal: Find – if any – $\mathbf{A} \in \mathrm{GL}_m(q)\mathbf{B} \in \mathrm{GL}_n(q)$ such that $\mathcal{D}_i = \mathbf{A}\mathcal{C}\mathbf{B}$ for some $i \in \{1, \ldots, r\}$.

The MCE problem has been shown to be at least as hard as the Code Equivalence problem in the Hamming metric [114]. Furthermore, under moderate assumptions, MCE is equivalent to the homogeneous version of the Quadratic Maps Linear Equivalence problem (QMLE) [248], which is considered the hardest among polynomial equivalence problems. An extensive security evaluation will be given in Section 5.7, encompassing an overview of the best attack techniques and concrete security estimates. From this, we infer a choice of parameters in Section 5.8.2.

To conclude, we now lay out the details of the MCE-based group action, given by the action of isometries on k-dimensional matrix codes. That is, the set X is formed by the k-dimensional matrix codes of size $m \times n$ over some base field $\mathcal{K}$, and the group $G = \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$ acts on this set via isometries as follows:

$$\begin{array}{rccc} \star: & G \times X & \to & X \\ & ((\mathbf{A}, \mathbf{B}), \mathcal{C}) & \mapsto & \mathbf{A}\mathcal{C}\mathbf{B} \end{array}$$

We write $\mathcal{G}_{m,n}(q)$ to denote this group of isometries and $\mathbf{M}_{k,m,n}(q)$ for the set of k-dimensional matrix codes; to simplify notation, we drop the indices k, m, n and q when clear from context. Then, for this MCE-based group action the Vectorization Problem is precisely Problem 5.2. This action is not commutative and in general neither transitive nor free. We can restrict the set $\mathbf{M}$ to a single well-chosen orbit to make the group action both transitive and free. In fact, picking any orbit generated from some starting code $\mathcal{C}$ ensures transitivity, and the group action is free if the chosen code $\mathcal{C}$ has trivial automorphism group $\mathrm{Aut}_{\mathcal{G}}(\mathcal{C}) := \{\phi \in \mathcal{G} : \phi(\mathcal{C}) = \mathcal{C}\}$, where trivial means up to scalars in $\mathbb{F}_q$[2]. The non-commutativity is both positive and negative: although it limits the cryptographical design possibilities, e.g. key exchange becomes hard, it prevents quantum attacks to which commutative cryptographic group actions are vulnerable, such as Kuperberg's algorithm for the dihedral hidden subgroup problem [192].

With regards to efficiency, it is immediate to notice that our group action is very promising, given that the entirety of the operations in the proposed protocols is simple linear algebra; this is in contrast with code-based literature (where complex decoding algorithms are usually required) and other group actions (e.g. isogeny-based) which are burdened by computationally heavy operations. Further details about performance are given in Section 5.8.

[2]More accurately, as the action of an isometry (A, B) is only interesting up to scalars $\lambda, \mu \in \mathbb{F}_q$, the group that is acting *freely* is $\mathrm{PGL}_m(q) \times \mathrm{PGL}_n(q)$.

Public Data
$q, m, n, k, \lambda \in \mathbb{N}$.
$\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2\lambda}$.

II. Commit(pk)

1. $\tilde{\mathbf{A}}, \tilde{\mathbf{B}} \xleftarrow{\$} \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$.
2. Compute $\tilde{\mathbf{G}} = \mathsf{SF}(\pi_{\tilde{\mathbf{A}},\tilde{\mathbf{B}}}(\mathbf{G}_0))$.
3. Compute $h = \mathsf{H}(\tilde{\mathbf{G}})$.
4. Set $\mathsf{cmt} = h$.
5. Send cmt to verifier.

IV. Response($\mathsf{sk}, \mathsf{pk}, \mathsf{cmt}, \mathsf{chal}$)

1. If $\mathsf{chal} = 0$ set $(\mu, \nu) = (\tilde{\mathbf{A}}, \tilde{\mathbf{B}})$.
2. If $\mathsf{chal} = 1$ set $(\mu, \nu) = (\tilde{\mathbf{A}}\mathbf{A}^{-1}, \mathbf{B}^{-1}\tilde{\mathbf{B}})$.
3. Set $\mathsf{resp} = (\mu, \nu)$.
4. Send resp to verifier.

I. Keygen()

1. $\mathbf{G}_0 \xleftarrow{\$} \mathbb{F}_q^{k \times mn}$ in standard form
2. $(\mathbf{A}, \mathbf{B}) \xleftarrow{\$} \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$.
3. Compute $\mathbf{G}_1 = \mathsf{SF}(\pi_{\mathbf{A},\mathbf{B}}(\mathbf{G}_0))$.
4. Set $\mathsf{sk} = (\mathbf{A}, \mathbf{B})$ and $\mathsf{pk} = (\mathbf{G}_0, \mathbf{G}_1)$.

III. Challenge()

1. $c \xleftarrow{\$} \{0,1\}$.
2. Set $\mathsf{chal} = c$.
3. Send chal to prover.

V. Verify($\mathsf{pk}, \mathsf{cmt}, \mathsf{chal}, \mathsf{resp}$)

1. If $\mathsf{chal} = 0$ compute $h' = \mathsf{H}(\mathsf{SF}(\pi_{\mu,\nu}(\mathbf{G}_0)))$.
2. If $\mathsf{chal} = 1$ compute $h' = \mathsf{H}(\mathsf{SF}(\pi_{\mu,\nu}(\mathbf{G}_1)))$.
3. Accept if $h' = \mathsf{cmt}$ or reject otherwise.

Figure 5.1: MCE Sigma Protocol

5.5 Protocols from Matrix Code Equivalence

The efficient non-commutative cryptographic group action provided by MCE from Section 5.4 yields a promising building block for post-quantum cryptographic schemes. In this section, we obtain a digital signature scheme by first designing a Sigma protocol and then applying the Fiat-Shamir transformation [**fiat1986prove**]. With a similar procedure, we are able to obtain (linkable) ring signatures as well.

5.5.1 Sigma Protocols and Signatures

The first building block in our work is the Sigma protocol in Figure 5.1, in which a Prover proves the knowledge of an isometry $(\mathbf{A}, \mathbf{B})$ between two equivalent matrix codes. The security result is given in Theorem 5.8.

Theorem 5.8. The Sigma protocol described above is complete, 2-special sound and honest-verifier zero-knowledge assuming the hardness of the MCE problem.

Proof. We prove the three properties separately.

Completeness. An honest prover will always have his response accepted by the verifier. In fact, for the case $c = 0$, we have

$$h' = \mathsf{H}(\mathsf{SF}(\pi_{\mu,\nu}(\mathbf{G}_0))) = \mathsf{H}(\mathsf{SF}(\pi_{\tilde{\mathbf{A}},\tilde{\mathbf{B}}}(\mathbf{G}_0)) = \mathsf{H}(\tilde{\mathbf{G}}) = h.$$

For the case $c = 1$, instead, observe that

$$\pi_{\tilde{\mathbf{A}}\mathbf{A}^{-1},\mathbf{B}^{-1}\tilde{\mathbf{B}}}(\mathbf{G}_1) = \pi_{\tilde{\mathbf{A}},\tilde{\mathbf{B}}}\left(\pi_{\mathbf{A}^{-1},\mathbf{B}^{-1}}(\mathbf{G}_1)\right).$$

Since $\mathbf{G}_1 = \mathsf{SF}(\pi_{\mathbf{A},\mathbf{B}}(\mathbf{G}_0))$, then $\mathbf{G}_1 = \mathbf{T}\left(\pi_{\mathbf{A},\mathbf{B}}(\mathbf{G}_0)\right)$ for some $\mathbf{T} \in \mathrm{GL}_k(q)$. Hence, we have that

$$\pi_{\tilde{\mathbf{A}}\mathbf{A}^{-1},\mathbf{B}^{-1}\tilde{\mathbf{B}}}(\mathbf{G}_1) = \pi_{\tilde{\mathbf{A}},\tilde{\mathbf{B}}}\left(\pi_{\mathbf{A}^{-1},\mathbf{B}^{-1}}\left(\mathbf{T}\left(\pi_{\mathbf{A},\mathbf{B}}(\mathbf{G}_0)\right)\right)\right) = \mathbf{T}\left(\pi_{\tilde{\mathbf{A}},\tilde{\mathbf{B}}}(\mathbf{G}_0)\right).$$

Now, computing the systematic form and applying H, we again have $h' = h$.

2-Special Soundness. To get the extractor which produces the witness, we prove that having obtained two valid transcripts with the same commitment and a different challenge, we can extract a solution for the underlying MCE problem. This demonstrates that this Sigma protocol is a Proof of Knowledge with soundness error $1/2$.

Let $(h, 0, \mathsf{resp}_0)$ and $(h, 1, \mathsf{resp}_1)$ be two valid transcripts, where $\mathsf{resp}_0 = (\mu_0, \nu_0)$ and $\mathsf{resp}_1 = (\mu_1, \nu_1)$. Since both pass verification, we have that

$$\mathsf{H}(\mathsf{SF}(\pi_{\mu_0,\nu_0}(\mathbf{G}_0))) = h = \mathsf{H}(\mathsf{SF}(\pi_{\mu_1,\nu_1}(\mathbf{G}_1))).$$

Then, since we assume that H is collision resistant, it must be that

$$\mathsf{SF}(\pi_{\mu_0,\nu_0}(\mathbf{G}_0)) = \mathsf{SF}(\pi_{\mu_1,\nu_1}(\mathbf{G}_1)).$$

From this, it is easy to see that $\mathsf{SF}(\pi_{\mu^*,\nu^*}(\mathbf{G}_0)) = \mathbf{G}_1$, for an isometry $(\mu^*, \nu^*) = = (\mu_1^{-1}\mu_0, \nu_0\nu_1^{-1})$, i.e. (μ^*, ν^*) is a solution to the MCE problem.

Honest-Verifier Zero-Knowledge. To show this, we provide a simulator S which, without the knowledge of the witness, is able to produce a transcript which is indistinguishable from one obtained after an interaction with an honest verifier. When the challenge is $c = 0$, the prover's response is $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}})$, and does not involve the witness. Hence, it is obvious that S can obtain a flawless simulation by simply performing the same steps as an honest prover would. Thus,

$$\mathsf{Pr}\Big[\mathsf{V}_2(\mathsf{pk},\mathsf{cmt},0,\mathsf{resp})=1 \mid (\mathsf{cmt},0,\mathsf{resp}) \leftarrow \mathsf{S}(\mathsf{pk})\Big] = 1.$$

When $c = 1$, the simulator generates two random matrices $(\mathbf{A}^*, \mathbf{B}^*)$, then sets the commitment to $\mathsf{cmt} = \mathsf{H}(SF(\pi_{\mathbf{A}^*,\mathbf{B}^*}(\mathbf{G}_1)))$ and the response to $\mathsf{resp} = (\mathbf{A}^*, \mathbf{B}^*)$. When $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}})$ is uniformly generated, then the matrices $(\tilde{\mathbf{A}}\mathbf{A}^{-1}, \mathbf{B}^{-1}\tilde{\mathbf{B}})$ are uniformly generated too, and hence follow the same distribution as $(\mathbf{A}^*, \mathbf{B}^*)$. Furthermore, the triple $(\mathsf{cmt}, 1, \mathsf{resp})$ is valid and therefore

$$\mathsf{Pr}\Big[\mathsf{V}_2(\mathsf{pk},\mathsf{cmt},1,\mathsf{resp})=1 \mid (\mathsf{cmt},1,\mathsf{resp}) \leftarrow \mathsf{S}(\mathsf{pk})\Big] = 1.$$

This concludes the proof. □
□

Applying the Fiat-Shamir transformation gives the signature scheme in Figure 5.2.

Public key and signature size.

We begin by calculating the communication costs for the Sigma protocol of Figure 5.1. Note that, for the case $c = 0$, the response (μ, ν) consists entirely of randomly-generated objects, and is efficiently represented by a single seed (that can be used to generate both matrices). This yields the following cost per round, in bits:

$$\begin{cases} 3\lambda + 1 & \text{if } c = 0 \\ 2\lambda + 1 + (m^2 + n^2)\lceil \log_2(q) \rceil & \text{if } c = 1 \end{cases}$$

remembering that seeds are λ bits and hash digests 2λ to avoid collision attacks.

For the signature scheme we calculate the sizes as follows. First, since the matrix $\mathbf{G}_0$ is random, it can also be represented via a short seed, and therefore can be included in the public key at negligible cost (see Algorithm I. of Figure 5.2). Keeping in mind that the number of rounds t is equal to the value of the desired security level λ, the protocol above yields the following sizes (in bits):

- Public key size: $\lambda + k(mn - k)\lceil \log_2(q) \rceil$
- Average signature size: $t\left(1 + \dfrac{\lambda + (m^2 + n^2)\lceil \log_2(q) \rceil}{2}\right)$.

5.5.2 Ring Signatures

The protocol of Figure 5.1 can be easily adapted to serve as a building block for a ring signature, by providing the ring of users with individual public keys, corresponding to equivalent matrix codes. Any user can then answer the verifier's challenge via the selected private key. The construction crucially utilizes a dedicated primitive known as *Index-Hiding Merkle Tree (IHMT)*. This primitive was first introduced in [51] as a variation on the traditional construction of Merkle trees. With this variant, in fact, the position of a leaf is not revealed, which is necessary to ensure anonymity. This can be accomplished by specifying a different method to construct the tree, based on an alternative ordering (e.g. lexicographic). For further details, we refer the reader to [51]. A visual representation is given in Figure 5.3, where we remind the reader that Algorithm I. Keygen is executed *once for each prover*, Algorithm II. Commit is the same regardless of the chosen prover, and Algorithm IV. Response is instead unique for the specific prover identified by $j^* \in \{1, \dots, r\}$.

It is easy to see that the construction in Figure 5.3 satisfies the Completeness, 2-Special Soundness and Honest-Verifier Zero-Knowledge properties, similarly to the protocol of Figure 5.1. The proof is analogue to the proof of Theorem 5.8, with only minor differences (such as the reversal in the roles of the challenges), that have no noticeable impact; this is therefore omitted in the interest of space. The protocol can then be turned into a ring signature scheme using Fiat-Shamir as usual, i.e. in the same manner as what was done for the protocol of Figure 5.1. Furthermore, we can make the ring signature scheme *linkable*, by means of a few modifications which we explain next. To avoid needless repetition, we avoid presenting the ring signature scheme in its extended form (as in Figure 5.2) and we move on instead to discussing the linkable variant; we will then present the linkable ring signature scheme, in its entirety, to conclude this section.

To begin, recall the group action $\star : G \times X \to X$, formalized in Section 5.4, with $X = \mathcal{M}$ the set of k-dimensional matrix codes, $G = \mathcal{G}$ the group of isometries for such codes, and the action given by $\star : ((\mathbf{A}, \mathbf{B}), \mathcal{C}) \mapsto \mathbf{A}\mathcal{C}\mathbf{B}$. We now require another group action $* : G \times Y \to Y$, satisfying the following properties.

Definition 5.9. Let $\star : G \times X \to X$ and $* : G \times Y \to Y$ be two group actions. We define the following properties for the pair $(\star, *)$:

- *Linkability*: Given $(x, y) \in X \times Y$, it is hard to output $g, g' \in G$ with $g \star x = g' \star x$ and $g * y \neq g' * y$.

- *Linkable Anonymity*: Given $(x, y) \in X \times Y$, the pair $(g \star x, g * y)$ is indistinguishable from (x', y'), where g and (x', y') are sampled uniformly at random from G and $X \times Y$, respectively.

- *Non-Frameability*: Given $(x, y) \in X \times Y$, $x' = g \star x$ and $y' = g * y$, for g sampled uniformly at random from G, it is hard to output $g' \in G$ with $g' * y = y'$.

Note how these three properties recall the Linkability, Linkable Anonymity and Non-Frameability properties introduced in Section 5.3.2. Indeed, showing that the pair of group actions satisfies the above properties is the key to constructing a secure linkable ring signature scheme. Also, as noted before, one could define unforgeability in a manner similar to the last property (by asking to output $g' \in G$ with $g' \star x = x'$) but this is not necessary, since this is a direct consequence of linkability and non-frameability. Informally, then, one can treat elements $y \in Y$ as "tags", that can be checked to establish the link. To this end, it is convenient to introduce an efficiently computable function $\mathsf{Link} : Y \times Y \to \{0, 1\}$, defined by $\mathsf{Link}(y, y') = 1 \Leftrightarrow y = y'$. For our purposes, we require $Y = X = \mathcal{M}$ and define $*$ as $\star^{-1}$. Thus, $* : ((\mathbf{A}, \mathbf{B}), \mathcal{C}) \mapsto \mathbf{A}^{-1}\mathcal{C}\mathbf{B}^{-1}$. We will then show that the required properties hold for any given code $\mathcal{C}$, with the IMCE problem as a basis for its security.

Theorem 5.10. The pair of group actions $(\star, *)$ described above is linkable, linkably anonymous and non-frameable assuming the hardness of the IMCE problem.

Proof. We prove the three properties separately.

Linkability. Consider a code $\mathcal{C}$ defined by $\mathbf{G}$, together with two isometries $g = (\mathbf{A}, \mathbf{B})$ and $g' = (\mathbf{A}', \mathbf{B}')$. Suppose that $\mathsf{SF}(\pi_{\mathbf{A},\mathbf{B}}(\mathbf{G})) = \mathsf{SF}(\pi_{\mathbf{A}',\mathbf{B}'}(\mathbf{G}))$ or, equivalently, that $\mathsf{SF}(\mathbf{G}(\mathbf{A}^\top \otimes \mathbf{B})) = \mathsf{SF}(\mathbf{G}((\mathbf{A}')^\top \otimes \mathbf{B}'))$. Then, it must be that $\mathbf{A}^\top \otimes \mathbf{B} = (\mathbf{A}')^\top \otimes \mathbf{B}'$, unless there is an automorphism of $\mathcal{C}$ hidden in the product. As noted in [248], it is plausible to assume that a random code $\mathcal{C}$ admits only the trivial automorphism, for all parameter choices we are interested in. If so, such trivial automorphism are pairs of scalar multiples of the identity for scalars $\alpha \in \mathbb{F}_q$. Hence, either we have $(\mathbf{A}^\top \otimes \mathbf{B})^{-1} = ((\mathbf{A}')^\top \otimes \mathbf{B}')^{-1}$; or, $(\mathbf{A}^\top \otimes \mathbf{B})^{-1} = \alpha^{-1}((\mathbf{A}')^\top \otimes \mathbf{B}')^{-1}$. In both cases, we have that $\mathsf{SF}(\pi_{\mathbf{A}^{-1},\mathbf{B}^{-1}}(\mathbf{G})) = \mathsf{SF}(\pi_{(\mathbf{A}')^{-1},(\mathbf{B}')^{-1}}(\mathbf{G}))$ i.e. $g * y = g' * y$.

Linkable Anonymity. This follows directly from the hardness of the IMCE problem. As discussed in Section 5.4, the problem is believed to be only marginally easier than MCE. In Section 5.7, we analyse dedicated attack techniques for IMCE.

Non-Frameability. For this property, an adversary A is given again a code $\mathcal{C}$ defined by $\mathbf{G}$ and codes x' and y' defined by $\mathsf{SF}(\pi_{\mathbf{A},\mathbf{B}}(\mathbf{G}))$ and $\mathsf{SF}(\pi_{\mathbf{A}^{-1},\mathbf{B}^{-1}}(\mathbf{G}))$ respectively, where $g = (\mathbf{A}, \mathbf{B})$ is chosen uniformly at random in $\mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$. The adversary A is asked to find $(\mathbf{A}', \mathbf{B}')$ such that

$$\mathsf{SF}(\pi_{(\mathbf{A}')^{-1},(\mathbf{B}')^{-1}}(\mathbf{G})) = y' = \mathsf{SF}(\pi_{\mathbf{A}^{-1},\mathbf{B}^{-1}}(\mathbf{G})).$$

We can use such an adversary to build a distinguisher D for the IMCE problem, as follows. Suppose $(\mathcal{C}, \mathcal{D}_1, \mathcal{D}_2)$ is the given instance of the problem, and let ϵ be the success probability of A. To begin, D calls A on $(\mathcal{C}, \mathcal{D}_1, \mathcal{D}_2)$, and A will reply with $g' = (\mathbf{A}', \mathbf{B}')$ that satisfies the equation above.

$$\mathsf{SF}(\pi_{(\mathbf{A}')^{-1},(\mathbf{B}')^{-1}}(\mathbf{G})) = y' = \mathsf{SF}(\pi_{\mathbf{A}^{-1},\mathbf{B}^{-1}}(\mathbf{G})).$$

From what we have seen in the proof of the Linkability property above, this implies that $((\mathbf{A}')^\top \otimes \mathbf{B}')^{-1} = (\mathbf{A}^\top \otimes \mathbf{B})^{-1}$, modulo trivial automorphisms. Either way, we have that $\mathsf{SF}(\pi_{\mathbf{A},\mathbf{B}}(\mathbf{G})) = \mathsf{SF}(\pi_{\mathbf{A}',\mathbf{B}'}(\mathbf{G}))$. Thus, it is enough for D to compute $\mathsf{SF}(\pi_{\mathbf{A}',\mathbf{B}'}(\mathbf{G}))$ and $\mathsf{SF}(\pi_{(\mathbf{A}')^{-1},(\mathbf{B}')^{-1}}(\mathbf{G})))$, and check whether they define $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. D then answers 1 if this is the case, or 0 otherwise. Since the probability that a randomly-drawn pair $(\mathcal{D}_1, \mathcal{D}_2)$ satisfies this condition is negligible, the probability of success of D is essentially the same as ϵ.

Together, these three properties complete the proof. □ □

Having clarified the nature of the tools at hand, we are now ready to introduce the linkable version of the ring signature scheme. We first explain compactly how to modify the protocol of Figure 5.3, then give a full description in Figure 5.4.

- As part of the public data, choose a collision-resistant hash function $\overline{\mathsf{H}}$.
- The commit procedure (Algorithm II.) now additionally uses a matrix $\mathbf{T} = \mathsf{SF}(\pi_{\mathbf{A}_{j^*}^{-1},\mathbf{B}_{j^*}^{-1}}(\mathbf{G}_0))$; this is the "tag" for prover j^*, which will be trasmitted alongside the commitment to the verifier. The prover then

computes $\tilde{\mathbf{T}} = \mathsf{SF}(\pi_{\tilde{\mathbf{A}}^{-1},\tilde{\mathbf{B}}^{-1}}(\mathbf{T}))$. After obtaining the root of the IHMT, the prover obtains $\bar{h} = \overline{\mathsf{H}}(\tilde{\mathbf{T}}, \mathsf{root})$ and the commitment is set to $\bar{h}$ instead of root.

- Algorithm IV. is identical, except that, in the case $c = 0$, the response has to include also the matrices $\bar{\mu}, \bar{\nu} = \mathbf{A}_{j^*}\tilde{\mathbf{A}}, \tilde{\mathbf{B}}\mathbf{B}_{j^*}$.

- Finally, Algorithm V. is modified to involve a check on the tag, as well. The case $c = 1$ is trivial, with the verifier essentially repeating the commitment procedure (Algorithm II.) as in the original protocol, only this time checking equality with $\bar{h}$ rather than root. For the case $c = 0$, instead, the verifier additionally computes $\hat{\mathbf{T}} = \mathsf{SF}(\pi_{\bar{\mu}^{-1},\bar{\nu}^{-1}}(\mathbf{G}_0))$, then uses this value to obtain $\overline{\mathsf{H}}(\hat{\mathbf{T}}, \mathsf{root}')$ and verify equality with $\bar{h}$.

It is relatively easy to see that the new protocols satisfy all the necessary security properties. For instance, the ring Sigma protocol, with the above modifications, still satisfies the Completeness, 2-Special Soundness and Honest-Verifier Zero-Knowledge properties. The linkable ring signature scheme of Figure 5.4 is then just an application of the Fiat-Shamir transform. Once again, such proofs are omitted due to space limitations; the interested reader can consult for example [51], where proofs are given in all generality (see e.g. Theorems 4.3, 4.4 and 4.7). We will present computational costs for the (linkable) ring signature scheme in the next section, after discussing optimizations.

5.6 Matrix Equivalence Digital Signature — MEDS

We apply the following optimizations from the literature to the basic Fiat-Shamir-based signature scheme described in Section 5.5, to obtain our Matrix Equivalence Digital Signature (MEDS).

Multiple keys.

The first optimization is a popular one in literature [33, 53, 117], and it consists of utilizing multiple public keys, i.e. multiple equivalent codes $\mathbf{G}_0, \ldots, \mathbf{G}_{s-1}$, each defined as $\mathbf{G}_i = S^{\mathbf{0}}(\pi_{\mathbf{A}_i,\mathbf{B}_i}(\mathbf{G}_0))$ for uniformly chosen secret keys[3] $(\mathbf{A}_i, \mathbf{B}_i)$. This allows to reduce the soundness error from $1/2$ to $1/2^\ell$, where $\ell = \lceil \log_2 s \rceil$. The optimization works by grouping the challenge bits into strings of ℓ bits,

[3] Again, for convenience, we choose $\mathbf{A}_0 = \mathbf{I}_m$, $\mathbf{B}_0 = \mathbf{I}_n$.

which can then be interpreted as binary representations of the indices $\{0, \ldots, s-1\}$, thus dictating which public key will be used in the protocol. Security is preserved since the proof of unforgeability can be easily modified to rely on a multi-instance version of the underlying problem: in our case, MIMCE (Problem 5.4). Note that, although in the literature s is chosen to be a power of 2, this does not have to be the case. In this work, we will instead select the value of s based on the best outcome in terms of performance and signature size.

Remark 5.3. This optimization comes at the cost of an s-fold increase in public-key size. As shown for instance in [117], it would be possible to reduce this impact by using Merkle trees to a hash of the tree commitment of all the public keys. This, however, would add some significant overhead to the signature size, because it would be necessary to include the paths for all openings. Considering the sizes of the objects involved, such an optimization is not advantageous in our case.

Partially seeding the public key.

The previous optimization comes at significant cost to the public key, so we propose a new optimization that trades public key size for private key size. This optimization is inspired by the trade-off in the key generation of Rainbow [129] and UOV [50]. It has not been previously used in Fiat-Shamir signatures, but we expect it can be used successfully in any scheme coming from equivalence problems especially the ones using the previous optimization, such as [33, 53, 117]. With this optimization, instead of generating the secret $(\mathbf{A}_i, \mathbf{B}_i)$ from a secret seed and then deriving the public $\mathbf{G}_i$, we generate $\mathbf{G}_i$ partially from a public seed and then use it to find $(\mathbf{A}_i, \mathbf{B}_i)$ and the rest of the public key $\mathbf{G}_i$. In more detail, in order to generate the public $\mathbf{G}_i$ and the corresponding secret $(\mathbf{A}_i, \mathbf{B}_i)$ we perform the following:

- We perform a secret change of basis of $\mathbf{G}_0$ by multiplying it by a secret matrix $\mathbf{T} \in \mathrm{GL}_k(q)$ to obtain $\mathbf{G}'_0$. Assume the codewords from $\mathbf{G}'_0$ are $\mathbf{P}^0_1, \mathbf{P}^0_2, \ldots, \mathbf{P}^0_k$.

- For each $i \in \{1, \ldots, s-1\}$, we generate from a public seed a complete $m \times n$ codeword $\mathbf{P}^i_1$ and the top $m-1$ rows of codeword $\mathbf{P}^i_2$ (depending on the parameters m, n one can get slightly more rows when $m \neq n$).

- Find $\mathbf{A}_i$ and $\mathbf{B}_i$ from the linear relations:

$$\mathbf{P}_1^i \mathbf{B}_i^{-1} = \mathbf{A}_i \mathbf{P}_1^0$$
$$\mathbf{P}_2^i \mathbf{B}_i^{-1} = \mathbf{A}_i \mathbf{P}_2^0$$

  by fixing the first (top left) value of $\mathbf{A}_i$.

- Find $\mathbf{P}_j^i = \mathbf{A}_i \mathbf{P}_j^0 \mathbf{B}_i$ for all $j \in \{3, \ldots, k\}$.

- Construct the public $\mathbf{G}_i$ from $\mathbf{P}_1^i, \mathbf{P}_2^i, \ldots, \mathbf{P}_k^i$.

The public key then is the public seed together with $\mathbf{P}_3^i, \ldots, \mathbf{P}_k^i$. For verification, the complete $\mathbf{G}_i$ are reconstructed using the seed.

Fixed-weight challenges.

Another common optimization is the use of fixed-weight challenges. The idea is to generate the challenge string h with a fixed number of 1s and 0s, i.e. Hamming weight, rather than uniformly at random. This is because, when $h_i = 0$, the response (μ_i, ν_i) consists entirely of randomly-generated objects, and so one can just transmit the seed used for generating them. This creates a noticeable imbalance between the two types of responses, and hence it makes sense to minimize the number of 1 values. To this end, one can utilize a so-called *weight-restricted hash function*, that outputs values in $\mathbb{Z}_{2,w}^t$, by which we denote the set of vectors with elements in $\{0, 1\}$ of length t and weight w. In this way, although the length of the challenge strings increases, the overall communication cost scales down proportionally to the value of w. In terms of security, this optimization only entails a small modification in the statement of the Forking Lemma, and it is enough to choose parameters such that $\log_2 \binom{t}{w} \geq \lambda$. Note that this optimization can easily be combined with the previous one, by mandating hash digests in $\mathbb{Z}_{s,w}^t$ and choosing parameters such that $\log_2 \left(\binom{t}{w}(s-1)^w\right) \geq \lambda$. In practice, this can be achieved with a hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$, by expanding the output to a t-tuple $(h_0, \ldots, h_{t-1})$, $0 \leq h_i < s$ of weight w.

Seed tree.

Finally, the signature size can be optimized again using a *seed tree*. This primitive allows to generate the many seeds used throughout the protocol in a recursive way, starting from a master seed mseed and building a binary tree, via repeated PRNG applications, having t seeds as leaves. When the required $t - w$

values need to be retrieved, it is then enough to reveal the appropriate sequence of nodes. This reduces the space required for the seeds from $\lambda(t-w)$ to λN_{seeds}, where N_{seeds} can be upper bounded by $2^{\lceil \log_2(w) \rceil} + w(\lceil \log_2(t) \rceil - \lceil \log_2(w) \rceil - 1)$, as shown in [164]. We refer the reader to Section 2.7 of [51] for more details. As suggested in [51], we are including a 256-bit salt to ward off multi-target collision attacks and the leaf address as identifier for domain separation in the inputs of the seed-tree hash functions.

To give a complete picture, we present the MEDS protocol in Figure 5.5, in its final form, including all applicable variants. The various parameters control different optimization: for instance s refers to the number of public keys used, whereas w refers to the fixed weight of the challenge hash string. Parameter choices will be thoroughly discussed in Section 5.8.2. Note that some of these optimizations can be applied in an identical way to the (linkable) ring signature scheme; however, we leave an explicit description of such a scheme, as well as a full-fledged implementation of it, to a dedicated future work.

Public key and signature size.

With these various optimizations, we obtain the following public key and signature size for MEDS:

- MEDS public key size: $\lambda + (s-1)((k-2)(mn-k)+n)\lceil \log_2(q) \rceil$
- MEDS signature size:

$$\underbrace{\lambda}_{h} + \underbrace{w(m^2+n^2)\lceil \log_2(q) \rceil}_{\{\mu_i,\nu_i\}_{h_i=1}} + \underbrace{\lambda N_{\text{seeds}}}_{\{\mu_i,\nu_i\}_{h_i=0}} + \underbrace{2\lambda}_{salt}$$

The optimizations described above can also be applied to the ring signature scheme, which gives us the following sizes:

- Ring signature public key size: $\lambda + rk(mn-k)\lceil \log_2(q) \rceil$
- Ring signature size:

$$\underbrace{w\lceil \log_2 t \rceil}_{h} + \underbrace{w((m^2+n^2)\lceil \log_2(q) \rceil + 2\lambda \lceil \log r \rceil + \lambda)}_{\{\mathsf{resp}_i\}_{h_i=0}} + \underbrace{\lambda N_{\text{seeds}}}_{\{\mathsf{resp}_i\}_{h_i=1}} + \underbrace{2\lambda}_{salt}$$

For the linkable variant, the cost of the middle term, i.e. $\{\mathsf{resp}_i\}_{h_i=0}$, is increased to $w(2(m^2+n^2)\lceil \log_2(q) \rceil + 2\lambda \lceil \log r \rceil + \lambda)$, and the signature additionally includes $k(mn-k)\lceil \log_2(q) \rceil$ bits for the tag $\mathbf{T}$.

5.7 Concrete Security Analysis

In this section, we will mostly use the Big O notation $\mathcal{O}$ to express the complexity of algorithms. Where we are not interested in the polynomial factor we will use $\mathcal{O}^*$. We note that despite the notation, the estimates are quite tight and provide a good basis for choosing parameters.

Recall that the goal of an adversary against MCE is to recover the matrices $\mathbf{A}$ and $\mathbf{B}$, given a description of the matrix codes $\mathcal{C}$ and $\mathcal{D}$. The most naïve attack would be to try every $\mathbf{A} \in \mathrm{GL}_m(q)$ and $\mathbf{B} \in \mathrm{GL}_n(q)$ until we find the correct isometry, amounting to a complexity of $\mathcal{O}(q^{n^2+m^2})$. The naïve attack can be improved by noting that once one of the matrices $\mathbf{A}$ or $\mathbf{B}$ is known, the resulting problem becomes easy [114]. Hence, we only need to brute-force one of $\mathbf{A}$ or $\mathbf{B}$, so the complexity becomes $\mathcal{O}^*(q^{\min\{m^2,n^2\}})$.

In the rest of the section, we will see that there exist several non-trivial attacks that perform much better than this upper bound.

5.7.1 Birthday-based graph-theoretical algorithms for solving MCE

Recent works [114, 248] investigate the hardness of MCE by connecting it to other equivalence problems, namely, the Code Equivalence problem in the Hamming metric [114] and the Quadratic Maps Linear Equivalence problem (QMLE) [248]. The latter provides complexity analysis by viewing MCE as an instance of QMLE. We recap their results here. For better understanding, we include the definition of the related QMLE problem.

Problem 5.5. QMLE $(k, N, \mathcal{F}, \mathcal{P})$:
Given: Two k-tuples of multivariate polynomials of degree 2 variables $x_1, \ldots, x_N$

$$\mathcal{F} = (f_1, f_2, \ldots, f_k),\ \mathcal{P} = (p_1, p_2, \ldots, p_k) \in \mathcal{K}[x_1, \ldots, x_N]^k.$$

Goal: Find – if any – matrices $\mathbf{S} \in \mathrm{GL}_N(q), \mathbf{T} \in \mathrm{GL}_k(q)$ such that

$$\mathcal{P}(\mathbf{x}) = (\mathcal{F}(\mathbf{x}\mathbf{S}))\mathbf{T}.$$

We denote by hQMLE, inhQMLE and BMLE the related problems when the polynomials are homogeneous of degree 2, inhomogeneous and bilinear, respectively. It was shown in [248] that, under the assumption that the two codes $\mathcal{C}$ and $\mathcal{D}$ have trivial automorphism groups (which is believed to be true with overwhelming probability for big enough parameters), MCE $(k, n, m, \mathcal{C}, \mathcal{D})$ is

Algorithm 3 Collision-search algorithm

```
1: function BuildList(F, P)
2:     L ← ∅
3:     repeat
4:         x ←$ F_q^(m+n)
5:         if P(F, x) then L ← L ∪ {x}
6:     until |L| = ℓ
7:     return L
8: function CollisionFind(F, P)
9:     L1 ← BuildList(F, P)
10:    L2 ← BuildList(P, P)
11:    for all (x, y) ∈ {L1 × L2} do
12:        φ ← inhQMLE(x, y)
13:        if φ ≠ ⊥ then
14:            return solution φ
15:    return ⊥
```

equivalent to hQMLE $(k, N, \mathcal{F}, \mathcal{P})$ where $N = m + n$. Concretely, an MCE instance with a solution $(\mathbf{A}, \mathbf{B})$ is transformed into an hQMLE instance with a solution $(\mathbf{S}, \mathbf{T})$ where $\mathbf{S} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}^\top \end{bmatrix}$ and $\mathbf{T}$ corresponds to a change of basis of $\mathcal{D}$. Therefore it is possible to apply algorithms for solving hQMLE to MCE instances such as the graph-theoretic algorithm of Bouillaguet et al. [64].

The algorithm is basically a collision-search algorithm comprised of two steps, as given in Algorithm 3. In the first step we build two lists L_1 and L_2 of size ℓ of elements in $\mathbb{F}_q^{(m+n)}$ that satisfy a predefined distinguishing property $\mathbb{P}$ related to the given systems of polynomials $\mathcal{F}$ and $\mathcal{P}$ and that is preserved under isometry. In the second step, we try to find a collision between the two lists that will lead us to the solution. For the property $\mathbb{P}$, the authors of [64] propose:

$$\mathbb{P}(\mathcal{F}, \mathbf{x}) = \top \Leftrightarrow Dim(Ker(D_{\mathbf{x}}(\mathcal{F}))) = \kappa$$

for a suitably chosen κ, where $D_{\mathbf{x}}(\mathcal{F}) : \mathbf{y} \mapsto \mathcal{F}(\mathbf{x} + \mathbf{y}) - \mathcal{F}(\mathbf{x}) - \mathcal{F}(\mathbf{y})$ is the *differential* of $\mathcal{F}$ at a point $\mathbf{x}$. Clearly, the rank of the differential is preserved under isometry, so this is an appropriate choice of $\mathbb{P}$. Other instantiations are possible as well, as long as they are invariant under isometry, although their success depends on the distribution of elements that satisfy the property for varying κ.

Once a collision $(\mathbf{a}, \mathbf{b})$ is found, it can be used to derive an associated *inhomogeneous* QMLE instance inhQMLE $(k, (m + n), \mathcal{F}', \mathcal{P}')$ as $\mathcal{F}'(\mathbf{x}) = \mathcal{F}(\mathbf{x} + \mathbf{a})$, $\mathcal{P}'(\mathbf{x}) = \mathcal{P}(\mathbf{x} + \mathbf{b})$ on which we call an inhomogeneous solver. Since it can not be directly checked whether a pair is a collision, the solver needs to be called for each pair, similar to the guess and check approach in ISD algorithms [244].

The inhomogeneous instance can be solved much more efficiently than the homogeneous one. Heuristic evidence suggests that solving *random* instances

of the inhQMLE problem using an algebraic approach takes $\mathcal{O}((m+n)^9)$ operations [**FP06**], however, the derived inhQMLE instances from the collision-search attack are not random enough. These specific instances have a solver with a complexity of $\mathcal{O}(q^\kappa)$ [63]. As κ is typically chosen to be small, this approach is still efficient in practice. Following the analysis from [248], the concrete complexity of the algorithm for $k \leqslant 2(m+n)$ follows a birthday argument and is the maximum of the complexity of the two steps, i.e.:

$$\mathsf{max}(\sqrt{q^{(m+n)}/d} \cdot C_{\mathbb{P}}, dq^{(m+n)} \cdot C_{\mathsf{iQ}}), \tag{5.1}$$

with success probability of $\approx 63\%$. Here, $C_{\mathbb{P}}$ denotes the cost of checking whether an element satisfies the property $\mathbb{P}$, d is the proportion of elements satisfying $\mathbb{P}$ and C_{iQ} denotes the cost of a single query to inhQMLE. Note that d can be calculated as $d = 1/\mathcal{O}(q^{\kappa^2+\kappa(k-(m+n))})$ and κ is chosen such that it minimizes Equation (5.1). Asymptotically, the complexity is $\mathcal{O}^*(q^{\frac{2}{3}(m+n)})$ by balancing the steps [248]. The memory complexity is simply the size of the lists.

It is pointed out in [248] that when $k \geq 2(m+n)$, we can no longer assume that we have distinguished elements, so we need to consider *all* elements in the collision search. Thus, we have $d = 1$ and the complexity of the algorithm is simply $\mathcal{O}(q^{m+n})$. In that case, we can consider choosing arbitrarily one element $\mathbf{x}$ and checking for a collision with all other elements $\mathbf{y} \in \mathbb{F}_q^{m+n}$. This approach yields the same complexity as the previous one, but is superior because it is deterministic and has negligible memory requirements. Note that this approach was also proposed in [64], but as an inferior (in terms of *time* complexity) deterministic variant, rather than as a solution for the lack of a distinguishing property. As this attack can be applied to any parameter set, it presents an upper-bound on the complexity of a classical collision-search algorithm.

For a quantum version of Algorithm 3, both BuildList and Collision-Find can be seen as searches of unstructured databases of a certain size, hence Grover's algorithm applies to both: we can build the list L using only $\sqrt{\ell \cdot d^{-1}}$ searches, and we can find a collision using only $\sqrt{|L_1 \times L_2|}$ queries to the solver. This requires both $\mathbb{P}$ and inhQMLE to be performed in superposition. The balance between both sides remains the same. In total, the complexity of the quantum version becomes $\mathcal{O}^*(q^{\frac{1}{3}(m+n)})$.

Collision-search algorithm using non-trivial roots.

When viewing an MCE instance as an hQMLE instance, it is possible to use certain bilinear properties to improve Algorithm 3. When $n = m$, such instances

have approximately q^{2n-k-1} non-trivial roots, which can be used to improve a subroutine of Algorithm 3, and to make it deterministic instead of probabilistic [248]. In practice, such non-trivial roots exist **i)** almost always when $k < 2n$, **ii)** with probability $1/q$ for $k = 2n$, **iii)** with probability $1/q^{k+1-2n}$ for $k > 2n$. The complexity of this approach is $\mathcal{O}^*(q^n)$, if such non-trivial roots exist. This complexity is proven under the assumption that the complexity of the inhomogenous QMLE solver is no greater than $\mathcal{O}(q^n)$, which holds trivially when $k \geq n$ [248], and heuristically when $k < n$. Finding the non-trivial roots can also be done using a bilinear XL algorithm [242]. We do not consider this approach in our analysis, as it is only interesting for a subset of parameters where the systems are (over)determined, i.e. when k is close to $m + n$.

5.7.2 Algebraic attacks

Direct modelling.

Recently, in [248], it was shown that MCE is equivalent to BMLE. which means that any solver for BMLE can be easily adapted to solve MCE. One of the natural attack avenues is thus to model the problem as an algebraic system of polynomial equations over a finite field. This approach was taken in [**FP06**], where the general Isomorphism of Polynomials (IP) problem was investigated. Here, we focus specifically on BMLE and perform a detailed complexity analysis.

First, fix arbitrary bases $(\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(k)})$ and $(\mathbf{D}^{(1)}, \ldots, \mathbf{D}^{(k)})$ of the codes $\mathcal{C}$ and $\mathcal{D}$ respectively. In terms of the bases, the MCE problem can be rephrased as finding $\mathbf{A} \in \mathrm{GL}_m(q)\mathbf{B} \in \mathrm{GL}_n(q)$ and $\mathbf{T} = (t_{ij}) \in \mathrm{GL}_k(q)$ such that:

$$\sum_{1 \leqslant s \leqslant k} t_{rs}\mathbf{D}^{(s)} = \mathbf{A}\mathbf{C}^{(r)}\mathbf{B}, \quad \forall r, 1 \leqslant r \leqslant k \tag{5.2}$$

The system (5.2) consists of knm equations in the $m^2 + n^2 + k^2$ unknown coefficients of the matrices $\mathbf{A}, \mathbf{B}$ and $\mathbf{T}$. The quadratic terms of the equations are always of the form $\gamma a_{ij} b_{i'j'}$ for some coefficients a_{ij} and $b_{i'j'}$ of $\mathbf{A}$ and $\mathbf{B}$ respectively which means the system (5.2) is bilinear. Note that the coefficients of $\mathbf{T}$ appear only linearly. As previously, we can guess the m^2 variables from $\mathbf{A}$, which will lead us to a linear system that can be easily solved. However, we can do better by exploiting the structure of the equations.

For ease of readability of the rest of the paragraph denote by $\mathbf{M}_{i\_}$ and $\mathbf{M}_{\_i}$ the i-th row and i-th column of a matrix $\mathbf{M}$. Note that, in (5.2), for $i \neq j$, the unknown coefficients from two rows $\mathbf{A}_{i\_}$ and $\mathbf{A}_{j\_}$ don't appear in the same equation. Symmetrically, the same holds for $\mathbf{B}_{\_i}$ and $\mathbf{B}_{\_j}$, but we will make

use of it for the matrix $\mathbf{A}$. Thus, we can consider only part of the system, and control the number of variables from $\mathbf{A}$. The goal is to reduce the number of variables that we need to guess before obtaining an overdetermined linear system, and we want to do this in an optimal way. Consider the first α rows from $\mathbf{A}$. Extracting the equations that correspond to these rows in (5.2) leads us to the system:

$$\sum_{1 \leqslant s \leqslant k} t_{rs} \mathbf{D}^{(s)}_{i\_} = \mathbf{A}_{i\_} \mathbf{C}^{(r)} \mathbf{B}, \quad \forall r, i, \ 1 \leqslant r \leqslant k, \ 1 \leqslant i \leqslant \alpha. \tag{5.3}$$

Guessing the αm coefficients from $\mathbf{A}_{i\_}$ $1 \leqslant i \leqslant \alpha$ leads to a linear system of αkn equations in $n^2 + k^2$ variables. Choosing $\alpha = \lceil \frac{n^2+k^2}{kn} \rceil$, the complexity of the approach becomes $\mathcal{O}(q^{m\lceil \frac{n^2+k^2}{kn} \rceil}(n^2+k^2)^3)$. For the usual choice of $m = n = k$, this reduces to at least $\alpha = 2$ and a complexity of $\mathcal{O}(q^{2n} n^6)$.

Note that, one can directly solve the bilinear system (5.3) using for example XL [111] and the analysis for bilinear systems from [242] (similar results can be obtained from [136]). We have verified, however, that due to the large number of variables compared to the available equations, the complexity greatly surpasses the one of the simple linearization attack presented above.

Improved modelling.

In order to improve upon this baseline algebraic attack, we will model the problem differently and completely avoid the t_{rs} variables. This modelling is in the spirit of the minors modellings of MinRank as in [30, 137].

As previously, let $\mathbf{G}$ and $\mathbf{G}'$ be the $k \times mn$ generator matrices of the equivalent codes $\mathcal{C}$ and $\mathcal{D}$ respectively. Then from Lemma 5.7, $\tilde{\mathbf{G}} = \mathbf{G}(\mathbf{A}^\top \otimes \mathbf{B})$ is a generator matrix of $\mathcal{D}$ for some invertible matrices $\mathbf{A}$ and $\mathbf{B}$. We will take the coefficients of $\mathbf{A}$ and $\mathbf{B}$ to be our unknowns. A crucial observation for this attack is that each row $\tilde{\mathbf{G}}_{i\_}$ of $\tilde{\mathbf{G}}$ is in the span of the rows of $\mathbf{G}'$, since $\mathbf{G}'$ and $\tilde{\mathbf{G}}$ define the same code. This means that adding $\tilde{\mathbf{G}}_{i\_}$ to $\mathbf{G}'$ does not change the code, i.e.,

$${}^{(i)}\mathbf{G}' = \begin{pmatrix} \mathbf{G}' \\ \tilde{\mathbf{G}}_{i\_} \end{pmatrix}$$

is not of full rank. From here, all maximal minors $\left| \left({}^{(i)}\mathbf{G}'_{\_j_1} \ {}^{(i)}\mathbf{G}'_{\_j_2} \dots {}^{(i)}\mathbf{G}'_{\_j_{k+1}} \right) \right|$ of ${}^{(i)}\mathbf{G}'$, for every $\{j_1, j_2, \dots, j_{k+1}\} \subset \{1, 2, \dots, mn\}$, are zero.

Now, as in a minors modeling of MinRank, we can form equations in the unknown coefficients of $\mathbf{A}$ and $\mathbf{B}$ by equating all maximal minors to zero, which

amounts to a total of $\binom{mn}{k+1}$ equations. Since the unknown coefficients of $\mathbf{A}$ and $\mathbf{B}$ appear only in the last row of the minors, and only bilinearly, the whole system is also bilinear. Thus we have reduced the problem to solving the bilinear system

$$\left\{ | \left({}^{(i)}\mathbf{G}'_{\_j_1} \ {}^{(i)}\mathbf{G}'_{\_j_2} \ldots {}^{(i)}\mathbf{G}'_{\_j_{k+1}} \right) | = 0, \quad \begin{array}{l} \text{for all } i \in \{1, 2, \ldots, k\} \text{ and all} \\ \{j_1, j_2, \ldots, j_{k+1}\} \subset \{1, 2, \ldots, mn\} \end{array} \right. \tag{5.4}$$

in the $m^2 + n^2$ unknown coefficients of $\mathbf{A}$ and $\mathbf{B}$.

At first sight, (5.4) seems to have more than enough equations to fully linearize the system. However, the majority of these equations are linearly dependent. In fact, there are only $(mn - k)k$ linearly independent equations. To see this, fix some i and consider a minor $| \left({}^{(i)}\mathbf{G}'_{\_j_1} \ {}^{(i)}\mathbf{G}'_{\_j_2} \ldots {}^{(i)}\mathbf{G}'_{\_j_{k+1}} \right) |$ of ${}^{(i)}\mathbf{G}'$. Since all rows except the first don't contain any variables, the equation

$$| \left({}^{(i)}\mathbf{G}'_{\_j_1} \ {}^{(i)}\mathbf{G}'_{\_j_2} \ldots {}^{(i)}\mathbf{G}'_{\_j_{k+1}} \right) | = 0$$

basically defines the linear dependence between the columns ${}^{(i)}\mathbf{G}'_{\_j_1}, \ldots {}^{(i)}\mathbf{G}'_{\_j_{k+1}}$. But the rank of the matrix is k, so all columns can be expressed through some set of k independent columns. Thus, in total, for a fixed i we have $mn - k$ independent equations and in total $(mn - k)k$ equations for all i.

Alternatively, we can obtain the same amount of equations from $\tilde{\mathbf{G}}$ and the generator matrix $\mathbf{G}'^{\perp}$ of the dual code of $\mathcal{D}$. Since $\tilde{\mathbf{G}}$ should also be a generator matrix of $\mathcal{D}$, we construct the system:

$$\mathbf{G}'^{\perp} \cdot \tilde{\mathbf{G}}^{\top} = \mathbf{0},$$

which is again a system of $(mn - k)k$ bilinear equations in $n^2 + m^2$ variables.

The complexity of solving the obtained system using either of the modellings strongly depends on the dimension of the code – it is the smallest for $k = mn/2$, and grows as k reduces (dually, as k grows towards mn). In Section 5.8 we give the concrete complexity estimate for solving the system for the chosen parameters using bilinear XL and the analysis from [242].

The attack does not seem to benefit a lot from being run on a quantum computer. Since the costly part comes from solving a huge linear system for which there are no useful quantum algorithms available, the only way is to 'Groverize' an enumeration part of the algorithm. One could enumerate over one set of the variables, either of $\mathbf{A}$ or $\mathbf{B}$, typically the smaller one, and solve a biliner system of less variables. Grover's algorithm could then speed up quadratically

this enumeration. However, since in the classical case the best approach is to not use enumeration, this approach only makes sense for quite small values of the field size i.e. only when $q < 4$. In this parameter regime, however, combinatorial attacks perform significantly better, so this approach becomes irrelevant.

5.7.3 Leon-like algorithm adapted to the rank metric

Leon [**Leon82**] proposed an algorithm against the code equivalence problem in the Hamming metric that relies on the basic property that isometries preserve the weight of the codewords and that the weight distribution of two equivalent codes is the same. Thus, finding the set of codewords of smallest weight in both codes reveals enough information to find a permutation that maps one set to the other, which with high probability is the unknown isometry between the codes. This algorithm is quite unbalanced and heavy on the 'codewords finding' side, since it requires finding all codewords of minimal weight. Beullens [**Beullens21**] proposed to relax the procedure and instead perform a collision based algorithm, much in the spirit of Algorithm 3: Build two lists of elements of the codes of particular weight (the distinguishing property from [**Beullens21**] actually also includes the multiset of entries of a codeword) and find a collision between them. As in Leon's algorithm and Algorithm 3, the 'collision finding' part employs an efficient subroutine for reconstructing the isometry.

The approach from the Hamming metric can be translated to matrix codes and can be used to solve MCE, but some adjustments are necessary. First of all note that finding codewords of a given rank r is equivalent to an instance of MinRank [112, 137] for k matrices of size $m \times n$ over $\mathbb{F}_q$. Depending on the parameters, we have noticed that the Kipnis-Shamir modelling [**KS99**] and Bardet's modelling [30] perform the best, so we use both in our complexity estimates.

For the collision part, notice that given two codewords $\mathbf{C}_1$ from $\mathcal{C}$ and $\mathbf{D}_1$ from $\mathcal{D}$, it is not possible to determine the isometry $(\mathbf{A}, \mathbf{B})$, as there are many isometries possible between single codewords. Thus, there is no efficient way of checking that these codewords collide nor finding the correct isometry. On the other hand, a pair of codewords is typically enough. For the pairs $(\mathbf{C}_1, \mathbf{C}_2)$ and $(\mathbf{D}_1, \mathbf{D}_2)$ we can form the system of $2mn$ linear equations

$$\begin{cases} \mathbf{A}^{-1}\mathbf{D}_1 = \mathbf{C}_1\mathbf{B} \\ \mathbf{A}^{-1}\mathbf{D}_2 = \mathbf{C}_2\mathbf{B} \end{cases} \tag{5.5}$$

in the $m^2 + n^2$ unknown coefficients of $\mathbf{A}$ and $\mathbf{B}$. When $m = n$, which is a typical choice, the system is expected to be overdetermined, and thus solved

in $\mathcal{O}(n^6)$. In practice, and since $\mathbf{C}_1$, $\mathbf{C}_2$, $\mathbf{D}_1$ and $\mathbf{D}_2$ are low-rank codewords, there are fewer than $2n^2$ linearly independent equations, so instead of a unique solution, we can obtain a basis of the solution space. However, the dimension of the solution space is small enough so that coupling this technique with one of the algebraic modelings in Section 5.7.2 results in a system that can be solved through direct linearization. It is then easy to check whether the obtained isometry maps $\mathcal{C}$ to $\mathcal{D}$. We will thus assume, as a lower bound, that we find collisions between pairs of codewords.

Now, let $C(r)$ denote the number of codewords of rank r in a k-dimensional $m \times n$ matrix code. Then, using a birthday argument, two lists of size $\sqrt{2C(r)}$ of rank r codewords of $\mathcal{C}$ and $\mathcal{D}$ are enough to find two collisions. To detect the two collisions, we need to generate and solve systems as in Equation (5.5) for all possible pairs of elements from the respective lists, so $\binom{\sqrt{2C(r)}}{2}^2$ systems in total. Since $C(r) \approx q^{r(n+m-r)-nm+k}$, the total complexity amounts to

$$\mathcal{O}(q^{2(r(n+m-r)-nm+k)}(m^2+n^2)^\omega).$$

Note that a deterministic variant of this approach has the same asymptotic complexity. Choosing two rank r codewords of $\mathcal{C}$ and checking them for a 2-collision against all pairs of rank r codewords of $\mathcal{D}$ requires solving $\binom{C(r)}{2}$ systems.

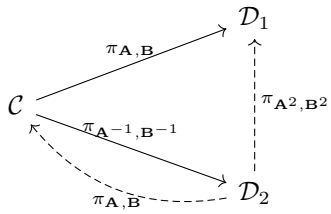
Finally, we choose r so that both parts – the MinRank and the collision part are as close to a balance as possible. Section 5.8 discusses further the complexity of this approach for the chosen parameters of our scheme.

When considering the quantum version of the algorithm, we apply the same reasoning as in the case of the collision based Algorithm 3, and obtain quadratic speedup in the collision part. Because hybridization is also possible for the MinRank part, it can also benefit from using Grover, especially for larger fields.

5.7.4 Attacks on IMCE

The security of the linkable version of the ring signature scheme relies on the hardness of the IMCE problem. Thus, given three codes $\mathcal{C}$, $\mathcal{D}_1$ and $\mathcal{D}_2$, we need to determine whether they form a triangle of the following form, where the full lines represent the exact mappings the problem asks for, whereas the dashed ones are mappings that if found also break the problem. (The diagram uses

loose notation in favor of readability.)



Extended collision-search algorithm.

Our extended collision-search attack consists of finding an isometry between *any* of the three codes. Specifically, finding a collision between $\mathcal{C}$ and $\mathcal{D}_1$ allows us to derive $\pi_{\mathbf{A},\mathbf{B}}$, one between $\mathcal{C}$ and $\mathcal{D}_2$ yields $\pi_{\mathbf{A}^{-1},\mathbf{B}^{-1}}$, and one between $\mathcal{D}_1$ and $\mathcal{D}_2$ yields $\pi_{\mathbf{A}^2,\mathbf{B}^2}$. We first reduce the problem to an equivalent QMLE-based problem: solve any of the three QMLE instances $(k, m+n, \mathcal{F}, \mathcal{P}_1)$, $(k, m+n, \mathcal{F}, \mathcal{P}_2)$ or $(k, m+n, \mathcal{P}_2, \mathcal{P}_1)$, with

$$\mathcal{P}_1(\mathbf{x}) = (\mathcal{F}(\mathbf{xS}))\mathbf{T_1},\ \mathcal{P}_2(\mathbf{x}) = (\mathcal{F}(\mathbf{xS}^{-1}))\mathbf{T_2},\ \mathcal{P}_1(\mathbf{x}) = (\mathcal{P}_2(\mathbf{xS}^2))\mathbf{T_2}^{-1}\mathbf{T_1}$$

and $\mathbf{S} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}^\top \end{bmatrix}$. Then, we apply a modified version of Algorithm 3, where we now build three lists instead of two, and on Line 11, we pick pairs $(\mathbf{x}, \mathbf{y})$ from $\{L_1 \times L_2\} \cup \{L_1 \times L_3\} \cup \{L_2 \times L_3\}$, instead of just $\{L_1 \times L_2\}$.

To derive the complexity of the attack, denote by $N = dq^{m+n}$ the total number of distinguished elements. Then it can be deduced, using a birthday based analysis that the probability of not having a collision when the lists contain ℓ elements each is $P = \prod_{i=1}^{\ell}(1 - \frac{i}{N})(1 - \frac{2i}{N}) \approx e^{-\frac{3\ell^2}{2N}}$. To have a 63% chance of collision, we need to build lists of size $\ell = c_2\sqrt{N}$, with $c_2 = \sqrt{\frac{2}{3}}$. For comparison, performing a similar analysis for the collision-search algorithm for the general MCE problem (where we have only two lists), this constant is $c_1 = \sqrt{2}$. constant is, however, neglected in the complexity analysis because the choice of having 63% chance of success is arbitrary, and an attack with a lower success probability still presents a security threat.

Remark 5.4. For Leon's algorithm for IMCE, the arguments are similar to the extended collision search algorithm. Thus it seems that apart from possibly a constant speed-up, the algorithm does not benefit from the IMCE context.

Extended algebraic attack.

Algebraically, IMCE can be treated in exactly the same manner as the minors modeling of MCE in the previous section. The types of equations are the same, and with the same structure. The only difference is that the algebraic system that we construct has twice as many equations, i.e $2(mn-k)k$, coming from the isometry between $\mathcal{C}$ and D_1 and between $\mathcal{D}_2$ and $\mathcal{C}$ but the amount of variables is the same – n^2+m^2, since the isometry is the same. This means that the cost of the attack is reduced for the same choice of parameters, so adding the linkability property requires larger parameters in the regime where the algebraic attack performs the best.

5.8 Implementation and Evaluation

In this section, we give an assessment of the performance of MEDS. We begin with important considerations about the implementation of the signature scheme. Then we provide concrete parameter choices for MEDS and a first preliminary evaluation of its performance based on a C reference implementation. The source code of our implementation is available at

https://github.com/MEDSpqc/meds.

5.8.1 Implementation

Besides performance, the security of a cryptographic implementation is of crucial importance. By "implementation security" here we mean the resilience of an implementation against threats such as timing attacks, physical side-channel attacks, and fault injection attacks. While the requirement for side-channel and fault attacks heavily depends on whether in practice physical access is possible for an attacker, it is widely considered best practice to provide protection against timing attacks as baseline for all cryptographic implementations. In order to do this, the implementation must be *constant time*, i.e., timing variations based on secret data must be prevented. This typically means that branching and memory access based on secret data must be avoided. There are generic constructions to achieve timing security for basically any given cryptographic scheme. However, if the design of a cryptographic primitive does not take constant-time requirements into consideration, such generic constructions can be computationally expensive. Therefore, defining a cryptographic scheme such that it supports an efficient protection against timing attacks can make it easier

to implement the scheme securely which in turn can make a scheme more secure and efficient in practice.

In the case of MEDS, we need to consider the implementation security of key generation and signing. Verification only operates on public data and hence does not require further protection. The basic operations of MEDS during key generation and signing are:

- field arithmetic,
- matrix multiplication,
- generating random invertible matrices, and
- computing a canonical form of a matrix.

All these operations must be implemented securely.

Field arithmetic.

We are using a relatively small prime field in MEDS. Hence, for finite field addition and multiplication, we can simply perform integer arithmetic followed by a reduction modulo the prime. On most architectures, constant-time operations for adding and multiplying small operands are available. The reduction modulo the prime can be implemented using standard approaches in literature (e.g., by Granlund and Montgomery [162], Barrett [36], or Montgomery [219]). Inversion of field elements can efficiently be implemented in constant time using Fermat's little theorem and optimal addition chains.

In the C reference implementation, we simply use the modulo operation for reduction and Fermat's little theorem for inversion to get a first impression on the performance of the scheme. For using MEDS in practice, the timing side-channel security in particular of the modulo operation needs to be verified.

Matrix multiplication.

The basic schoolbook algorithm for multiplying matrices is not data dependent and hence constant time. More sophisticated approaches like the method by Arlazarov, Dinic, Kronrod, and Faradžev [15] may depend on potentially secret data, but most likely do not significantly improve the performance for the finite field and the matrix sizes used in MEDS. Asymptotically more efficient matrix multiplication algorithms likely are not more efficient for the given matrix dimensions neither. Hence, for the C reference implementation, we are simply

using schoolbook multiplication. If a more efficient algorithm is used for optimized implementations, it must be ensured that a constant time algorithm is used.

Random invertible matrix generation.

On several occasions throughout the MEDS operations, we need to generate random invertible matrices: For the partially seeded key generation (cf. Section 5.6) we need to compute an invertible matrix $\mathbf{T} \in \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$ and for signing (cf. step i.. in Figure 5.5) we need to generate potentially secret matrices $\tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i \in \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$. (During verification we need to recompute $\tilde{\mathbf{A}}_i$ and $\tilde{\mathbf{B}}_i$ for cases where $h_i = 0$.)

There are several approaches for generating random invertible matrices:

1. *Trial-and-error approach:* Generate a random matrix $\mathbf{M}$ and attempt to compute its inverse. If $\mathbf{M}$ does not have an inverse, try again with a new random matrix. This approach requires constant-time Gaussian elimination if $\mathbf{M}$ needs to be kept secret and it might require several attempts before a random invertible matrix has been found.

2. *Constructive approach:* Construct a matrix $\mathbf{M}$ that is known to be inveritble. One approach for this is described in [246]: Generate a random lower-left triangular matrix $\mathbf{L}$ with the diagonal all 1 and an upper-right triangular matrix $\mathbf{U}$ with the diagonal all $\neq 0$ as well as a corresponding permutation matrix $\mathbf{P}$ akin to the result of an LUP decomposition. Then compute the random invertible matrix $\mathbf{M}$ as $\mathbf{M} = \mathbf{P}^{-1}\mathbf{L}\mathbf{U}$.

   Since generation of and multiplication with $\mathbf{P}$ are expensive to implement in constant time, one can follow the approach of [**Tang22**], leave out $\mathbf{P}$, and compute $\mathbf{M}$ as $\mathbf{M} = \mathbf{L}\mathbf{U}$ directly. This, however, covers only a subset of $((q-1)/q)^n$ matrices of all invertibe matrices in $\mathbb{F}_q^{n \times n}$. The inverse $\mathbf{M}^{-1}$ of the matrix can then be computed as $\mathbf{M}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}$.

The fastest approach in our experiments was the constructive approach following [**Tang22**]. Hence, we are using the constructive approach in all cases.

Canonical matrix form.

MEDS requires to compute a canonical form of matrices before hashing. However, during signing, this must be computed in constant time. Computing the reduced row-echelon form of a matrix in constant time is expensive. Canonical

matrix forms that can be computed in constant time more efficiently include the *systematic form* and the *semi-systematic* form of a matrix, used, e.g., in Classic McEliece [8]. Both the systematic and the semi-systematic form are special cases of the reduced row-echelon form.

For the systematic form, all pivoting elements are required to reside on the left diagonal of the matrix, i.e., a matrix $\mathbf{G} \in \mathbb{F}_q^{k \times mn}$ in systematic form has the shape $(\mathbf{I}_k | \mathbf{G}')$ with $\mathbf{G}' \in \mathbb{F}_q^{k \times (mn-k)}$ and $\mathbf{I}_k$ denoting the $k \times k$ identity matrix. The requirements for the semi-systematic form are more relaxed: Following [8, Sect. 2.2.1], we say that a matrix $\mathbf{G}$ is in (μ, ν)-semi-systematic form if $\mathbf{G}$ has r rows (i.e., no zero rows), the pivot element in row $i \leq r - \mu$ also is in column i and the pivot element in row $i > r - \mu$ is in a column $c \leq i - \mu + \nu$.

However, not all matrices admit a systematic or semi-systematic form. In this case, we need to restart the computation with new random data. The probability that a matrix $\mathbf{G} \in \mathbb{F}_q^{k \times mn}$, $k \leq mn$ is full rank is $\prod_{i=1}^{k}(q^{mn} - q^{i-1})/q^{kmn}$. Therefore, the probability that $\mathbf{G}$ has a systematic form is $\prod_{i=1}^{k}(q^k - q^{i-1})/q^{k^2}$ and the probability that it has a semi-systematic form is $\prod_{i=1}^{k-\mu}(q^k - q^{i-1})/q^{(k-\mu)k}$. $\prod_{i=1}^{\mu}(q^\nu - q^{i-1})/q^{\mu\nu}$. The probability and the cost of constant-time implementation for the semi-systematic form depend on μ and ν.

This gives us the following three options for avoiding to compute a reduced row-echelon form in constant time for $\tilde{\mathbf{G}}_i$ during signing:

1. *Basis change:* After computing $\tilde{\mathbf{G}}'_i = \pi_{\tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i}(\mathbf{G}_0)$, perform a basis change $\tilde{\mathbf{G}}''_i = \mathbf{M}_i \tilde{\mathbf{G}}'_i$ with a secret random invertible matrix $\mathbf{M}_i \in \mathbb{F}_q^{k \times k}$. Then compute a canonical form $\tilde{\mathbf{G}}_i = S^{\mathbf{0}}(\tilde{\mathbf{G}}''_i)$ on a public $\tilde{\mathbf{G}}''_i$ without the need for a constant time implementation. This removes the requirement of a constant time computation of the canonical form but introduces extra cost for the generation of and multiplication with a random invertible matrix $\mathbf{M}_i$. Instead of an invertible matrix $\mathbf{M}_i$, just a random matrix can be used. With low probability (see above), a random matrix does not have full rank and the computation of the canonical form fails. In that case, the process needs to be restarted with a different $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$.

2. *Semi-systematic form:* Compute the semi-systematic form. This requires a less expensive constant-time implementation than for the reduced-row-echelon form. However, computing the canonical form might fail if no semi-systematic form exists, in which case the computation needs to be restarted.

3. *Systematic form:* Compute the systematic form. This can be imple-

$\lceil \log_2 q \rceil$	$n = k$	Birthday	Algebraic	Leon	SIG
9	16	235.29	181.55	131.20	13 296
9	17	249.04	194.55	149.65	16 237
10	15	244.62	174.75	130.50	12 428
11	14	250.79	160.24	131.21	12 519
12	14	272.40	160.24	141.17	13 548
13	**13**	**274.10**	**146.76**	**130.41**	**11 586**
14	13	294.10	146.76	134.41	13 632
20	12	383.75	138.46	135.40	16 320

Table 5.1: Cost of the investigated attacks in log scale, and 'SIG' for 'signature size in bytes. Preferred choice in bold.

mented even more easily and cheaper in constant time than computing the semi-systematic form. However, systemization failure is more frequent and it is more likely that computations need to be restarted.

We performed generic experiments to investigate the performance impact of these variants. Our experiments indicate that the implementation cost for variant 1 is higher than that of the other two. For the specific parameter sets we propose in Section 5.8.2, the probability that $\tilde{\mathbf{G}}_i$ does not have a systematic form is about 0.0015%. Therefore, even though the failure probability can be reduced by computing the semi-systematic form compared to the systematic form with well-chosen μ and ν, the overall overhead (including cost for constant-time implementation) of computing the semi-systematic form (variant 2) is likely higher than the overall overhead for of computing the systematic form (variant 3). Hence, we decided to use the systematic form throughout as canonical form.

5.8.2 Parameter choice and evaluation

A summary of the cost of the three different attacks described in Section 5.7 is given in Table 5.1. First, we decide to set $n = k$, as this seems to be the Goldilocks zone for our scheme. For k larger, the algebraic attack becomes significantly better, and the same is true for Leon's attack when k is smaller. Then, for finite fields of different sizes, we find the smallest value of n that achieves the required security level of 128 bits. We see that Leon's algorithm performs the best in most cases, although the algebraic approach is almost as

good. Finally, to determine the optimal value for q, we choose the optimization parameters (s, t, and w) such that the sizes of the public key and the signature are comparable, and we report the signature size in the last column of Table 5.1. We conclude that the sweet spot for 128-bit security is given for the 13-bit prime $q = 8191$ and $n = k = 13$.

Remark 5.5. Given these parameters, we heuristically assume that the automorphism group of the codes is trivial with overwhelming probability. It is computationally infeasible to compute the automorphism group of codes of this size; however, data on smaller-sized codes shows that the probability of a random code having a trivial automorphism group grows rapidly as q, n, and m increase.

In this setting, we can vary s, t, and w for different trade-offs of public key and signature sizes as well as performance. We also checked the impact of q if we aim for small public keys or small signatures (instead if balancing these two as in Table 5.1). In such cases, both 11-bit and 13-bit primes for q seem to perform similarly well. Hence, we stick to the 13-bit prime $q = 8191$ in our discussion.

Table 5.2 provides an overview of 128-bit security parameters for MEDS, highlighting different performance and key/signature size trade-offs. The best attack for all parameter set based on $q = 8191$, $n = 13$, and $k = 13$ is the Leon-like attack as shown in Table 5.1 with an expected cost of slightly over 2^{130} operations. The best quantum attack is obtained by Groverizing Leon's algorithm and has a cost of around 2^{88} operations. We select s, t, and w such that the probability of an attack on the Fiat-Shamir construction is around 2^{-128}. To improve the efficiency of vectorized implementations using SIMD instructions in the future, we select t as multiple of 16. In general, we are using all optimizations discussed in Section 5.6. However, we provide one parameter set without using the seed tree (without '-st' in the name of the parameter set).

Table 5.3 shows the resulting performance of these parameter sets from our constant-time C reference implementation on an AMD Ryzen 7 PRO 5850U CPU. The C reference implementation follows the implementation discussion above but does not apply any further algorithmic or platform-specific optimizations. We expect that optimized and vectorized implementations can significantly increase the performance.

TODO: FIX ISSUE WITH SI UNITS

[4] https://bench.cr.yp.to/supercop.html

The parameter set MEDS-2826-st with $s = 2$ provides the smallest public key of about 2.8TODO FIXTODO FIX and a signature of about 18TODO FIXTODO FIX. MEDS-8445-st increases the public key size with $s = 4$ to slightly over 8TODO FIXTODO FIX while reducing the signature size to about 10.5TODO FIXTODO FIX. MEDS-8445-st-f is a 'fast' variant of this parameter set with a smaller $t = 160$ but a larger $w = 23$, resulting in a larger signature size of about 14TODO FIXTODO FIX. MEDS-8445-st-s is 'small' and goes the opposite direction, providing a smaller signature size of about 8.5TODO FIXTODO FIX due to a smaller $w = 13$ at a larger computational cost due to $t = 1760$. These three sibling parameter sets illustrate the impact of t and w on performance and signature size.

MEDS-11255-st provides balanced public key and signature sizes, with both around 11TODO FIXTODO FIX, and a small sum of signature and public key size at moderate computational cost for signing and verification due to $t = 224$. Removing the seed tree optimization comes with an increase in signature size of about 2TODO FIXTODO FIX, which illustrates the impact of the seed tree.

Finally, sets MEDS-42161-st, MEDS-356839-st, and MEDS-716471-st push the public key size to an extreme at the expense of key generation time in the pursue of reducing signature size and computational cost for signing and verification. However, we expect that at least the key generation time can significantly be improved by optimizing the computation of solving the medium-size sparse linear system used for partially seeding the public key.

Overall, Table 5.2 and Table 5.3 highlight the large degree of flexibility offered by the MEDS scheme. All parameter sets are competitive with respect to existing literature, such as the LESS-FM scheme.

Finally, we discuss performance for the ring signature scheme. With the MEDS-2696-st parameter set, the size of a signature is in fact given by approximately $(\log_2 r + 19.26)$ kB; in the linkable case, this increases to $(\log_2 r + 39.22)$ kB. These numbers are close to the lattice instantiation (Falafl) given in [51]; judging by the timings in Table 5.3, we also expect it to be much faster than the isogeny instantiation (Calamari). On the other hand, the work of [32] obtains smaller sizes, but does not propose a linkable variant. Note that both sizes and timings can be improved by choosing dedicated parameters and designing an ad-hoc implementation, which we leave as part of a future work.

5.8.3 Comparison to related signature schemes

Table 5.4 shows a comparison of public key and signature sizes as well as computational performance of our new MEDS scheme with some established schemes

and related recent proposals. While the comparison of public key and signature sizes is accurate, the comparison of the performance needs to be taken with a large grain of salt: While we provide numbers in the same performance metric (mega cycles – mcyc.), a direct comparison is still quite hard since not all schemes have received the same degree of optimization and since not all performance data has been obtained on the same CPU architecture.

The performance data from the 'classical' scheme ed25519 as well as from the NIST PQC schemes CRYSTALS-Dilithium [133], Falcon [146], and SPHINCS+[170] has been obtained from the SUPERCOP website[5]. We selected the performance data from the AMD64 Zen CPU, which is an AMD Ryzen 7 1700 from 2017, i.e., the same microarchitecture (but a different CPU) as we used for our measurements of MEDS. We are reporting median cycles directly from the website.

For UOV [50], LESS [33] and Wavelet [23] we list the performance data as reported in the respective papers unless such data was unavailable. In the case of SDitH [5], only reports of performance data in milliseconds on a 3.1 GHz Intel Core i9-9990K are available. We computed the corresponding number of cycles from this to enable a rough comparison to the other schemes, but note that this data is therefore not entirely accurate.

Table 5.4 shows that, although code-based schemes do not compete well with pre-quantum or lattice-based PQC schemes, MEDS fills a gap that was not previously available for multivariate or code-based schemes, with a relatively small combined size of public key and signature. Furthermore, its versatility in parameter selection allows for great flexibility for specific applications. In terms of performance, the current implementation of MEDS is still unoptimized. We expect speed-ups of at least one order of magnitude from SIMD parallelization on AVX256 and AVX512 CPUs, since both the data-independent loop of the Fiat-Shamir construction and the matrix arithmetic lend themselves to efficient parallelization. Providing optimized implementations of MEDS for modern SIMD architectures as well as embedded systems is an open task for future work.

[5] https://bench.cr.yp.to/results-sign.html – amd64; Zen (800f11); 2017 AMD Ryzen 7 1700; 8 x 3000MHz; rumba7, supercop-20220506

Public Data

$q, m, n, k, t = \lambda \in \mathbb{N}$.
$\mathsf{H} : \{0,1\}^* \to \{0,1\}^t$.

II. Sign(sk)

1. For all $i = 0, \ldots, t-1$:
   i. $\tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i \xleftarrow{\$} \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$.
   ii. Compute $\tilde{\mathbf{G}}_i = \mathsf{SF}(\pi_{\tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i}(\mathbf{G}_0))$.
2. Compute $h = \mathsf{H}(\tilde{\mathbf{G}}_0, \ldots, \tilde{\mathbf{G}}_{t-1}, \mathsf{msg})$.
3. Parse $h = [h_0| \ldots |h_{t-1}]$, $h_i \in \{0,1\}$.
4. For all $i = 0, \ldots, t-1$:
   i. Set $(\mu_i, \nu_i) = (\tilde{\mathbf{A}}_i \mathbf{A}_{h_i}^{-1}, \mathbf{B}_{h_i}^{-1} \tilde{\mathbf{B}}_i)$.
5. Set $\sigma = (h, \mu_0, \ldots, \mu_{t-1}, \nu_0, \ldots, \nu_{t-1})$.
6. Send σ to verifier.

I. Keygen()

1. $\mathbf{G}_0 \xleftarrow{\$} \mathbb{F}_q^{k \times mn}$ in standard form
2. Set $\mathbf{A}_0 = \mathbf{I}_m$, $\mathbf{B}_0 = \mathbf{I}_n$.
3. $\mathbf{A}_1, \mathbf{B}_1 \xleftarrow{\$} \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$.
4. Compute $\mathbf{G}_1 = \mathsf{SF}(\pi_{\mathbf{A}_1, \mathbf{B}_1}(\mathbf{G}_0))$.
5. Set $\mathsf{sk} = (\mathbf{A}_1, \mathbf{B}_1)$ and $\mathsf{pk} = (\mathbf{G}_0, \mathbf{G}_1)$.

III. Verify($\mathsf{pk}, \mathsf{msg}, \sigma$)

1. Parse $h = [h_0| \ldots |h_{t-1}]$, $h_i \in \{0,1\}$.
2. For all $i = 0, \ldots, t-1$:
   i. Set $\hat{\mathbf{G}}_i = \mathsf{SF}(\pi_{\mu_i, \nu_i}(\mathbf{G}_{h_i}))$.
3. Compute $h' = \mathsf{H}(\hat{\mathbf{G}}_0, \ldots, \hat{\mathbf{G}}_{t-1}, \mathsf{msg})$.
4. Accept if $h' = h$ or reject otherwise.

Figure 5.2: The basic signature scheme

Public Data

$q, m, n, k, \lambda, r \in \mathbb{N}$.
$\mathsf{Com} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{2\lambda}$.

II. Commit(pk)

1. $\tilde{\mathbf{A}}, \tilde{\mathbf{B}} \xleftarrow{\$} \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$.
2. For all $j = 1, \ldots, r$:
   i. Sample $\mathsf{r}_j \xleftarrow{\$} \{0,1\}^\lambda$.
   ii. Compute $\tilde{\mathbf{G}}_j = \mathsf{SF}(\pi_{\tilde{\mathbf{A}},\tilde{\mathbf{B}}}(\mathbf{G}_j))$.
   iii. Compute $h_j = \mathsf{Com}(\mathsf{r}_j, \tilde{\mathbf{G}}_j)$.
3. Build IHMT from $(h_1, \ldots, h_r)$ and get root.
4. Set $\mathsf{cmt} = \mathsf{root}$.
5. Send cmt to verifier.

IV. Response($\mathsf{sk}_{j^*}, \mathsf{pk}, \mathsf{cmt}, \mathsf{chal}$)

1. If $\mathsf{chal} = 0$:
   i. Set $(\mu, \nu) = (\tilde{\mathbf{A}}\mathbf{A}_{j^*}, \mathbf{B}_{j^*}\tilde{\mathbf{B}})$.
   ii. Get path corresponding to leaf j^* in IHMT for $(h_1, \ldots, h_r)$.
   iii. Set $\mathsf{bits} = \mathsf{r}_{j^*}$.
   iv. Set $\mathsf{resp} = (\mu, \nu, \mathsf{path}, \mathsf{bits})$.
2. If $\mathsf{chal} = 1$:
   i. Set $(\mu, \nu) = (\tilde{\mathbf{A}}, \tilde{\mathbf{B}})$.
   ii. Set $\mathsf{bits} = (\mathsf{r}_1, \ldots, \mathsf{r}_r)$.
   iii. Set $\mathsf{resp} = (\mu, \nu, \mathsf{bits})$.
3. Send resp to verifier.

I. Keygen(j)

1. $\mathbf{G}_0 \xleftarrow{\$} \mathbb{F}_q^{k \times mn}$ in standard form.
2. $(\mathbf{A}_j, \mathbf{B}_j) \xleftarrow{\$} \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$.
3. Compute $\mathbf{G}_j = \mathsf{SF}(\pi_{\mathbf{A}_j,\mathbf{B}_j}(\mathbf{G}_0))$.
4. Set $\mathsf{sk}_j = (\mathbf{A}_j, \mathbf{B}_j)$ and $\mathsf{pk}_j = \mathbf{G}_j$.
5. Set $\mathsf{pk} = (\mathbf{G}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_r)$

=**III. Challenge**()

1. $c \xleftarrow{\$} \{0,1\}$.
2. Set $\mathsf{chal} = c$.
3. Send chal to prover.

V. Verify($\mathsf{pk}, \mathsf{cmt}, \mathsf{chal}, \mathsf{resp}$)

1. If $\mathsf{chal} = 0$:
   i. Compute $\hat{\mathbf{G}} = \mathsf{SF}(\pi_{\mu,\nu}(\mathbf{G}_0))$.
   ii. Compute $h' = \mathsf{Com}(\mathsf{bits}, \hat{\mathbf{G}})$.
   iii. Get root' from IHMT using h' and path.
2. If $\mathsf{chal} = 1$:
   i. For all $j = 1, \ldots, r$:
      a. Compute $\hat{\mathbf{G}}_j = \mathsf{SF}(\pi_{\mu,\nu}(\mathbf{G}_j))$.
      b. Compute $h'_j = \mathsf{Com}(\mathsf{r}_j, \hat{\mathbf{G}}_j)$.
   ii. Build IHMT from $(h'_1, \ldots, h'_r)$ and get root'.
3. Accept if $\mathsf{root}' = \mathsf{cmt}$, reject otherwise.

Figure 5.3: MCE Ring Sigma Protocol

Public Data
$q, m, n, k, t = \lambda, r \in \mathbb{N}$. $\mathsf{Com} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{2\lambda}$. $\mathsf{H}, \overline{\mathsf{H}} : \{0,1\}^* \to \{0,1\}^t$.

I. Keygen(j)

1. $\mathbf{G}_0 \xleftarrow{\$} \mathbb{F}_q^{k \times mn}$ in standard form.
2. $(\mathbf{A}_j, \mathbf{B}_j) \xleftarrow{\$} \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$.
3. Compute $\mathbf{G}_j = \mathsf{SF}(\pi_{\mathbf{A}_j, \mathbf{B}_j}(\mathbf{G}_0))$.
4. Set $\mathsf{sk}_j = (\mathbf{A}_j, \mathbf{B}_j)$ and $\mathsf{pk}_j = \mathbf{G}_j$.
5. Set $\mathsf{pk} = (\mathbf{G}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_r)$

II. Sign$(\mathsf{sk}_{j^*}, \mathsf{pk})$

1. Compute the tag $\mathbf{T} = \mathsf{SF}(\pi_{\mathbf{A}_{j^*}^{-1}, \mathbf{B}_{j^*}^{-1}}(\mathbf{G}_0))$.
2. For all $i = 0, \ldots, t-1$:
   i. $\tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i \xleftarrow{\$} \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$.
   ii. Set $\tilde{\mathbf{T}}_i = \mathsf{SF}(\pi_{\tilde{\mathbf{A}}_i^{-1}, \tilde{\mathbf{B}}_i^{-1}}(\mathbf{T}))$.
   iii. For all $j = 1, \ldots, r$:
      a. Sample $\mathsf{r}_{i,j} \xleftarrow{\$} \{0,1\}^\lambda$.
      b. Set $\tilde{\mathbf{G}}_{i,j} = \pi_{\tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i}(\mathbf{G}_j)$.
      c. Set $h_{i,j} = \mathsf{Com}(\mathsf{r}_{i,j}, \mathsf{SF}(\tilde{\mathbf{G}}_{i,j}))$.
   iv. Build IHMT from $(h_{i,1}, \ldots, h_{i,r})$ and get root_i.
   v. Compute $\bar{h}_i = \overline{\mathsf{H}}(\tilde{\mathbf{T}}_i, \mathsf{root}_i)$.
3. Conpute $h = \mathsf{H}(\bar{h}_0, \ldots, \bar{h}_{t-1}, \mathbf{T}, \mathsf{msg})$.
4. Parse $h = [h_0 | \ldots | h_{t-1}]$, $h_i \in \{0,1\}$.
5. For all $i = 0, \ldots, t-1$:
   i. If $h_i = 0$:
      a. Set $(\mu_i, \nu_i) = (\tilde{\mathbf{A}}_i \mathbf{A}_{j^*}, \mathbf{B}_{j^*} \tilde{\mathbf{B}}_i)$.
      b. Set $(\bar{\mu}_i, \bar{\nu}_i) = (\mathbf{A}_{j^*} \tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i \mathbf{B}_{j^*})$.
      c. Get path_i corresponding to leaf j^* in IHMT for $(h_{i,1}, \ldots, h_{i,r})$.
      d. Set $\mathsf{bits}_i = \mathsf{r}_{i,j^*}$.
      e. Set $\mathsf{rsp}_i = \{\mu_i, \nu_i, \bar{\mu}_i, \bar{\nu}_i, \mathsf{path}_i, \mathsf{bits}_i\}$.
   ii. If $h_i = 1$:
      a. Set $(\mu_i, \nu_i) = (\tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i)$.
      b. Set $\mathsf{bits}_i = (\mathsf{r}_{i,1}, \ldots, \mathsf{r}_{i,r})$.
      c. Set $\mathsf{rsp}_i = \{\mu_i, \nu_i, \mathsf{bits}_i\}$.
6. Set $\sigma = (h, \mathsf{resp}_0, \ldots, \mathsf{resp}_{t-1}, \mathbf{T})$.
7. Send σ to verifier.

III. Verify$(\mathsf{pk}, \mathsf{msg}, \sigma)$

1. Parse $h = [h_0 | \ldots | h_{t-1}]$, $h_i \in \{0,1\}$.
2. For all $i = 0, \ldots, t-1$:
   i. If $h_i = 0$:
      a. Compute $\hat{\mathbf{G}}_i = \mathsf{SF}(\pi_{\mu_i, \nu_i}(\mathbf{G}_0))$.
      b. Compute $h'_i = \mathsf{Com}(\mathsf{bits}_i, \hat{\mathbf{G}}_i)$.
      c. Get root_i from IHMT using h'_i and path_i.
      d. Compute $\hat{\mathbf{T}}_i = \mathsf{SF}(\pi_{\bar{\mu}_i^{-1}, \bar{\nu}_i^{-1}}(\mathbf{G}_0))$.
   ii. If $h_i = 1$:
      a. For all $j = 1, \ldots, r$:
         † Compute $\hat{\mathbf{G}}_{i,j} = \mathsf{SF}(\pi_{\mu_i, \nu_i}(\mathbf{G}_j))$.
         ‡ Compute $h'_{i,j} = \mathsf{Com}(\mathsf{r}_{i,j}, \hat{\mathbf{G}}_{i,j})$.
      b. Build IHMT from $(h'_{i,1}, \ldots, h'_{i,r})$ and get root'_i.
      c. Compute $\hat{\mathbf{T}}_i = \mathsf{SF}(\pi_{\mu_i^{-1}, \nu_i^{-1}}(\mathbf{T}))$.
   iii. Compute $\bar{h}'_i = \overline{\mathsf{H}}(\hat{\mathbf{T}}_i, \mathsf{root}'_i)$.
3. Compute $h' = \mathsf{H}(\bar{h}'_0, \ldots, \bar{h}'_{t-1}, \mathbf{T}, \mathsf{msg})$.
4. Accept if $h' = h$ or reject otherwise.

Figure 5.4: MCE linkable ring signature scheme

Public Data

$q, m, n, k, \lambda, t, s, w \in \mathbb{N}$.
$\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$.

I. Keygen()

1. $\mathbf{G}_0 \xleftarrow{\$} \mathbb{F}_q^{k \times mn}$ in standard form.
2. Set $\mathbf{A}_0 = \mathbf{I}_m$, $\mathbf{B}_0 = \mathbf{I}_n$.
3. $\mathbf{T} \xleftarrow{\$} \mathrm{GL}_k(q)$
4. Compute $\mathbf{G}'_0 = \mathbf{T}\mathbf{G}_0$
5. Parse the first two rows of $\mathbf{G}'_0$ into $\mathbf{P}_1^0, \mathbf{P}_2^0 \in \mathbb{F}_q^{m \times n}$
6. For all $j = 1, \ldots, s-1$:
   i. $\mathbf{P}_1^j, \mathbf{P}_2^j \xleftarrow{\$} \mathbb{F}_q^{m \times n}$
   ii. Find $\mathbf{A}_j$ and $\mathbf{B}_j$ from:
   $$\mathbf{P}_1^j \mathbf{B}_j^{-1} = \mathbf{A}_j \mathbf{P}_1^0$$
   $$\mathbf{P}_2^j \mathbf{B}_j^{-1} = \mathbf{A}_j \mathbf{P}_2^0$$
   iii. Compute $\mathbf{G}_j = S^{\mathbf{0}}(\pi_{\mathbf{A}_j, \mathbf{B}_j}(\mathbf{G}'_0))$.
7. Set $\mathsf{sk} = (\mathbf{A}_1^{-1}, \mathbf{B}_1^{-1}, \ldots, \mathbf{A}_{s-1}^{-1}, \mathbf{B}_{s-1}^{-1})$.
8. Set $\mathsf{pk} = (\mathbf{G}_0, \mathbf{G}_1, \ldots, \mathbf{G}_{s-1})$.

II. Sign(sk)

1. For all $i = 0, \ldots, t-1$:
   i. $\tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i \xleftarrow{\$} \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$.
   ii. Compute $\tilde{\mathbf{G}}_i = S^{\mathbf{0}}(\pi_{\tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i}(\mathbf{G}_0))$.
2. Compute $h = \mathsf{H}(\tilde{\mathbf{G}}_0, \ldots, \tilde{\mathbf{G}}_{t-1}, \mathsf{msg})$.
3. Expand h to $(h_0, \ldots, h_{t-1})$, $0 \leq h_i < s$.
4. For all $i = 0, \ldots, t-1$:
   i. Set $(\mu_i, \nu_i) = (\tilde{\mathbf{A}}_i \mathbf{A}_{h_i}^{-1}, \mathbf{B}_{h_i}^{-1} \tilde{\mathbf{B}}_i)$.
5. Set $\sigma = (\mu_0, \ldots, \mu_{t-1}, \nu_0, \ldots, \nu_{t-1}, h)$.
6. Send σ to verifier.

III. Verify($\mathsf{pk}, \mathsf{msg}, \sigma$)

1. Expand h to $(h_0, \ldots, h_{t-1})$, $0 \leq h_i < s$.
2. For all $i = 0, \ldots, t-1$:
   i. Set $\hat{\mathbf{G}}_i = S^{\mathbf{0}}(\pi_{\mu_i, \nu_i}(\mathbf{G}_{h_i}))$.
3. Compute $h' = \mathsf{H}(\hat{\mathbf{G}}_0, \ldots, \hat{\mathbf{G}}_{t-1}, \mathsf{msg})$.
4. Accept if $h' = h$ or reject otherwise.

Figure 5.5: The MEDS Protocol

Parameter Set	q	n	m	k	s	t	w	ST	PK	SIG	FS
MEDS-2826-st	8191	13	13	13	2	256	30	✓	2826	18 020	−129.74
MEDS-8445-st-f	8191	13	13	13	4	160	23	✓	8445	13 946	−128.01
MEDS-8445-st	8191	13	13	13	4	464	17	✓	8445	10 726	−128.76
MEDS-8445-st-s	8191	13	13	13	4	1760	13	✓	8445	8702	−128.16
MEDS-11255-st	8191	13	13	13	5	224	19	✓	11 255	11 618	−128.45
MEDS-11255	8191	13	13	13	5	224	19	–	11 255	13 778	−128.45
MEDS-42161-st	8191	13	13	13	16	128	16	✓	42 161	9616	−128.85
MEDS-356839-st	8191	13	13	13	128	80	12	✓	356 839	7288	−129.64
MEDS-716471-st	8191	13	13	13	256	64	11	✓	716 471	6530	−127.37

Table 5.2: Parameters for MEDS, for $\lambda = 128$ bits of classical security. 'ST' for seed tree. 'PK' for 'public key size' and 'SIG' for 'signature size in bytes, 'FS' for 'Fiat-Shamir' probability logarithmic to base 2.

Parameter Set	Key Generation		Signing		Verification	
	(ms)	(mcyc.)	(ms)	(mcyc.)	(ms)	(mcyc.)
MEDS-2826-st	71.13	135.14	102.79	195.30	98.00	186.21
MEDS-8445-st-f	211.45	401.75	63.21	120.09	60.14	114.27
MEDS-8445-st	211.35	401.57	185.68	352.79	178.42	339.01
MEDS-8445-st-s	211.77	402.36	697.00	1324.31	673.19	1279.05
MEDS-11255-st	258.18	490.54	88.12	167.44	84.47	160.48
MEDS-11255	258.99	492.08	88.19	167.56	84.50	160.56

Scheme	pk size (byte)	sig size (byte)	key gen (mcyc.)	sign (mcyc.)	verify (mcyc.)
ed25519 (scop)	32	64	0.0484	0.0513	0.182
[133] dilithium2 (scop)	1 312	2 420	0.151	0.363	0.163
[146] falcon512dyn (scop)	897	666	19.5	0.880	0.0856
[170] sphincsf128shake256 (scop)	32	16 976	6.86	220	9.91
[170] sphincss128shake256 (scop)	32	8 080	218	3 500	4.04
[50] UOV ov-Ip	278 432	128	2.90	0.105	0.0903
[33] LESS-I	8 748	12 728	—	—	—
[23] Wavelet	3 236 327	930	7 400	1 640	1.09
[5] SDitH Var3f	144	12 115	—	4.03	3.04
[5] SDitH Var3sss	144	5 689	—	994	969
MEDS-8445-st-f	8 445	13 914	402	120	114
MEDS-11255-st	11 255	11 586	491	168	160
MEDS-42161-st	42 161	9 584	1 840	96.0	92.0
MEDS-716471-st	716 471	6 498	34 200	48.6	55.0

Table 5.4: Performance comparison to other relevant schemes (mcyc. rounded to three significant figures). Data marked with '(scop)' is from the SUPERCOP website. For SPHINCS+ we list results for the 'simple' variant.

Chapter 6

Automorphism group

6.1 Placeholder

The paper will go here.

Considerations

Some considerations on future research on isometries

Part II

Isogenies (CSIDH)

Intro

Some introduction on CSIDH

Chapter 7

Patient Zero & Patient Six

7.1 Placeholder

The paper will go here.

7.2 Introduction

Isogeny-based cryptography is a promising candidate for replacing pre-quantum schemes with practical quantum-resistant alternatives. In general, isogeny-based schemes feature very small key sizes, while suffering from running times that are at least an order of magnitude slower than e.g. lattice- or code-based schemes. Therefore, they present a viable option for applications that prioritize bandwidth over performance. SIKE [175], a key encapsulation mechanism (KEM) based on the key exchange SIDH [178], is the lone isogeny-based participant of the NIST post-quantum cryptography standardization process, and proceeded to the fourth round. In 2018, only after the NIST standardization process started, the key exchange scheme CSIDH was published [**CLM+18**]. Due to its commutative structure, a unique feature among the known post-quantum schemes, CSIDH allows for a non-interactive key exchange, which gained much attention among the research community. Together with its efficient key validation, which enables a static-static key setting, this makes CSIDH a promising candidate for a drop-in replacement of classical Diffie–Hellman-style schemes.

In this work, we focus on a side-channel attack against CSIDH and SIKE. We follow the main idea of [143], which reconstructs SIKE private keys through

zero-value attacks. This attack approach tries to force zero values for some intermediate values of computations related to secret key bits. By recognizing these zero values via side-channel analysis (SCA), this allows an attacker to recover bits of the secret key. While *coordinate randomization* is an effective method to mitigate general *Differential Power Analysis* (DPA) and *Correlation Power Analysis* (CPA), it has no effect on zero values, such that forcing their occurrence bypasses this countermeasure, which is incorporated in SIKE [175]. Similar to [143], the recent *Hertzbleed attack* exploits zero values in SIKE [**hertzbleed**].

While [143] focuses on forcing values connected to elliptic curve points becoming zero, we discuss the occurrence of zero values as curve parameters. This was first proposed in [191], yet [143] concludes that this idea is unlikely to be applicable in a realistic scenario, since curve representations in SIKE are such that they cannot produce a zero. In spite of this fact, we show that some curves in SIKE and CSIDH, as e.g. the zero curve, have a special correlation in these representations, which admits noticing their occurrence via side-channel analysis.

The secret isogeny computation in SIKE essentially consists of two phases: scalar multiplication and isogeny computation. In general, the first phase is believed to be more vulnerable to physical attacks, since private key bits are directly used there (see [**C19**]). We propose the first passive implementation attack using side-channel analysis that exclusively targets the second phase of the SIKE isogeny computation. Notably, countermeasures like coordinate/coefficient randomization [**C19**] or the *CLN test* [108, 143] do not prevent this attack.

Our contributions.

In this work, we present zero-value and correlation attacks against state-of-the-art implementations of CSIDH and SIKE. For CSIDH, we use the fact that the zero curve E_0, i.e., the Montgomery curve with coefficient $a = 0$, represents a valid curve. Thus, whenever a secret isogeny walk passes over this curve, this can be detected via side-channel analysis. We present a passive adaptive attack that recovers one bit of the secret key per round by forcing the target to walk over the zero curve.

Some implementations, like SQALE and SIKE, represent the zero curve without using zero values. Nevertheless, in such a case there is often (with probability 1/2 in SQALE and probability 1 in SIKE) a strong correlation between certain variables, which also occurs for the supersingular six curve E_6 with coefficient

$a = 6$. Via CPA, we exploit this correlation to detect these curves, and mount a similar adaptive attack.

Using these two approaches, we present a generic attack framework, and apply this attack to the state-of-the-art CSIDH implementations SQALE [91] and CTIDH [21] (Section 7.4), and to SIKE (Section 7.5). We explore the practical feasibility of the proposed attacks (Section 7.6), simulations (Section 7.7), and different types of countermeasures (Section 7.8). Our code is available in the public domain:

https://github.com/PaZeZeVaAt/simulation

Related work.

The analysis of physical attacks on isogeny-based schemes has only recently gained more attention, including both side-channel [**hertzbleed**, 143, 157, 191, 292] and fault attacks [2, 69, 70, 156, 196, 278, 281]. Introduced for classical elliptic curve cryptography (ECC) in [6, 161, 174], zero-value attacks were adapted to SIKE in [143], which applies t-tests to determine zero values within power traces [257].

An approach to identify certain structures within traces, similar to the ones occurring in non-zero representations of the zero curve and six curve in our case, are correlation-enhanced power analysis collision attacks [221], such as [37] for ECC. This attack combines the concept of horizontal side-channel analysis [227] with correlation-enhanced power analysis collision attacks to extract leakage from a single trace.

We note that from a constructive perspective, this attack follows the idea of steering isogeny paths over special curves, as proposed for the zero curve in [191]. Furthermore, the attack on SIKE uses the framework of [2] to produce suitable public keys. However, our attack is a *passive* attack that is much easier to perform in practice compared to the elaborate fault injection required for [2].

7.3 Preliminaries

We briefly introduce mathematical background related to isogeny-based cryptography, and the schemes CSIDH [**CLM+18**] and SIKE [175]. For more mathematical details, we refer to [**D17**].

Mathematical background.

Let $\mathbb{F}_q$ with $q = p^k$ denote the finite field of order q, with a prime $p > 3$. Supersingular elliptic curves over $\mathbb{F}_q$ are characterized by the condition $\#E(\mathbb{F}_q) \equiv 1 \mod p$. Throughout this work, we will only encounter group orders that are multiples of 4, and hence elliptic curves E over $\mathbb{F}_q$ with $j(E) \in \mathbb{F}_q$ can be represented in Montgomery form:

$$E_a \colon y^2 = x^3 + ax^2 + x, \quad a \in \mathbb{F}_q. \tag{7.1}$$

Given two such elliptic curves E_a and $E_{a'}$, an isogeny is a morphism $\phi : E_a \to E_{a'}$ such that $\mathcal{O}_{E_a} \mapsto \mathcal{O}_{E_{a'}}$ for the neutral elements of E_a and $E_{a'}$. In the context of isogeny-based cryptography, we are only interested in separable isogenies, which are characterized by their kernel (up to isomorphism): A finite subgroup $G \subset E_a(\overline{\mathbb{F}_q})$ defines a separable isogeny $\phi : E_a \to E_a/G$ and vice versa. In such a case, the degree of ϕ is equal to the size of its kernel, $|G|$. For any isogeny $\phi : E_a \to E_{a'}$, there is a unique isogeny $\hat{\phi} : E_{a'} \to E_a$ such that $\hat{\phi} \circ \phi = [\deg(\phi)]$ is the scalar point multiplication on E_a by $\deg(\phi)$. We call $\hat{\phi}$ the dual isogeny. Two elliptic curves E_a and $E_{a'}$ over $\mathbb{F}_q$ are isogenous, i.e., there exists an isogeny between them, if and only if $\#E_a(\mathbb{F}_q) = \#E_{a'}(\mathbb{F}_q)$.

7.3.1 CSIDH

In the context of CSIDH, we choose p of the form $p + 1 = h \cdot \prod_{i=1}^{n} \ell_i$ and work with supersingular elliptic curves over $\mathbb{F}_p$. Each ℓ_i is a small odd prime, and h is a suitable cofactor to ensure p is prime, with the additional requirement that $4 \mid h$. Usually, we pick p such that $p \equiv 3 \mod 8$ and work with the set $\mathcal{E}$ of supersingular elliptic curves with minimal endomorphism ring $\mathcal{O} \cong \mathbb{Z}[\sqrt{-p}]$. This ensures that the group order $p + 1$ is a multiple of 4, and any such supersingular elliptic curve can be represented uniquely in Montgomery form [**CLM+18**], as given by Equation (10.1) with $a \in \mathbb{F}_p$.

The main operation in CSIDH is the group action of the ideal class group of $\mathcal{O}$ acting on the set $\mathcal{E}$. We are interested in specific ideals $\mathfrak{l}_i$ of $\mathcal{O}$, whose action $\mathfrak{l}_i * E$ on some curve $E \in \mathcal{E}$ is given by an isogeny of degree ℓ_i that is defined by the kernel $G = E[\ell_i] \cap E[\pi - 1]$, where π denotes the Frobenius endomorphism, i.e., $\mathbb{F}_p$-rational points that have ℓ_i-torsion. For $E_a \in \mathcal{E}$ we get that $\#E_a = p + 1$, and $E_a(\mathbb{F}_p) \xrightarrow{\sim} \mathbb{Z}_h \times \prod_{i=1}^{n} \mathbb{Z}_{\ell_i}$. This implies there are ℓ_i of such points $P \in E[\ell_i] \cap E[\pi - 1]$, and $\ell_i - 1$ of these (all but the point $\mathcal{O}_{E_a}$) will generate G. The codomain $E_{a'}$ of such an isogeny is again supersingular and so $|E_{a'}(\mathbb{F}_p)| = p + 1$, which implies $\mathfrak{l}_i$ can also be applied to $E_{a'}$. This

implies a group action of the ideals $\mathfrak{l}_i$ on the supersingular curves E_a over $\mathbb{F}_p$, which we denote by $[\mathfrak{l}_i] * E_a$. In particular, this group action is commutative: $[\mathfrak{l}_i\mathfrak{l}_j] * E_a \xrightarrow{\sim} [\mathfrak{l}_i] * [\mathfrak{l}_j] * E_a \xrightarrow{\sim} [\mathfrak{l}_j] * [\mathfrak{l}_i] * E_a \xrightarrow{\sim} [\mathfrak{l}_j\mathfrak{l}_i] * E_a$. For each $\mathfrak{l}_i$ there exists an inverse $\mathfrak{l}_i^{-1}$, whose action on $E \in \mathcal{E}$ is given by an ℓ_i-isogeny that is defined by the kernel $G = E[\ell_i] \cap E[\pi + 1]$.

For reasons of brevity, in the following we will sometimes abuse notation and identify the ideals $\mathfrak{l}_i^{\pm 1}$ with the ℓ_i-isogenies that their action implies.

The CSIDH scheme.

The CSIDH scheme is based on the group action as described above: We apply each of the n different $\mathfrak{l}_i^{\pm 1}$ a number of times to a given curve E_a, and we denote this number by e_i. Hence, the secret key is some vector of n integers $(e_1, \ldots, e_n)$ defining an element $\mathfrak{a} = \prod_{i=1}^{n} \mathfrak{l}_i^{e_i}$ which we can apply to supersingular curves E_a over $\mathbb{F}_p$. There is some variation between different proposals on where e_i is chosen from: The original proposal of CSIDH-512 picks $e_i \in \{-m, \ldots, m\}$ with $m = 5$, but one can also define individual bounds $m_i \in \mathbb{Z}$ per e_i. The key space is of size $\prod(2m_i + 1)$. For the original CSIDH-512 proposal with $m_i = 5$ and $n = 74$, this gives roughly size 2^{256}.

The public key is the supersingular curve E_a corresponding to applying the secret key $\mathfrak{a}$ to the publicly known starting curve $E_0 : y^2 = x^3 + x$:

$$E_a = \mathfrak{a} * E_0 = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_0. \tag{7.2}$$

To derive a shared secret between Alice and Bob with secret keys $\mathfrak{a}$ and $\mathfrak{b}$ and given public keys $E_a = \mathfrak{a} * E_0$ and $E_b = \mathfrak{b} * E_0$, Alice simply computes $E_{ab} = \mathfrak{a} * E_b$ and Bob computes $E_{ba} = \mathfrak{b} * E_a$. From the commutativity of the group action, we get $E_{ab} \xrightarrow{\sim} E_{ba}$.

Security of CSIDH.

The classical security relies mostly on the size of the keyspace $\prod(2m_i + 1)$, but the quantum security of CSIDH is heavily dependent on the size of the group generated by these elements $\mathfrak{l}_i$. It is heuristically assumed that the $\mathfrak{l}_i$ generate a group of size approximately $\sqrt{p}$. While the original CSIDH proposal considered a 512-bit prime p sufficient for NIST security level 1 [**CLM+18**], its exact quantum security is debated [**2020/peikert**, **2020/bonnetain**, 47, 91]. For instance, [91] claims that 4096-bit primes are required for level 1 security. Note that the key space is not required to cover the full group of size roughly $\sqrt{p}$, but can be chosen as a large enough subset, except for particularly bad choices like

subgroups. At larger prime sizes, the number n of small primes ℓ_i grows, and therefore it becomes natural to pick secret key vectors from $\{-1, 0, 1\}^n$ resp. $\{-1, 1\}^n$ for primes sizes of at least 1792 resp. 2048 bits. This allows for a large enough key space for classical security, while increasing p for sufficient quantum security.

We note that the exact quantum security of CSIDH remains unclear, and thus work on efficient and secure implementations for both smaller and larger parameters continues to appear, e.g. in [21, 91].

Constant-time implementations.

CSIDH is inherently difficult to implement in constant time, as this requires that the timing of the execution is independent of the respective secret key $(e_1, \ldots, e_n)$. However, picking a secret key vector $(e_1, \ldots, e_n)$ translates to the computation of $|e_i|$ isogenies of degree ℓ_i, which directly affects the timing of the group action evaluation. One way to mitigate this timing leakage is by using dummy isogenies: We can keep the total number of isogenies per degree constant by computing m_i isogenies of degree ℓ_i, but discarding the results of $m_i - |e_i|$ of these, effectively making them dummy computations [**MR18**, 212]. Several optimizations and different techniques have been proposed in the literature [**CCC+19**, **CR20**, 236].

The latest and currently most efficient variant of constant-time implementations of CSIDH is CTIDH [21]. In contrast to sampling private key vectors such that $e_i \in \{-m_i, \ldots, m_i\}$, CTIDH uses a different key space that exploits the approach of batching the primes ℓ_i. We define *batches* $B_1, \ldots, B_N$ of consecutive primes of lengths $n_1, \ldots, n_N$, i.e., $B_1 = (\ell_{1,1}, \ldots, \ell_{1,n_1}) = (\ell_1, \ldots, \ell_{n_1})$, $B_2 = (\ell_{2,1}, \ldots, \ell_{2,n_2}) = (\ell_{n_1+1}, \ldots, \ell_{n_1+n_2})$, et cetera. We write $e_{i,j}$ for the (secret) coefficient associated to $\ell_{i,j}$. Instead of defining bounds m_i for each individual ℓ_i so that $|e_i| \leq m_i$, CTIDH uses bounds M_i for the batch B_i, i.e., we compute at most M_i isogenies of those degrees that are contained in B_i. That is, the key sampling requires $|e_{i,1}| + \cdots + |e_{i,n_i}| \leq M_i$. CTIDH then adapts the CSIDH algorithm such that the distribution of the M_i isogenies among degrees of batch B_i does not leak through the timing channel. Among other techniques, this involves Matryoshka isogenies, first introduced in [47], that perform the exact same sequence of instructions independent of its isogeny degree $\ell_{i,j} \in B_i$.

The main advantage of CTIDH is the ambiguity of the isogeny computations: From a time-channel perspective, a Matryoshka isogeny for B_i could be an $\ell_{i,j}$-isogeny for any $\ell_{i,j} \in B_i$. Thus, in comparison to the previous CSIDH algorithms, CTIDH covers the same key space size in fewer isogenies. For instance,

the previously fastest implementation of CSIDH-512 required 431 isogenies in total [1] (including dummies), whereas CTIDH [21] requires only 208 isogenies (including dummies) for the same key space size. This leads to an almost twofold speedup.

Representation of Montgomery coefficient.

To decrease computational cost by avoiding costly inversions, the curve E_a is almost always represented using *projective* coordinates for $a \in \mathbb{F}_p$. The following two are used most in current CSIDH-based implementations:

- the Montgomery form $(A : C)$, such that $a = A/C$, with C non-zero,
- and the alternative Montgomery form $(A + 2C : 4C)$, such that $a = A/C$, with C non-zero.

The alternative Montgomery form is most common, as it is used in projective scalar point multiplication formulas. Hence, in most state-of-the-art implementations of CSIDH-based systems, the Montgomery coefficient a is mapped to alternative Montgomery form and remains in this form until the end, where it is mapped back to affine form for the public key resp. shared secret (e.g., in SQALE [91]). CTIDH [21] switches between both representations after each isogeny, and maps back to affine $a = A/C$ at the end. For most values of $(A : C)$ and $(A + 2C : 4C)$, $a = A/C$ represents either an ordinary or a supersingular curve. The exceptions are $C = 0$, which represents no algebraic object, and $A = \pm 2C$, which represents the singular curves $E_{\pm 2}$. Specifically the supersingular zero curve E_0 is represented as $(0 : C)$ in Montgomery form and $(2C : 4C)$ in alternative Montgomery form, where $C \in \mathbb{F}_p$ can be any non-zero value.

Isogeny computation in projective form.

When using projective representations to compute isogenies with domain E_a where a is represented as $(A : C)$, most implementations use projectivized versions of Vélu's formulas, described in [**velu**, 43, 220]. To compute the action of $\mathfrak{l}_i^{\pm 1}$ on E_a, one finds a point P of order ℓ_i on E_a and computes the x-coordinates of the points $\{P, [2]P, \ldots, [\frac{\ell-1}{2}]P\}$. Let $(X_k : Z_k)$ denote the x-coordinate of $[k]P$ in projective form. Then, the projective Montgomery coefficient $(A' : C')$ of $E_{a'} = \mathfrak{l}_i * E_a$ using Montgomery form $(A : C)$ is computed by

$$B_z = \prod_{k=1}^{\frac{\ell-1}{2}} Z_k, \quad A' = (A + 2C)^{\ell} \cdot B_z^8, \tag{7.3}$$

$$B_x = \prod_{k=1}^{\frac{\ell-1}{2}} X_k, \quad C' = (A-2C)^\ell \cdot B_x^8, \tag{7.4}$$

and when using alternative Montgomery form $(\alpha : \beta) = (A + 2C : 4C)$ by

$$B_z = \prod_{k=1}^{\frac{\ell-1}{2}} Z_k, \quad \alpha' = \alpha^\ell \cdot B_z^8, \tag{7.5}$$

$$B_x = \prod_{k=1}^{\frac{\ell-1}{2}} X_k, \quad \beta' = \alpha' - (\alpha-\beta)^\ell \cdot B_x^8, \tag{7.6}$$

where $(\alpha' : \beta')$ represents $E_{a'}$ in alternative Montgomery form. Note that the values $(A + 2C)$ in (7.3), $(A - 2C)$ in (7.4), α in (7.5) and $(\alpha - \beta)$ in (7.6) are never zero: In all cases, this implies $A/C = \pm 2$, i.e., the singular curves $E_{\pm 2}$.

Remark 7.1. So far, we know of no deterministic implementations based on the class group action. This is because in order to perform the isogenies, all current implementations sample a random point P on the curve and compute the scalar multiple of P required to perform isogenies. The projective coordinates $(X_k : Z_k)$ are then non-deterministic, and hence the output of Equations (7.3) to (7.6) is non-deterministic. This implies that the representation of a as $(A : C)$ or $(A + 2C : 4C)$ is non-deterministic after the first isogeny. A deterministic approach, e.g. as sketched in [47] using Elligator, ensures a deterministic representation of a, but has so far not been put into practice.

7.3.2 SIKE

In SIKE, we pick a prime of the form $p = 2^{e_A} \cdot 3^{e_B} - 1$ such that $2^{e_A} \approx 3^{e_B}$, and work with supersingular elliptic curves over $\mathbb{F}_{p^2}$ in Montgomery form. We choose to work with curves such that $E_a(\mathbb{F}_{p^2}) = (p+1)^2$, and we have $E_a(\mathbb{F}_{p^2}) \xrightarrow{\sim} \mathbb{Z}_{2^{e_A}}^2 \times \mathbb{Z}_{3^{e_B}}^2$ for these curves. Thus, the full 2^{e_A}- and 3^{e_B}-torsion subgroups lie in $E_a(\mathbb{F}_{p^2})$. Any point R_A of order 2^{e_A} then uniquely (up to isomorphism) determines a 2^{e_A}-isogeny and codomain curve $E_{a'} = E_a/\langle R_A \rangle$ with kernel $\langle R_A \rangle$. For choosing an appropriate point, the SIKE setup defines basis points P_A and Q_A of the 2^{e_A}-torsion of the public starting curve. Picking an integer $\mathsf{sk_A} \in [0, 2^{e_A} - 1]$ and computing $R_A = P_A + [\mathsf{sk_A}]Q_A$ then results in choosing such a kernel generator R_A of order 2^{e_A}.

In practice, such a 2^{e_A}-isogeny is computed as a sequence of 2-isogenies of length e_A. This can be interpreted as a sequence of steps through a graph: For

a prime ℓ with $\ell \neq p$, the ℓ-isogeny graph consists of vertices that represent (j-invariants of) elliptic curves, and edges representing ℓ-isogenies. Due to the existence of dual isogenies, edges are undirected. For supersingular curves, this graph is an $(\ell+1)$-regular expander graph and contains approximately $p/12$ vertices. Hence, a sequence of 2-isogenies of length e_A corresponds to a walk of length e_A through the 2-isogeny graph. An analogous discussion applies to the case of 3^{e_B}-isogenies. Note that for reasons of efficiency, we often combine two 2-isogeny steps into one 4-isogeny.

The secret keys $\mathsf{sk_A}$, $\mathsf{sk_B}$ can be decomposed as

$$\mathsf{sk_A} = \sum_{i=0}^{e_2-1} \mathsf{sk}_i \cdot 2^i \quad \mathsf{sk}_i \in \{0,1\}, \quad \mathsf{sk_B} = \sum_{i=0}^{e_3-1} \mathsf{sk}_i \cdot 3^i \quad \mathsf{sk}_i \in \{0,1,2\}.$$

We refer to these sk_i as the *bits* resp. the *trits* of the secret key $\mathsf{sk_A}$ resp. $\mathsf{sk_B}$. For a given sk, we use $\mathsf{sk}_{<k}$ to represent the key up to the k-th bit/trit sk_{k-1}.

The SIKE scheme.

The main idea behind SIDH and SIKE is to use secret isogenies to set up a key exchange scheme resp. key encapsulation mechanism. SIDH fixes E_6 as starting curve, and torsion basis points P_A, Q_A and P_B, Q_B. It uses the following subroutines:

- Keygen_A samples a secret key $\mathsf{sk_A} \in [0, 2^{e_A} - 1]$, computes $R_A = P_A + [\mathsf{sk_A}]Q_A$, and the secret isogeny $\varphi_A : E_6 \to E_6/\langle R_A \rangle$. It outputs the key pair $(\mathsf{sk_A}, \mathsf{pk_A})$, where $\mathsf{pk_A} = (\varphi_A(P_B), \varphi_A(Q_B), \varphi_A(Q_B - P_B))$. We write $\mathsf{Keygen}_A(\mathsf{sk})$ if Keygen_A does not sample a secret key, but gets sk as input.

- Keygen_B proceeds analogously with swapped indices A and B. The public key is $\mathsf{pk_B} = (\varphi_B(P_A), \varphi_B(Q_A), \varphi_B(Q_A - P_A))$.

- Derive_A takes as input $(\mathsf{sk_A}, \mathsf{pk_B}) = (S_A, T_A, T_A - S_A)$. It computes the starting curve E_B from the points in $\mathsf{pk_B}$, the secret point $R'_A = S_A + [\mathsf{sk_A}]T_A$, and the isogeny $\varphi'_A : E_B \to E_B/\langle R'_A \rangle$.

- Derive_B proceeds analogously with input $(\mathsf{sk_B}, \mathsf{pk_A})$, and computes the codomain curve $E_A/\langle R'_B \rangle$.

When running this key exchange, both parties arrive at a curve (isomorphic to) $E_6/\langle R_A, R_B \rangle$, and (a hash of) its j-variant can serve as a shared secret.

SIKE uses the SIDH subroutines Keygen and Derive to construct three algorithms Keygen, Encaps, and Decaps. Furthermore, we define h and h' to be cryptographic hash functions.

- Keygen generates a (static) key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Keygen}_B$.
- Encaps encapsulates a random value m in the following way:
  - Get an ephemeral key pair $(\mathsf{ek}, c) \leftarrow \mathsf{Keygen}_A(\mathsf{ek})$ with $\mathsf{ek} = h(\mathsf{pk}, m)$.
  - Compute the shared secret $s \leftarrow \mathsf{Derive}_A(\mathsf{ek}, \mathsf{pk})$.
  - Compute the ciphertext $\mathsf{ct} = (c, h'(s) \oplus m)$.
- Decaps receives a ciphertext (c_0, c_1), and proceeds as follows:
  - Compute the shared secret $s' \leftarrow \mathsf{Derive}_B(\mathsf{sk}, c_0)$.
  - Recover $m' \leftarrow c_1 \oplus h'(s')$.
  - Recompute $\mathsf{ek}' = h(\mathsf{pk}, m')$.
  - Compute $(\mathsf{ek}', c') \leftarrow \mathsf{Keygen}_A(\mathsf{ek}')$ and check if $c' = c_0$.

  Passing this check guarantees that the ciphertext has been generated honestly, and $m' = m$ can be used to set up a session key.

Representation of Montgomery coefficients.

As in CSIDH, the curve E_a is almost always represented using projective coordinates, with the caveat that $a \in \mathbb{F}_{p^2}$. The following two representations are used throughout SIKE computations, although in different subroutines.

- The alternative Montgomery form $(A+2C : 4C)$, such that $a = A/C$ with C non-zero. This representation is used for Alice's computations as it is the most efficient for computing 2-isogenies. It is often written as $(A_{24}^+ : C_{24})$ with $A_{24}^+ = A + 2C$ and $C_{24} = 4C$ so that $a = 2(2A_{24}^+ - C_{24})/C_{24}$.
- The form $(A + 2C : A - 2C)$, such that $a = A/C$, with C non-zero. This representation is used for Bob's computations as it is the most efficient for computing 3-isogenies. It is often written as $(A_{24}^+ : A_{24}^-)$ with $A_{24}^+ = A+2C$ and $A_{24}^- = A - 2C$ so that $a = 2(A_{24}^+ + A_{24}^-)/(A_{24}^+ - A_{24}^-)$.

Note that the values A, C, A_{24}^+, A_{24}^- and C_{24} are in $\mathbb{F}_{p^2}$. When necessary, we write them as $\alpha + \beta i$ with $\alpha, \beta \in \mathbb{F}_p$ and $i^2 = -1$. Equal to CSIDH, both forms represent either an ordinary or a supersingular curve, with the exceptions $C = 0$, which represents no algebraic object, and $A = \pm 2C$, which represents the singular curves $E_{\pm 2}$. For the rest of the paper, we are interested in representations of the supersingular six curve E_6. Fortunately, E_6 is represented in *both* forms as $(8C : 4C)$, with $C = \alpha + \beta i \in \mathbb{F}_{p^2}$ any non-zero element. For the goal of the paper, this means that the analysis is similar for both forms.

Isogeny computation in projective form.

SIKE uses the above projective representations to compute the codomain $E_{\tilde{a}}$ of a 3- or 4-isogeny $\varphi : E_a \to E_{\tilde{a}}$.

4-isogeny. Given a point P of order 4 on E_a with x-coordinate $x(P) = (X : Z)$, the codomain $E_{\tilde{a}} = E_a/\langle P \rangle$ with $\tilde{a}$ represented by $(\tilde{A}_{24}^+ : \tilde{C}_{24})$ is computed by

$$\tilde{A}_{24}^+ = 4 \cdot X^4, \qquad \tilde{C}_{24} = 4 \cdot Z^4. \tag{7.7}$$

3-isogeny. Given a point P of order 3 on E_a with x-coordinate $x(P) = (X : Z)$, the codomain $E_{\tilde{a}} = E_a/\langle P \rangle$ with $\tilde{a}$ represented by $(\tilde{A}_{24}^+ : \tilde{A}_{24}^-)$ is computed by

$$\tilde{A}_{24}^+ = (3X + Z)^3 \cdot (X - Z), \quad \tilde{A}_{24}^- = (3X - Z)^3 \cdot (X + Z). \tag{7.8}$$

7.4 Recovering CSIDH keys with E_0 side-channel leakage

In this section, we explore how side-channel information can leak information on secret isogeny walks. As shown in [143], it is possible to detect zero values in isogeny computations using side-channel information. In Section 7.4.1, we specifically explore how both representations of the zero curve E_0, i.e. $(0 : C)$ and $(2C : 4C)$, leak secret information, even though the value $C \in \mathbb{F}_p$ is assumed to be a uniformly random non-zero value. As E_0 is always a valid supersingular $\mathbb{F}_p$-curve in CSIDH, we can always construct a walk that potentially passes over E_0. This allows us to describe a generic approach to leak a given bit of information of the secret isogeny walk, hence, a general attack on the class group action as introduced in CSIDH. We apply this attack in more detail to

the two current state-of-the-art cryptosystems based on this class group action: SQALE in Section 7.4.2 and CTIDH in Section 7.4.3. We discuss their practical feasibility in Section 7.6 and simulate these attacks in Section 7.7. We note that the proposed attack applies to all variants of CSIDH that we know of, e.g. from [**CLM+18**, **CCC+19**].

Throughout this work, we assume a static-key setting, i.e., that a long-term secret key $\mathfrak{a}$ is used, and that the attacker can repeatedly trigger key exchange executions on the target device using public key curves of their choice. Formally, this means that we adaptively feed curves $E_{\mathtt{PK}}$ and get side-channel information on the computations $\mathfrak{a} * E_{\mathtt{PK}}$. We exploit this information to reveal $\mathfrak{a}$ bit by bit.

7.4.1 Discovering a bit of information on a secret isogeny walk

Detecting E_0 in Montgomery form.

As described in Remark 7.1, the representation of the Montgomery coefficient as $(A : C)$ or $(A + 2C : 4C)$ is non-deterministic after the first isogeny, so they effectively contain random $\mathbb{F}_p$-values, representing the affine Montgomery coefficient a. This makes it hard to get any information on E_a using side channels. However, in Montgomery form the curve E_0 is special: It is simply represented by $(0 : C)$ for some $C \in \mathbb{F}_p$. We define such a representation containing a zero a *zero-value representation*.

Definition 7.1. Let E_a be an elliptic curve over $\mathbb{F}_p$. A *zero-value representation* is a representation of the Montgomery coefficient a in projective coordinates $(\alpha : \beta)$ such that either $\alpha = 0$ or $\beta = 0$.

Clearly, a representation of E_0 in Montgomery form must be a zero-value representation. As is known for ECC and SIKE, an attacker can observe zero-value representations in several different ways using side-channel analysis [143]. We will expand on this in section 7.6 to show that E_0 leaks secret information in implementations that use Montgomery form.

Detecting E_0 in alternative Montgomery form.

Using the alternative Montgomery form, no non-singular curve has a zero-value representation, as $(A+2C : 4C)$ can only be zero for $A = -2C$ corresponding to $a = -2$, which represents the singular curve E_{-2}. Thus, the alternative Montgomery form avoids the side-channel attack described above. Nevertheless, the

representation of E_0 is still unusual: Whenever $2C$ is smaller than $p/2$, doubling $2C$ does not require a modular reduction, and hence the bit representation of $4C$ is precisely a bit shift of $2C$ by one bit to the left. Such strongly correlated values can be observed in several ways using side-channel analysis, as we detail later in section 7.6.

Definition 7.2. Let E_a be an elliptic curve over $\mathbb{F}_p$. A *strongly-correlated representation* is a representation of the Montgomery coefficient a in projective coordinates $(\alpha : \beta)$ such that the bit representations of α and β are bit shifts.[1]

For E_0, for any non-zero value C with $2C \leq p/2$, the representation in alternative Montgomery form by $(2C : 4C)$ is a strongly-correlated representation. As C is effectively random during the computation of the class group action, in roughly 50% of the cases where we pass over E_0, the representation is strongly correlated. For random values of a, the values of $(A + 2C : 4C)$ are indistinguishable from random $(\gamma : \delta)$, and so an attacker can differentiate E_0 from such curves. From this, an attacker only needs a few traces to determine accurately whether a walk passes over E_0 or not, as discussed in Section 7.6.

Remark 7.2. Other curves have strongly-correlated representations too, e.g., the curve E_6 requires $A = 6C$ which gives $(8C : 4C)$ with $C \in \mathbb{F}_p$ random and non-zero, and so E_6 can be detected in precisely the same way as E_0. For simplicity, we focus on the zero curve in the CSIDH attack. We note that analyzing this attack to any curve with strongly-correlated representations is of independent interest for CSIDH and other isogeny-based schemes (such as SIKE).

Remark 7.3. In the case where $2C$ is larger than $p/2$, the modular reduction by p decreases the correlation between $2C$ and $4C$ significantly, which is why we disregard these cases. However, a modular reduction does not affect all bits, and so this correlation remains for unaffected bits. Especially for primes with large cofactor 2^k in $p + 1$, or primes close to a power of 2, the correlation between unaffected bits should be exploitable. For the primes used in the CSIDH instances in this work, this effect is negligible. However, the primes used in SIDH and SIKE do have this form and we exploit this in Section 7.5.

The idea is now to detect E_0 in a certain step k of the computation $\mathfrak{a} * E_{\mathtt{PK}}$. In order to ensure that this happens the computation needs to be performed

[1] This definition may be expanded to cover other types of correlation, whenever such correlation can be distinguished from random values using side-channel information.

in a known order of isogeny steps $E \to \mathfrak{l}^{(k)} * E$. In general, by the way how isogenies are computed, such a step can fail with a certain probability. The following definition takes this into account.

Definition 7.3. Let $\mathfrak{a}$ be a secret isogeny walk. An *ordered evaluation* of $\mathfrak{a} * E$ is an evaluation in a fixed order

$$\mathfrak{l}^{(n)} * \ldots * \mathfrak{l}^{(1)} * E$$

of n steps, assuming that no step fails. We write $\mathfrak{a}_k * E$ for the first k steps of of such an evaluation,

$$\mathfrak{l}^{(k)} * \ldots * \mathfrak{l}^{(1)} * E.$$

We define $p_{\mathfrak{a}}$ resp. $p_{\mathfrak{a}_k}$ as the probability that $\mathfrak{a}$ resp. $\mathfrak{a}_k$ is evaluated without failed steps.

Generic approach to discover isogeny walks using E_0.

Given the ability to detect E_0 in a walk for both the Montgomery form and the alternative Montgomery form, we sketch the following approach to discover bits of a secret isogeny walk $\mathfrak{a}$ that has an ordered evaluation. Assuming we know the first $k-1$ steps $\mathfrak{l}^{(k-1)} * \ldots * \mathfrak{l}^{(1)}$ in the secret isogeny walk $\mathfrak{a}$, denoted by $\mathfrak{a}_{k-1}$, we want to see if the k-th step $\mathfrak{l}^{(k)}$ equals $\mathfrak{l}_i$ or $\mathfrak{l}_i^{-1}$ for some i. We compute $E_a = \mathfrak{l}_i^{-1} * E_0$ and $E_{a'} = \mathfrak{l}_i^{-1} * E_a$, and as a public key we use $E_{\mathrm{PK}} = \mathfrak{a}_{k-1}^{-1} * E_a$. Then, when applying the secret walk $\mathfrak{a}$ to E_{PK}, the k-th step either goes over E_0 or over $E_{a'}$. From side-channel information, we observe if the k-th step applies $\mathfrak{l}_i^e = \mathfrak{l}_i^1$ or $\mathfrak{l}_i^e = \mathfrak{l}_i^{-1}$, and set $\mathfrak{l}^{(k)} = \mathfrak{l}_i^e$, as shown in Figure 7.1. Then we repeat with $\mathfrak{a}_k = \mathfrak{l}_i^e \cdot \mathfrak{a}_{k-1}$.

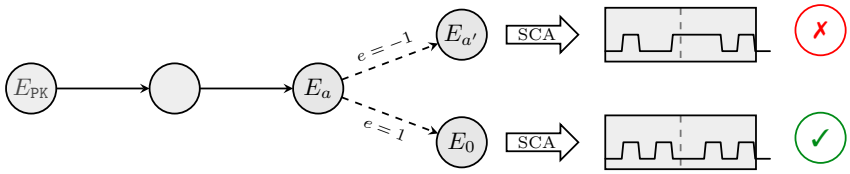


Figure 7.1: Generic approach to discover secret bits using side-channel information.

If E_0 is not detected in the above setting, i.e. $e = -1$, we can confirm this by an additional measurement: We compute $\tilde{E_a} = \mathfrak{l} * E_0$ and $\tilde{E_{a'}} = \mathfrak{l} * \tilde{E_a}$, and

use $\tilde{E}_{\mathtt{PK}} = \mathfrak{a}_{k-1}^{-1} * \tilde{E}_a$ as public key. If $e = -1$, the isogeny walk now passes over E_0, which can be recognized via side-channel analysis. More formally, we get:

Lemma 7.4. Let $\mathfrak{a}$ be any isogeny walk of the form $\mathfrak{a} = \prod \mathfrak{l}_i^{e_i}$. Assume the evaluation of $\mathfrak{a}$ is an ordered evaluation. Then, there exists a supersingular curve $E_{\mathtt{PK}}$ over $\mathbb{F}_p$ such that $\mathfrak{a} * E_{\mathtt{PK}}$ passes over E_0 in the k-th step.

After successfully detecting all steps $\mathfrak{l}^{(k)}$, the private key elements e_i can simply be recovered by counting how often $\mathfrak{l}_i$ resp. $\mathfrak{l}_i^{-1}$ appeared in the evaluation.

This generic approach has a nice advantage: If one detects the k-th step to walk over E_0, this confirms all previous steps were guessed correctly. In other words, guessing wrongly in a certain step will be noticed in the next step: Denote a wrong guess by $\mathfrak{a}_k^{\text{wrong}} = \mathfrak{l}^{-e} \cdot \mathfrak{a}_{k-1}$. The attacker computes E_a from E_0 so that $\mathfrak{l}' * E_a = E_0$ and gives the target $E_{\mathtt{PK}}$ such that $\mathfrak{a}_k^{\text{wrong}} * E_{\mathtt{PK}} = E_a$. Due to the wrong guess, neither $e = 1$ nor $e = -1$ lead to E_0, as the *actual* secret walk $\mathfrak{a}$ leads to $E_{a'} = \mathfrak{a}_k * E_{\mathtt{PK}}$, and the case $e = 1$ leads to $E_{\neg 0} = \mathfrak{l}' * E_{a'} = \mathfrak{l}^{-2e} * E_0$, as shown in Figure 7.2.

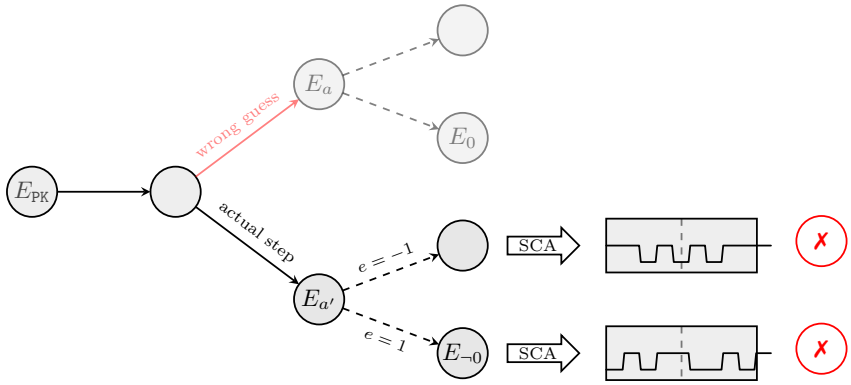


Figure 7.2: Due to a wrong guess of the isogeny path $\mathfrak{a}_k$, an attacker miscomputes $E_{\mathtt{PK}}$ and the actual walk does not pass over E_0.

Remark 7.4. Note that $E_{\mathtt{PK}}$ given by Lemma 7.4 is a valid CSIDH public key, so public key validation (see [**CLM+18**]) does not prevent this attack.

Probability of a walk passing over E_0.

Due to the probabilistic nature of the computation of the class group action, not every evaluation $\mathfrak{a} * E_{\mathtt{PK}}$ passes over E_0 in the k-th step: One of the steps $\mathfrak{l}^{(j)}$ for $1 \leq j \leq k-1$ can fail with probability $1/\ell^{(j)}$, and if so, the k-th step passes over a different curve. With $E_{\mathtt{PK}}$ as given by Lemma 7.4, the probability that an ordered evaluation $\mathfrak{a} * E_{\mathtt{PK}}$ passes over E_0 is then described by $p_{\mathfrak{a}_k}$, which we compute in Lemma 7.5.

Lemma 7.5. Let $\mathfrak{a}$ be an isogeny walk computed as an ordered evaluation $\mathfrak{l}^{(n)} * \ldots * \mathfrak{l}^{(1)} * E_{\mathtt{PK}}$. Then $p_{\mathfrak{a}_k}$, the probability that the first k isogenies succeed, is

$$p_{\mathfrak{a}_k} := \prod_{j=1}^{k} \frac{\ell^{(j)} - 1}{\ell^{(j)}}$$

where $\ell^{(j)}$ is the degree of the isogeny $\mathfrak{l}^{(j)}$ in the j-th step.

As $p_{\mathfrak{a}_k}$ describes the chance that we pass over E_0 in the k-th step, $1/p_{\mathfrak{a}_k}$ gives us the estimated number of measurements of $\mathfrak{a} * E_{\mathtt{PK}}$ we need in order to pass over E_0 in step k. We apply this more concretely in Sections 7.4.2 and 7.4.3.

Remark 7.5. Instead of learning bit by bit starting from the beginning of the secret isogeny walk, we can also start at the end of the walk. To do so, we use the twist E_{-t} of the target's public key E_t, for which $\mathfrak{a} * E_{-t} = E_0$. As for the generic attack, we feed $E_{\mathtt{PK}} = \mathfrak{l}^{-1} * E_{-t}$ and $\tilde{E}_{\mathtt{PK}} = \mathfrak{l} * E_{-t}$. The computation then passes over E_0 in the *last* step instead of the *first*. This approach requires the same probability $p_{\mathfrak{a}_k}$ to recover the k-th bit, but assumes knowledge of all bits after k instead of before. Hence, we can discover starting and ending bits of $\mathfrak{a}$ in parallel.

7.4.2 Recovering secret keys in SQALE

SQALE [91] is the most recent and most efficient constant-time implementation of CSIDH for large parameters, featuring prime sizes between 1024 and 9216 bit. In this section, we explain how the attack from section 7.4.1 can be applied to SQALE, leading to a full key recovery. For concreteness, we focus on SQALE-2048, which uses parameters $n = 231$ and secret exponents $e_i \in \{-1, 1\}$ for $1 \leq i \leq 221$. The ℓ_i with $i > 221$ are not used in the group action.

Algorithmic description of SQALE.

Given a starting curve E_A, the SQALE implementation computes the group action in the following way:

- Sample random points $P_+ \in E_A[\pi - 1]$ and $P_- \in E_A[\pi + 1]$, and set $E \leftarrow E_A$.
- Iterate through $i \in \{1, \ldots, n\}$ in ascending order, and attempt to compute $\varphi : E \to \mathfrak{l}_i^{e_i} * E$ using P_+ resp P_-. Push both points through each φ.
- In case of point rejections, sample fresh points and attempt to compute the corresponding isogenies, until all $\mathfrak{l}_i^{e_i}$ have been applied.

In order to speed up computations, SQALE additionally pushes intermediate points through isogenies, which saves computational effort in following steps [**CR20**]. However, the exact design of the computational strategy inside CSIDH is not relevant for the proposed attack. Using the above description, we sketch the adaptive attack on SQALE-2048 to recover the secret key bit by bit. In case of no point rejections, the order of steps in which $\mathfrak{a} * E_{\mathtt{PK}}$ is computed in SQALE is deterministic, and thus we can immediately apply Lemmas 7.4 and 7.5:

Corollary 7.1. If no point rejections occur, the computation $\mathfrak{a} * E_{\mathtt{PK}}$ in SQALE is an ordered evaluation with

$$\mathfrak{l}^{(n)} * \ldots * \mathfrak{l}^{(1)} * E_{\mathtt{PK}} = \mathfrak{l}_{221}^{e_{221}} * \ldots * \mathfrak{l}_1^{e_1} * E_{\mathtt{PK}}.$$

Hence, $p_{\mathfrak{a}_k} = \prod_{i=1}^{k} \frac{\ell_i - 1}{\ell_i}$.

SQALE uses coefficients in alternative Montgomery form $(A + 2C : 4C)$, so that passing over the curve E_0 can be detected as described in Section 7.4.1.

Recovering the k-th bit.

Recovering the k-th bit of a SQALE secret key works exactly as described in fig. 7.1, as in a successful run SQALE performs each step $\mathfrak{l}_i^{\pm 1}$ in ascending order. Thus, the k-th step, in a run where the first k steps succeed, computes $E \to \mathfrak{l}_k^{\pm 1} * E$. For the attack, we assume knowledge of the first $k - 1$ bits of the secret to produce public keys $E_{\mathtt{PK}}$ resp. $\tilde{E}_{\mathtt{PK}}$ that lead the target through E_0 via an application of $\mathfrak{l}_k^{-1}$ resp. $\mathfrak{l}_k$, as given by Lemma 7.4. For one of these cases,

with probability $p_{\mathfrak{a}_k}$ (Lemma 7.5), the target passes over E_0 on the k-th step, and we learn the k-th secret bit e_k from side-channel information.

As k increases, $p_{\mathfrak{a}_k}$ decreases: In order for the target to pass over E_0 in one of the two cases, *all* previous isogenies have to succeed, for which Corollary 7.1 gives the probability $p_{\mathfrak{a}_k}$. Thus, the fact that SQALE first computes small-degree isogenies is slightly inconvenient for the attack, due to their low success probabilities. Nevertheless, attacking the last round of SQALE-2048 has a success probability of roughly $p_{\mathfrak{a}_{221}} = \prod_{j=1}^{221}(\ell_j - 1)/\ell_j \approx 19.3\%$, so that in about 1 in 5 runs, every isogeny succeeds and we pass over E_0 for the 221-th bit, compared to 2 in 3 runs to pass over E_0 for the first bit ($p_{\mathfrak{a}_1} = \frac{2}{3}$). This means that we need about three times as many measurements to discover the last bit, than the first bit. Nonetheless the required total number of measurements for all bits is very managable; we get with Lemma 7.5:

Corollary 7.2. Assuming a pass over E_0 leaks the k-th bit when the representation is strongly correlated, the estimated number of measurements to recover a SQALE-2048 key is

$$4 \cdot \sum_{k=1}^{221} \frac{1}{p_{\mathfrak{a}_k}} = 4 \cdot \sum_{k=1}^{221} \prod_{i=1}^{k} \frac{\ell_i}{\ell_i - 1} \approx 4 \cdot 1020.$$

Here, the factor 4 represents the fact that we need to feed both $E_{\mathtt{PK}}$ and $\tilde{E}_{\mathtt{PK}}$, and that only half the time ($2C : 4C$) is strongly-correlated. In practice, for more certainty, we increase the number of attempts per bit by some constant α, giving a total of $\alpha \cdot 4 \cdot 1020$ expected attempts. We detail this in section 7.7.

7.4.3 Recovering secret keys in CTIDH

CTIDH [21] is the most efficient constant-time implementation of CSIDH to date, although the work restricts to the CSIDH-512 and CSIDH-1024 parameter sets. We note that techniques from CTIDH can be used to significantly speed up CSIDH for larger parameters too, yet this appears to require some modifications that have not been explored in the literature yet. In this section, we explain how zero-value curve attacks can be mounted on CTIDH, leading to a partial or full key recovery, depending on the number of measurements that is deemed possible. For concreteness, we focus on the CTIDH parameter set with a 220-bit key space, dubbed CTIDH-511 in [21], which uses 15 batches of up to 8 primes. The bounds satisfy $M_i \leq 12$.

Algorithmic description of CTIDH.

Given a starting curve E_A, CTIDH computes the group action by multiple rounds of the following approach:

- Set $E \leftarrow E_A$, sample random points $P_+ \in E[\pi - 1]$ and $P_- \in E[\pi + 1]$.
- Per batch B_i, (attempt to) compute $\varphi : E \to \mathfrak{l}_{i,j}^{\mathtt{sign}(e_{i,j})} * E$ using P_+ resp P_- (or dummy when all $\mathfrak{l}_{i,j}^{e_{i,j}}$ are performed). Push both points through each φ.
- Repeat this process until all $\mathfrak{l}_{i,j}^{e_{i,j}}$ and dummy isogenies have been applied.

Furthermore, the following design choices in CTIDH are especially relevant:

- Per batch B_i, CTIDH computes real isogenies first, and (potential) dummy isogenies after, to ensure M_i isogenies are computed, independent of $(e_{i,j})$.
- Per batch B_i, CTIDH computes the actual $\ell_{i,j}$-isogenies in ascending order.
- Per batch B_i, CTIDH scales the point rejection probability to the largest value, $1/\ell_{i,1}$. This slightly changes the computation of $p_{\mathfrak{a}_k}$.
- The order in which batches are processed is deterministic.

Example 7.1. Let $B_1 = \{3, 5\}$ with $M_1 = 6$, and let $e_{1,1} = 2$ and $e_{1,2} = -3$. For B_1, we first try to compute $E \to \mathfrak{l}_1 * E$, until this succeeds twice. Then, we try to compute $E \to \mathfrak{l}_2^{-1} * E$, until this succeeds three times. After the real isogenies, we try to compute the remaining B_1-dummy isogeny. All B_1-isogenies, including dummies, have success probability $2/3$. If all six of the B_1-isogenies are performed but other B_i are unfinished, we skip B_1 in later rounds.

As for SQALE, the above description gives us that the order in which each $\mathfrak{l}$ is applied in CTIDH is deterministic, assuming that none of the steps fail, and so we get with Lemmas 7.4 and 7.5 again:

Corollary 7.3. If no point rejections occur, the computation $\mathfrak{a} * E$ in CTIDH is an ordered evaluation $\mathfrak{l}^{(n)} * \ldots * \mathfrak{l}^{(1)} * E$, with $n = \sum M_i$, including dummy isogenies.

Hence we can perform the adaptive attack on CTIDH-511 to recover the secret key bit by bit. The CTIDH implementation of [21] uses coefficients in

alternative Montgomery form $(A+2C:4C)$, but passes over Montgomery form $(A:C)$ after each isogeny. Hence, E_0 always has a zero-value representation and we detect E_0 as described in Section 7.4.1. We argue in section 7.6 that zero-value representations are easier to detect than strongly-correlated representations.

Recovering the k-th bit.

CTIDH introduces several difficulties for the attack, compared to SQALE. In particular, let $B_i = \{\ell_{i,1}, \ldots, \ell_{i,n_i}\}$ be the batch to be processed at step k. Then, since usually $n_i > 1$, we do not get a binary decision at each step as depicted in fig. 7.1, but a choice between $2 \cdot n_i$ real isogeny steps $\mathfrak{l}_{i,j}^{\pm 1}$, or possibly a dummy isogeny. In practice, with high probability, we do not need to cover all $2 \cdot n_i + 1$ options, as the following example shows.

Example 7.2. As CTIDH progresses through the batch ascendingly from $\ell_{i,1}$ to ℓ_{i,n_i}, the first step of a batch can often be recovered as in fig. 7.1, using public keys that are one $\ell_{i,1}$-isogeny away from E_0 respectively. If both do not pass over E_0, we deduce that $e_{i,1} = 0$, and we repeat this approach using an $\ell_{i,2}$-isogeny. In case of a successful attempt for $\ell_{i,j}$, we learn that the respective key element satisfies $e_{i,j} \leq -1$ resp. $e_{i,j} \geq 1$, depending on which of the binary steps was successful.[2] If we do not succeed in detecting E_0 after trying all $\ell_{i,j}^{\pm 1}$ in B_i, we learn that the target computes a B_i-dummy isogeny, and so all $e_{i,j} = 0$ for $\ell_{i,j} \in B_i$. We can easily confirm dummy isogenies: If the k-th step is a dummy isogeny, then using $E_{\mathtt{PK}}$ such that $\mathfrak{a} * E_{\mathtt{PK}}$ passes over E_0 in step $k-1$, we do not move to a different curve in step k and so we observe E_0 using side-channel information after steps $k-1$ *and* k.

This approach to recover the k-th bit in CTIDH-511 only differs slightly from Section 7.4.2: Given the knowledge of the secret path up to step $k-1$, we recover the k-th step by iterating through the target batch $B_i = \{\ell_{i,1}, \ldots, \ell_{i,n_i}\}$, until we detect E_0 for a given degree $\ell_{i,j}$, or otherwise assume a dummy isogeny. This iteration becomes easier in later rounds of each batch:

- If a previous round found that some $e_{i,j}$ is positive, we only have to check for positive $\ell_{i,j}$-isogeny steps later on (analogously for negative).
- If a previous round computed an $\ell_{i,j}$-isogeny, we immediately know that the current round cannot compute an $\ell_{i,h}$-isogeny with $h < j$.

[2] Note that the $e_{i,j}$ are not limited to $\{-1, 1\}$ in CTIDH, in contrast to e_i in SQALE.

- If a previous round detected a dummy isogeny for batch B_i, we can skip isogenies for B_i in all later rounds, since only dummy isogenies follow.

Thus, knowledge of the previous isogeny path significantly shrinks the search space for later steps. As in SQALE, the probability $p_{\mathfrak{a}_k}$ decreases the further we get: Batches containing small degrees ℓ_i appear multiple times, and steps with small ℓ_i have the most impact on $p_{\mathfrak{a}_k}$. For the last step $\mathfrak{l}^{(n)}$, the probability that *all* steps $\mathfrak{l}^{(k)}$ in CTIDH-511 succeed without a single point rejection, is roughly 0.3%. This might seem low at first, but the number of measurements required to make up for this probability does not explode; we are able to recover the full key with a reasonable amount of measurements as shown in Section 7.7. Furthermore, this probability represents the absolute lower bound, which is essentially the worst-case scenario: It is the probability that for the worst possible key, with no dummy isogenies, all steps must succeed in one run. In reality, almost all keys contain dummy isogenies, and we can relax the requirement that none of the steps fail, as failing dummy isogenies do not impact the curves passed afterwards.

Example 7.3. Let $B_1 = \{3, 5\}$ with $M_1 = 6$ as in CTIDH-511. Say we want to detect some step in the eighth round of some B_i for $i > 1$; it is not relevant in which of the seven former rounds the six B_1-isogenies are computed, and thus we can effectively allow for one point rejection in these rounds. This effect becomes more beneficial when dummy isogenies are involved. For example, if three of these six B_1-isogenies are dummies, we only need the three actual B_1-isogenies to be computed within the first seven rounds. Furthermore, after detecting the first dummy B_1-isogeny, we do not need to attack further B_1-isogenies as explained above, and therefore save significant attack effort.

Remark 7.6. The generic attack requires that all first k steps succeed. This is not optimal: Assuming that some steps fail increases the probability of success of passing over E_0. For example, to attack isogenies in the sixth round and knowing that $e_{1,1} = 5$, it is better to assume that one or two out of these five fail and will be performed after the $\ell_{i,j}$-isogeny we want to detect, than it is to assume that all five of these succeed in the first five rounds. This improves the success probability of passing over E_0 per measurement, but makes the analysis of the required number of measurements harder to carry out. Furthermore, this optimal approach highly depends on the respective private key. We therefore do not pursue this approach in our simulations. A concrete practical attack against a single private key that uses this improved strategy should require a smaller number of measurements.

Remark 7.7. For CTIDH with large parameters, one would expect more large ℓ_i and fewer isogenies of low degrees, relative to CTIDH-511. This improves the performance of the attack, as the probability of a full-torsion path increases, and so we expect more measurements to pass over E_0. However, the details of such an attack are highly dependent on the implementation of a large-parameter CTIDH scheme. As we know of no such implementation, we do not analyze such a hypothetical implementation in detail.

Remark 7.8. At a certain point, it might be useful to stop the attack, and compute the remaining key elements via a simple meet-in-the-middle search. Especially for later bits, if some dummy isogenies have been detected and most of the key elements $e_{i,j}$ are already known, performing a brute-force attack may be faster than this side-channel attack.

7.5 Recovering SIKE keys with side-channel leakage of E_6

We now apply the same strategy from Section 7.4 to SIKE. In this whole section, we focus on recovering Bob's static key $\mathsf{sk_B}$ by showing side-channel leakage in Derive_B, used in Decaps. In general, the idea would apply as well to recover Alice's key $\mathsf{sk_A}$ in static SIDH or SIKE with swapped roles, as we do not use any specific structure of 3-isogenies. One can easily verify that the attack generalizes to SIDH based on ℓ_A or ℓ_B-isogenies for *any* ℓ_A, ℓ_B. We repeat many of the general ideas from Section 7.4, with some small differences as SIKE operates in isogeny graphs over $\mathbb{F}_{p^2}$ instead of $\mathbb{F}_p$. Fortunately, these differences make the attack *easier*.

Detecting E_6.

As remarked in Section 12.3, for both representations used in SIDH and SIKE, the curve E_6 is represented as $(8C : 4C)$, with $C = \alpha + \beta i \in \mathbb{F}_{p^2}$ non-zero. Similar to the CSIDH situation, whenever 4α or 4β is smaller than $p/2$, doubling $4C$ does not require a modular reduction for these values, and hence the bit representation of 8α resp. 8β of $8C$ is precisely a bit shift of 4α resp. 4β of $4C$ by one bit to the left. Such strongly-correlated values can be observed in several ways using side-channel analysis, as we detail later in section 7.6. Different from the CSIDH situation are the following key observations:

- The prime used in SIKE is of the form $p = 2^{e_A} \cdot 3^{e_B} - 1$. As observed in Remark 7.3, this large cofactor 2^{e_A} in $p+1$ implies a modular reduction does *not* affect the lowest $e_A - 1$ bits, except for the shift. Hence, even when 4α or 4β is larger than $p/2$, we see strong correlation between their lowest bits.
- C is now an $\mathbb{F}_{p^2}$ value, so we get strong correlation between 8α and 4α *and* between 8β and 4β. This implies at least $2 \cdot (e_A - 1)$ strongly-correlated bits in the worst case (25 %), up to $2 \cdot (\log_2(p) - 1)$ strongly-correlated bits in the best case (25%).

For random curves E_a, the representations of a are indistinguishable from random $(\alpha + \beta i : \gamma + \delta i)$, and so an attacker can differentiate E_6 from such curves. From this, an attacker only needs a few traces to determine accurately whether a walk passes over E_6 or not, as discussed in Section 7.6.

General approach to recover the k-th trit.

Assuming we know the first $k-1$ trits sk_i of a secret key sk, i.e. $\mathsf{sk}_{<k-1} = \sum_{i=0}^{k-2} \mathsf{sk}_i \cdot 3^i$, we want to find $\mathsf{sk}_{k-1} \in \{0,1,2\}$. We construct three candidate secret keys, $\mathsf{sk}^{(0)}, \mathsf{sk}^{(1)}, \mathsf{sk}^{(2)}$ as

$$\mathsf{sk}^{(0)} = \mathsf{sk}_{<k-1} + 0 \cdot 3^{k-1}, \quad \mathsf{sk}^{(1)} = \mathsf{sk}_{<k-1} + 1 \cdot 3^{k-1}, \quad \mathsf{sk}^{(2)} = \mathsf{sk}_{<k-1} + 2 \cdot 3^{k-1}.$$

We must have $\mathsf{sk}_{<k} = \mathsf{sk}^{(i)}$ for some $i \in \{0,1,2\}$. Thus, we use these three keys to construct (see Lemma 7.6) three public keys $\mathsf{pk}^{(0)}, \mathsf{pk}^{(1)}, \mathsf{pk}^{(2)}$ such that $\mathsf{Derive}_B(\mathsf{sk}^{(i)}, \mathsf{pk}^{(i)})$ computes E_6. When we feed these three keys to Bob, the computation $\mathsf{Derive}_B(\mathsf{sk}, \mathsf{pk}^{(i)})$ will then pass over E_6 in the k-th step *if and only if* $\mathsf{sk}_{k-1} = i$. By observing E_6 from side-channel information, we find sk_{k-1}.

In this attack scenario, another key observation makes the attack on SIDH and SIKE easier than the attack on CSIDH: The computation $\mathsf{Derive}_B(\mathsf{sk}, \mathsf{pk}^{(i)})$ *always* passes over the same curves, as there are no "steps that can fail" as in CSIDH. We know *with certainty* that Bob will pass over E_6 in step k in precisely one of these three computations. Hence, the number of traces required reduces drastically, as we do not need to worry about probabilities, such as $p_{\mathfrak{a}_k}$, that we have for CSIDH.

Constructing $\mathsf{pk}^{(i)}$ from $\mathsf{sk}^{(i)}$ using backtracking.

Whereas in CSIDH it is trivial to compute a curve E_{PK} such that $\mathfrak{a} * E_{\mathsf{PK}}$ passes over E_0 in the k-th step (see Lemma 7.4), in SIDH and SIKE it is not immediatly

clear how to construct $\mathsf{pk}^{(i)}$ for $\mathsf{sk}^{(i)}$. We follow [2, § 3.3], using *backtracking* to construct such a pk.[3] The main idea is that any $\mathsf{sk}_{<k}$ corresponds to some *kernel point* R_B of order 3^k for some k, so to an *isogeny* $\varphi^{(k)} : E_6 \to E^{(k)}$. Here, the trits sk_i determine the steps

$$E_6 = E^{(0)} \xrightarrow{\mathsf{sk}_0} E^{(1)} \xrightarrow{\mathsf{sk}_1} \dots \xrightarrow{\mathsf{sk}_{k-1}} E^{(k)}.$$

The dual isogeny $\hat{\varphi}^{(k)} : E^{(k)} \to E_6$ then corresponds to the kernel generator $\varphi^{(k)}([3^{e_B-k}]Q_B)$ (see [**NR19**]). This leads to [2, Lemma 2].

Lemma 7.6 ([2])**.** Let sk be a secret key, and let $R_k = [3^{e_B-k}](P_B + [\mathsf{sk}_{<k}]Q_B)$ so that $\varphi : E_6 \to E^{(k)}$ is the corresponding isogeny for the first k steps. Let $T \in E^{(k)}[3^{e_B}]$ such that $[3^{e_B-k}]T \neq \pm[3^{e_B-k}]\varphi(Q_B)$. Then

$$\mathsf{pk}' = (\varphi(Q_B) + [\mathsf{sk}_{<k}]T,\ -T,\ \varphi(Q_B) + [\mathsf{sk}_{<k} - 1]T)$$

is such that $\mathsf{Derive}_B(\mathsf{sk}, \mathsf{pk}')$ passes over E_6 in the k-th step.

It is necessary that such a pk' is not rejected by a SIKE implementation.

Corollary 7.4 ([2])**.** The points P' and Q' for a $\mathsf{pk}' = (P', Q', Q' - P')$ as constructed in Lemma 7.6 form a basis for the 3^{e_B}-torsion of $E^{(k)}$. This implies they are of order 3^{e_B} and pass the CLN test.

Given Lemma 7.6 and $\mathsf{sk}_{<k-1}$, we can therefore easily compute the $\mathsf{pk}^{(i)}$ corresponding to $\mathsf{sk}^{(i)}$ for $i \in \{0, 1, 2\}$. One of the three attempts $\mathsf{Derive}_B(\mathsf{sk}, \mathsf{pk}^{(i)})$ will then pass over E_6 in the k-th step, while the other two will not. Only the representation of E_6 by $(8C : 4C)$ is then strongly-correlated, and by detecting this representation using side-channel information, we recover sk_{k-1}.

Remark 7.9. A straightforward attack computes $\mathsf{pk}^{(0)}, \mathsf{pk}^{(1)}$ and $\mathsf{pk}^{(2)}$, and feeds all three to Bob, and so requires 3 traces to recover a single trit sk_{k-1}. Clearly, when we already detect E_6 in the trace of $\mathsf{Derive}_B(\mathsf{sk}, \mathsf{pk}^{(0)})$, we do not need the traces of $\mathsf{pk}^{(1)}$ and $\mathsf{pk}^{(2)}$, similarly for $\mathsf{Derive}_B(\mathsf{sk}, \mathsf{pk}^{(1)})$. This approach would require on average $\frac{1}{3} \cdot 1 + \frac{1}{3} \cdot 2 + \frac{1}{3} \cdot 3 = 2$ traces per trit. We can do even better: If we do not detect E_6 in both $\mathsf{Derive}_B(\mathsf{sk}, \mathsf{pk}^{(0)})$ and $\mathsf{Derive}_B(\mathsf{sk}, \mathsf{pk}^{(1)})$, we do not need a sample for $\mathsf{Derive}_B(\mathsf{sk}, \mathsf{pk}^{(2)})$, as sk_{k-1} *must* equal 2. This gives $\frac{5}{3}$ samples per trit, giving a total of $\frac{5}{3} \cdot e_B$ traces.

[3] It is important that such a $\mathsf{pk} = (P, Q, Q - P)$ passes the *CLN test* [108]: P and Q are both of order 3^{e_B} and $[3^{e_B-1}]P \neq [\pm 3^{e_B-1}]Q$, so that they generate $E[3^{e_B}]$.

7.6 Feasibility of obtaining the side-channel information

In this section, we discuss the practical feasibility of obtaining the required side-channel information.

Zero-value representations.

For zero-value representations as in CTIDH, where E_0 is represented by $(0 : C)$ in Montgomery form, we exploit side-channel analysis methods to distinguish between the zero curve and others. In particular, as shown in [143], one can apply Welch's t-test [257] to extract the required information from the power consumption of the attacked device. Further, as mentioned in [143], one can use correlation-collision SCA methods to identify zero values using multiple measurements. Therefore, the attack scheme as demonstrated in [143] to SIKE can analogously be applied whenever zero-value representations occur.

Strongly-correlated representations.

The attacks presented in Sections 7.4.2 and 7.5 for implementations using strongly-correlated representations, such as SQALE and SIKE, are more challenging in practice, since no zero values occur. A naïve approach to mount the proposed attack for such instances would be to apply side-channel attacks like CPA or DPA, and estimate or guess the values of intermediate codomain curves. Revealing those intermediate values would require a fitting power model and a sufficiently high signal-to-noise ratio (SNR[4]). By exploiting the pattern similarity in the strongly-correlated representation $(2C : 4C)$ for SQALE or $(8C : 4C)$ for SIKE, as mentioned in Section 7.4.1 and Section 7.5, we reduce the SNR required to successfully perform the attack. To achieve this, we apply the concept of correlation-collision attacks, so that there is no need to reveal the actual value of C via a sophisticated power model.

We exploit side-channel correlation-collision attacks [221] to find similar values by searching for strongly-correlated patterns versus non-correlated patterns. Instead of measuring *multiple* computations to identify similar or identical patterns, as in [221], we apply the concept of a horizontal side-channel attack as in [227]. That is, we extract the required side-channel information from a *single* segmented power trace. Such a segmented power trace contains the power

[4]SNR is the ratio between the variance of the signal and the variance of noise. Too small SNR values make information and noise indistinguishable.

values of the processed limbs (each limb is 64 bits), required to represent $\mathbb{F}_p$-values, which form a fingerprint characteristic of such a value. These fingerprints then serve as input to calculate the correlation between $2C$ and $4C$ for SQALE, or $4\alpha, 4\beta, 8\alpha$ and 8β for SIKE, from which we judge their similarity. For strongly-correlated representations of E_0 and E_6, this gives a higher correlation between the fingerprints than for representations of random curves E_a as either $(A + 2C : 4C)$ or $(A + 2C : A - 2C)$, with $A, C \neq 0$.

For both CSIDH attacks, we assume no point rejections prior to the respective isogeny computation, so that the specific isogeny steps are known in advance. For SIKE, there are no such probabilities involved in the isogeny computation, and so here too the specific isogeny steps are known in advance. This implies that an attacker will know where the values of interest are computed and used within the power trace, and can distinguish the relevant information from the rest of the trace. Thus, in all cases, the points of interest (position of the limbs) within the power trace are known in advance, and segmenting each power trace into vectors of the corresponding processed limbs for mounting the correlation-collision attack is easy.

7.7 Simulating the attacks on SQALE, CTIDH and SIKE

To demonstrate the proposed attacks, we implemented Python (version 3.8.10) simulations for our CTIDH-511[5] and SQALE-2048[6] attacks, and a C simulation of the attack on SIKE.[7] The C code for key generation and collecting the simulated power consumption were compiled with gcc (version 9.4.0). Security-critical spots of the attacked C code remained unchanged in both cases.

For the SQALE and CTIDH attacks, the implemented simulation works as follows: First, we generate the corresponding public keys $E_{\mathtt{PK}}$ and $\tilde{E}_{\mathtt{PK}}$ for the current k-th step, as described in Section 7.4.1. Then we collect the bit values of the resulting codomain curve after the computation of the k-th step $E \leftarrow \mathfrak{l}^{(k)} * E$ in the group action $\mathfrak{a} * E_{\mathtt{PK}}$ resp. $\mathfrak{a} * \tilde{E}_{\mathtt{PK}}$ to simulate the power consumption.

We calculate the Hamming weight of these values and add a zero-mean Gaussian standard distribution to simulate noise in the measurement. We picked different values of the standard deviation to mimic realistic power measurements

[5] http://ctidh.isogeny.org/high-ctidh-20210523.tar.gz
[6] https://github.com/JJChiDguez/sqale-csidh-velusqrt, commit a95812f
[7] https://github.com/Microsoft/PQCrypto-SIDH, commit ecf93e9

with different SNR values. By varying the SNR in such a way, we can determine up to which SNR the attacks are successful, and compare this to known SNR values achieved in physical attacks. For SQALE and CTIDH, we are only interested in power traces passing over E_0, and so we need the first k steps to succeed. We therefore take enough samples to ensure high probability that passing over E_0 happens multiple times for either E_{PK} or $\tilde{E_{\mathsf{PK}}}$. Finally, based on the set of collected bit vectors for all these samples, we decide on which of the two cases contains paths over E_0, and therefore reveal the k-th bit of the secret key.

For the SIKE attack, we generate $\mathsf{pk}^{(0)}, \mathsf{pk}^{(1)}$ and $\mathsf{pk}^{(2)}$ for the current k-th step, as described in Section 7.5, and collect the bit values of the resulting codomain curve in the computation of the k-th step of Derive_B in Decaps. For simplicity, there is no noise in the simulation, as the results are exactly the same as for the SQALE situation after extracting the bit values. Deciding which sample has strongly-correlated values is easy, as is clear from Figure 7.3.

As described in previous sections, due to the different representations, the decision step differs between CTIDH and SQALE. For SIKE, the probability to pass over E_6 is 100%, and so a single sample per $\mathsf{pk}^{(i)}$ is enough to decide what the k-th trit sk_{k-1} is.

In order to reduce the running time of our simulations for SQALE and CTIDH, we terminate each group action run after returning the required bit values of the k-th step. Furthermore, we implemented a threaded version so that we collect several runs in parallel, which speeds up the simulation. All experiments were measured on AMD EPYC 7643 CPU cores.

Attacking CTIDH-511.

As shown in [143, § 4] a practical differentiation between zero and non-zero values, even with low SNR, is feasible with a single trace containing the zero value. Hence, in CTIDH, where E_0 is represented by $(0 : C)$, a single occurrence of E_0 leaks enough information for the decision in each step. Thus, the number of required attempts can be calculated as follows: Given $p_{\mathfrak{a}_k}$ from Lemma 7.5, the success probability of having *at least one* sequence that passes over E_0 in the k-th step in t_k attempts is $P_{exp}(X \geq 1) = 1-(1-p_{\mathfrak{a}_k})^{t_k}$. We can calculate t_k to achieve an expected success rate P_{exp} by $t_k = \log_{(1-p_{\mathfrak{a}_k})}(1-P_{exp})$. For CTIDH-511, to achieve $P_{exp} \geq 99\%$ for all k, we get an estimate of $\sum t_k \approx 130,000$ attempts for full key recovery. In simulations, the required number of attempts for full key recovery was $\approx 85,000$ on average over 100 experiments, due to effects mentioned in Section 7.4.3. The average execution time was about 35

minutes (single core) or 5 minutes (120 threads). As described in Remark 7.8, finding the last few key bits by brute force drastically reduces the required measurements, as $p_{\mathtt{a}_k}$ is low.

Attacking SQALE-2048.

In this case, we simulate a correlation-collision attack as described in Section 7.6: We calculate the correlation between the 64-bit limbs that represent the $\mathbb{F}_p$-values, and apply the standard Hamming-weight model with noise drawn from a normal distribution. Even with an SNR as low as 1.40, strongly-correlated representations leak enough information to guess the k-th bit, as can be seen in Figure 7.3 for $k = 1$. Both without noise and with SNR 1.40, we are able to determine the right bit in 74% of measurements (where 75% is the theoretical optimum, as $2C \leq \frac{p}{2}$ only half the time). An SNR value of 1.40 is considered *low*: The SNR value of a common embedded device, using a measurement script[8] provided by the ChipWhisperer framework for a ChipWhisperer-Lite board with an ARM Cortex-M4 target, obtains an SNR of 8.90. Figure 7.4 shows the success rate for different values. We evaluated the following methods for decision-making:

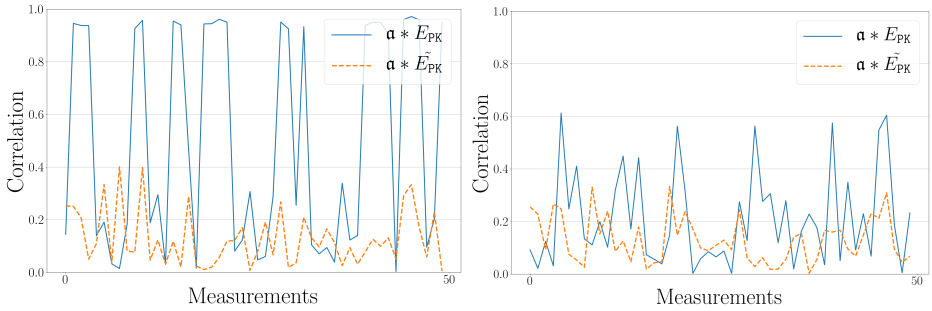
- Decide based on the number of cases with a higher resulting correlation, as exemplified in Figure 7.3.
- Decide based on the sum of the resulting correlations for each case, to reduce the number of attempts required for a given success rate.

Empirical results show that the sum-based approach reduces the required number of attempts for key recovery by a factor 3 on average (from $\approx 24,819$ to $\approx 8,273$), which leads to an average execution time of 35 minutes (120 threads).

Attacking SIKE.

For SIKE, the analysis after collecting the bit values is similar to that of the SQALE case, and hence the results from Figure 7.3 apply to these simulated sampels too. Furthermore, for SIKE we have the advantage that **i)** we know that one of the three samples per trit must be an E_6-sample, **ii)** we know that even with modular reduction, there is strong correlation between the lowest limbs and **iii)** we can use both $\mathbb{F}_p$-values α and β for $C = \alpha + \beta i \in \mathbb{F}_{p^2}$.

[8] https://github.com/newaetech/chipwhisperer-jupyter/blob/master/archive/PA_Intro_3-Measuring_SNR_of_Target.ipynb, commit 44112f6



(a) Correlation results without noise. (b) Correlation results with SNR of 1.40.

Figure 7.3: Experimental results to discover bit $k = 1$: the correct hypothesis ($\mathfrak{a} * E_{\mathsf{PK}}$) in blue and the wrong hypothesis ($\mathfrak{a} * \tilde{E_{\mathsf{PK}}}$) in orange, for SQALE-2048.

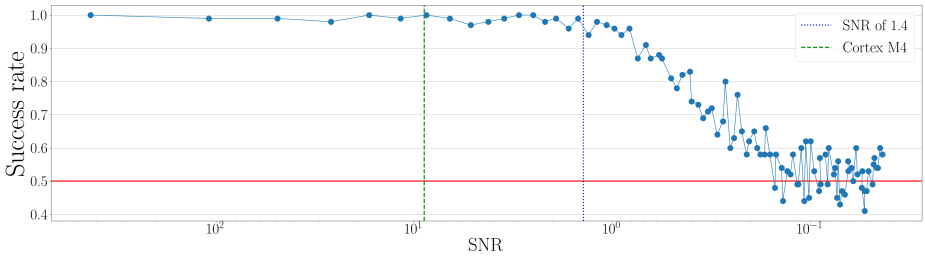


Figure 7.4: Relation between SNR and success rate. Rate of 0.5 equals random guess.

As explained in Section 7.5, we need on average 5/3 samples per trit to find sk_i, for all e_B trits. For SIKEp434, this gives an average of 228 samples to recover sk_B. The average running time over 100 evaluations in each case was $\approx$ 4 seconds for SIKEp434, $\approx$ 8 seconds for SIKEp503, $\approx$ 17 seconds for SIKEp610, and $\approx$ 42 seconds for SIKEp751 respectively.

7.8 Countermeasures and conclusion

We have shown that both CSIDH and SIKE are vulnerable to leakage of specific curves. For CSIDH, we have shown that both Montgomery form and alternative Montgomery form leak secret information when passing over E_0, and for SIKE we have shown that in both forms, the representation of E_6 by $(8C : 4C)$ leaks secret information. As described in Section 7.6, zero-value representations

Scheme	SQALE-2048	CTIDH-511	SIKEp434	SIKEp503	SIKEp610	SIKEp751
Samples	8,273	85,000	228	265	320	398

Table 7.1: Required number of samples to reconstruct secret key in simulations.

are easiest to detect, and accordingly one should prefer the alternative Montgomery form over the Montgomery form throughout the whole computation for CSIDH variants. However, more effective countermeasures are required to avoid strongly-correlated representations in CSIDH and SIKE.

7.8.1 Public key validation

As mentioned in Section 7.4, public keys in the proposed attacks on CSIDH variants consist of valid supersingular elliptic curves. Hence, the attack cannot be prevented by public key validation.

For the SIKE attack, the situation is different: Instead of containing valid points $(\varphi_A(P_B), \varphi_A(Q_B), \varphi_A(Q_B - P_B))$ (see Section 12.3), we construct public keys differently, as described in Lemma 7.6. However, such public key points are not detected by partial validation methods contained in the current SIKE software, such as the CLN test (see Corollary 7.4). In general, the full validation of SIDH or SIKE public keys is believed to be as hard as breaking the schemes themselves [**GV18**]. It remains an open question if there is an efficient partial validation method to detect the specific public key points generated by our attack.

7.8.2 Avoiding E_0 or E_6

A straightforward way of mitigating the attacks is to avoid paths that lead over E_0 or E_6, or any other vulnerable curve. As argued in [2, 143], avoiding them altogether seems difficult. We discuss techniques to achieve this.

Danger zone.

Similar to the rejection proposal from [2], it may appear intuitive to define a certain *danger zone* around vulnerable curves, e.g. for CSIDH, containing all curves $\mathfrak{l}_i^{\pm 1} * E_0$ for $1 \leq i \leq n$, and abort the execution of the protocol whenever an isogeny path enters this zone. In the SIKE attack, this zone could include the four curves that are 3-isogenous to E_6. However, the attacker can simply

construct public keys that would or would not pass through this zone, and observe that the protocol aborts or proceeds. This leaks the same information as in the attack targeting only E_0 or E_6.[9]

Masking on isogeny level.

One can fully bypass this danger zone by masking by a (small) isogeny before applying secret isogeny walks (see [2, 191]). For CSIDH, for a masking isogeny $\mathfrak{z}$ and a secret $\mathfrak{a}$ we have that by commutativity, $\mathfrak{a} * E = \mathfrak{z}^{-1} * (\mathfrak{a} * (\mathfrak{z} * E))$, so this route avoids the danger zone when $\mathfrak{z}$ is sufficiently large. Drawing $\mathfrak{z}$ from a masking key space of k bits would require the attacker to guess the random ephemeral mask correctly in order to get a successful walk over E_0, which happens with probability 2^{-k}. Thus, a k-bit mask increases the number of samples needed by 2^k. Similarly, as detailed in [2], the secret isogeny in the SIKE attack can be masked by a 2^k-isogeny, where keeping track of the dual requires some extra cost. Although masking comes at a significant cost if the masking isogeny needs to be large, this appears to be the only known effective countermeasure that fully avoids the proposed attacks.

Randomization of order (CSIDH).

For CSIDH variants, intuitively, randomizing the order of isogenies, and as proposed in [196] the order of real and dummy isogenies, might seem beneficial to achieve this. However, we can then simply *always* attack the first step of the isogeny path, with a success probability of $1/n$. With enough repetitions, we can therefore statistically guess the secret key, where the exact success probabilities highly depend on the respective CSIDH variant. This countermeasure also significantly impacts performance, making it undesirable.

Working on the surface (CSIDH).

[10] An interesting approach to avoid vulnerable curves, specific to CSIDH, is to move to the *surface* of the isogeny graph. That is, we use curves E_A with $\mathbb{F}_p$-rational endomorphism ring $\mathbb{Z}[\frac{1+\pi}{2}]$ instead of $\mathbb{Z}[\pi]$, and use a prime $p = 7 \mod 8$. This idea was proposed in [**csurf**] and dubbed CSURF. We can still work with elliptic curves in Montgomery form, although Montgomery coefficients a are not unique in this setting. However, when following the setup described

[9] Pun aficionados may wish to dub this scenario the *highway to the danger zone*.

[10] We thank the anonymous reviewers of SAC 2022 for this suggestion.

in [**C21**], we are not aware of any vulnerable curves on the surface, but it seems difficult to prove that vulnerable curves do not exist there. More analysis is necessary to rule out such curves. Nevertheless, working on the surface offers other benefits, and we see no reason to work on the floor with known vulnerabilities, instead of on the surface.

Precomposition in SIKE.

A potential countermeasure specific to SIKE is precomposing with a random isomorphism, as proposed in [191]. In our attack scenario, the isogeny walk then passes a curve isomorphic to E_6 instead of E_6, which may eliminate the leakage. However, as discussed in [**CLNRV20**], each isomorphism class contains exactly six Montgomery curves, and the isomorphism class of E_6 also contains E_{-6}, which shares the same vulnerability as E_6. Thus, in 1/3 of cases, leakage still occurs, only moderately increasing the number of required measurements. On the other hand, finding an isomorphism that guarantees the isogeny walk not to pass E_6 or E_{-6} only from public key information seems infeasible. Furthermore, the computation of isomorphisms usually contains expensive square root computations.

7.8.3 Avoiding correlations

Another approach to mitigate the attacks is to ensure that vulnerable curves such as E_0 and E_6 do not leak information when passing over them. This requires adapting the representations of such curves.

Avoiding correlations in alternative Montgomery form.

As noted for CSIDH variants, the representation $(2C : 4C)$ leaks secret information whenever $2C < \frac{p}{2}$. In order to avoid this, we can try to represent the alternative Montgomery form $(A + 2C : 4C)$ differently and use a *flipped* alternative Montgomery form $(A + 2C : -4C)$ instead, which we write as ɐlʇǝɹuɐʇıʌǝ Montgomery form for brevity. In the case of E_0, this means that the coefficients $2C$ and $-4C$ are *not* simple shifts of each other for $2C < \frac{p}{2}$, which prevents the correlation attack. In order to still achieve constant-time behavior, we should flip $4C$ for all curves, since otherwise E_0 would easily be detectable via side channels. The correctness of computations can be guaranteed by corresponding sign flips in computations that would normally include $4C$. Analogously, we can define a flipped representation of curves in SIKE. Although the ɐlʇǝɹuɐʇıʌǝ

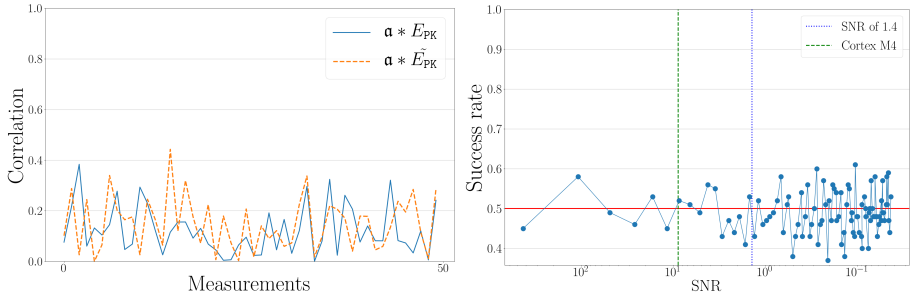
Montgomery form is effective in preventing leakage of E_0, it creates other vulnerable curves. We discuss this in more detail in Section 7.A. It remains an open question to find a representation without both zero-value representations and strongly-correlated representations.

Masking a single value.

Assuming we are working with the representations $(A_{24}^+ : A_{24}^-)$ or $(A_{24}^+ : C_{24})$ for either CSIDH or SIKE, masking is non-trivial, as it needs to respect the ratio A/C during the computation. However, it is possible to multiply by some random α during the computation of A_{24}^+, and to multiply by $1/\alpha$ in the next computations that use A_{24}^+. This requires a careful analysis and implementation, in order to guarantee that no leak of A_{24}^+ or some different correlation occurs at a given point in the computation.

7.A Flipping $4C$ as a countermeasure.

In this appendix, we discuss the effectiveness of ɐlʇɐɹuɐʇıʌǝ Montgomery form as a countermeasure. As Figure 7.5 shows, the countermeasure prevents detection of $(2C : 4C)$, and therefore prevents leakage on E_0. Similar techniques can also be applied for other strongly-correlated representations, such as those for E_6.



(a) Correlation results with countermeasures (without noise).

(b) SNR/Success rate with countermeasures.

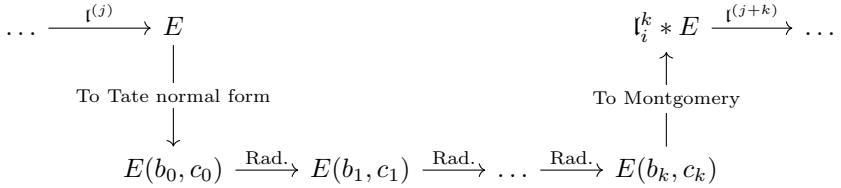
Figure 7.5: Correlation values including the countermeasures leak no information.

Nevertheless, ɐlʇɐɹuɐʇıʌǝ Montgomery form creates new problems. In alternative Montgomery form, the curve E_{-6}, a valid supersingular curve for most CSIDH-primes, is represented using $(-4C : 4C)$, which is not strongly-correlated. However, by switching to ɐlʇɐɹuɐʇıʌǝ Montgomery form, we get the strongly-correlated representation $(-4C : -4C)$. The attack as described in the paper is then applicable by replacing E_0 by E_{-6}.

It seems difficult to discover if a curve E_a will leak information before computing the next step: doing so would require knowledge on a, and so requires a representation of a in some form. Hence, we cannot decide to use either alternative Montgomery form or ɐlʇɐɹuɐʇıʌǝ Montgomery form before we compute the actual curve.

7.B CSIDH implementations using radical isogenies

In [83] an alternative method to compute the action of $\mathfrak{a}$ is proposed, using *radical isogenies*. The evaluation of $\mathfrak{a} * E$ is still an ordered evaluation $\mathfrak{l}^{(n)} * \ldots * \mathfrak{l}^{(1)} * E$, due to a change in the evaluation algorithm we get chains $\mathfrak{l}_i * \ldots * \mathfrak{l}_i$ of specific degrees $\ell_i \in \{4, 5, 7, 9, 11, 13\}$ in the evaluation. These chains are computed on a different curve form, namely the Tate normal form for that specific degree ℓ_i, instead of as steps between Montgomery curves.



The Tate normal form for a degree ℓ_i in general requires two coefficients $b, c \in \mathbb{F}_p$ instead of the single Montgomery coefficient $a \in \mathbb{F}_p$, and the radical isogeny computes b', c' associated to $\mathfrak{l}_i * E_a$.

In an efficient implementation, both b and c would be represented in projective coordinates. We know of only one such implementation, given in [**JesusAndKrijn**]. We sketch two attack approaches to extend the proposed attacks to such an implementation:

1. Find a Tate normal curve of degree ℓ_i such that either b or c has a strongly-correlated representation. The generic adaptive attack then works exactly the same.

2. Find the length of the chain by feeding a curve $E_{\mathtt{PK}}$ such that we map back to E_0 when we map back to Montgomery form at the end of the chain. This requires feeding several different $E_{\mathtt{PK}j}$ representing several different lengths of chains.

Note that the attack becomes easier when using radical isogenies: these chains are computationally very distinct from ordinary isogeny evaluations, and so we only need to discover the length of the chain. Furthermore, radical isogenies are performed for low degrees (up to 13), which implies that we do not perform these degrees in the rest of the steps $\mathfrak{l}^{(j)}$. This increases $p_{\mathfrak{a}_k}$ substantially.

Chapter 8

Disorientation faults

8.1 Placeholder

The paper will go here.

8.2 Introduction

Isogeny-based cryptography is a contender in the ongoing quest for post-quantum cryptography. Perhaps the most attractive feature is small key size, but there are other reasons in favor of isogenies: Some functionalities appear difficult to construct from other paradigms. For instance, the *CSIDH* [**CLM+18**] scheme gives rise to non-interactive key exchange. CSIDH uses the action of an ideal-class group on a set of elliptic curves to mimic (some) classical constructions based on discrete logarithms, most notably the Diffie–Hellman key exchange. Recently, more advanced cryptographic protocols have been proposed based on the CSIDH group action: the signature schemes SeaSign [118] and CSI-FiSh [53], threshold schemes [123], oblivious transfer [193], and more. The main drawback of isogeny-based cryptography is speed: CSIDH takes hundreds of times longer to complete a key exchange than pre-quantum elliptic-curve cryptography (ECC).

The group action in CSIDH and related schemes is evaluated by computing a sequence of small-degree *isogeny steps*; the choice of degrees and "directions" is the private key. Thus, the control flow of a straightforward implementation is directly related to the secret key, which complicates side-channel resistant

implementations [**MR18**, **BBCCLMSS21**, 47, 71, 173, 212].

In a side-channel attack, passive observations of physical leakage (such as timing differences, electromagnetic emissions, or power consumption) during the execution of sensitive computations help an attacker infer secret information. A more intrusive class of physical attacks are *fault-injection attacks* or *fault attacks*: By actively manipulating the execution environment of a device (for instance, by altering the characteristics of the power supply, or by exposing the device to electromagnetic radiation), the attacker aims to trigger an error during the execution of sensitive computations and later infer secret information from the outputs, which are now potentially incorrect, i.e., *faulty*.

Two major classes of faults are *instruction skips* and *variable modifications*. Well-timed skips of processor instructions can have far-reaching consequences, e.g., omitting a security check entirely, or failing to erase secrets which subsequently leak into the output. Variable modifications may reach from simply randomized CPU registers to precisely targeted single-bit flips. They cause the software to operate on unexpected values, which (especially in a cryptographic context) may lead to exploitable behavior. In practice, the difficulty of injecting a particular kind of fault (or a combination of multiple faults) depends on various parameters; generally speaking, less targeted faults are easier.

Our contributions. We analyze the behavior of existing CSIDH implementations under a new class of attacks that we call *disorientation faults*. These faults occur when the attacker confuses the algorithm about the *orientation* of a point used during the computation: The effect of such an error is that a subset of the secret-dependent isogeny steps will be performed in the opposite direction, resulting in an incorrect output curve.

The placement of the disorientation fault during the algorithm influences the distribution of the output curve in a key-dependent manner. We explain how an attacker can post-process a set of faulty outputs to fully recover the private key. This attack works against almost all existing CSIDH implementations.

To simplify exposition we first assume access to a device that applies a secret key to a given public key (i.e., computing the shared key in CSIDH) and returns the result (e.g., a hardware security module providing a CSIDH accelerator). We also discuss variants of the attack with weaker access; this includes a *hashed* version where faulty outputs are not revealed as-is, but passed through a key-derivation function first, as is commonly done for a Diffie–Hellman-style key exchange, and made available to the attacker only indirectly, e.g., as a MAC under the derived key.

Part of the tooling for the post-processing stage of our attack is a some-

what optimized meet-in-the-middle *path-finding* program for the CSIDH isogeny graph, dubbed `pubcrawl`. This software is intentionally kept fully generic with no restrictions specific to the fault-attack scenario we are considering, so that it may hopefully be usable out of the box for other applications requiring "small" neighborhood searches in CSIDH in the future. Applying expensive but feasible precomputation can speed up post-processing for all attack variants and is particularly beneficial to the hashed version of the attack.

To defend against disorientation faults, we provide a set of *countermeasures*. We show different forms of protecting an implementation and discuss the pros and cons of each of the methods. In the end, we detail two of the protections that we believe give the best security. Both of them are lightweight, and they do not significantly add to the complexity of the implementation.

Note on security.

We emphasize that CSIDH, its variants, and the protocols based on the CSIDH group action are not affected by the recent attacks that break the isogeny-based scheme SIDH [**cryptoeprint:2022/1038**, 78, 202]. These attacks exploit specific auxiliary information which is revealed in SIDH but does not exist in CSIDH.

CSIDH is a relatively young cryptosystem, being introduced only in 2018, but it is based on older systems due to Couveignes [**couveignes:hhs**] and Rostovtsev and Stolbunov [**rostovtsev-stolbunov**] which have received attention since 2006. The best non-quantum attack is a meet-in-the-middle attack running in $O(\sqrt[4]{p})$; a low-memory version was developed in [125]. On a large quantum computer Kuperberg's attack can be mounted as shown in [97]. This attack runs in $L_{\sqrt{p}}(1/2)$ calls to a quantum oracle. The number of oracle calls was further analyzed in [**2020/bonnetain**] and [**2020/peikert**] for concrete parameters, while [47] analyzes the costs per oracle call in number of quantum operations. Combining these results shows that breaking CSIDH-512 requires around 2^{60} qubit operations on logical qubits, i.e., not taking into account the overhead for quantum error correction. Implementation papers such as CTIDH [**BBCCLMSS21**] use the CSIDH-512 prime for comparison purposes and also offer larger parameters. Likewise, we use the CSIDH-512 and CTIDH-512 parameters for concrete examples.

Related work. Prior works investigating fault attacks on isogeny-based cryptography mostly target specific variants or implementations of schemes and are different from our approach. *Loop-abort* faults on the SIDH cryptosys-

tem [**GelinW17**], discussed for CSIDH in [69], lead to leakage of an intermediate value of the computation rather than the final result. Replacing torsion points with other points in SIDH [278, 281] can be used to recover the secret keys; faulting intermediate curves in SIDH [**ACMR22**] to learn if secret isogeny paths lead over subfield curves can also leak information on secret keys. But the two latter attacks cannot be mounted against CSIDH due to the structural and mathematical differences between SIDH and CSIDH.

Recently, several CSIDH-specific fault attacks were published. One can modify memory locations and observe if this changes the resulting shared secret [**CamposKM21**]. A different attack avenue is to target fault injections against dummy computations in CSIDH [69, 196]. We emphasize that these are attacks against specific implementations and variants of CSIDH. Our work, in contrast, features a generic approach to fault attacks, exploiting an operation and data flow present in almost all current implementations of CSIDH.

8.3 Background

CSIDH is based on a group action on a certain set of elliptic curves. We explain the setup of CSIDH in Section 8.3.1 and relevant algorithmic aspects in Section 8.3.2. We assume some familiarity with elliptic curves and isogenies; the reader may consult [**CLM+18**] for more details.

8.3.1 CSIDH

We fix a prime p of the form $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ with distinct odd primes ℓ_i. We define $\mathcal{E}$ to be the set of supersingular elliptic curves over $\mathbb{F}_p$ in Montgomery form, up to $\mathbb{F}_p$-isomorphism. All such curves admit an equation of the form $E_A : y^2 = x^3 + Ax^2 + x$ with a unique $A \in \mathbb{F}_p$. For $E_A \in \mathcal{E}$, the group of rational points $E_A(\mathbb{F}_p)$ is cyclic of order $p + 1$. The quadratic twist of $E_A \in \mathcal{E}$ is $E_{-A} \in \mathcal{E}$.

Isogeny steps. For any ℓ_i and any $E_A \in \mathcal{E}$ there are two ℓ_i-isogenies, each leading to another curve in $\mathcal{E}$. One has kernel generated by any point P_+ of order ℓ_i with both coordinates in $\mathbb{F}_p$. We say this ℓ_i-isogeny is in the *positive direction* and the point P_+ has *positive orientation*. The other ℓ_i-isogeny has kernel generated by any point P_- of order ℓ_i with x-coordinate in $\mathbb{F}_p$ but y-coordinate in $\mathbb{F}_{p^2} \setminus \mathbb{F}_p$. We say this isogeny is in the *negative direction* and the point P_- has *negative orientation*. Replacing E_A by the codomain of a

positive and negative ℓ_i-isogeny from E_A is a *positive and negative ℓ_i-isogeny step*, respectively. As the name suggests, a positive and a negative ℓ_i-isogeny step cancel.

Fix $i \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$ with $i^2 = -1 \in \mathbb{F}_p$ and note that a negatively oriented point is necessarily of the form (x, iy) with $x, y \in \mathbb{F}_p$. Moreover, $x \in \mathbb{F}_p^*$ defines a positively oriented point on E_A whenever $x^3 + Ax^2 + x$ is a square in $\mathbb{F}_p$, and a negatively oriented point otherwise.

The group action. It is a non-obvious, but very useful fact that the isogeny steps defined above *commute*: Any sequence of them can be rearranged arbitrarily without changing the final codomain curve [**CLM+18**]. Thus, taking a combination of various isogeny steps defines a group action of the abelian group $(\mathbb{Z}^n, +)$ on $\mathcal{E}$: The vector $(e_1, \ldots, e_n) \in \mathbb{Z}^n$ represents $|e_i|$ individual ℓ_i-isogeny steps, with the sign of e_i specifying the orientation: if $\mathfrak{l}_i$ denotes a single positive ℓ_i-isogeny step, the action of $(e_1, \ldots, e_n) \in \mathbb{Z}^n$ on a curve E denotes the sequence of steps

$$(\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_n^{e_n}) * E\,.$$

We refer to $(e_1, \ldots, e_n)$ as an *exponent vector*.

8.3.2 Algorithmic aspects

Every step is an oriented isogeny, so applying a single $\mathfrak{l}_i^{\pm 1}$ step requires a point P with two properties: P has order ℓ_i and the right orientation. The codomain of $E \to E/\langle P \rangle$ is computed using either the Vélu [**V71**] or $\sqrt{\text{élu}}$ [43] formulas.

Determining orientations. All state-of-the-art implementations of CSIDH use x-only arithmetic and completely disregard y-coordinates. So, we sample a point P by sampling an x-coordinate in $\mathbb{F}_p$. To determine the orientation of P, we then find the field of definition of the y-coordinate, e.g., through a Legendre symbol computation. An alternative method is the "Elligator 2" map [44] which generates a point of the desired orientation.

Sampling order-ℓ points. There are several methods to compute points of given order ℓ. The following Las Vegas algorithm is popular for its simplicity and efficiency: As above, sample a uniformly random point P of either positive or negative orientation, and compute $Q := [(p+1)/\ell]P$. Since P is uniformly random in a cyclic group of order $p+1$, the point Q has order ℓ with probability $1 - 1/\ell$. With probability $1/\ell$, we get $Q = \infty$. Retry until $Q \neq \infty$. Filtering for

points of a given orientation is straightforward.

Multiple isogenies from a single point. To amortize the cost of sampling points and determining orientations, implementations usually pick some set S of indices of exponents of the same sign, and attempt to compute one isogeny per degree ℓ_i with $i \in S$ from one point. If $d = \prod_{i \in S} \ell_i$ and P a random point, then the point $Q = [\frac{p+1}{d}]P$ has order dividing d. If $[d/\ell_i]Q \neq \infty$ we can use it to construct an isogeny step for $\ell_i \in S$. The image of Q under that isogeny has the same orientation as P and Q and order dividing d/ℓ_i, so we continue with the next ℓ_j.

In CSIDH and its variants, the set S of isogeny degrees depends on the secret key and the orientation s of P. For example in Algorithm 4 (from [**CLM+18**]), for the first point that is sampled with positive orientation, the set S is $\{i \mid e_i > 0\}$.

Algorithm 4 Evaluation of CSIDH group action

Input: $A \in \mathbb{F}_p$ and a list of integers $(e_1, \ldots, e_n)$.
Output: $B \in \mathbb{F}_p$ such that $\prod [\mathfrak{l}_i]^{e_i} * E_A = E_B$

while some $e_i \neq 0$ **do**
  Sample a random $x \in \mathbb{F}_p$, defining a point P.
  Set $s \leftarrow \texttt{IsSquare}(x^3 + Ax^2 + x)$.
  Let $S = \{i \mid e_i \neq 0,\ \text{sign}(e_i) = s\}$. **Restart** with new x if S is empty.
  Let $k \leftarrow \prod_{i \in S} \ell_i$ and compute $Q \leftarrow [\frac{p+1}{k}]P$.
  for each $i \in S$ **do**
    Set $k \leftarrow k/\ell_i$ and compute $R \leftarrow [k]Q$. If $R = \infty$, **skip** this i.
    Compute $\varphi : E_A \to E_B$ with kernel $\langle R \rangle$.
    Set $A \leftarrow B$, $Q \leftarrow \varphi(Q)$, and $e_i \leftarrow e_i - s$.
return A.

The order of a random point P is not divisible by ℓ_i with probability $1/\ell_i$. This means that in many cases, we will not be able to perform an isogeny for *every* $i \in S$, but only for some (large) subset $S' \subset S$ due to P lacking factors ℓ_i in its order for those remaining $i \in S \setminus S'$. In short, a point P performs the action $\prod_{i \in S'} \mathfrak{l}_i^s$ for some $S' \subset S$, with s the orientation of P (interpreted as ± 1). Sampling a point and computing the action $\prod_{i \in S'} \mathfrak{l}_i^s$ is called a *round*; we perform rounds for different sets S until we compute the full action $\mathfrak{a} = \prod \mathfrak{l}_i^{e_i}$.

Strategies. There are several ways of computing the group action as efficiently

as possible, usually referred to as *strategies*. The strategy in Algorithm 4 is called *multiplicative strategy*[**CLM+18**, **MR18**, 47]. Other notable strategies from the literature are the *SIMBA strategy* [212], *point-pushing strategies* [**CR20**], and *atomic blocks* [**BBCCLMSS21**].

1-point and 2-point approaches. The approach above and in Algorithm 4 samples a single point, computes some isogenies with the same orientation, and repeats this until all steps $\mathfrak{l}_i^{\pm 1}$ are processed. This approach, introduced in [**CLM+18**], is called *1-point approach*. In contrast, one can sample two points per round, one with positive and one with negative orientation, and attempt to compute isogenies for each degree ℓ_i per round, independent of the sign of the e_i [236]. Constant-time algorithms require choosing S independent of the secret key, and all state-of-the-art constant-time implementations use the *2-point approach*, e.g., [**BBCCLMSS21**, 91].

Keyspace. In both CSIDH and CTIDH, each party's private key is an integer vector $(e_1, \ldots, e_n)$ sampled from a bounded subset $\mathcal{K} \subset \mathbb{Z}^n$, the *keyspace*. Different choices of $\mathcal{K}$ have different performance and security properties. The original scheme [**CLM+18**] uses $\mathcal{K}_m = \{-m, \ldots, m\}^n \subset \mathbb{Z}^n$, e.g. $m = 5$ for CSIDH-512. As suggested in [**CLM+18**] and shown in [212], using different bounds m_i for each i can improve speed. The shifted keyspace $\mathcal{K}_m^+ = \{0, \ldots, 2m_i\}^n$ was used in [212]. Other choices of $\mathcal{K}$ were made in [**CCC+19**, 91, 236], and CTIDH [**BBCCLMSS21**] (see Section 8.6.2).

8.4 Attack scenario and fault model

Throughout this work, we assume physical access to some hardware device containing an unknown CSIDH private key $\mathfrak{a}$. In the basic version of the attack, we suppose that the device provides an interface to pass in a CSIDH public-key curve E and receive back the result $\mathfrak{a} * E$ of applying $\mathfrak{a}$ to the public key E as in the second step of the key exchange.

Remark 8.1. Diffie–Hellman-style key agreements typically *hash* the shared secret to derive symmetric key material, instead of directly outputting curves as in our scenario. Our attacks are still applicable in this *hashed version* of the attack, although the complexity for post-processing steps from Section 8.5 will increase significantly. To simplify exposition, we postpone this discussion to Section 8.8.

We assume that the attacker is able to trigger an error during the computation of the orientation of a point in a specific round of the CSIDH algorithm: whenever a point P with orientation $s \in \{-1, 1\}$ is sampled during the algorithm, we can flip the orientation $s \mapsto -s$ as shown below. This leads to some isogenies being computed in the opposite direction throughout the round. The effect of this flip will be explored in Section 8.5.

Square check. In CSIDH, cf. Algorithm 4, the point P is generated in Step 2 and its orientation s is determined in Step 3. The function `IsSquare` determines s by taking as input the non-zero value $z = x^3 + Ax^2 + x$, and computing the Legendre symbol of z. Hence, $s = 1$ when z is a square and $s = -1$ when z is not a square. Many implementations simply compute $s \leftarrow z^{\frac{p-1}{2}}$.

A successful fault injection in the computation $z \leftarrow x^3 + Ax^2 + x$, by skipping an instruction or changing the value randomly, ensures random input to `IsSquare` and so in about half of the cases the output will be flipped by $s \mapsto -s$. In the other half of the cases, the output of `IsSquare` remains s. The attacker knows the outcome of the non-faulty computation and can thus discard those outputs and continue with those where the orientation has been flipped.

Remark 8.2. There are other ways to flip the orientation s. For example, one can also inject a random fault into x after s has been computed, which has a similar effect. The analysis and attack of Sections 8.5 and 8.6 apply to all possible ways to flip s, independent of the actual fault injection. The countermeasures introduced in Section 8.10 prevent all possible ways to flip s that we know of.

Faulting the Legendre symbol computation in `IsSquare`, in general, leads to a random $\mathbb{F}_p$-value as output instead of ± 1. The interpretation of this result is heavily dependent on the respective implementation. For instance, the CSIDH implementation from [**CLM+18**] interprets the output as boolean value by setting $s = 1$ if the result is $+1$, and -1 otherwise. In this case, faults mostly flip in one direction: from positive to negative orientation. Thus, faulting the computation of z is superior in our attack setting.

Elligator. Implementations using a 2-point strategy often use Elligator 2 [44]. On input of a random value, Elligator computes two points P and P' of opposite orientations. An `IsSquare` check is used to determine the orientation of P. If P has positive orientation, we set $P_+ \leftarrow P$ and $P_- \leftarrow P'$. Otherwise, set $P_+ \leftarrow P'$ and $P_- \leftarrow P$. Again, we can fault the input to this `IsSquare` check, which flips the assignments to P_+ and P_-; hence, the orientation of *both* points is flipped.

As before, this means that all isogenies computed using either of these points are pointing in the wrong direction. A notable exception is CTIDH, where two independent calls to Elligator are used to produce points for the 2-point strategy. This is due to security considerations, and the algorithmic and attack implications are detailed in Section 8.6.2.

8.5 Exploiting orientation flips

In Section 8.4, we defined an attack scenario that allows us to flip the orientation s in Line 3. If this happens, the net effect is that we will select an incorrect set S' with opposite orientation, and hence perform an isogeny walk in the *opposite* direction for all the indices in S'. Equivalently, the set S selected in Line 3 has opposite orientation to the point P. For simplicity, we will always fix the set S first and talk about the point P being flipped. We assume that we can successfully flip the orientation in any round r, and that we get the result of the faulty evaluation, which is some *faulty curve* $E_t \neq \mathfrak{a} * E$.

We first study the effect of orientation flips for full-order points in Section 8.5.2, and then discuss effects of torsion in Section 8.5.3 and Section 8.5.4. We organize the faulty curves into components according to their orientation and round in Section 8.5.5 and study the distance of components from different rounds in Section 8.5.6. In Section 8.5.7, we use faulty curves to recover the secret key $\mathfrak{a}$.

8.5.1 Implications of flipping the orientation of a point

In this section, all points will have full order, so Line 7 never skips an i.

Suppose we want to evaluate the group action $\prod_{i \in S} \mathfrak{l}_i * E_A$ for some set of steps S. Suppose we generate a negatively oriented point P, but flipped its orientation. This does not change the point (still negatively oriented), but if we use P to evaluate the steps in what we believe is the *positive* direction, we will in fact compute the steps in the negative direction: $E_f = \prod_{i \in S} \mathfrak{l}_i^{-1} * E_A$. More generally, if we want to take steps in direction s and use a point of opposite orientation, we actually compute the curve $E_f = \prod_{i \in S} \mathfrak{l}_i^{-s} * E_A$.

Suppose we flip the orientation of a point in one round of the isogeny computation $E_B = \mathfrak{a} * E_A$ and the rest of the computation is performed correctly. The resulting curve E_t is called a *faulty curve*. If the round was computing steps for isogenies in S with direction s, the resulting curve satisfies $E_B = \prod_{i \in S} \mathfrak{l}_i^{2s} * E_t$, that is, the faulty curve differs from the correct curve by an isogeny whose de-

gree is given by the (squares of) primes ℓ_i for $i \in S$, the set S in the round we faulted. We call S the *missing set* of E_t.

Distance between curves. We define the *distance* d between two curves E and E' as the lowest number of different degrees for isogenies $\varphi : E \to E'$. Note that the distance only tells us how many primes we need to connect two curves, without keeping track of the individual primes ℓ_i or their multiplicity. Specifically for a faulty curve with $E_B = \prod_{i \in S} \mathfrak{l}_i^{2s} * E_t$, we define the distance to E_B as the number of flipped steps $|S|$. Note that each $\mathfrak{l}_i$ appears as a square; this gets counted *once* in the distance.

Positive and negative primes. Suppose the secret key $\mathfrak{a}$ is given by the exponent vector (e_i). Then every ℓ_i is used to take e_i steps in direction $\text{sign}(e_i)$. Define the set of *positive* primes $L_+ := \{i \mid e_i > 0\}$, *negative* primes $L_- := \{i \mid e_i < 0\}$, and neutral primes $L_0 := \{i \mid e_i = 0\}$. For 1-point strategies and any faulty curve E_t with missing set S, we always have $S \subset L_+$ or $S \subset L_-$. However, using 2-point strategies, the sets S may contain positive and negative primes. We use the terminology 'flipping a batch' when we refer to the effect of an orientation flip to the primes being performed: when we flip the orientation s of a negative point from negative to positive, the final result has performed a batch of positive primes in the negative direction.

Example 8.1. Take CSIDH-512. Assume we flip the orientation $s \mapsto -s$ of the first point P. From Algorithm 4, we see the elements of S are exactly those i such that $|e_i| \geq 1$ and $\text{sign}(e_i) = -s$. Therefore, we have $S = L_{-s}$.

8.5.2 Faulty curves and full-order points

We continue to assume that all points have full order, so Line 7 never skips an i, and analyze which faulty curves we obtain by flipping the orientation in round r. We treat the general case in Section 8.5.3 and Section 8.5.4.

Effective curves. For any strategy (cf. Section 8.3.2), the computation in round r depends on what happened in previous rounds. In a 2-point strategy, we sample both a negative and a positive point and use them to perform the isogenies in both directions. So assuming points of full order, the round-r computation and the set S do not depend on the previous round but only the secret key.

In a 1-point strategy, we sample 1 point per round, and only perform isogenies in the direction of that point. So the set S in round r depends additionally on what was computed in previous rounds. However, the computation in round r only depends on previous rounds with *the same orientation*. The orientation of a round refers to which primes were used. Hence, a *positive* round means that the steps were performed for the positive primes, in the positive or negative direction.

Notation. Let $+$ and $-$ denote the positive and negative orientation, respectively. For a 1-point strategy, we encode the choices of orientations by a sequence of $\pm$. We denote the round r in which we flip the orientation of a point by parentheses $(\cdot)$. We truncate the sequence at the moment of the fault because the rest of the computation is computed correctly. Hence, $++(-)$ means a computation starting with the following three rounds: the first two rounds were positive, the third one was a negative round with a flipped orientation, so the steps were computed for the negative primes, but in the positive direction.

Consider a flip of orientation in the second round. There are four possible scenarios:

$+(+)$. Two positive rounds, but the second positive batch of primes was flipped and we took the steps in negative direction instead.

$+(-)$. One positive round, one negative batch flipped to the positive direction.

$-(+)$. One negative round, one positive batch flipped to the negative direction.

$-(-)$. Two negative rounds, the second negative batch flipped to positive.

All four cases are equally likely to appear for 1-point strategies, but result in different faulty curves. Since the computation only depends on previous rounds with the same orientation, the case $+(-)$ is easily seen to be the same as $(-)$ and $++(-)$: all three are cases where the orientation of the point was flipped the first time a negative round occurred. However, the cases $+(+)$ and $-(+)$ are different: the latter is equivalent to $(+)$. For example, in CSIDH, the set S for $(+)$ is $\{i \mid e_i \geq 1\}$, and the set S' for $+(+)$ is $\{i \mid e_i \geq 2\}$, differing exactly at the primes for which $e_i = 1$.

Example 8.2 (CSIDH)**.** For a secret key $(1, -2, -1, 3)$ in CSIDH with primes $L = \{3, 5, 7, 11\}$, the case $+(-)$ takes us to a faulty curve that is two $\{5, 7\}$-isogenies away from the desired curve, whereas the case $-(-)$ results in a curve two 5-isogenies away.

Effective round. Let $E^{r,+}$ be the faulty curve produced by the sequence $+\cdots+(+)$ of length r, and $E^{r,-}$ the curve produced by sequence $-\cdots-(-)$. We call the curves $E^{r,\pm}$ *effective round-r curves*. For a 2-point strategy, all faulty curves from round r are effective round-r curves. For 1-point strategies, effective round-r curves can be produced from other sequences as well, e.g. $+(-)$ produces the effective round 1 curve $E^{1,-}$ and $++--+(-)$ produces an effective round-3 curve $E^{3,-}$. To get an effective round-r sample $E^{r,+}$ from a round n, the last sign in the sequence must be $(+)$, and the sequence must contain a total of r pluses.

Lemma 8.1. Assume we use a 1-point strategy. The probability to get any effective round-r sample if we successfully flip in round n is equal to $\binom{n-1}{r-1} \cdot \frac{1}{2^{n-1}}$.

Remark 8.3. For a 2-point strategy, all curves resulting from a fault in round r are effective round-r curves.

Torsion sets $S^{r,+}$ and $S^{r,-}$. Define the set $S^{r,s}$ as the missing set of the effective round-r curve with orientation s, i.e., $E_B = \prod_{i \in S^{r,s}} \mathfrak{l}_i^{2s} * E^{r,s}$.

Example 8.3 (CSIDH)**.** The sets $S^{1,\pm}$ were already discussed in Example 8.1. In general, $S^{r,+} = \{i \mid e_i \geq r\}$ and $S^{r,-} = \{i \mid e_i \leq -r\}$.

8.5.3 Missing torsion: faulty curves and points of non-full order

In Section 8.5.2, we worked under the unrealistic assumption that all points we encounter have full order. In this section, we relax this condition somewhat: we assume that every point had full order (and hence all isogenies were computed) up until round r, but the point P generated in round r potentially has smaller order. We call this the *missing torsion* case. The remaining relaxation of non-full order points in earlier rounds will be concluded in Section 8.5.4.

If the point P used to compute isogenies in round r does not have full order, the faulty curve E_t will differ from the effective round-r curve $E^{r,s}$ by the primes ℓ_i with $i \in S^{r,s}$ which are missing in the order of P.

Round-r faulty curves. For simplicity, assume that we are in round r, in the case $+\cdots+(+)$, and that none of the isogenies in the previous rounds failed. In round r, a negative point P is sampled, but we flip its orientation, so the batch of positive primes will be computed in the negative direction.

If the point P has full order, we obtain the curve $E^{r,+}$ at the end of the computation, which differs from E_B exactly at primes contained in $S^{r,+}$. If,

however, the point P does not have full order, a subset $S \subset S^{r,+}$ of steps will be computed, leading to a different faulty curve E_t. By construction, the curve E_t is related to E_B via $E_B = \prod_{i \in S} \mathfrak{l}_i^2 * E_t$.

Assume we repeat this fault in T runs, leading to different faulty curves E_t. Let $n(E_t)$ be the number of times the curve E_t occurs among the T samples. For each such E_t, we know $E_B = \prod_{i \in S_t} \mathfrak{l}_i^{2s} * E_t$, where $S_t \subset S^{r,+}$ is determined by the order of P_t. As P_t is a randomly sampled point, it has probability $\frac{\ell_i - 1}{\ell_i}$ that its order is divisible by ℓ_i, and so probability $\frac{1}{\ell_i}$ that its order is not divisible by ℓ_i. This gives us directly the probability to end up at E_t: the order of the point P_t should be divisible by all ℓ_i for $i \in S_t$, but not by those ℓ_i for $i \in S^{r,+} \setminus S_t$. This is captured in the following result.

Proposition 8.2. Let P_t be a random negative point, where we flip the orientation s to positive. The probability that we compute the faulty curve $E_t = \prod_{i \in S_t} \mathfrak{l}_i^{-2} * E_B$ is exactly $p_t = \prod_{i \in S_t} \frac{\ell_i - 1}{\ell_i} \cdot \prod_{i \in S^{r,+} \setminus S_t} \frac{1}{\ell_i}$.

Proof. The probability of obtaining E_t is equal to the probability that the order of the point P_t is divisible by all the primes in S_t and not divisible by all the primes in $S^{r,+} \setminus S_t$. The first happens with probability $\prod_{i \in S_t} \frac{\ell_i - 1}{\ell_i}$; the second is an independent event happening with probability $\prod_{i \in S^{r,+} \setminus S_t} \frac{1}{\ell}$. □

In CTIDH, the success probability of each point to match that of the smallest prime in the batch to hide which prime is handled. But for fixed batches, an analogous results to Proposition 8.2 can be given.

The expected number of appearances $n(E_t)$ of a curve E_t is $n(E_t) \approx p_t \cdot T$ for T runs. As $\frac{\ell_i - 1}{\ell_i} \geq \frac{1}{\ell_i}$ for all ℓ_i, the probability p_t is maximal when $S_t = S^{r,+}$. We denote this probability by $p^{r,+}$. Hence, the curve that is likely to appear the most in this scenario over enough samples, is the curve $E^{r,+}$ which we defined as precisely that curve with missing set $S^{r,+}$. For now, we focused solely on the positive curves. Taking into account the negative curves too, we get:

Corollary 8.1. Let $E^{r,+} = \prod_{i \in S^{r,+}} \mathfrak{l}_i^{-2} * E_B$ and let $E^{r,-} = \prod_{i \in S^{r,-}} \mathfrak{l}_i^{2} * E_B$. Then $E^{r,+}$ and $E^{r,-}$ have the highest probability to appear among the effective round-r faulty curves. As a consequence, the largest two values $n(E)$ of all effective round-r curves are most likely $n(E^{r,+})$ and $n(E^{r,-})$

Example 8.4 (CSIDH)**.** Take the set $S^{1,+} = \{i \mid e_i \geq 1\}$ and let $p^{1,+}$ denote the probability that a random point P has order divisible by all primes in $S^{1,+}$. This probability depends on the secret key (e_i), but can be estimated if we collect enough faulty curves. Moreover, if $e_1 \neq 0$, then $\ell_1 = 3$ dominates either

$p^{1,+}$ or $p^{1,-}$ through the relatively small probability of $2/3$ that P has order divisible by 3. Thus, if the largest pile of faulty curves is $E^{1,\pm}$, we expect $S^{1,\pm}$ not to contain 1. For instance, if e_1 is positive, $p^{1,-}$ is larger than $p^{1,+}$ and so we expect $n(E^{1,-})$ to be larger than $n(E^{1,+})$. In this case, we would expect to see another faulty curve E_t with $n(E_t)$ half the size of $n(E^{1,+})$; this curve E_t has *almost* full missing set $S^{1,+}$, but does not miss the 3-isogeny. That is, $S_t = S^{1,+} \setminus \{1\}$, with probability $p_t := \frac{1}{\ell_1} \cdot \frac{\ell_1}{\ell_1 - 1} \cdot p^{1,+} = \frac{1}{2} \cdot p^{1,+}$. This curve E_t is very "close" to $E^{1,+}$; they are distance 1 apart, precisely by $\mathfrak{l}_1^2$.

The precise probabilities $p^{r,+}$ and $p^{r,-}$ depend highly on the specific implementation we target. Given an implementation, the values of $p^{r,+}$ and $p^{r,-}$ allow for a concrete estimate on the size of $n(E)$ for a specific curve E. Because ℓ_i that are missing in the order of P_t skip the misoriented steps, the curves in the neighborhood of $E^{r,+}$ differ by two ℓ_i-isogenies for $i \in S^{r,+} \setminus S_t$ in positive direction while those around $E^{r,-}$ differ by two ℓ_i-isogenies for $i \in S^{r,-} \setminus S_t$ in negative direction.

Distance between samples. We can generalize Example 8.4 for any two faulty curves E_t and $E_{t'}$ that are effective round-r samples of the same orientation, using Proposition 8.2.

Corollary 8.2. Let E_t and $E_{t'}$ both be effective round-r samples with the same orientation s and missing torsion sets S_t and $S_{t'}$. Let S_Δ denote the difference in sets S_t and $S_{t'}$, i.e., $S_\Delta = (S_t \setminus S_{t'}) \cup (S_{t'} \setminus S_t)$. Then E_t and $E_{t'}$ are distance $|S_\Delta|$ apart, by $E_t = \left(\prod_{i \in S_{t'} \setminus S_t} \mathfrak{l}_i^{2s} \cdot \prod_{i \in S_t \setminus S_{t'}} \mathfrak{l}_i^{-2s}\right) * E_{t'}$. In particular, any effective round-r curve E_t with orientation s is close to $E^{r,s}$: since $S_t \subset S^{r,s}$, S_Δ is small.

Example 8.5 (CSIDH)**.** For a secret key $(2, 3, 1, 2)$ in CSIDH with primes $L = \{3, 5, 7, 11\}$, the first positive point with full torsion P will perform a 3, 5, 7 and 11-isogeny, so $S^{1,+} = \{1, 2, 3, 4\}$, with $S^{2,+} = \{1, 2, 4\}$ and $S^{3+} = \{2\}$. In one faulted run, the first point P_{t_1} might only have $\{5, 7, 11\}$-torsion, while in another faulted run the first point P_{t_2} might only have $\{3, 7, 11\}$-torsion. The faulty curves E_{t_1} and E_{t_2} differ from E^+ by two 3-isogenies and two 5-isogenies, respectively, and have a distance 2 towards each other: their S_Δ is $\{1, 2\}$, so they are two $\{3, 5\}$-isogenies apart. The two samples E_{t_1} and E_{t_2} therefore show that both 1 and 2 are in S^+ and show that $e_1 \geq 1$ and $e_2 \geq 1$.

Corollary 8.2 will be essential to recover information on $S^{r,+}$ out of the samples E_t: Recovering small isogenies between samples allows us to deduce which i are in $S^{r,+}$ or $S^{r,-}$, and so leaks information about e_i.

8.5.4 Torsion noise

Orthogonally to Section 8.5.3, we now examine the case that missing torsion occurred in an earlier round than the round we are faulting.

Example 8.6 (CSIDH)**.** Suppose that $e_1 = 1$ and that in the first positive round, the point generated in Line 2 of Algorithm 4 had order not divisible by ℓ_1, but all other points have full order. Thus, the ℓ_1-isogeny attempt fails in the first positive step. Consider now the second positive round. From Section 8.5.2, we would expect to be computing steps in $S^{2,+} = \{i \mid e_i \geq 2\}$. But no ℓ_1-isogeny has been computed in the first round, so it will be attempted in this second positive round. If we now fault the second positive point, we obtain a faulty curve that is *also missing* ℓ_1, that is, $E_t = \mathfrak{l}_1^{-2} * E^{2,+}$. Unlike the faulty curves from 8.5.3, the positively oriented isogeny goes from E_t *towards* $E^{2,+}$. Also, note that in this scenario if $e_1 = 2$, a fault in round 2 would still result in the curve $E^{2,+}$, because the set $S^{2,+}$ contains ℓ_1 already, and so the missed ℓ_1-isogeny from round 1 will be computed in later rounds.

We refer to the phenomenon observed in Example 8.6 as *torsion noise*. More concretely, torsion noise happens when we fault the computation in round r for a run which is computing an ℓ_i-isogeny in round r for $|e_i| < r$ because it was skipped in a previous round.

Torsion noise is rarer than missing torsion but can still happen: the isogeny computation needs to fail and the fault must come when we are "catching up" with the computation. For CSIDH, torsion noise can only happen if $r > |e_i|$ and the computation of the ℓ_i-isogeny failed in at least $r - |e_i|$ rounds. Torsion noise is unlikely for large ℓ_i because the probability that an isogeny fails is about $1/\ell_i$.

For small primes, such as $\ell_i \in \{3, 5, 7\}$, we observe a lot of torsion noise. This can slightly affect the results as described in Section 8.5.3, but has no major impact on the results in general. Concretely, torsion noise may make it impossible to determine the correct e_i for the small primes given only a few faulted curves. Nevertheless, their exact values can be brute-forced at the end of the attack.

Remark 8.4 (Orientation of torsion noise)**.** Faulty curves affected by torsion noise require contrarily oriented isogenies to the curves $E^{r,s}$ than the remaining faulty curves. Therefore, if torsion noise happens and we find a path from such a curve $E_t \to E^{r,s}$, then we can infer not just the orientation of the primes in this path, but often also bound the corresponding exponents e_i.

8.5.5 Connecting curves from the same round

Suppose we have a set of (effective) round-r faulty curves with the same orientation s, and suppose r and s are fixed. In Corollary 8.2, we show that such curves are close to each other. In particular, the path from E_t to $E^{r,s}$ uses only degrees contained in the set $S^{r,s}$. Finding short paths among faulty curves gives us information about $S^{r,s}$, and hence about the secret key.

Component graphs. Starting from a set $\{E_t\}$ of round-r faulty curves with orientation s, we can use them to define the graph $G^{r,s}$ as follows: The vertices of $G^{r,s}$ are given by $\{E_t\}$, and the edges are steps between the curves, labeled by i if the curves are connected by two ℓ_i-isogenies. For convenience, we sparsify the graph $G^{r,s}$ and regard it as a tree with the curve $E^{r,s}$ as the root.

Remark 8.5 (Edges)**.** Starting from a set of faulty curves, it is easy to build the graphs $G^{r,s}$. We can identify the *roots* of these graphs $E^{r,s}$ using Corollary 8.1. Then the distance from the root to any round-r faulty curve with the same orientation is small (cf. Corollary 8.2). Therefore, we can find the edges by applying short walks in this graph. Note that edges of $G^{r,s}$ give information on $S^{r,s}$.

Remark 8.6 (Missing vertices)**.** If we do not have enough faulty curves $\{E_t\}$, it may not be possible to connect all the curves with single steps (understood as isogenies of square degree, see Corollary 8.2). For convenience, we assume that we have enough curves. In practice, we include in the graph $G^{r,s}$ any curve on the path between E_t to $E^{r,s}$ (again, taking steps with square prime degree).

Remark 8.7 (Components)**.** We imagine the graphs $G^{r,s}$ as subgraphs of the *isogeny graph* of supersingular elliptic curves with edges given by isogenies. Computing short paths from $E^{r,s}$ will give us enough edges so that we can consider the graphs $G^{r,s}$ to be connected. Hence we call them *components*.

Secret information. An effective round-r faulty curve E_t with torsion set $S_t \subset S^{r,+}$ can easily be connected by a path with labels $S^{r,+} \setminus S_t$. Moreover, the orientation $E^{r,+} \to E_t$ is positive. Therefore, we can identify which components are positive, and all the labels of the edges are necessarily in $S^{r,+}$, that is, the prime ℓ_i is positive. Torsion noise can be recognized from the opposite direction of the edges (see Remark 8.4). In either case, the components $G^{r,s}$ give us the orientation of all the primes occurring as labels of the edges.

Sorting round-r samples. Suppose we are given a set of round-r faulty curves $\{E_t\}$, but we do not have information about the orientation yet. We can again use Corollary 8.1 to find the root of the graph; then we take small isogeny steps until we have two connected components G_1, G_2. It is easy to determine the direction of the edges given enough samples; ignoring torsion noise, the positively oriented root will have outgoing edges.

In summary, we try to move curves E_t from a pile of unconnected samples to one of the two graphs by finding collisions with one of the nodes in $G^{r,+}$ resp. $G^{r,-}$. The degrees of such edges reveal information on $S^{r,+}$ and $S^{r,-}$: An edge with label i in $G^{r,+}$ implies $i \in S^{r,+}$, and analogously for $G^{r,-}$ and $S^{r,-}$. Figure 8.1 summarizes the process, where, e.g., $E^{r,+} \to E_7$ shows missing torsion and $E_8 \to E^{r,+}$ is an example of torsion noise.

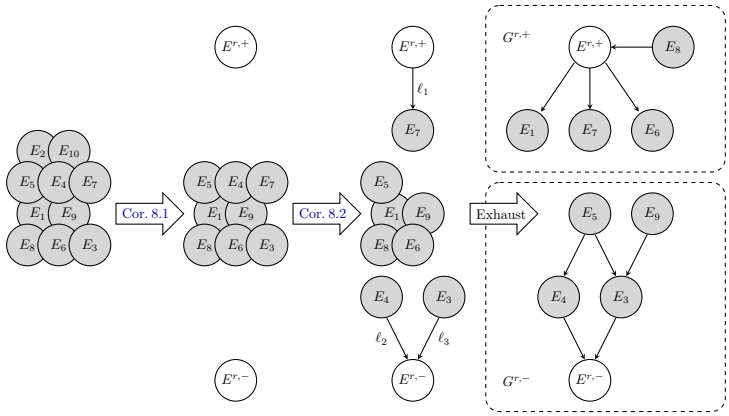


Figure 8.1: Building up the component graphs of faulty curves.

8.5.6 Connecting the components $G^{r,s}$

Now, we explain how to connect the components $G^{r,s}$ for different rounds r. The distance of these components is related to the sets $S^{r,+}$ and $S^{r,-}$. We then show that it is computationally feasible to connect the components via a meet-in-the-middle attack. Connecting two components gives us significantly more knowledge on the sets $S^{r,+}$ and $S^{r,-}$, such that connecting all components is

enough to reveal the secret $\mathfrak{a}$ in Section 8.5.7.

Information from two connected components. We start with an example:

Example 8.7 (CSIDH)**.** Recall that we have $S^{r,+} = \{i \mid e_i \geq r\}$, and so $E^{r,+} = \prod_{i \in S^{r,+}} \mathfrak{l}_i^{-2} * E_B$. This means that, e.g., we have $S^{3,+} \subset S^{2,+}$, and $E^{2,+}$ has a larger distance from E_B than $E^{3,+}$. The path between $E^{3,+}$ and $E^{2,+}$ then only contains steps of degrees ℓ_i such that $i \in S^{2,+} \backslash S^{3,+}$, so $e_i = 2$. In general, it is easy to see that finding a single isogeny that connects a node E_{t_3} from $G^{3,+}$ and a node E_{t_2} in $G^{2,+}$ immediately gives the connection from $E^{3,+}$ to $E^{2,+}$. Hence, we learn all ℓ_i with $e_i = 2$ from the components $G^{3,+}$ and $G^{2,+}$.

In the general case, if we find an isogeny between two such graphs, say $G^{r,+}$ and $G^{r',+}$, we can compute the isogeny between the two roots $E^{r,+}$ and $E^{r',+}$ of these graphs. The degree of this isogeny $E^{r,+} \to E^{r',+}$ describes precisely the *difference* between the sets $S^{r,+}$ and $S^{r',+}$. The example above is the special case $r' = r + 1$, and in CSIDH we always have $S^{(r+1),+} \subset S^{r,+}$, so that the difference between $S^{r,+}$ and $S^{(r+1),+}$ is the set of ℓ_i such that $e_i = r$. In other CSIDH-variants, such sets are not necessarily nested, but connecting all components still reveals e_i as Section 8.5.7 will show. In general, we connect two subgraphs by a distributed meet-in-the-middle search which finds the shortest connection first.

Distance between connected components. As we have shown, connecting two components $G^{r,+}$ and $G^{r',+}$ is equivalent to finding the difference in sets $S^{r,+}$ and $S^{r',+}$. The distance between these sets heavily depends on the implementation, as these sets are determined by the key $\mathfrak{a}$ and the evaluation of this key. For example, in CSIDH-512, the difference between $S^{r,+}$ and $S^{(r+1),+}$ are the $e_i = r$, which on average is of size $\frac{74}{11} \approx 6.7$. In practice, this distance roughly varies between 0 and 15. For an implementation such as CTIDH-512, the sets $S^{r,+}$ are smaller in general, on average of size 7, and the difference between such sets is small enough to admit a feasible meet-in-the-middle connection. See Section 8.7 for more details on how we connect these components in practice.

8.5.7 Revealing the private key

So far, we showed how connecting different components $G^{r,+}$ and $G^{r',+}$ reveals information on the difference between the sets $S^{r,+}$ and $S^{r',+}$. In this section, we show that when all components are connected, we can derive the secret $\mathfrak{a}$.

This wraps up Section 8.5: Starting with disorientations in certain rounds r, we derive the secret $\mathfrak{a}$ from the resulting graph structure, assuming enough samples.

From differences of sets to recoveries of keys. By connecting the graphs of all rounds, including the one-node-graph consisting of just the correct curve E_B, we learn the difference between the sets $S^{r,+}$ and $S^{(r+1),+}$ for all rounds r (as well as for $S^{r,-}$ and $S^{(r+1),-}$). A single isogeny from some $G^{r,+}$ to $E_B = \mathfrak{a} * E_A$ then recovers $S^{r,+}$ for this round r: Such an isogeny gives us an isogeny from $E^{r,+} = \prod_{i \in S^{r,+}} \mathfrak{l}_i^{-2} * E_B$ to E_B, whose degree shows us exactly those $\ell_i \in S^{r,+}$. From a connection between the components $G^{r,+}$ and $G^{r',+}$, we learn the difference in sets $S^{r,+}$ and $S^{r',+}$. From $S^{r,+}$, we can then deduce $S^{r',+}$. Therefore, if all graphs $G^{r,+}$ for different r are connected, and we have at least one isogeny from a node to E_B, we learn the sets $S^{r,+}$ for all rounds r (and equivalently for $S^{r,-}$). From the knowledge of all sets $S^{r,+}$ and $S^{r,-}$ we then learn $\mathfrak{a} = (e_i)$: the sign of e_i follows from observing in which of the sets $S^{r,+}$ or $S^{r,-}$ the respective ℓ_i appears, and $|e_i|$ equals the number of times of these appearances.

In practice however, due to missing torsion and torsion noise, connecting all components may not give us the *correct* sets $S^{r,+}$ resp. $S^{r,-}$. In such a case, one can either gather more samples to gain more information, or try to brute-force the difference. In practice, we find that the actual set $S^{r,+}$ as derived from $\mathfrak{a}$ and the set $\tilde{S}^{r,+}$ derived from our attack (leading to some $\mathfrak{a}'$) always have a small distance. A simple meet-in-the-middle search between $\mathfrak{a}' * E_A$ and $\mathfrak{a} * E_A$ then quickly reveals the errors caused by missing torsion and torsion noise.

8.5.8 Complexity of recovering the secret $\mathfrak{a}$

The full approach of this section can be summarized as follows:

1. Gather enough effective round-r samples E_t per round r, using Lemma 8.1.
2. Build up the components $G^{r,+}$ and $G^{r,-}$ using Corollaries 8.1 and 8.2.
3. Connect components to learn the difference in sets $S^{r,+}$ and $S^{r',+}$.
4. Compute the sets $S^{r,+}$ and $S^{r,-}$ for every round and recover $\mathfrak{a}$.

The overall complexity depends on the number of samples per round, but is in general dominated by Item 3. For Item 2, nodes are in most cases relatively close to the root $E^{r,+}$ or to an already connected node E_t, as shown in Corollary 8.2.

For Item 3, components are usually further apart than nodes from Item 2. In general, the distance between components $G^{r,+}$ and $G^{r',+}$ depends heavily on the specific design choices of an implementation. In a usual meet-in-the-middle approach, where n is the number of ℓ_i over which we need to search and d is the distance between $G^{r,+}$ and $G^{r',+}$, the complexity of finding a connection is $\mathcal{O}(\binom{n}{d/2})$. Note that we can use previous knowledge from building components or finding small-distance connections between other components to reduce the search space and thus minimize n for subsequent connections. We analyze this in detail for specific implementations in Section 8.6.

8.6 Case studies: CSIDH and CTIDH

We previously defined a general strategy in four steps. In practice, those steps are dependent on the actual implementation. Concretely, we select two main implementations: CSIDH-512 and CTIDH-512. We discuss CSIDH-512 in Section 8.6.1, CTIDH-512 in Section 8.6.2, and we analyze other implementations in Section 8.6.3.

In this section we will specialize to inputting E_0 into the target which thus computes a faulty version of $E_B = \mathfrak{a} * E_0$, its own public key.

8.6.1 Breaking CSIDH-512

The primes used in CSIDH-512 [**CLM+18**] are $L = \{3, 5, \ldots, 377, 587\}$, and exponent vectors are sampled as $(e_i) \in \{-5, \ldots, 5\}^{74}$ uniformly at random. For any $k \in \{-5, \ldots, 5\}$ we expect about $\frac{1}{11} \cdot 74$ primes ℓ_i with $e_i = k$; this count obeys a binomial distribution with parameters $(74, 1/11)$. We expect to see about $\frac{5}{11} \cdot 74 \approx 33.6$ positive and negative primes each, and about $\frac{1}{11} \cdot 74 \approx 6.7$ neutral primes.

In CSIDH-512, the group action is evaluated as displayed in Algorithm 4, using a 1-point strategy. In particular, after generating a point with orientation s, we set $S = \{i \mid e_i \neq 0,\ \mathrm{sign}(e_i) = s\}$. If the value of s is flipped, we set $S = \{i \mid e_i \neq 0,\ \mathrm{sign}(e_i) = -s\}$, but we perform the steps in direction s.

Now, we specialize the four steps to secret-key recovery defined in Section 8.5.8.

Building components $G^{r,+}$ and $G^{r,-}$. Item 2 of the attack on CSIDH-512 works exactly as described in Section 8.5.5. If E_t and $E_{t'}$ are effective samples

from the same round with the same orientation, their distance is small (Corollary 8.2). We can thus perform a neighborhood search on all of the sampled curves until we have 10 connected components $G^{r,\pm}$ for $r \in \{1, \ldots, 5\}$, as in Figure 8.1. This step is almost effortless: most curves will be distance 1 or 2 away from the root $E^{r,s}$. In practice, using round information and number of occurrences, we identify the 10 curves $E^{r,\pm}$ for $r = 1, \ldots, 5$, and explore all paths of small length from those 10 curves, or connect them via a meet-in-the-middle approach (e.g., using `pubcrawl`, see Section 8.7). The degrees of the isogenies corresponding to the new edges in $G^{r,\pm}$ reveal information on the sets $S^{r,\pm}$, which can be used to reduce the search space when connecting the components $G^{r,\pm}$.

Filter-and-break it, until you make it. Item 3 is the most computationally intensive step, as it connects 11 components ($G^{r,\pm}$ and E_B) into a single large connected component. We argue that it is practical for CSIDH-512.

More specifically, we want to find connections between $G^{r,\pm}$ and $G^{(r+1),\pm}$, as well as connections from $G^{5,\pm}$ to E_B. This gives us 10 connections, corresponding to the gaps $\{i \mid e_i = k\}$ for $k \in [-5, 5] \setminus \{0\}$. Figure 8.2 shows an abstraction of this large connected component.

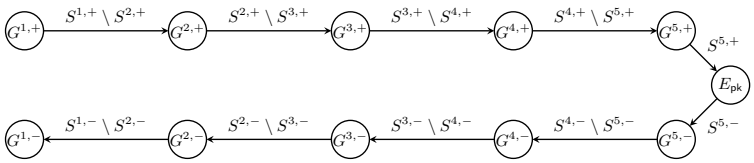


Figure 8.2: Large connected component associated to an attack on CSIDH-512.

Since there are 74 primes in total, and only 10 gaps, at least one of these gaps is at most 7 primes. If we assume that at least 5 of the exponents are 0 (we expect ≈ 7 to be 0), then the smallest distance is at most 6 steps. Such gaps are easily found using a meet-in-the-middle search, see Section 8.7.

Let us call *support* the set of isogeny degrees used in a meet-in-the-middle neighborhood search. We can connect all components by a meet-in-the-middle search with support $\{\ell_1, \ldots, \ell_{74}\}$. This becomes infeasible for large distances, so instead, we adaptively change the support. We start by finding short connections, and use the labels we find to pick a smaller support for searching between

certain components, i.e., *filter* some of the ℓ_i out of the support.

First, we learn the orientation of the components by identifying $G^{1,\pm}$ and considering the direction of the edges. Effective round-1 samples do not have torsion noise, so the root $E^{1,+}$ has only outgoing edges, whereas the root $E^{1,-}$ has only incoming edges. The labels of the edges of $G^{1,+}$ must be positive primes, and all components with a matching label are also positive. Next, all the labels that appear as degrees of edges in $G^{r,+}$ for any r are necessarily positive. Finally, positively oriented components can only be connected by positive primes, so we can remove from the support all the primes that we know are negative. Similarly for negative orientations.

After finding the first connection we restrict the support even more: we know that any label i appears in at most *one* connection. Hence, whenever we find a connection, we get more information about orientation and can reduce the support for further searches, allowing us to find larger connections. We repeat this procedure with more and more restrictions on the support until we find the full connected component.

Recovering the secret key. From the connected components, we recover all of the sets $S^{r,\pm}$ and we compute the secret key as described in Section 8.5.7.

Example 8.8 (Toy CSIDH-103)**.** Figure 8.3 shows the resulting connected graph for a toy version of CSIDH using Algorithm 4 with the first $n = 21$ odd primes and private keys in $\{-3, \dots, +3\}^n$. Each round was faulted 10 times.

The distances between the components are very small and hence connecting paths are readily found. We sparsify the graph to plot it as a spanning tree; the edges correspond to positive steps of the degree indicated by the label. This graph comes from the secret key

$$(-1, +1, +2, +3, -2, +3, +2, +3, +1, +2, -3, -3, +2, +3, -2, -3, -2, +2, +1, -3,\ 0).$$

Required number of samples. Recovering the full secret exponent vector in CSIDH-512 equates to computing the sets $S^{r,+}$ and $S^{r,-}$ for $r \in \{1, \dots, 5\}$. Recall that to compute these sets we need to build a connected component including subcomponents $G^{r,+}$ and $G^{r,-}$ for $r \in \{1, \dots, 5\}$, and E_B (the one-node-graph consisting of just the public key). We build the components $G^{r,+}$ and $G^{r,-}$ by acquiring enough effective round-r samples. More effective round-r samples may give more vertices in $G^{r,\pm}$, and more information about $S^{r,\pm}$.

Let T_r be the number of effective round-r samples and let $T = \sum T_r$. A first approach is to inject in round r until the probability is high enough that we have enough effective round-r samples. For CSIDH-512, we take $T_1 = 16$, $T_2 = 16$, $T_3 = 32$, $T_4 = 64$ and $T_5 = 128$, so that $T = 256$. From Lemma 8.1,

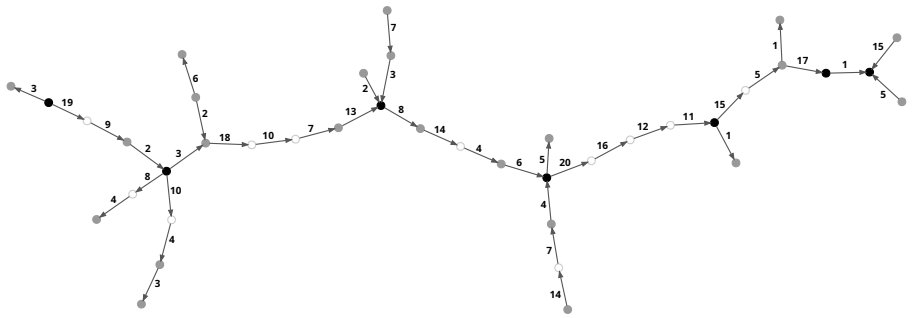


Figure 8.3: Example isogeny graph of faulty curves obtained from attacking the fictitious CSIDH-103 implementation from Example 8.8. An edge labeled i denotes the isogeny step $\mathfrak{l}_i$. The E_B curve and the root faulty curves $E^{r,s}$ are rendered in black (from left to right: $E^{1,+}$, $E^{2,+}$, $E^{3,+}$, E_B, $E^{3,-}$, $E^{2,-}$, $E^{1,-}$), other faulty curves appearing in the dataset are gray, and white circles are "intermediate" curves discovered while connecting the components. The primes appearing on the connecting path between $E^{i,\pm}$ and $E^{i+1,\pm}$ are exactly the primes appearing i times with orientation $\pm$. For example, the primes indexed by 2, 9, 19 appearing between $E^{1,+}$ and $E^{2,+}$ have exponent $+1$ in the secret key.

we then expect 8 round-5 samples (4 per orientation) and the probability that we do not get any of the elements of $G^{5,+}$ or $G^{5,-}$ is about 1.7%.

This strategy can be improved upon. Notice that we need round-5 samples, and so in any case we need T_5 rather large (in comparison to T_i with $i < 5$) to ensure we get such samples. But gathering samples from round 5 already gives us many samples from rounds before. Using Lemma 8.1 with $T_5 = 128$, we get on average 8 effective round-1 samples, 32 effective round-2 samples, 48 effective round-3 samples, 32 effective round-4 samples and 8 effective round-5 samples. In general, attacking different rounds offers different tradeoffs: attacking round 9 maximizes getting effective round-5 samples, but getting a round-1 sample in round 9 is unlikely. Faulting round 1 has the benefits that all faulty curves are effective round-1 curves, making them easy to detect in later rounds; that no torsion noise appears; and that missing torsion quickly allows to determine the orientation of the small primes, reducing the search space for connecting the components. Finally, note that gathering T faulty samples requires approx-

imately $2T$ fault injections, since, on average, half of the faults are expected to will flip the orientation.

8.6.2 Breaking CTIDH-512

CTIDH [**BBCCLMSS21**] partitions the set of primes ℓ_j into b batches, and bounds the number of isogenies per batch. For a list $N \in \mathbb{Z}_{>0}^b$ with $\sum N_k = n$ and a list of non-negative bounds $m \in \mathbb{Z}_{\geq 0}^b$ define the keyspace as

$$\mathcal{K}_{N,m} := \left\{(e_1, \ldots, e_n) \in \mathbb{Z}^n \;\middle|\; \textstyle\sum_{j=1}^{N_i} |e_{i,j}| \leq m_i \text{ for } 1 \leq i \leq b\right\},$$

where $(e_{i,j})$ is a reindexed view of (e_i) given by the partition into batches.

CTIDH-512 uses 14 batches with bounds $m_i \leq 18$, requiring at least 18 rounds. In every round, we compute one isogeny per batch; using a 2-point strategy, we compute isogenies in both positive and negative direction. So, all round-r samples are effective round-r samples.

Injecting faults. To sample oriented points, CTIDH uses the Elligator-2 map twice. First, Elligator is used to sample two points P_+ and P_- on the starting curve E_A. A direction s is picked to compute an isogeny, the point P_s is used to take a step in that direction to a curve $E_{A'}$, and the point P_s is mapped through the isogeny. Then another point P'_{-s} is sampled on $E_{A'}$ using Elligator.

We will always assume that we inject a fault into only one of these two Elligator calls (as in Section 8.4). Hence, as for CSIDH and 1-point strategies, we again always obtain either positively or negatively oriented samples.

Different rounds for CTIDH-512. Per round, CTIDH performs one $\ell_{i,j}$ per batch $\mathcal{B}_i$. Within a batch, the primes $\ell_{i,j}$ are ordered in ascending order: if the first batch is $\mathcal{B}_1 = \{3, 5\}$ and the exponents are $(2, -4)$, then we first compute 2 rounds of 3-isogenies in the positive direction, followed by 4 rounds of 5-isogenies in the negative direction. We can visualize this as a queue $[3+, 3+, 5-, 5-, 5-, 5-]$ (padded on the right with dummy isogenies for the remaining rounds up to m_1). CTIDH inflates the failure of each isogeny to that of the smallest prime in the batch to hide how often each prime is used; in our example, the failure probability is $1/3$.

This implies that the sets $S^{r\pm}$ contain precisely the r-th prime in the queue for the batch $\mathcal{B}_i$. With 14 batches and an equal chance for either orientation, we expect that each $S^{r\pm}$ will contain about 7 primes. Furthermore, each set $S^{r\pm}$ can contain only one prime per batch $\mathcal{B}_i$.

The small number of batches and the ordering of primes within the batches make CTIDH especially easy to break using our disorientation attack.

Components for CTIDH-512. Given enough samples, we construct the graphs $G^{r,s}$; the slightly higher failure probability of each isogeny (because of inflating) somewhat increases the chances of missing torsion and torsion noise. The distance of the root curves $E^{r,s}$ to the non-faulted curve E_B is bounded by the number of batches. Per round r, the sum of the distances of $E^{r,\pm}$ to E_B is at most 14, so we expect the distance to be about 7.

The distance between two graphs $G^{r,s}$ and $G^{(r+1),s}$ is often much smaller. We focus on positive orientation (the negative case is analogous). The distance between $G^{r,+}$ to $G^{(r+1),+}$ is given by the set difference of $S^{r,+}$ and $S^{(r+1),+}$. If these sets are disjoint and all primes in round r and $r+1$ are positive, the distance is 28, but we expect significant overlap: The set difference contains the indices i such that either the last ℓ_i-isogeny is computed in round r or the first ℓ_i-isogeny is computed in round $r+1$. Note that these replacements need not come in pairs. In the first case, the prime ℓ_i is replaced by the next isogeny ℓ_j from the same batch only if ℓ_j is also positive. In the second case, the prime ℓ_i might have followed a negative prime that preceded it in the batch.

Therefore, given $S^{r,+}$, one can very quickly determine $S^{(r+1),+}$ by leaving out some ℓ_i's or including subsequent primes from the same batch. In practice, this step is very easy. Finding one connection $E_B \to E^{r,+}$ determines some set $S^{r,+}$, which can be used to quickly find other sets $S^{r',+}$. This approach naturally also works going backwards, to the set $S^{(r-1),+}$.

Directed meet-in-the-middle. Using a meet-in-the-middle approach, we compute the neighborhood of E_B and all the roots $E^{r,\pm}$ (or components $G^{r,\pm}$) of distance 4. This connects E_B to all the curves at distance at most 8. Disregarding orientation and information on batches, if we have N curves that we want to connect, the naive search will require about $2 \cdot \binom{74}{4} \cdot N \approx 2^{21} \cdot N$ isogenies. The actual search space is even smaller as we can exclude all paths requiring two isogenies from the same batch.

Moreover, isogenies in batches are in ascending order. So, if in round r we see that the 3rd prime from batch $\mathcal{B}_i$ was used, none of the rounds $r' > r$ involves the first two prime, and none of the rounds $r' < r$ can use the fourth and later primes from the batch for that direction.

Late rounds typically contain many dummy isogenies and the corresponding faulty curves are especially close to the public key. We expect to rapidly recover $S^{r,\pm}$ for the late round curves, and work backwards to handle earlier rounds.

Required number of samples. In CTIDH, we can choose to inject a fault into the first call of Elligator or the second one. We do not see a clear benefit of prioritizing either call. Unlike for CSIDH and 1-point strategies, there is no clear benefit from targeting a specific round. Assume we perform c successful faults per round per Elligator call, expecting to get samples for both orientations per round. As CTIDH-512 performs 18 rounds (in practice typically up to 22 because of isogeny steps failing), we require $T = 18 \cdot 2 \cdot c$ successful flips. It seems possible to take $c = 1$ and hence $T = 36$ (or up to $T = 44$) samples.

With just one sample per round r (and per orientation s), the torsion effects will be significant and we will often not be able to recover $S^{r,s}$ precisely. Let $\tilde{S}^{r,s}$ denote the index set recovered for round r and sign s. We can correct for some of these errors, looking at $\tilde{S}^{r',\pm}$ for rounds r' close to r. Consider only primes from the same batch $\mathcal{B}$, then the following can happen:

- *No* prime from $\mathcal{B}$ is contained in either $\tilde{S}^{r,+}$ or $\tilde{S}^{r,-}$: all primes from $\mathcal{B}$ are done or *missing torsion* must have happened. We can examine the primes from the batch $\mathcal{B}$ which occur in neighboring rounds $\tilde{S}^{(r\pm1),\pm}$ and use the ordering in the batch to obtain guesses on which steps should have been computed if any.

- *One* prime from $\mathcal{B}$ is contained in $\tilde{S}^{r,+} \cup \tilde{S}^{r,-}$: we fix no errors.

- *Two* primes from $\mathcal{B}$ are contained in $\tilde{S}^{r,+} \cup \tilde{S}^{r,-}$: the smaller one must have come from torsion noise in a previous round and can be removed.

Remark 8.8. It is possible to skip certain rounds to reduce the number of samples, and recover the missing sets $S^{r,s}$ using information from the neighboring rounds. We did not perform the analysis as to which rounds can be skipped, we feel that already two successful faults per round are low enough.

Even a partial attack (obtaining information only from a few rounds) reveals a lot about the secret key thanks to the batches being ordered, and can reduce the search space for the secret key significantly. One may also select the rounds to attack adaptively, based on the information recovered from $S^{r,s}$.

Recovering the secret key. Once we recover all the sets $S^{r,s}$, the secret key can be found as $\mathfrak{a} = \prod_r \left(\prod_{i \in S^{r,+}} \mathfrak{l}_i \cdot \prod_{j \in S^{r,-}} \mathfrak{l}_j^{-1} \right)$. Note that as before, if we misidentify $S^{r,s}$ due to torsion effects, we may have to perform a small search to correct for the mistakes.

8.6.3 Other variants of CSIDH

In this section, we discuss some of the other implementations of CSIDH: all of these use `IsSquare` checks in the process of point sampling and are vulnerable to our attack. We analyze SIMBA [212], dummy-free implementations [**CCC+19**, **CR20**, 1], and SQALE [91].

SIMBA. Implementations using SIMBA [212] can be attacked similarly to CSIDH (cf. Section 8.6.1). SIMBA divides the n primes ℓ_i into m *prides* (batches), and each round only computes ℓ_i-isogenies from the same pride. That is, each round only involves up to $\lceil n/m \rceil$ isogenies, and the setup of the prides is publicly known. In each round, fewer isogenies are computed, the sets $S^{r,s}$ are smaller and the distances between the components $G^{r,s}$ are shorter. It is therefore easier to find isogenies connecting the components, and recover the secret key.

Dummy-free CSIDH. Dummy-free implementations [**CCC+19**, **CR20**, 1] replace pairs of dummy ℓ_i-isogenies by pairs of isogenies that effectively cancel each other [**CCC+19**]. This is due to the fact that $\mathfrak{l}_i * (\mathfrak{l}_i^{-1} * E) = \mathfrak{l}_i^{-1} * (\mathfrak{l}_i * E) = E$. Thus, computing one ℓ_i-isogeny in positive direction and one ℓ_i-isogeny in negative direction has the same effect as computing two dummy ℓ_i-isogenies. However, this approach requires fixing the parity of each entry of the private key e_i, e.g., by sampling only even numbers from $[-10, 10]$ to reach the same key space size as before. The implementation of [**CCC+19**] therefore suffers a slowdown of factor 2. Nevertheless, such dummy-free implementations mitigate certain fault attacks, such as skipping isogenies, which in a dummy-based implementation would directly reveal if the skipped isogeny was a dummy computation and give respective information on the private key. Dummy-free CSIDH [1] computes $|e_i|$ ℓ_i-isogenies per i in the appropriate direction, and then computes equally many ℓ_i isogenies in both directions which cancel out, until all required isogenies have been computed. For instance, for an even e_i sampled from $[-10, 10]$, choosing $e_i = 4$ would be performed by applying $\mathfrak{l}_i^1$ in the first 5 rounds, applying $\mathfrak{l}_i^{-1}$ in round 6 and 7, applying $\mathfrak{l}_i^1$ again in round 8 and 9, and finishing with $\mathfrak{l}_i^{-1}$ in round 10.

Notice that all isogenies start in the correct direction, and that we learn $|e_i|$ from disorientation faults if we know in which round the first $\mathfrak{l}_i$ is applied in the opposite direction. Therefore, if we apply the attack of Section 8.5 and learn all sets $S^{r,+}$ and $S^{r,-}$, we can determine e_i precisely. Even better, it suffices to only attack every second round: It is clear that each prime will have the same orientation in the third round as in the second round, in the fifth and fourth, et

cetera. Due to the bounds used in [1], large degree ℓ_i do not show up in later rounds, which decreases the meet-in-the-middle complexity of connecting the components $G^{r,+}$ and $G^{(r+1),+}$ for later rounds r.

SQALE. SQALE [91] only uses exponent bounds $e_i \in \{-1, 1\}$. To get a large enough key space, more primes ℓ_i are needed; the smallest instance uses 221 ℓ_i. SQALE uses a 2-point strategy and only requires one round (keeping in mind the isogeny computation may fail and require further rounds).

Set $S^+ = S^{1,+} = \{i \mid e_i = 1\}$ and $S^- = S^{1,-} = \{i \mid e_i = -1\}$. If the sampled points in round 1 have full order, the round 1 faulty curves are either:

- the 'twist' of E_B: all the directions will be flipped (if both points are flipped),
- or the curve $E^+ = (\prod_{S^+} \mathfrak{l}_i^{-2}) * E_B$, if the positive point was flipped,
- or the curve $E^- = (\prod_{S^-} \mathfrak{l}_i^{2}) * E_B$, if the negative point was flipped.

As $|S^+| \approx |S^-| \approx n/2 > 110$, we will not be able to find an isogeny to either of these curves using a brute-force or a meet-in-the-middle approach.

However, SQALE samples points randomly, and some of the isogeny computation will fail, producing faulty curves close to $E^{\pm}$ (and curves with the same orientation will be close to each other, as in Section 8.5.5). Getting enough faulty curves allows the attacker to get the orientation of all the primes ℓ_i, and the orientation of the primes is exactly the secret key in SQALE. We note that [**CR20**] in another context proposes to include points of full order into the system parameters and public keys such that missing torsion and torsion noise do not occur. If this is used for SQALE, our attack would not apply.

8.7 The `pubcrawl` tool

The post-processing stage of our attack relies on the ability to reconstruct the graph of connecting isogenies between the faulty CSIDH outputs. We solve this problem by a meet-in-the-middle neighborhood search in the isogeny graph, which is sufficiently practical for the cases we considered. In this section, we report on implementation details and performance results for our `pubcrawl` software.[1]

[1] The name refers to *crawl*ing the graph of *pub*lic keys, and to the act of visiting several pubs or bars in succession, consuming one or more drinks at each.

We emphasize that the software is *not* overly specialized to the fault-attack setting and may therefore prove useful for other "small" CSIDH isogeny searches appearing in unrelated contexts.

Algorithm. `pubcrawl` implements a straightforward meet-in-the-middle graph search: Grow isogeny trees from each input node simultaneously and check for collisions; repeat until there is only one connected component left. The set of admissible isogeny degrees ("support") is configurable, as are the directions of the isogeny steps ("sign", cf. CSIDH exponent vectors), the maximum number of isogeny steps to take from each target curve before giving up ("distance"), and the number of prime-degree isogenies done per graph-search step ("multiplicity", to allow for restricting the search to square-degree isogenies).

Size of search space.

The number of vectors in $\mathbb{Z}^n$ of 1-norm $\leq m$ is [99, § 3]

$$G_n(m) = \sum_{k=0}^{m} \binom{n}{k} \binom{m-k+n}{n}.$$

Similarly, the number of vectors in $\mathbb{Z}^n_{\geq 0}$ of 1-norm $\leq m$ equals

$$H_n(m) = \sum_{k=0}^{m} \binom{k+n-1}{n-1}.$$

Implementation. The tool is written in C++ using modern standard library features, most importantly hashmaps and threading. It incorporates the latest version of the original CSIDH software as a library to provide the low-level isogeny computations. Public-key validation is skipped to save time. The shared data structures (work queue and lookup table) are protected by a simple mutex; more advanced techniques were not necessary in our experiments.

We refrain from providing detailed benchmark results for the simple reason that the overwhelming majority of the cost comes from computing isogeny steps in a breadth-first manner, which parallelizes perfectly. Hence, both time and memory consumption scale almost exactly linearly with the number of nodes visited by the algorithm.

Concretely, on a server with two Intel Xeon Gold 6136 processors (offering a total of 24 hyperthreaded Skylake cores) using GCC 11.2.0, we found that

each isogeny step took between 0.6 and 0.8 core milliseconds, depending on the degree. Memory consumption grew at a rate of $\approx$ 250 bytes per node visited, although this quantity depends on data structure internals and can vary significantly. Example estimates based on these observations are given in Table 8.1.

There is no doubt that `pubcrawl` could be sped up if desired, for instance by computing various outgoing isogeny steps at once instead of calling the CSIDH library as a black box for each individually.

Code. The `pubcrawl` software is available at

`https://yx7.cc/code/pubcrawl/pubcrawl-latest.tar.xz`.

8.8 Hashed version

As briefly mentioned in Remark 8.1, the attacker-observable output in Diffie–Hellman-style key agreements is not the shared elliptic curve, but a certain derived value. Typically, the shared elliptic curve is used to compute a key k using a key derivation function, which is further used for symmetric key cryptography. So we cannot expect to obtain (the Montgomery coefficient of) a faulty curve E_t but only a derived value such as $k = \text{SHA-256}(E_t)$ or $\text{MAC}_k(\texttt{str})$ for some known fixed string `str`.

The attack strategies from Section 8.5 and Section 8.6 exploit the connections between the various faulty curves, but when we are only given a derived value, we are unable to apply isogenies. We argue that our attack, however, still extends to this more realistic setting as long as the observable value is computed deterministically from E_t and collisions do not occur.

For simplicity, we will refer to the observable values as *hashes* of the faulty curves. Starting from a faulty curve E_t, we assume we can easily compute the hashed value $H(E_t)$, but we cannot recover E_t from the hash $h = H(E_t)$.

As we lack the possibility to apply isogenies to the hashes, we must adapt the strategy from Section 8.5. Given a set of faulty curves, we can no longer generate the neighborhood graphs, nor find connecting paths between these graphs, and it is harder to learn the orientation of primes, which helped to reduce the possible degrees of the isogenies when applying `pubcrawl`. If we only see hashes of the faulty curves, we cannot immediately form the neighborhood graphs and determine orientations. But from the frequency analysis (Corollary 8.1), we can

still identify the two most frequent new hashes h_1, h_2 per round as the probable hashes of $H(E^{r,\pm})$.

Example 8.9 (CSIDH)**.** When faulting the first point, the two most common hashed values are our best guesses for the hashes of $E^{1,\pm}$. Considering faults in the second point, we guess $H(E^{2,\pm})$ to be the most common hashes that have not appeared in round 1. Similarly for later points.

To recover E given a hash $H(E)$, we run a one-sided `pubcrawl` search starting from E_B, where we hash all the curves we reach along the way, until we find a curve that hashes to $H(E)$. In practice, we run `pubcrawl` with one orientation (or both, in parallel) until we recognize $H(E^{r,\pm})$. Having identified $E^{r,\pm}$, we can then run a small neighborhood search around $E^{r,\pm}$ to identify the hashes of the faulty curves E_t close to $E^{r,\pm}$. In contrast to the unhashed version, in the hashed version we can only recover the faulty curves E_t by a one-sided search from a known curve E, instead of a meet-in-the-middle attack. In particular, the only known curve at the beginning of the attack is E_B.

Example 8.10 (CSIDH-512)**.** The distance of the curves $E^{r,s}$ to E_B is given by $|\{i \mid s \cdot e_i \geq r\}|$. Therefore, the curves $E^{5,\pm}$ have the smallest distance to E_B. Starting from the public key E_B, we thus first search the paths to the curves $E^{5,\pm}$. We do this by growing two neighborhoods (with positive and negative orientation) from E_B. Recall from Section 8.6.1 that the expected distance of the faulty curves is about $74/11 \approx 7$. But the distance from E_B to $E^{5,s}$ can be a lot larger (it is equal to $|\{e_i \mid s \cdot e_i = 5\}|$). Such large distances are rare: the probability of both $E^{5,\pm}$ having distance larger than 10 from E_B is, e.g., $\sum_{n=11}^{74-11} \sum_{m=11}^{74-n} \left(\binom{74}{n}\binom{74-n}{m}9^{74-n-m}\right)/11^{74} \approx 0.3\%$. Hence, we do expect to find a connection to at least one of the curves $E^{5,\pm}$ within distance 10, meaning that we expect the first connection to cost no more than $2\sum_{i=0}^{10}\binom{74}{i} \approx 2^{40.6}$ isogeny step evaluations and likely less for at least some $H(E_t)$ in the neighborhood. From there, we will identify orientation for some primes, hence the search will be more efficient at each successive step because we need to search through fewer than 74 primes.

Example 8.11 (CTIDH-512)**.** The faulty curves for *any* round in CTIDH are closer to the public key E_B than in the CSIDH case: it is 14 in the worst case (one prime per batch all having the same orientation) and the distance is 7 on average (Section 8.6.2). So the directed `pubcrawl` searches up to distance ≈ 7 (one with positive and one with negative orientation) are very likely to identify many of the hashed curves. Once we identify some faulty curves, we can identify

other faulty curves quickly by small neighborhood searches thanks to the extra ordered structure of the CTIDH keyspace. We also benefit from the slightly increased probability of failure leading to more curves in the neighborhood of $E^{r,s}$.

Summary. In the hashed version, the main difference compared to the approach in Section 8.6 is that we can no longer mount a meet-in-the-middle attack starting from E_B and from all faulty curves but can only search starting from E_B. Hence, we do not get the square-root speedup from meeting in the middle. Despite this increase in the costs, it is still possible to attack the hashed version. Other sizes and variants work the same way with the concrete numbers adjusted. The brute-force searches to connect the effective round-r curves in large CSIDH versions do get very expensive but will still remain cheaper than the security level for average gaps between E_B and $E^{r,s}$ for the maximum r values.

8.9 Exploiting the twist to allow precomputation

In this section, we use quadratic twists and precomputation to significantly speed up obtaining the private key $\mathfrak{a}$ given enough samples E_t, especially for the "hashed" version described in Section 8.8.

Using the twist. The attack target is a public key $E_B = \mathfrak{a} * E_0$. Previously (Section 8.4), we attacked the computation of $\mathfrak{a} * E_0$ with disorientation faults. In this section, we will use E_{-B} as the input curve instead: Negating B is related to inverting $\mathfrak{a}$ because $E_{-B} = \mathfrak{a}^{-1} * E_0$. Moreover, applying $\mathfrak{a}$ to E_{-B} gives us back the curve E_0 and faulting this computation then produces faulty curves close to the fixed curve E_0. As E_{-B} is the *quadratic twist* of E_B, we will refer to this attack variant as *using the twist*.

The main trick is that twisting induces a symmetry around the curve E_0. This can be used to speed up `pubcrawl`: the opposite orientation of E_t (starting from E_0) reaches E_{-t}, so we can check two curves at once. By precomputing a set $\mathcal{C}$ of curves of distance at most d to E_0, a faulty curve E_t at distance $d' \leq d$ is in $\mathcal{C}$ and can immediately be identified via a table lookup. Note that $\mathcal{C}$ can be precomputed once and for all, independent of the target instance, as for any secret key $\mathfrak{a}'$ the faulty curves end up close to E_0. The symmetry of E_{-t} and

E_t also reduces storage by half.

Finally, this twisting attack cannot be prevented by simply recognizing that E_{-B} is the twist of E_B and refusing to apply the secret $\mathfrak{a}$ to such a curve: An attacker can just as easily pick a random masking value $\mathfrak{z}$ and feed $\mathfrak{z} * E_{-B}$ to the target device. The faulty curves E_t can then be moved to the neighborhood of E_0 by computing $\mathfrak{z}^{-1} * E_t$ at some cost per E_t, or the attacker can precompute curves around $\mathfrak{z} * E_0$. The latter breaks the symmetry of E_t and E_{-t} and does not achieve the full speedup or storage reduction, but retains the main benefits.

Twisting CTIDH. The twisting attack is at its most powerful for CTIDH. As noted before, the sets $S^{r,\pm}$ are small in every round for CTIDH. The crucial observation is that in each round and for each orientation, we use at most one prime per batch (ignoring torsion noise, see Section 8.5.4). For a faulty curve E_t, the path $E_t \to E_0$ includes only steps with the same orientation and uses at most one prime per batch. With batches of size N_i, the total number of possible paths per orientation is $\prod_i (N_i+1)$, which is about $2^{35.5}$ for CTIDH-512. Hence, it is possible to precompute *all* possible faulty curves that can appear from orientation flips from *any* possible secret key $\mathfrak{a}$.

Extrapolating the performance of `pubcrawl` (Section 8.7), this precomputation should take no more than a few core years. The resulting lookup table occupies $\approx 3.4\,\text{TB}$ when encoded naively, but can be compressed to less than 250 GB using techniques similar to [283, § 4.3].

Twisting CSIDH. For this speed-up to be effective, the distance d we use to compute $\mathcal{C}$ must be at least as large as the smallest $|S^{r,\pm}|$. Otherwise, no faulty curves end up within $\mathcal{C}$. For CSIDH, the smallest such sets are $S^{r_{\max},\pm}$, where $r_{\max}$ is the maximal exponent permitted by the parameter; e.g., for CSIDH-512 $r_{\max} = 5$ and $S^{5,\pm}$ have an expected size ≈ 7. Precomputing $\mathcal{C}$ for $d \leq 7$ creates a set containing $\sum_{i=0}^{7} \binom{74}{i} \approx 2^{31}$ curves. Such a precomputation will either identify $S^{5,\pm}$ immediately, or allow us to find these sets quickly by considering a small neighborhood of the curves $E^{5,\pm}$.

Note that for all the earlier rounds $r < r_{\max}$, the sets $S^{r,s}$ include $S^{r_{\max},s}$. Therefore, if we have the orientation s and the set $S^{r_{\max},s}$, we can shift all the faulty curves by two steps for every degree in $S^{r_{\max},s}$. If we have misidentified the orientation, this shift moves the faulty curves in the wrong direction, away from E_0. This trick is particularly useful for larger r as eventually many isogenies need to be applied in the shifts and we will have identified the orientation of enough primes so that the search space for `pubcrawl` becomes small enough to be faster.

Twisting in the hashed version. Precomputation extends to the hashed version from Section 8.8: we simply precompute $\mathcal{C}'$ which instead of E_t includes $H(E_t)$ for all E_t in the neighborhood of E_0. Again, this works directly for attacking a hashed version of CTIDH and the effective round-$r_{\max}$ curves in CSIDH. To use precomputation for different rounds, one can replace the starting curve E_{-B} that is fed to the target device by the shift given exactly by the primes in $S^{r_{\max},s}$ (or, adaptively, by the part of the secret key that is known). This has the same effect as above: shifting all the curves E_t with the same orientation *closer* towards E_0, hopefully so that the $H(E_t)$ are already in our database. If they are not then likely the opposite orientation appeared when we faulted the computation.

Summary. The benefit of using the twist with precomputation is largest for the hashed versions: we need a brute force search from E_0 in any case, and so we would use on average as many steps per round as the precomputation takes. For the non-hashed versions, the expensive precomputation competes with meet-in-the-middle attacks running in square root time. This means that in the hashed version we do not need to amortize the precomputation cost over many targets and have a clear tradeoff between memory and having to recompute the same neighborhood searches all over again and again.

8.10 Countermeasures

In this section, we present countermeasures against disorientation fault attacks from Section 8.4. We first review previous fault attacks on CSIDH and their countermeasures, as well as their influence on our attack in Section 8.10.1. We then discuss new countermeasures for one-point sampling from CSIDH and Elligator in Section 8.10.2, and estimate the costs of the countermeasures in Section 8.10.3.

8.10.1 Previous fault attacks and countermeasures

One way to recover secret keys is to target dummy isogenies with faults [69, 196].

Although these attacks are implementation-specific, the proposed countermeasures impact our attack too. Typically, real isogenies are computed prior to dummy isogenies, but the order of real and dummy isogenies can be ran-

domized [69, 196] with essentially no computational overhead. When applied to dummy-based implementations, e.g., from [212, 236], this randomization means dummy isogenies can appear in different rounds for each run, which makes the definitions of the curves $E^{r,\pm}$ almost obsolete. However, we can instead simply collect many faulted round-1 samples. Each faulty curve E_t reveals a different set S_t due to the randomization, and with enough samples, a statistical analysis will quickly reveal all the $e_{i,j}$ just from the number of appearances among the sets S_t, again recovering the secret key.

Adapted to CTIDH, there are two possible variants of this randomization countermeasure: One could either keep the queue of real isogenies per batch as described above, but insert dummy isogenies randomly instead of at the end of the queue, or fully randomize the order of isogeny computations per batch including the dummy operations. In the first case, faulting round r if a dummy isogeny is computed in batch $\mathcal{B}_i$ means that no prime from this batch appears in the missing set. This effect is the same as missing torsion and thus our attack remains feasible. The net effect matches increased failure probabilities p_i and the larger neighborhoods simplify finding orientations. Note also that p_i is inflated more for batches with more dummy isogenies. In the second case when the entire queue is randomized, the same arguments as for CSIDH apply, and we can recover the secret key from statistical information with round-1 samples only.

Many fault attacks produce invalid intermediate values. In [69] some low-level protections for dummy isogenies to detect fault injections are proposed. This approach does not prevent our disorientation attack, and is orthogonal to our proposed countermeasures. Its performance overhead for the CSIDH-512 implementation from [236] is reported to be 7%.

Faulting memory locations can identify dummy isogenies [**CamposKM21**]. In addition to the countermeasures above, the authors of [**CamposKM21**] recommend using dummy-free implementations when concerned about fault attacks, with a roughly twofold slowdown [**CCC+19**]. However, as described in Section 8.6.3, dummy-free implementations are vulnerable to disorientation faults too.

Lastly, [69] reports that its fault attack theoretically could lead to disorientation of a point. Although the probability for this to happen is shown to be negligible, the authors of [69] propose to counter this attack vector by checking the field of definition of each isogeny kernel generator (point R in Step 7 of Algorithm 4). This is rather expensive, with an overhead of roughly 30% for the implementation from [236], but also complicates the disorientation faults proposed in this work. We further discuss this in Section 8.10.2. We note that our

countermeasures are significantly cheaper, but do not prevent the theoretical fault effect from [69].

8.10.2 Protecting square checks against fault attacks

The attack described in Section 8.4 can be applied to all implementations of CSIDH that use a call to `IsSquare` to determine the orientations of the involved point(s). The main weakness is that the output of `IsSquare` is always interpreted as $s = 1$ or $s = -1$, and there is no obvious way of reusing parts of the computation to verify that the output is indeed related to the x-coordinate of the respective point. For instance, faulting the computation of the Legendre-input $z = x^3 + Ax^2 + x$ results in a square check for a point unrelated to the actual x-coordinate in use, and yields a fault success probability of 50%.

Repeating square checks.

One way to reduce the attacker's chances for a successful fault is to add redundant computations and repeat the execution of `IsSquare` k times. In principle, this means that the attacker has to fault all k executions successfully, hence reducing the overall fault success probability to $1/2^k$. However, if an attacker manages to reliably fault the computation of z or the Legendre symbol computation or to skip instructions related to the redundant computations, they might be able to circumvent this countermeasure.

Repeated square checks have been proposed for a different fault attack scenario [69]. There, `IsSquare` is used to verify the correct orientation for each point that generates an isogeny kernel. However, this countermeasure significantly impacts the performance of CSIDH, and could be bypassed as above.

Using y-coordinates.

In CSIDH, the field of definition of the y-coordinate determines the orientation of a point. So, another simple countermeasure relying on redundant computation is to work with both x- and y-coordinates, instead of x-only arithmetic. We can then easily recognize the orientation of each point. But this leads again to a significant performance loss due to having to keep y-coordinates during all point multiplications and isogeny evaluations. We expect that this countermeasure is significantly more expensive than repeating `IsSquare` k times for reasonable choices of k.

Using pseudo y-coordinates.

We propose a more efficient countermeasure: compute *pseudo* y-coordinates after sampling points. We sample a random x-coordinate and set $z = x^3 + Ax^2 + x$. If z is a square in $\mathbb{F}_p$, we can compute the corresponding y-coordinate $\tilde{y} \in \mathbb{F}_p$ through the exponentiation $\tilde{y} = \sqrt{z} = z^{(p+1)/4}$, and hence $\tilde{y}^2 = z$. Conversely, if z is a non-square in $\mathbb{F}_p$, the same exponentiation outputs $\tilde{y} \in \mathbb{F}_p$ such that $\tilde{y}^2 = -z$. Thus, as an alternative to `IsSquare`, we can determine the orientation of the sampled point by computing $z = x^3 + Ax^2 + x$, and the pseudo y-coordinate $\tilde{y} = z^{(p+1)/4}$. If $\tilde{y}^2 = z$, the point has positive orientation, if $\tilde{y}^2 = -z$ it has negative orientation. If neither of these cases applies, i.e., $\tilde{y}^2 \neq \pm z$, a fault must have occurred during the exponentiation, and we reject the point.

This method may seem equivalent to computing the sign s using `IsSquare` as it does not verify that z has been computed correctly from x. But having an output value $\tilde{y} \in \mathbb{F}_p$ instead of the `IsSquare` output -1 or 1 allows for a much stronger verification step in order to mitigate fault attacks on the point orientation. We present the details of the original CSIDH algorithm including this countermeasure in Algorithm 5.

Algorithm 5 Evaluation of CSIDH group action with countermeasure

Input: $A \in \mathbb{F}_p$ and a list of integers $(e_1, \ldots, e_n)$.
Output: $B \in \mathbb{F}_p$ such that $\prod [\mathfrak{l}_i]^{e_i} * E_A = E_B$

1: **while** some $e_i \neq 0$ **do**
2: Sample a random $x \in \mathbb{F}_p$, defining a point P.
3: Set $z \leftarrow x^3 + Ax^2 + x$, $\tilde{y} \leftarrow z^{(p+1)/4}$.
4: Set $s \leftarrow 1$ if $\tilde{y}^2 = z$, $s \leftarrow -1$ if $\tilde{y}^2 = -z$, $s \leftarrow 0$ otherwise.
5: Let $S = \{i \mid e_i \neq 0,\ \mathrm{sign}(e_i) = s\}$. **Restart** with new x if S is empty.
6: Let $k \leftarrow \prod_{i \in S} \ell_i$ and compute $Q' = (X_{Q'} : Z_{Q'}) \leftarrow [\frac{p+1}{k}]P$.
7: Compute $z' \leftarrow x^3 + Ax^2 + x$.
8: Set $X_Q \leftarrow s \cdot z' \cdot X_{Q'}$, $Z_Q \leftarrow \tilde{y}^2 \cdot Z_{Q'}$.
9: Set $Q = (X_Q : Z_Q)$.
10: **for each** $i \in S$ **do**
11: Set $k \leftarrow k/\ell_i$ and compute $R \leftarrow [k]Q$. If $R = \infty$, **skip** this i.
12: Compute $\varphi : E_A \to E_B$ with kernel $\langle R \rangle$.
13: Set $A \leftarrow B$, $Q \leftarrow \varphi(Q)$, and $e_i \leftarrow e_i - s$.
14: **return** A.

Steps 3 and 4 of Algorithm 5 contain our proposed method to determine

the orientation s without using `IsSquare`. In order to verify the correctness of these computations, we add a verification step. First, we recompute z via $z' = x^3 + Ax^2 + x$, and in case of a correct execution, we have $z = z'$. Thus, we have $s \cdot z' = \tilde{y}^2$, which we can use as verification of the correctness of the computations of s, z, z', and $\tilde{y}$. If this were implemented through a simple check, an attacker might be able to skip this check through fault injection. Hence, we perform the equality check through the multiplications $X_Q = s \cdot z' \cdot X_{Q'}$ and $Z_Q = \tilde{y}^2 \cdot Z_{Q'}$, and initialize $Q = (X_Q : Z_Q)$ only afterwards, in order to prevent an attacker from skipping Step 8. If $s \cdot z' = \tilde{y}^2$ holds as expected, this is merely a change of the projective representation of Q', and thus leaves the point and its order unchanged. However, if $s \cdot z' \neq \tilde{y}^2$, this changes the x-coordinate X_Q/Z_Q of Q to a random value corresponding to a point of different order. If Q does not have the required order before entering the isogeny loop, the isogeny computation will produce random outputs in $\mathbb{F}_p$ that do not represent supersingular elliptic curves with overwhelming probability. We can either output this random $\mathbb{F}_p$-value, or detect it through a supersingularity check (see [**CLM+18**, 27]) at the end of the algorithm and abort. The attacker gains no information in both cases. The supersingularity check can be replaced by a cheaper procedure [69]: Sampling a random point P and checking if $[p+1]P = \infty$ is much cheaper and has a very low probability of false positives, which is negligible in this case.

There are several ways in which an attacker may try to circumvent this countermeasure. A simple way to outmaneuver the verification is to perform the *same* fault in the computation of z and z', such that $z = z'$, but $z \neq x^3 + Ax^2 + x$. To mitigate this, we recommend computing z' using a different algorithm and a different sequence of operations, so that there are no simple faults that can be repeated in both computations of z and z' that result in $z = z'$. Faults in the computation of both z and z' then lead to random $\mathbb{F}_p$-values, where the probability of $z = z'$ is $1/p$.

The attacker may still fault the computation of s in Step 4 of Algorithm 5. However, this will now also flip the x-coordinate of Q to $-x$, which in general results in a point of random order, leading to invalid outputs. The only known exception is the curve $E_0 \colon y^2 = x^3 + x$: In this case, flipping the x-coordinate corresponds to a distortion map taking Q to a point of the same order on the quadratic twist. Thus, for E_0, flipping the sign s additionally results in *actually* changing the orientation of Q, so these two errors effectively cancel each other in Algorithm 5 and the resulting curve is the correct output curve after all.

Protecting Elligator. Recall from Section 8.4 that two-point variants of CSIDH, including CTIDH, use the Elligator map for two points simultaneously,

which requires an execution of `IsSquare` in order to correctly allocate the sampled points to P_+ and P_-.

We can adapt the pseudo y-coordinate technique from Section 8.10.2: we determine orientations and verify their correctness by applying this countermeasure for both P_+ and P_- separately. We dub this protected version of the Elligator sampling `Elligreator`. An additional benefit is that faulting the computations of the x-coordinates of the two points within Elligator (see [**CCC+19**]) is prevented by `Elligreator`.

In CTIDH, each round performs two Elligator samplings, and throws away one point respectively. Nevertheless, it is not known a priori which of the two points has the required orientation, so `Elligreator` needs to check *both* points anyway in order to find the point of correct orientation.

On the one hand, adding dummy computations, in this case sampling points but directly discarding some of them, might lead to different vulnerabilities such as safe-error attacks. On the other hand, sampling both points directly with `Elligreator` at the beginning of each round (at the cost of one additional isogeny evaluation) may lead to correlations between the sampled points, as argued in [**BBCCLMSS21**]. It is unclear which approach should be favored.

8.10.3 Implementation costs

Implementing this countermeasure is straightforward. While `IsSquare` requires an exponentiation by $(p-1)/2$, our pseudo y-coordinate approach replaces this exponent by $(p+1)/4$, which leads to roughly the same cost. (Note that neither has particularly low Hamming weight.) Furthermore, we require a handful of extra operations for computing z', X_Q, and Z_Q in Steps 7 and 8 of Algorithm 5. For the computation of z' we used a different algorithm than is used for the computation of z, incurring a small additional cost, for the reason discussed above. Therefore, using this countermeasure in a 1-point variant of CSIDH will essentially not be noticeable in terms of performance, since the extra operations are negligible in comparison to the overall cost of the CSIDH action.

In 2-point variants, we use `Elligreator`, which requires two exponentiations instead of one as Elligator does. Thus, the countermeasure is expected to add a more significant, yet relatively small overhead in 2-point variants as in CTIDH. CTIDH uses two calls to `Elligreator` per round, and both executions contain two pseudo-y checks respectively. We estimate the cost of our countermeasure in CTIDH-512. The software of [**BBCCLMSS21**] reports an exponentiation by $(p-1)/2$ to cost 602 multiplications (including squarings). Since CTIDH-512 requires roughly 20 rounds per run, we add two additional exponentiations by

$(p+1)/4$ per round, and these have almost the same cost of 602 multiplications, the overhead is approximately $2 \cdot 20 \cdot 602 = 24080$ multiplications. Ignoring the negligible amount of further multiplications we introduce, this comes on top of a CTIDH-512 group action, which takes 438006 multiplications on average. Thus, we expect the total overhead of our countermeasure to be roughly 5.5% in CTIDH-512.

Table 8.1: Example cost estimates per target curve for various `pubcrawl` instances, assuming each isogeny step takes 0.7 milliseconds and consumes 250 bytes. For example, an isogeny walk of length up to 10 between two given curves can be recovered using approximately 10 core days and 300 gigabytes of RAM, as it involves exploring *two* neighborhoods up to distance 5 from the two curves.

sign	|support|	distance	cardinality of search space	core time	memory
both	74	≤ 2	$11{,}101 \approx 2^{13.44}$	7.8 s	2.8 MB
both	74	≤ 3	$551{,}449 \approx 2^{19.07}$	6.4 min	137.9 MB
both	74	≤ 4	$20{,}549{,}801 \approx 2^{24.29}$	4.0 h	5.1 GB
both	74	≤ 5	$612{,}825{,}229 \approx 2^{29.19}$	5.0 d	153.2 GB
both	74	≤ 6	$15{,}235{,}618{,}021 \approx 2^{33.83}$	123.4 d	3.8 TB
both	74	≤ 7	$324{,}826{,}290{,}929 \approx 2^{38.24}$	7.2 y	81.2 TB
both	74	≤ 8	$6{,}063{,}220{,}834{,}321 \approx 2^{42.46}$	134.6 y	1.5 PB
both	74	≤ 9	$100{,}668{,}723{,}849{,}029 \approx 2^{46.52}$	2234.5 y	25.2 PB
one	74	≤ 2	$2{,}850 \approx 2^{11.48}$	2.0 s	712.5 kB
one	74	≤ 3	$73{,}150 \approx 2^{16.16}$	51.2 s	18.3 MB
one	74	≤ 4	$1{,}426{,}425 \approx 2^{20.44}$	16.6 min	356.6 MB
one	74	≤ 5	$22{,}537{,}515 \approx 2^{24.43}$	4.4 h	5.6 GB
one	74	≤ 6	$300{,}500{,}200 \approx 2^{28.16}$	2.4 d	75.1 GB
one	74	≤ 7	$3{,}477{,}216{,}600 \approx 2^{31.70}$	28.2 d	869.3 GB
one	74	≤ 8	$35{,}641{,}470{,}150 \approx 2^{35.05}$	288.8 d	8.9 TB
one	74	≤ 9	$328{,}693{,}558{,}050 \approx 2^{38.26}$	7.3 y	82.2 TB
both	37	≤ 2	$2{,}813 \approx 2^{11.46}$	2.0 s	703.2 kB
both	37	≤ 3	$70{,}375 \approx 2^{16.10}$	49.3 s	17.6 MB
both	37	≤ 4	$1{,}321{,}641 \approx 2^{20.33}$	15.4 min	330.4 MB
both	37	≤ 5	$19{,}880{,}915 \approx 2^{24.24}$	3.9 h	5.0 GB
both	37	≤ 6	$249{,}612{,}805 \approx 2^{27.90}$	2.0 d	62.4 GB
both	37	≤ 7	$2{,}691{,}463{,}695 \approx 2^{31.33}$	21.8 d	672.9 GB
both	37	≤ 8	$25{,}450{,}883{,}345 \approx 2^{34.57}$	206.2 d	6.4 TB
both	37	≤ 9	$214{,}483{,}106{,}715 \approx 2^{37.64}$	4.8 y	53.6 TB
one	37	≤ 3	$9{,}880 \approx 2^{13.27}$	6.9 s	2.5 MB
one	37	≤ 4	$101{,}270 \approx 2^{16.63}$	1.2 min	25.3 MB
one	37	≤ 5	$850{,}668 \approx 2^{19.70}$	9.9 min	212.7 MB
one	37	≤ 6	$6{,}096{,}454 \approx 2^{22.54}$	1.2 h	1.5 GB
one	37	≤ 7	$38{,}320{,}568 \approx 2^{25.19}$	7.5 h	9.6 GB
one	37	≤ 8	$215{,}553{,}195 \approx 2^{27.68}$	1.7 d	53.9 GB
one	37	≤ 9	$1{,}101{,}716{,}330 \approx 2^{30.04}$	8.9 d	275.4 GB

Chapter 9

Projective Radical Isogenies

9.1 Placeholder

The paper will go here.

9.2 Introduction

The first proposal of an isogeny-based Diffie-Hellman key exchange was done by Couveignes [**couveignes06**] and centered on the action of an ideal class group on a set of ordinary elliptic curves. Later Rostovtsev and Stolbunov [**StolbunovIsogenyStar**, 275] independently rediscovered it and recognized its potential as a possible post-quantum candidate. In the last decade, isogeny-based key exchange developed further, notably with SIDH in [**DBLP:conf/pqcrypto/JaoF11**, **DBLP:journals/jmc/FeoJP14**, 17]. In Asiacrypt 2018, Castryck, Lange, Martindale, Panny, and Renes introduced CSIDH (a *non-interactive* key exchange) as a reformulation of the Couveignes-Rostovtsev-Stolbunov system using supersingular curves defined over a prime field [**Castryck0MPR18**]. With the hope to improve the performance of CSIDH, Castryck and Decru proposed CSURF, which exploits 2-isogenies [80] on the surface of the isogeny graph. Later on, Castryck, Decru, and Vercauteren in Asiacrypt 2020 expanded on the ideas in CSURF to construct isogenies with small odd degree based on radical computations (N-th roots) [83]. Using radical isogenies, they claimed a speed-up of about 19% over CSIDH-512, however both of the implementations in [80] and [83] focus on non-constant-time instantiations. In particular, Castryck, Decru, and Vercauteren left the analysis of a

constant-time implementation of CSURF and radical isogenies as an open problem. A constant-time algorithm refers to an algorithm whose running time is independent of (or uncorrelated with) the secret input. This implies the variability in the running time depends on randomness and not on the leakage of information on secret values.

Dealing with constant-time implementations of CSIDH (and CSURF) can be tricky as there are multiple approaches, such as using dummy isogenies or a dummy-free approach. The first constant-time CSIDH instantiation is the procedure using dummy isogenies proposed by Meyer, Campos, and Reith in [**DBLP:conf/pqcrypto/MeyerCR19**], later improved by Onuki *et al.* in [**DBLP:conf/iwsec/OnukiAYT19**]. Subsequently, Cervantes-Vázquez *et al.* proposed a dummy-free variant of CSIDH [**DBLP:conf/latincrypt/Cervantes-Vazquez** and more recently, Banegas *et al.* presented CTIDH [21]. This covers the literature that we are aware of.

The general idea to make CSIDH implementations run in constant-time is to perform a fixed number m of isogenies of a certain degree ℓ_i, independent of the secret key e_i. For example, take the CSIDH-512 prime $p = 4 \cdot \prod_{i=1}^{74} \ell_i - 1$, where ℓ_1 up to ℓ_{73} are the smallest 73 odd prime numbers and $\ell_{74} = 587$. Let $E/\mathbb{F}_p \colon y^2 = x^3 + Ax^2 + x$ be a supersingular Montgomery curve with $(p+1)$ rational points. Assuming we require exactly $m = 5$ isogenies per ℓ_i, then our key space corresponds with the integer exponent[1] vectors $(e_1, \ldots, e_{74}) \in [\![-m \,..\, m]\!]^{74}$. A dummy-based variant of constant-time CSIDH performs $|e_i|$ secret ℓ_i-isogenies and then proceeds by performing $(m - |e_i|)$ dummy-isogenies. The ℓ_i-isogeny kernel belongs to either $E[\pi - 1]$ or $E[\pi + 1]$, which is determined by the sign of e_i. A dummy-free variant (which prevents e.g. fault injection attacks) does not perform the $(m - |e_i|)$ dummy-isogeny constructions, but instead requires e_i to have the same parity as m. It then alternates between using kernels in $E[\pi - 1]$ and $E[\pi + 1]$ in such a way that one effectively *applies* e_i isogenies while *performing* m isogenies.

The experiments presented in [83] suggest a speed-up of about 19% when using radical isogenies instead of Vélu's formulas (for a prime of 512 bits). As mentioned above, these experiments focused on a non-constant-time Magma implementation for both the group-action evaluation and the chain of radical isogenies. More specifically, the Magma-code implementation of [83] performs field inversions in variable time depending on the input. Furthermore, the implementation computes exactly $|e_i|$ ℓ_i-radical isogenies, where $e_i \in [\![-m_i \,..\, m_i]\!]$ is a secret exponent of the private key (for instance when $e_i = 0$ the group

[1] The word *exponent* comes from the associated group action, see Section 9.3.2

action is trivial). Clearly, when measuring random non-constant time instances of CSURF or radical isogenies the average number of ℓ_i-radical isogenies to be performed is $\frac{m_i}{2}$, whereas in constant-time implementations the number of isogenies of degree ℓ_i is the fixed bound m_i.

A straightforward constant-time implementation of CSURF and radical isogenies would replace all non-constant-time techniques with constant-time techniques. This would, however, drastically reduce the performance of CSURF and radical isogenies, as inversions become costly and we need to perform more (dummy) isogenies per degree. Furthermore, radical isogenies currently do not use relatively 'cheap' Vélu isogenies of low degree because they are replaced by relatively expensive radical isogenies. Such an implementation would be outperformed by any state-of-the-art CSIDH implementation in constant-time.

Contributions. In this paper, we are interested in constant-time implementations of CSURF and radical isogenies. We present two improvements to radical isogenies which reduce their algorithmic cost. Then, we analyze the cost and efficiency of constant-time CSURF and radical isogenies, and benchmark their performance when implemented in CSIDH in number of finite field multiplications. More concretely, our contributions are

1. *fully projective radical isogenies*, a non-trivial reformulation of radical isogenies in projective coordinates, and of the required isomorphisms between curve models. This allows us to perform radical isogenies without leaving the projective coordinates used in CSIDH. This saves an inversion per isogeny and additional inversions in the isomorphisms between curve models, which in total reduces the cost of radical isogenies in constant-time by almost 50%.

2. a *hybrid strategy* to integrate radical isogenies in CSIDH, which allows us to 're-use' torsion points that are used in the CSIDH group action evaluation to initiate a 'chain' of radical isogenies, and to keep cheap low degree Vélu isogenies. This generalizes the 'traditional' CSIDH evaluation, optimizes the evaluation of radical isogenies, and does not require sampling an extra torsion point to initiate a 'chain' of radical isogenies.

3. a *cost analysis* of the efficiency of radical isogenies in constant-time, which describes the overall algorithmic cost to perform radical isogenies, assuming the aforementioned improvements. We show that, although these improvements greatly reduce the total cost in terms of finite field operations, radical isogenies of degree 5, 7, 11 and 13 are too costly in comparison to

Vélu and $\sqrt{\text{élu}}$ isogenies. We conclude that only radical isogenies of degree 4 and 9 are an improvement to 'traditional' CSIDH. Furthermore, we show that radical isogenies scale worse than Vélu isogenies with regards to the size of the base field, which reduces the effectiveness of CSURF and radical isogenies in comparison to CSIDH for large primes.

4. the *first constant-time implementation* of CSURF and radical isogenies, optimized with concern to the exponentiations used in radical isogenies, and optimal bounds and approximately optimal strategies as in [**DBLP:journals/jmc/H HLKA**, 95], which allow for a more precise comparison in performance between CSIDH, CSURF and implementations using radical isogenies (CRADS) than [80] and [83]. Our Python-code implementation allows isogeny evaluation strategies using both traditional Vélu and $\sqrt{\text{élu}}$ formulas, as well as radical isogenies and 2-isogenies (on the surface), and can thus be used to compare CSURF and CRADS against a state-of-the-art constant-time implementation of CSIDH.

5. a *performance benchmark* of CSURF and CRADS in comparison to 'traditional' CSIDH, in total finite field operations. Our benchmark is more accurate than the original benchmarks from [80] and [83] and shows that the 5% and 19% speed-up (respectively) diminishes to roughly 3% in a precise constant-time comparison. These results gives a detailed view of the performance of radical isogenies in terms of finite field operations, and their performance when increasing the size of the base field. We show that in low parameter sets, with the additional cost of moving to constant-time, CSURF-512 and CRADS-512 perform a bit better than CSIDH-512 implementations, with a 2.53% and 2.15% speed-up respectively. Additionally, with the hybrid strategy this speed-up is slightly larger for small primes. However, we get no speed-up for larger base fields: For primes of 1792 bits and larger, CSIDH outperforms both CSURF and CRADS due to the better scaling of Vélu isogenies in comparison to radical isogenies.

The (Python) implementation used in this paper is freely available at

`https://github.com/Krijn-math/Constant-time-CSURF-CRADS`.

The results from the benchmark answer the open question from Castryck, Decru, and Vercauteren in [83]: in constant-time, the CSIDH protocol gains only a small speed-up by using CSURF or radical isogenies, and only for small primes. Even stronger, our hybrid strategy shows that radical isogenies require significant improvements to make them cost effective, as computing a radical per

isogeny is in most cases too expensive. Our results illustrate that constant-time CSURF and radical isogenies perform worse than large CSIDH instantiations (i.e. $\log(p) \geq 1792$), at least at the level of finite field operations.[2]

Outline. In Section 9.3, we recap the theoretical preliminaries on isogenies, CSIDH, CSURF, and radical isogenies. In Section 9.4, we introduce our first improvement: fully projective radical isogenies. This allows us to analyse the effectiveness and cost of constant-time radical isogenies in Section 9.5. Then we improve the integration of radical isogenies in CSIDH in Section 9.6, using a hybrid strategy. In Section 9.7, we benchmark constant-time CSIDH, CSURF and CRADS in terms of finite field operations, using state-of-the-art techniques for all three. Finally, in Section 9.8 we present our conclusions concerning the efficiency of radical isogenies in comparison to CSIDH in constant-time.

9.3 Preliminaries

In this section we describe the basics of isogenies, CSIDH, CSURF and radical isogenies.

Given two elliptic curves E and E' over a prime field $\mathbb{F}_p$, an isogeny is a morphism $\phi : E \to E'$ such that $\mathcal{O}_E \mapsto \mathcal{O}_{E'}$. A separable isogeny ϕ has a degree $\deg(\phi)$ equal to the size of its kernel, and for any isogeny $\phi : E \to E'$ there is a unique isogeny $\hat{\phi} : E' \to E$ called the *dual isogeny*, with the property that $\hat{\phi} \circ \phi = [\deg(\phi)]$ is the scalar point multiplication on E. A separable isogeny is uniquely defined by its kernel and *vice versa*; a finite subgroup $G \subset E(\overline{\mathbb{F}_p})$ defines a unique separable isogeny $\phi_G : E \to E/G$ (up to isomorphism).

Vélu's formulas [**Velu71**] provide the construction and evaluation of separable isogenies with cyclic kernel $G = \langle P \rangle$ for some $P \in E(\overline{\mathbb{F}_p})$. Both the isogeny construction of ϕ_G and the evaluation of $\phi_G(R)$ for a point $R \in E(\overline{\mathbb{F}_p})$ have a running time of $O(\#G)$, which becomes infeasible for large subgroups G. A new procedure presented by Bernstein, De Feo, Leroux, and Smith in ANTS-2020 based on the baby-step giant-step algorithm decreases this cost to $\tilde{O}(\sqrt{\#G})$ finite field operations [**BernsteinFLS20**]. We write this procedure as $\sqrt{\text{élu}}$. This new approach is based on multi-evaluations of a given polynomial, although at its core it is based on traditional Vélu's formulas.

[2] We explicitly do not focus on performance in clock cycles; a measurement in clock cycles (on our python-code implementation) could give the impression that the underlying field arithmetic is optimized, instead of the algorithmic performance.

Isogenies from E to itself are *endomorphisms*, and the set of all endomorphisms of E forms a ring, which is usually denoted as $\mathrm{End}(E)$. The scalar point multiplication map $(x, y) \mapsto [N](x, y)$ and the Frobenius map $\pi : (x, y) \mapsto (x^p, x^p)$ are examples of such endomorphisms over the finite field of characteristic p. In particular, the order $\mathcal{O} \cong \mathbb{Z}[\pi]$ is a subring of $\mathrm{End}(E)$. An elliptic curve E is *ordinary* if it has a (commutative) endomorphism ring isomorphic to a suborder $\mathcal{O}$ of the ring of integers $\mathcal{O}_K$ for some quadratic number field K. A *supersingular* elliptic curve has a larger endomorphism ring: $\mathrm{End}(E)$ is isomorphic to an order $\mathcal{O}$ in a quaternion algebra, and thus non-commutative.

9.3.1 CSIDH and its Surface

CSIDH works with the smaller (commutative) subring $\mathrm{End}_p(E)$ of $\mathrm{End}(E)$, which are rational endomorphisms of a supersingular elliptic curve E. This subring $\mathrm{End}_p(E)$ is isomorphic to an order $\mathcal{O} \subset \mathcal{O}_K$. As both $[N]$ and π are defined over $\mathbb{F}_p$, we get $\mathbb{Z}[\pi] \subset \mathrm{End}_p(E)$. To be more precise, the CSIDH protocol is based on the commutative action of the class group $\mathcal{Cl}(\mathcal{O})$ on the set $\mathcal{E}\ell\ell_p(\mathcal{O})$ of supersingular elliptic curves E such that $\mathrm{End}_p(E)$ is isomorphic to the specific order $\mathcal{O} \subset \mathcal{O}_K$. The group action for an ideal class $[\mathfrak{a}] \in \mathcal{Cl}(\mathcal{O})$ maps a curve $E \in \mathcal{E}\ell\ell_p(\mathcal{O})$ to another curve $[\mathfrak{a}] \star E \in \mathcal{E}\ell\ell_p(\mathcal{O})$ (see Section 9.3.2). Furthermore, the CSIDH group action is believed to be a *hard homogeneous space* [**couveignes06**] that allows a Merkle-Diffie-Hellman-like key agreement protocol with commutative diagram

$$\begin{array}{ccc} E & \xrightarrow{\quad \mathfrak{a} \quad} & [\mathfrak{a}] \star E \\ \Big\downarrow{\scriptstyle \mathfrak{b}} & & \Big\downarrow{\scriptstyle \mathfrak{b}} \\ [\mathfrak{b}] \star E & \xrightarrow{\quad \mathfrak{a} \quad} & [\mathfrak{a}\mathfrak{b}] \star E \end{array}$$

The original CSIDH protocol uses the set $\mathcal{E}\ell\ell_p(\mathcal{O})$ with $\mathcal{O} \cong \mathbb{Z}[\pi]$ and $p = 3 \mod 4$ (named the *floor*). To also benefit from 2-isogenies, the CSURF protocol switches to elliptic curves on the *surface* of the isogeny graph, that is, $\mathcal{E}\ell\ell_p(\mathcal{O})$ with $\mathcal{O} \cong \mathbb{Z}[\frac{1+\pi}{2}]$. Making 2-isogenies useful requires $p = 7 \mod 8$.

9.3.2 The Group Action of CSIDH and CSURF

The traditional way of evaluating the group action of an element $[\mathfrak{a}] \in \mathcal{Cl}(\mathcal{O})$ is by using 'traditional' Vélu's [**Velu71**] or $\sqrt{\text{élu}}$ [**BernsteinFLS20**] formulas.

The group action maps $E \to [\mathfrak{a}] \star E$ and can be described by the kernel $E[\mathfrak{a}]$ of an isogeny $\phi_{\mathfrak{a}}$ of finite degree. Specifically, $[\mathfrak{a}] \star E = E/E[\mathfrak{a}]$ where

$$E[\mathfrak{a}] = \bigcap_{\phi \in \mathfrak{a}} \ker(\phi).$$

In both CSIDH and CSURF, we apply specific elements $[\mathfrak{l}_i] \in \mathcal{C}\ell(\mathcal{O})$ such that $\mathfrak{l}_i^{\pm 1} = (\ell_i, \pi \mp 1)$ and ℓ_i is the i-th odd prime dividing $(p+1)$. For $\mathfrak{l}_i$, we have

$$E[\mathfrak{l}_i^{\pm 1}] = E[\ell_i] \cap E[\pi \mp 1],$$

where $P \in E[\ell_i]$ means P is a point of order ℓ_i and $P \in E[\pi \mp 1]$ implies $\pi(P) = \pm P$, so P is either an $\mathbb{F}_p$-rational point or a zero-trace point over $\mathbb{F}_{p^2}$. Thus, the group action $E \to [\mathfrak{l}_i^{\pm 1}] \star E$ is usually calculated by sampling a point $P \in E[\mathfrak{l}_i^{\pm 1}]$ and applying Vélu's formulas with input point P. A secret key for CSIDH is then a vector (e_i), which is evaluated as $E \to \prod_i [\mathfrak{l}_i]^{e_i} \star E$. CSURF changes the order $\mathcal{O}$ used to $\mathbb{Z}[\frac{1+\pi}{2}]$ to also perform 2-isogenies on the surface of the isogeny graph; these 2-isogenies do not require the sampling of a 2-order point but can instead be calculated by a specific formula based on radical computations.

Key space. Originally, the secret key $e = (e_i)$ was sampled from $[\![-m \,..\, m]\!]^n$ for some bound $m \in \mathbb{N}$. This was improved in [**HLKA**, **DBLP:conf/pqcrypto/MeyerCR19**, 95] by varying the bound m per degree ℓ_i (a weighted L_∞-norm ball). Further developments with regards to improving the key space are presented in [226], using an $(L_1 + L_\infty)$-norm ball, and in CTIDH ([21]). These methods can give significant speed-ups. In their cores, they rely on (variations of) Vélu isogenies to evaluate the group action. In [80, 83], the authors compare the performance of radical isogenies to CSIDH by using an unweighted L_∞-norm ball for CSIDH-512 versus a weighted L_∞-norm ball for the implementation using radical isogenies. This gives a skewed benchmark, which favors the performance of CSURF and CRADS. In this paper, to make a fair comparison to the previous work, we continue in the line of [**HLKA**, **DBLP:conf/pqcrypto/MeyerCR19**, 95] by using *weighted* L_∞-norm balls for the implementations of CSIDH, CSURF and CRADS. It remains interesting to analyse the impact of radical isogenies in key spaces that are not based on weighted L_∞-norm balls. As radical isogenies can easily be made to have exactly the same cost per degree (with only slightly extra cost), they are interesting to analyse with respect to CTIDH.

9.3.3 The Tate Normal Form

CSURF introduced the idea to evaluate a 2-isogeny by radical computations. [83] extends this idea to higher degree isogenies, using a different curve model than the Montgomery curve. To get to that curve model, fix an N-order point P on E with $N \geq 4$. Then, there is a unique isomorphic curve $E(b, c)$ over $\mathbb{F}_p$ such that P is mapped to $(0, 0)$ on $E(b, c)$. The curve $E(b, c)$ is given by Equation (9.1), and is called the Tate normal form of (E, P):

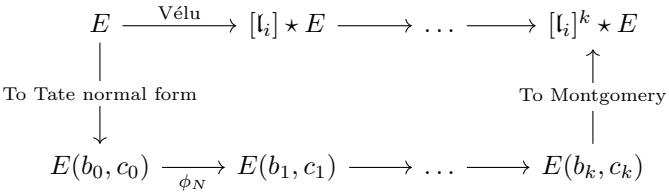
$$E(b, c)/\mathbb{F}_p \colon y^2 + (1 - c)x - by = x^3 - bx^2, \quad b, c \in \mathbb{F}_p. \tag{9.1}$$

The curve $E(b, c)$ has a non-zero discriminant $\Delta(b, c)$ and in fact, it can be shown that the reverse is also true: for $b, c \in \mathbb{F}_p$ such that $\Delta(b, c) \neq 0$, the curve $E(b, c)$ is an elliptic curve over $\mathbb{F}_p$ with $(0, 0)$ of order $N \geq 4$. Thus the pair (b, c) uniquely determines a pair (E, P) with P having order $N \geq 4$ on some isomorphic curve E over $\mathbb{F}_p$. In short, there is a bijection between the set of isomorphism classes of pairs (E, P) and the set of $\mathbb{F}_p$-points of $\mathbb{A}^2 - \{\Delta = 0\}$. The connection with *modular curves* is explored in more detail in [237].

9.3.4 Radical Isogenies

Let E_0 be a supersingular Montgomery curve over $\mathbb{F}_p$ and P_0 a point of order N with $N \geq 4$. Additionally, let $E_1 = E_0/\langle P_0 \rangle$, and P_1 a point of order N on E_1 such that $\hat{\phi}(P_1) = P_0$ where $\hat{\phi}$ is the dual of the N-isogeny $\phi \colon E_0 \to E_1$. The pairs (E_0, P_0) and (E_1, P_1) uniquely determine Tate normal parameters (b_0, c_0) and (b_1, c_1) with $b_i, c_i \in \mathbb{F}_p$.

Castryck, Decru, and Vercauteren proved the existence of a function ϕ_N that maps (b_0, c_0) to (b_1, c_1) in such a way that it can be applied iteratively. This computes a chain of N-isogenies without the need to sample points of order N per iteration. As a consequence, by mapping a given supersingular Montgomery curve $E/\mathbb{F}_p$ and some point P of order N to its Tate normal form, we can evaluate $E \to [\mathfrak{l}_i] \star E$ without any points (except for sampling P). Thus, it allows us to compute $E \to [\mathfrak{l}_i]^k \star E$ without having to sample k points of order N.



Notice that the top row and the bottom row of the diagram are isomorphic. The map ϕ_N is an elementary function in terms of b, c and $\alpha = \sqrt[N]{\rho}$ for a specific element $\rho \in \mathbb{F}_p(b, c)$: hence the name 'radical' isogeny. Over $\mathbb{F}_p$, an N-th root is unique whenever N and $p-1$ are co-prime (as the map $x \mapsto x^N$ is then a bijection). Notice that this in particular holds for all odd primes ℓ_i of a CSIDH prime $p = h \cdot \prod \ell_i - 1$ for a suitable cofactor h. Castryck, Decru, and Vercauteren provided the explicit formulas of ϕ_N for $N \in \{2, 3, 4, 5, 7, 9, 11, 13\}$. For larger degrees the formulas could not be derived yet. They also suggest the use of radical isogenies of degree 4 and 9 instead of 2 and 3, respectively.

Later work by Onuki and Moriya [237] provides similar radical isogenies on Montgomery curves instead of Tate normal curves. Although their results are of theoretical interest, they only provide such radical isogenies for degree 3 and 4. For degree 3, the use of degree 9 radical isogenies on Tate normal curves is more efficient, while for degree 4 the difference between their formulas and those presented in [83] are negligible. We, therefore, focus only on radical isogenies on Tate normal curves for this work.

9.4 Fully Projective Radical Isogenies

In this section we introduce our first improvement to radical isogenies: *fully projective* radical isogenies. These allow to us bypass all inversions required for radical isogenies. We perform (a) the radical isogenies on Tate normal curves in projective coordinates, and (b) the switch between the Montgomery curve and the Tate normal curve, and back, in projective coordinates. (a) requires non-trivial work which we explain in Section 9.4.1, whereas (b) is only tediously working out the correct formulas. The savings are worth it: (a) saves an inversion per radical isogeny and (b) saves numerous inversions in overhead costs. All in all, it is possible to remain in projective coordinates throughout the whole implementation, which saves about 50% in terms of finite field operations in comparison to affine radical isogenies in constant time.

9.4.1 Efficient Radicals for Projective Coordinates

The cost of an original (affine) radical isogenies of degree N in constant-time is dominated by the cost of the N-th root and one inversion per iteration. We introduce projective radical isogenies so that we do not require this inversion. In a constant-time implementation, projective radical isogenies save approximately 50% of finite field operations in comparison to affine radical isogenies. A straightforward translation to projective coordinates for radical isogenies would save an inversion by writing the Tate normal parameter b (when necessary c) as $(X : Z)$. However, this comes at the cost of having to calculate both $\sqrt[N]{X}$ and $\sqrt[N]{Z}$ in the next iteration. Using the following lemma, we save one of these exponentiations.

Lemma 9.1. Let $N \in \mathbb{N}$ such that $\gcd(N, p-1) = 1$. Write $\alpha \in \mathbb{F}_p$ as $(X : Z)$ in projective coordinates with $X, Z \in \mathbb{F}_p$. Then $\sqrt[N]{\alpha} = (\sqrt[N]{XZ^{N-1}} : Z)$.

Proof. As $\alpha = (X : Z) = (XZ^{N-1} : Z^N)$, we only have to show that the N-th root is unique. But N is co-prime with $p-1$, so the map $x \mapsto x^N$ is a bijection. Therefore, the N-th root $\sqrt[N]{\rho}$ is unique for $\rho \in \mathbb{F}_p$, so $\sqrt[N]{Z^N} = Z$. □

Crucially for radical isogenies, we want to compute N-th roots where $N = \ell_i$ for some i, working over the base field $\mathbb{F}_p$ with $p = h \cdot \prod_i \ell_i - 1$, and so for such an N we get $\gcd(N, p-1) = 1$. This leads to the following corollary.

Corollary 9.1. The representation $(XZ^{N-1} : Z^N)$ saves an exponentiation in the calculation of a radical isogeny of degree $N = \ell_i$ in projective coordinates.

This brings the cost of a projective radical isogeny of small degree ℓ_i down to below $1.25 \log(p)$. Compared with affine radical isogeny formulas in constant-time, which cost roughly two exponentiations, such projective formulas cost approximately half of the affine ones in terms of finite field operations. The effect this has for degrees 2, 3, 4, 5, 7 and 9 can be seen in Table 9.1. A similar approach as Lemma 9.1 works for radical isogenies of degree $N = 4$.

9.4.2 Explicit Projective Formulas for Low Degrees

We give the projective radical isogeny formulas for three cases: degree 4, 5 and 7. For larger degrees, it becomes increasingly more tedious to work out the projective isogeny maps. In the repository, we provide formulas for $N \in \{2, 3, 4, 5, 7, 9\}$.

Projective isogeny of degree 4. The Tate normal form for degree 4 is $E : y^2 + xy - by = x^3 - bx^2$ for some $b \in \mathbb{F}_p$. From [83], we get $\rho = -b$ and $\alpha = \sqrt[4]{\rho}$, and the affine radical isogeny formula is

$$\alpha \mapsto b' = -\frac{\alpha(4\alpha^2 + 1)}{(2\alpha + 1)^4}.$$

Projectively, write α as $(X : Z)$ with $X, Z \in \mathbb{F}_p$. Then the projective formula is

$$\begin{aligned} (X : Z) &\mapsto (X'Z'^4 : Z') \quad \text{with} \\ X' = (4X^2 + Z^2)XZ, &\quad \text{and} \quad Z' = 2X + Z. \end{aligned} \tag{9.2}$$

This isogeny is a bit more complex than it seems. First, notice that the denominator of the affine map is a fourth power. One would assume that it is therefore enough to map to $(X' : Z')$ and continue by taking only the fourth root of X' and re-use $Z' = \sqrt[4]{Z'^4}$. However, as $\gcd(4, p-1) = 2$, the root $\delta = \sqrt[4]{Z'}$ is not unique. Following [83] we need to find the root δ that is a quadratic residue in $\mathbb{F}_p$. We can force δ to be a quadratic residue: notice that $(X' : Z'^4)$ is equivalent to $(X'Z'^4 : Z'^8)$, so that taking fourth roots gives $(\sqrt[4]{X'Z'^4} : \sqrt[4]{Z'^8}) = (\sqrt[4]{X'Z'^4} : Z'^2)$, where we have forced the second argument to be a square, and so we get the correct fourth root.

Therefore, by mapping to $(X'Z'^4 : Z')$ we compute $\sqrt[4]{-b'}$ as $(\sqrt[4]{X'Z'^4} : Z'^2)$ using only one 4-th root. This allows us to repeat Equation (9.2) using only one exponentiation, without the cost of the inversion required in the affine version.

Projective isogeny of degree 5. The Tate normal form for degree 5 is $E : y^2 + (1-b)xy - by = x^3 - bx^2$ for some $b \in \mathbb{F}_p$. From [83] we get $\rho = b$ and $\alpha = \sqrt[5]{\rho}$, and the affine radical isogeny formula is

$$\alpha \mapsto b' = \alpha \cdot \frac{\alpha^4 + 3\alpha^3 + 4\alpha^2 + 2\alpha + 1}{\alpha^4 - 2\alpha^3 + 4\alpha^2 - 3\alpha + 1}.$$

Projectively, write α as $(X : Z)$ with $X, Z \in \mathbb{F}_p$. Then the projective formula is

$$\begin{aligned} (X : Z) &\mapsto (X'Z'^4 : Z') \quad \text{with} \\ X' &= X(X^4 + 3X^3Z + 4X^2Z^2 + 2XZ^3 + Z^4), \quad \text{and} \\ Z' &= Z(X^4 - 2X^3Z + 4X^2Z^2 - 3XZ^3 + Z^4). \end{aligned} \tag{9.3}$$

Notice that the image is $(X'Z'^4 : Z')$ instead of $(X' : Z') = (X'Z'^4 : Z'^5)$, following Lemma 9.1. This allows us in the next iteration to compute $\sqrt[5]{b} =$

$(\sqrt[5]{X} : \sqrt[5]{Z}) = (\sqrt[5]{X'Z'^4} : Z')$ using only one 5-th root. This allows us to repeat Equation (9.3) using only one exponentiation, without the cost of the inversion required in the affine version.

Projective isogeny of degree 7. The Tate normal form for degree 7 is $E : y^2 + (-b^2 + b + 1)xy + (-b^3 + b^2)y = x^3 + (-b^3 + b^2)x^2$ for some $b \in \mathbb{F}_p$, with $\rho = b^5 - b^4$ and $\alpha = \sqrt[7]{\rho}$. However, the affine radical isogeny is already too large to display here, and the projective isogeny is even worse. However, we can still apply Lemma 9.1. The projective isogeny maps to $(X'Z'^6 : Z')$ and in a next iteration we can compute $\alpha = \sqrt[7]{\rho} = \sqrt[7]{b^5 - b^4}$ as $(\sqrt[7]{X^4Z^2(X - Z)} : Z)$.

9.4.3 Cost of Projective Radical Isogenies per Degree

In Table 9.1, we compare the cost of affine radical isogenies to projective radical isogenies. In Table 9.2, we compare the cost in switching between the different curve models for affine and projective coordinates.

Table 9.1: Comparison between affine radical isogenies from [83] and the projective radical isogenies in this work. The letters **E**, **M**, **S**, **A** and **I** denote exponentiation, multiplication, squaring, addition and inversion respectively. The last column expresses the ratio projective/affine in terms of finite field multiplications over $\mathbb{F}_p$ for a prime of 512 bits, using close-to-optimal addition chains for exponentiation and inversion, assuming $\mathbf{S} = \mathbf{M}$ and ignoring **A**.

Degree	**Affine** ([83])	**Projective** (This work.)	**Ratio** projective/affine
2-isogeny	**E** + 4**M** + 6**A** + **I**	**E** + 3**M** + 5**S** + 10**A**	50.4%
3-isogeny	**E** + 6**M** + 3**A**	**E** + 2**M** + 10**A**	99.3%
4-isogeny	**E** + 4**M** + 3**A** + **I**	**E** + 6**M** + 4**S** + 3**A**	50.5%
5-isogeny	**E** + 7**M** + 6**A** + **I**	**E** + 8**M** + 6**S** + 18**A**	50.7%
7-isogeny	**E** + 24**M** + 20**A** + **I**	**E** + 14**M** + 4**S** + 64**A**	50.5%
9-isogeny	**E** + 69**M** + 58**A** + **I**	**E** + 61**M** + 10**S** + 202**A**	52.1%

In summary, fully projective radical isogenies are almost twice as fast as the original affine radical isogenies for constant-time implementations. Nevertheless, as we will see in the analysis of Section 9.5.1, the radical isogenies of degree 5, 7, 11 and 13 still perform worse than 'traditional' Vélu isogenies in realistic scenarios.

Table 9.2: Comparison between the cost of different functions to switch curve models, necessary to perform radical isogenies. Affine results from [83] and projective results from this work.

Function	Affine ([83])	Projective (This work.)	Ratio projective/affine
`Mont+ to Mont-`	$\mathbf{E}+\ \mathbf{M}+\ \mathbf{S}+\ 2\mathbf{A}+\ \mathbf{I}$	$\mathbf{E}+\ 2\mathbf{M}+\ 2\mathbf{S}+\ 4\mathbf{A}$	50.1%
`Mont- to Mont+`	$\mathbf{E}+\ \mathbf{M}+\ \mathbf{S}+\ 2\mathbf{A}+\ \mathbf{I}$	$\mathbf{E}\ +\ 2\mathbf{S}+\ 4\mathbf{A}$	50.0%
`Mont- to Tate4`	$7\mathbf{M}+\ \mathbf{S}+\ \mathbf{A}+\ \mathbf{I}$	$5\mathbf{M}+\ 8\mathbf{S}+\ 7\mathbf{A}$	2.1%
`Tate4 to Mont-`	$2\mathbf{E}+\ 3\mathbf{M}+\ \mathbf{S}+\ 7\mathbf{A}+2\mathbf{I}$	$2\mathbf{E}+\ 6\mathbf{M}+\ \mathbf{S}+11\mathbf{A}$	50.1%
`Full overhead CSURF`	$7\mathbf{E}+17\mathbf{M}+\ 6\mathbf{S}+19\mathbf{A}+5\mathbf{I}$	$7\mathbf{E}+18\mathbf{M}+16\mathbf{S}+35\mathbf{A}$	58.5%
`Mont+ to TateN`	$\mathbf{E}+\ 9\mathbf{M}+\ \mathbf{S}+11\mathbf{A}+\ \mathbf{I}$	$\mathbf{E}+13\mathbf{M}+\ 7\mathbf{S}+13\mathbf{A}$	51.1%
`TateN to Mont+`	$3\mathbf{E}+20\mathbf{M}+\ 7\mathbf{S}+34\mathbf{A}+\ \mathbf{I}$	$3\mathbf{E}+33\mathbf{M}+11\mathbf{S}+65\mathbf{A}$	75.9%
`Full overhead CRADS`	$4\mathbf{E}+34\mathbf{M}+14\mathbf{S}+54\mathbf{A}+4\mathbf{I}$	$4\mathbf{E}+54\mathbf{M}+22\mathbf{S}+83\mathbf{A}$	50.9%

9.5 Cost Analysis of Constant-time Radical Isogenies

In this section, we analyze the cost and effectiveness of radical isogenies on Tate normal curves in constant-time. In a simplified model, the cost of performing n radical ℓ-isogenies can be divided into 4 steps.

1. Sample a point P on E_A of order ℓ;
2. Map (E_A, P) to the (isomorphic) Tate normal curve E_0 with $P \mapsto (0,0)$;
3. Perform the radical isogeny formula n times: $E_0 \to E_1 \to \ldots \to E_n$;
4. Map E_n back to the correct Montgomery curve $E_{A'} = [\mathfrak{l}]^n \star E_A$.

In each of these steps, the cost is dominated by the number of exponentiations ($\mathbf{E}$) and inversions ($\mathbf{I}$). Using Tables 9.1 and 9.2, in an affine constant-time implementation, moving to the Tate normal curve (step 2) will cost close to $1\ \mathbf{E} + 1\ \mathbf{I}$, an affine radical isogeny (step 3) costs approximately $1\ \mathbf{E} + 1\ \mathbf{I}$ per isogeny, and moving back to the Montgomery curve (step 4) will cost about $3\ \mathbf{E} + 1\ \mathbf{I}$.

Inversions. In contrast to ordinary CSIDH, radical isogenies would require these inversions to be constant-time, as the value that is inverted can reveal valuable information about the isogeny walk related to the secret key. Two methods to compute the inverse of an element $\alpha \in \mathbb{F}_p$ in constant-time are 1) by

Fermat's little theorem[3]: $\alpha^{-1} = \alpha^{p-2}$, or 2) by masking the value that we want to invert with a random value $r \in \mathbb{F}_p$, computing $(r\alpha)^{-1}$ and multiplying by r again. Method 1 makes inversion as costly as exponentiation, while method 2 requires a source of randomness, which is an impediment from a crypto-engineering point of view. Using Fermat's little theorem almost doubles the cost of CSURF and of a radical isogeny in low degrees (2, 3, 4, 5, 7) and significantly increases the cost of a radical isogeny of degree 9, 11 or 13. Furthermore, such constant-time inversions increase the overhead of switching to Tate normal form and back to Montgomery form, which in total makes performing n radical isogenies less effective. Both methods of inversion are unfavorable from a crypto-engineering view, and thus we implement the fully projective radical isogenies from Section 9.4 to by-pass all inversions completely for radical isogenies.

Approximate cost of radical isogenies. We can now approximate the cost of evaluating fully projective radical isogenies in constant-time. We can avoid (most of) the cost of step 1 with a hybrid strategy (see Section 9.6). Projective coordinates avoid the inversion required in step 2 to move from the Montgomery curve to the correct Tate normal curve, and the inversion required in step 4 to move from the Tate normal curve back to the Montgomery curve. In step 3, projective radical isogenies save an inversion per isogeny, and so step 3 costs approximately n **E**. In total, performing n radical ℓ-isogenies therefore costs approximately $(n+4)$**E**.

At first sight, this approximated cost does not seem to depend on ℓ. However, there is some additional cost besides the exponentiation per isogeny in step 3, and this additional cost grows with ℓ. But, the cost of an exponentiation is larger than $\log_2(p)$ **M** and so overshadows the additional cost. For more details, see Table 9.1. For this analysis, the approximated cost fits for small degrees.

The cost of exponentiation is upperbounded by $1.5\log(p)$ by the (suboptimal) square-and-multiply method, assuming squaring (**S**) costs as much as multiplication (**M**). In total, we get the following approximate cost:

Lemma 9.2. The cost to perform n radical isogenies (using Tate normal curves) of degree $\ell \in \{5, 7, 9, 11, 13\}$ is at least

$$(n+4) \cdot \alpha \cdot \log_2(p),$$

finite field multiplications (**M**) where $\alpha \in [1, 1.5]$ depends on the method to perform exponentiation (assuming **S** = **M**).

[3]Bernstein and Yang [48] give a constant-time inversion based on gcd-computations. We have not implemented this, as avoiding inversions completely is cheaper.

9.5.1 Analysis of Effectiveness of Radical Isogenies

In this subsection, we analyze the efficiency of radical isogenies in comparison to Vélu isogenies, assuming the results from the previous sections. We argue that the cost of $(n+4)\cdot\alpha\cdot\log_2(p)$ from Lemma 9.2 for radical isogenies is too high and it is therefore not worthwhile to perform radical isogenies for degrees $5, 7, 11$ and 13. Degrees 2 and 3, however, benefit from the existence of radical isogenies of degree 4 and 9. Degree 4 and 9 isogenies cost only one exponentiation, but evaluate as two. This implies that performing radical isogenies is most worthwhile in degrees 2 and 3. We write 2/4 and 3/9 as shorthand for the combinations of degree 2 and 4, resp. degree 3 and 9 isogenies.

The three crucial observations in our analysis are

1. Current faster $\sqrt{\text{élu}}$ isogeny formulas require $\mathcal{O}(\sqrt{\ell^{\log_2 3}})$ field multiplications, whereas the cost of a radical isogeny scales as a factor of $\log_2(p)$(for more details, see [**BernsteinFLS20**] and [3]);

2. The group action evaluation first performs one block using $\sqrt{\text{élu}}$ isogeny formulas, and then isolates the radical isogeny computations. What is particularly important among these $\sqrt{\text{élu}}$ isogeny computations, is that removing one specific ℓ'-isogeny does not directly decrease the number of points that need to be sampled. Internally, the group action looks for a random point R and performs all the possible ℓ_i-isogenies such that $\left[\frac{p+1}{\ell_i}\right] R \neq \mathcal{O}$.

3. Replacing the smallest Vélu ℓ-isogeny with a radical isogeny could reduce the sampling of points in that specific Vélu isogeny block. This is because the probability of reaching a random point R of order ℓ is $\frac{\ell-1}{\ell}$, which is small for small ℓ. Additionally, the cost of verifying $\left[\frac{p+1}{\ell}\right] R \neq \mathcal{O}$ is about $1.5\log_2\left(\frac{p+1}{\ell}\right)$ point additions $\approx 9\log_2\left(\frac{p+1}{\ell}\right)$ field multiplications (for more details see [**DBLP:conf/latincrypt/Cervantes-Vazquez19**]). In total, sampling n points of order ℓ costs

$$\begin{aligned}\texttt{sampling}(n,p,\ell) &= 9\left\lfloor\frac{n\ell}{\ell-1}\right\rceil\log_2\left(\frac{p+1}{\ell}\right)\mathbf{M}\\ &\approx 9\left\lfloor\frac{n\ell}{\ell-1}\right\rceil(\log_2(p)-\log_2(\ell))\mathbf{M}.\end{aligned}$$

Nevertheless, using radical isogenies for these degrees does not save the sampling of n points, just a fraction of them. To be more precise, let $\ell' > \ell$ be the next smallest prime such that the group action requires n' ℓ'-isogenies and $\left\lfloor \frac{n\ell}{\ell-1} \right\rceil \geq \left\lfloor \frac{n'\ell'}{\ell'-1} \right\rceil$. Then the savings are given by their difference with respect to the cost sampling such torsion-points (see Equation (9.4)).

$$9 \left(\left\lfloor \frac{n\ell}{\ell - 1} \right\rceil - \left\lfloor \frac{n'\ell'}{\ell' - 1} \right\rceil \right) (\log_2(p) - \log_2(\ell))\mathbf{M}. \tag{9.4}$$

Whenever $\left\lfloor \frac{n\ell}{\ell-1} \right\rceil < \left\lfloor \frac{n'\ell'}{\ell'-1} \right\rceil$, using radical isogenies does not reduce the number of points that need to be sampled.

As an example for the cost in a realistic situation, we take the approximately optimal bounds analyzed in [**DBLP:conf/iwsec/OnukiAYT19**] and [95]. In both works, $\log_2(p) \approx 512$ and the first five smallest primes ℓ_i's in $\{3, 5, 7, 11, 13\}$ have bounds m_i that satisfy

$$\left\lfloor \frac{m_0\ell_0}{\ell_0 - 1} \right\rceil = \left\lfloor \frac{m_1\ell_1}{\ell_1 - 1} \right\rceil = \left\lfloor \frac{m_2\ell_2}{\ell_2 - 1} \right\rceil = \left\lfloor \frac{m_3\ell_3}{\ell_3 - 1} \right\rceil = \left\lfloor \frac{m_4\ell_4}{\ell_4 - 1} \right\rceil.$$

Thus, there are no savings concerning sampling of points when including small degree radical isogenies. Clearly, performing n radical ℓ-isogenies becomes costlier than using $\sqrt{\text{élu}}$ isogenies, and thus the above analysis suggests radical isogenies need their own optimal bounds to be competitive. The analysis is different for degree 2 and 3, where we can perform 4-and 9-isogenies in $\lceil \frac{n}{2} \rceil$ radical computations instead of n computations. In fact, 4-isogenies directly reduces the sampling of points by decreasing the bounds of the other primes ℓ_i's. Nevertheless, performing n radical isogenies takes at least $(n + 4)\log_2(p)$ field multiplications (Lemma 9.2), which implies higher costs (and then lower savings) for large prime instantiations. For example, a single radical isogeny in a 1024-bit field costs twice as much as a single radical isogeny in a 512-bit field, and in a 2048-bit field this becomes four times as much. These expected savings omit the cost of sampling an initial point of order ℓ_i, as we show in Section 9.6 how we can find such points with little extra cost with high probability.

9.5.2 Further Discussion

In this subsection, we describe the two further impacts on performance in constant-time and higher parameter sets in more detail: Radical isogenies scale badly to larger primes, as their cost scales with $\log(p)$, and dummy-free isogenies are more expensive, as we need to switch direction often for a dummy-free evaluation.

Radical isogenies do not scale well. Using the results in Tables 9.1 and 9.2, the cost of a single radical isogeny is approximately 600 finite field operations, with an overhead of about 2500 finite field operations for a prime of 512 bits. Thus, a CSURF-512 implementation (which uses 2/4- radical isogenies) or a CRADS-512 implementation (which uses 2/4- and 3/9- radical isogenies) could be competitive with a state-of-the-art CSIDH-512 implementation. However, implementations using radical isogenies scale worse than CSIDH implementations, due to the high cost of exponentiation in larger prime fields. For example, for a prime of 2048 bits, just the overhead of switching curve models is already over 8500 finite field operations, which is close to 1% of total cost for a 'traditional' CSIDH implementation. Therefore, CSIDH is expected to outperform radical isogenies for larger primes. In Section 9.7, we demonstrate this using a benchmark we have performed on CSIDH, CSURF, CRADS, and an implementation using the hybrid strategy we introduce in Section 9.6, for six different prime sizes, from 512 bits up to 4096 bits. These prime sizes are realistic: several analyses, such as [**BS20**, **Pei20**, 92], call the claimed quantum security of the originally suggested prime sizes for CSIDH (512, 1024 and 1792 bits) into question. We do not take a stance on this discussion, and therefore provide an analysis that fits both sides of the discussion.

Dummy-free radical isogenies are costly. Recall that radical isogenies require an initial point P of order N to switch to the right Tate normal form, depending on the direction of the isogeny. So, two kinds of curves in Tate normal form arise: P belongs either to $E[\pi-1]$ or to $E[\pi+1]$. Now, a dummy-free chain of radical isogenies requires (at some steps of the group action) to switch the direction of the isogenies, and therefore to switch to a Tate normal form where P belongs to either $E[\pi-1]$ or $E[\pi+1]$. As we switch direction $m_i - |e_i|$ times, this requires $m_i - |e_i|$ torsion points. That is, a dummy-free implementation of a chain of radical isogenies will require at least $(m_i - |e_i|)$ torsion points, which leaks information on e_i. We can make this procedure secure by sampling m_i points every time, but this costs too much. These costs could be decreased by

pushing points through radical isogenies, however, this is still not cost-effective. In any case, we will only focus on dummy-based implementations of radical isogenies.

9.6 A Hybrid Strategy for Radical Isogenies

In this section we introduce our second improvement to radical isogenies: a *hybrid strategy* for integrating radical isogenies into CSIDH. In [83], radical ℓ-isogenies replace Vélu ℓ-isogenies, and they are performed before the CSIDH group action, by sampling a point of order ℓ to initiate a 'chain' of radical ℓ-isogenies. Such an approach replaces cheap Vélu ℓ-isogenies with relatively expensive radical ℓ-isogenies and requires finding a point of order ℓ to initiate this 'chain'. The hybrid strategy combines the 'traditional' CSIDH group action evaluation with radical isogenies in an optimal way, so that we do not sacrifice cheap Vélu isogenies of low degree and do not require another point of order ℓ to initiate the 'chain'. This substantially improves the efficiency of radical isogenies.

Concretely, in 'traditional' CSIDH isogeny evaluation, one pushes a torsion point T through a series of ℓ-isogenies with Vélu's formulas. This implies that at the end of the series of Vélu isogenies, such a point T might still have suitable torsion to initiate a chain of radical isogenies. Re-using this point saves us having to specifically sample a torsion point to initiate radical isogenies. Furthermore, with this approach we can do *both* radical and Vélu isogenies for such ℓ where we have radical isogenies. We show this hybrid strategy generalizes CSIDH, CSURF and CRADS (an implementation with radical isogenies) and gives an improved approach to integrate radical isogenies on Tate normal curves in CSIDH.

In this section, V refers to the set of primes for which we have Vélu isogenies (i.e. all ℓ_i), and $R \subset V$ refers to the set of degrees for which we have radical isogenies. Currently, $R = \{3, 4, 5, 7, 9, 11, 13\}$.

9.6.1 A Hybrid Strategy for Integration of Radical Isogenies

Vélu isogenies for degree ℓ_i with $i \in R$ are much cheaper than the radical isogenies for those degrees, as a single radical isogeny always requires at least one exponentiation which costs $O(\log(p))$. The downside to Vélu isogenies is that they require a torsion point per isogeny. However, torsion points can be re-used for Vélu isogenies of many degrees by pushing them through the isogeny,

and so the cost of point sampling is amortized over all the degrees where Vélu isogenies are used. Thus, although for a single degree n radical isogenies are much cheaper than n Vélu isogenies, this does not hold when the cost of point sampling is distributed over many other degrees. The idea of our hybrid strategy is to do both types of isogeny per degree: we need to perform a certain amount of Vélu blocks in any CSIDH evaluation, so we expect a certain amount T_i of points of order ℓ_i. So, for $i \in R$, we can use $T_i - 1$ of these points to perform Vélu ℓ_i-isogenies, and use the last one to initiate the chain of radical ℓ_i-isogenies. Concretely, we split up the bound m_i for $i \in R$ into m_i^v and m_i^r. Here, m_i^v is the number of Vélu isogenies, which require m_i^v points of order ℓ_i, and m_i^r is the number of radical isogenies, which require just 1 point of order ℓ_i.

Thus, our *hybrid strategy* allows for evaluation of CSIDH with radical isogenies, in such a way that we can the following parameters

- for $i \in R$, m_i^r is the number of radical isogenies of degree ℓ_i,
- for $i \in V$, m_i^v is the number of Vélu/$\sqrt{\text{élu}}$ isogenies of degree ℓ_i.

For $i \in R$, we write $m_i = m_i^v + m_i^r$ for simplicity. What makes the difference with the previous integration of radical isogenies, is that this hybrid strategy allows R and V to overlap! That is, the hybrid strategy does not require you to pick between Vélu *or* radical isogenies for $\{3, 5, 7, 9, 11, 13\}$. In this way, hybrid strategies generalize both CSIDH/CSURF and CRADS:

Lemma 9.3. The CSIDH/CSURF group action evaluation as in [**Castryck0MPR18**] and [80], and the radical isogenies evaluation from [83] are both possible in this hybrid strategy.

Proof. Take $m_i^r = 0$ for all $i \in R$ to get the CSIDH/CSURF group action evaluation, and take $m_i^v = 0$ for all $i \in R$ to get the radical isogenies evaluation. □

In the rest of this section, we look at non-trivial hybrid parameters (i.e. there is some i such that both m_i^v and m_i^r are non-zero) to improve the performance of CSIDH and CRADS. As we can predict T_i (the number of points of order ℓ_i) in a full CSIDH group action evaluation, we can choose our parameter m_i^v optimally with respect to T_i: for $i \in R$, we take $m_i^v = T_i - 1$ and use the remaining point of order ℓ_i to initiate the 'chain' of radical ℓ_i isogenies of length m_i^r.

9.6.2 Choosing Parameters for Hybrid Strategy.

As we explained above, the parameter m_i^v is clear given T_i, and we are left with optimizing the value m_i^r. Denote the cost of the overhead to switch curve models by c_{overhead} and the cost of a single isogeny by $c_{\text{single}}(\ell)$, then the cost of performing the m_i^r radical isogenies is clearly $c = c_{\text{overhead}} + m_i^r \cdot c_{\text{single}}(\ell_i)$ (see also Lemma 9.2). Furthermore, given m_i^v, the increase in key space is

$$b = \log_2 \left(\frac{2 \cdot (m_i^r + m_i^v) + 1}{2 \cdot m_i^v + 1} \right).$$

So, we can minimize c/b for a given m_i^v (independent of p). This minimizes the number of field operations per bits of security. Notice that for degree 3, the use of 9-isogenies means we get a factor $\frac{1}{2}$ for the cost of single isogenies, as we only need to perform half as many.

It is possible that the 'optimal' number m_i^r is higher, when c/b is still lower than in CSIDH. However, we heuristically argue that such an optimum can only be slightly higher, as the increase in bits of security decreases quite rapidly.

9.6.3 Algorithm for Evaluation of Hybrid Strategy.

Evaluating the hybrid strategy requires an improved evaluation algorithm, as we need to re-use the 'left-over' torsion point at the right moment of a 'traditional' CSIDH evaluation. We achieve this by first performing the CSIDH evaluation for all $i \in V$ and decreasing m_i^v by 1 if a Vélu ℓ_i-isogeny is performed. If m_i^v becomes zero, we remove i from V, so that the next point of order ℓ_i initiates the 'chain' of radical isogenies of length m_i^r. After this, we remove i from R too.

Effectively, for $i \in V \cap R$, we first check if we can perform a Vélu ℓ_i-isogeny in the loop in lines 5-9 (a Vélu block). If m_i^v of these have been performed, we check in lines 11-14 if the 'left-over' point Q has order ℓ_i for some $i \in S \cap R$, so that it can initiate the 'chain' of radical ℓ_i-isogenies in line 13. Algorithm 6 does not go into the details of CSURF (i.e. using degree 2/4 isogenies), which can easily be added and does not interfere with the hybrid strategy. Furthermore, Algorithm 6 does not leak any timing information about the secret values e_i, only on m_i, which is public information. For simplicity's sake, we do not detail many lower-level improvements.

Algorithm 6 High-level evaluation of hybrid strategy for radical isogenies

Input: $A \in \mathbb{F}_p$, a key $(e_1, \ldots, e_n)$, a set V (Vélu isog.) and a set R (radical isog.).

Output: $B \in \mathbb{F}_p$ such that $\prod [\mathfrak{l}_i]^{e_i} \star E_A = E_B$

1: **while** $m_i \neq 0$ for $i \in V \cup R$ **do**
2: Sample $x \in \mathbb{F}_p$, set $s \leftarrow 1$ if $x^3 + Ax^2 + x$ is a square in $\mathbb{F}_p$, else $s \leftarrow -1$.
3: Let $S = \{i \in V \cup R \mid m_i^v \neq 0, \text{sign}(e_i) = s\}$. **Restart** if S is empty.
4: Let $k \leftarrow \prod_{i \in S} \ell_i$ and compute $T \leftarrow [(p+1)/k]P$.
5: **for** $i \in S \cap V$ **do**
6: Compute $Q \leftarrow [k/\ell_i]T$. If $Q = \infty$, **skip** this i and set $S \leftarrow S - \{i\}$.
7: Compute $\varphi : E_A \to E_B$ with kernel $\langle Q \rangle$ using Vélu.
8: When $e_i \neq 0$, set $A \leftarrow B$, $T \leftarrow \varphi(T)$, $e_i \leftarrow e_i - s$
9: Set $m_i^v \leftarrow m_i^v - 1$ and set $S \leftarrow S - \{i\}$
10: If $m_i^v = 0$, set $V \leftarrow V - \{i\}$
11: **for** $i \in S \cap R$ **do**
12: Compute $Q \leftarrow [k/\ell_i]T$. If $Q = \infty$, **skip** this i.
13: Compute $E_B = [\mathfrak{l}_i]^{e_i} \star E_A$ using m_i^r radical ℓ_i-isogenies
14: Set $A \leftarrow B$, $R \leftarrow R - \{i\}$, and **start over** at line 1
return A.

9.7 Implementation and Performance Benchmark

All the experiments presented in this section are centred on constant-time CSIDH, CSURF and CRADS implementations, for a base field of 512-, 1024-, 1792-, 2048-, 3072-, and 4096-bits. To be more precise, in the first subsection we restrict our experiments to i) the most competitive CSIDH-configurations according to [**HLKA**, 95], ii) the CSURF-configuration presented in [80, 83] and iii) the radical isogenies-configuration presented in [83] (i.e. without the hybrid strategy). As mentioned in Section 9.5, we only focus on dummy-based variants such as MCR-style [**DBLP:conf/pqcrypto/MeyerCR19**] and OAYT-style [**DBLP:conf/iwsec/OnukiAYT19**]. The experiments using only radical isogenies of degree 2/4 are labelled CSURF, whereas the experiments using both radical isogenies of degree 2/4 and 3/9 are labelled CRADS. In the second subsection, we integrate the hybrid strategy to our experiments, and focus on dummy-based OAYT-style implementations. This allows us to compare the improvement of the hybrid strategy against the previous section. When comparing totals, we assume one field squaring costs what a field multiplication costs ($\mathbf{S} = \mathbf{M}$). Primes used are of the form $p = h \cdot \prod_{i=1}^{74} \ell_i - 1$, with $h = 2^k \cdot 3$. The key space is about 2^{256}.

On the optimal exponent bounds (fixed number of ℓ_i-isogenies required), the results from [**HLKA**] give $\approx 0.4\%$ of saving in comparison to [95] (see Table 5 from [95]). The results from [**HLKA**] are mathematically rich: analysis on the permutations of the primes and the (integer) convex programming technique for determining an approximately optimal exponent bound. However, their current Matlab-based code implementation from [**HLKA**] only handles CSIDH-512 using OAYT-style prioritizing multiplicative-based strategies. Both works essentially give the approximate same expected running time, and by simplicity, we choose to follow [95], which more easily extends to any prime size (for both OAYT and MCR styles). Furthermore, all CSIDH-prime instantiations use the approximately optimal exponent bounds presented in [95].

To reduce the cost of exponentiations in radical isogenies, we used short addition chains (found with [208]), which reduces the cost from $1.5\log(p)$ (from square-and-multiply) to something in the range $[1.05\log(p), 1.18\log(p)]$. These close-to-optimal addition chains save at least 20% of the cost of an exponentiation used per (affine or projective) radical isogeny in constant-time.

Our CSURF and CRADS constant-time implementations evaluate the group action by first performing the evaluation as CSIDH does on the floor of the isogeny graph, with the inclusion of radical isogenies as in Algorithm 6. Afterwards we move to the surface to perform the remaining 4-isogenies. So, the only

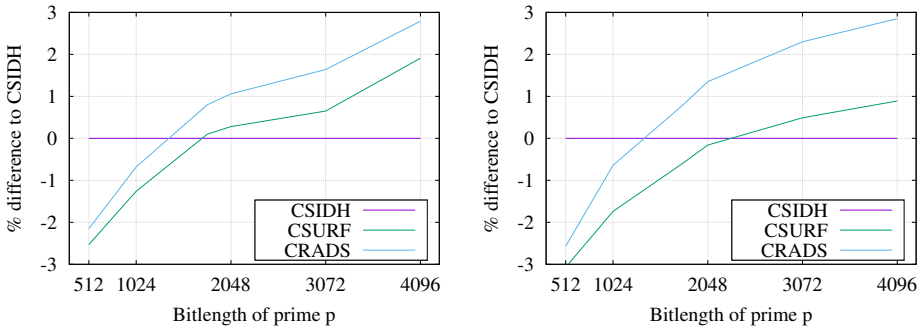
curve arithmetic required is on Montgomery curves of the form $E/\mathbb{F}_p : By^2 = x^3 + Ax^2 + x$. Concluding, we compare three different implementations which we name CSIDH, CSURF and CRADS. The CSIDH implementation uses traditional Vélu's formulas to perform an ℓ_i-isogeny for $\ell_i \leq 101$ and switches to $\sqrt{\text{élu}}$ for $\ell_i > 101$. The CSURF implementation adds the functionality of degree 2/4 radical isogenies, while the CRADS implementation uses radical isogenies of degree 2/4 and 3/9.

9.7.1 Performance Benchmark of Radical Isogenies

We compare the performance using a different keyspace (i.e., different bounds (e_i)) for CSIDH, CSURF, and CRADS than in [80, 83], where they have used weighted L_∞-norm balls for CSURF and CRADS to compare against an unweighted L_∞-norm ball for CSIDH. Analysis from [**HLKA**, **DBLP:conf/pqcrypto/MeyerCR19**, 95] shows that such a comparison is unfair against CSIDH. We therefore use approximately optimal keyspaces (using weighted L_∞-norm) for CSIDH, CSURF and CRADS.

Suitable bounds. We use suitable exponent bounds for approximately optimal keyspaces that minimize the cost of CSIDH, CSURF, and CRADS by using a slight modification of the greedy algorithm presented in [95], which is included in the provided repository. In short, the algorithm starts by increasing the exponent bound $m_2 \leq 256$ of two used in CSURF, and then applies the exponent bounds search procedure for minimizing the group action cost on the floor (the CSIDH computation part). Once having the approximately optimal bounds for CSURF, we proceed in a similar way for CRADS: this time m_2 is fixed and the algorithm increases the bound $m_3 \in [\![1 \,..\, m_2]\!]$ until it is approximately optimal.

Comparisons. The full results are given in Table 9.3. From Figure 9.1a we see that CSURF and CRADS outperform CSIDH for primes of sizes 512 and 1024 bits, and is competitive for primes of sizes 1792 and 2048 bits. For larger primes, CSIDH outperforms both CSURF and CRADS. Using OAYT-style, CSURF-512 provides a speed-up over CSIDH-512 of 2.53% and CRADS-512 provides a speed-up over CSIDH-512 of 2.15%. The speed-up is reduced to 1.26% and 0.68% respectively for 1024 bits. For larger primes both CSURF and CRADS do not provide speed-ups, because radical isogenies scale worse than Vélu's (or $\sqrt{\text{élu}}$'s) formulas (see Section 9.5.2). This is visible in Figure 9.1a and Figure 9.1b.



(a) OAYT-style (b) MCR-style

Figure 9.1: Relative difference between the number of finite field multiplications required for CSURF and CRADS in comparison to CSIDH. The percentage is based on the numbers in Table 9.3.

Table 9.3: Results for different prime sizes. The numbers are given in millions of finite field multiplications, and the results are the average over 1024 runs. The results count multiplication (**M**) and squaring (**S**) operations, assuming **S** = **M**. Numbers in bold are optimal results for that prime size.

Dummy-style	512-bits	1024-bits	1792-bits	2048-bits	3072-bits	4096-bits
CSIDH-OAYT	0.791	0.873	**0.999**	**1.039**	**1.217**	**1.361**
CSURF-OAYT	**0.771**	**0.862**	1.000	1.042	1.225	1.387
CRADS-OAYT	0.774	0.867	1.007	1.050	1.237	1.399
CSIDH-MCR	1.011	1.093	1.218	1.255	1.436	1.580
CSURF-MCR	0.980	1.074	1.211	1.253	1.443	1.594
CRADS-MCR	0.985	1.086	1.228	1.272	1.469	1.625

Furthermore, the approximately optimal bounds we computed show that the exponents m_2 and m_3 decrease quickly: from $m_2 = 32$ and $m_3 = 12$ for 512-bits, to $e_0 = e_1 = 4$ for 1792-bits, to $e_0 = e_1 = 2$ for 4096-bits. When using MCR-style CSURF and CRADS are slightly more competitive, although the overall cost is significantly higher than OAYT-style. Table 9.3 presents the results obtained in this benchmark and highlights the best result per parameter set. Notice that CSURF outperforms CRADS in every parameter set, which implies that replacing Vélu 3-isogenies with radical 9-isogenies is not cost effective.

9.7.2 Performance of Radical Isogenies using the Hybrid Strategy

In this subsection, we benchmark the performance of radical isogenies when the hybrid strategy of Section 9.6 is used to integrate radical isogenies into CSIDH. Concretely, in comparison to the previous benchmark, we allow a strategy using both Vélu and radical isogenies for degree 3/9. We denote this by CRAD-H, and used our code to estimate the expected running time. We compared this against the results from Table 9.3. As a corollary of Lemma 9.3, CRAD-H may effectively become a 'traditional' CSIDH implementation, when $m_i^r = 0$ turns out to be most optimal for all $i \in R$ (i.e. we get the trivial 'hybrid' paramaters of ordinary CSIDH). We see this happening for primes larger than 1024 bits due to the scaling issues with radical isogenies. Table 9.4 shows the results[4].

Table 9.4: Results for different prime sizes. The first row is given in millions of finite field multiplications, counting multiplication (**M**) and squaring (**S**) operations, assuming $\mathbf{S} = \mathbf{M}$. The second row gives the speed-up in comparison to CSIDH. Trivial parameters are denoted by a superscript t.

CRAD-H	512-bits	1024-bits	1792-bits	2048-bits	3072-bits	4096-bits
performance	0.765	0.861	0.999^t	1.039^t	1.217^t	1.361^t
speed-up	3.3%	2.2%	-	-	-	-

Interestingly, an implementation using non-trivial hybrid parameters outperforms any of the other implementations for a prime of 512 bits. This shows that radical isogenies *can* improve performance in constant-time when using the hybrid strategy. We conclude that replacing Vélu 3-isogenies with radical 9-isogenies is too costly, but adding radical 9-isogenies after Vélu 3-isogenies can be cost effective. However, due to the scaling issues of radical isogenies, the value m_i^r quickly drops to 0 and we get trivial parameters for primes larger than 1024 bits, implying that traditional CSIDH performs better for large primes than any implementation using radical isogenies, even when such an implementation does not have to sacrifice the cheap Vélu isogenies of small degree. We conclude that 'horizontally' expanding the degrees ℓ_i for large primes p is more efficient than 'vertically' increasing the bounds m_i using radical isogenies. Nevertheless, for small primes, our hybrid strategy for radical isogenies can speed-up CSIDH and requires only slight changes to a 'traditional' CSIDH implementation.

[4]Only for OAYT-style. Using the code, we analysed the cost of m_i^r projective radical isogenies and m_i^v Vélu isogenies. The results are realistic; there are no extra costs.

Remark. The CTIDH proposal from [21] is about twice as fast as CSIDH by changing the key space in a clever way: CTIDH reduces the number of isogenies by half compared to CSIDH, using this new key space, which requires the *Matryoshka structure* for Vélu isogenies. The radical part of CSIDH is isolated from the Vélu part, and this will be the same case for CTIDH , when using radical isogenies. The CTIDH construction decreases the cost of this Vélu part by approximately a factor half, but the radical part remains at the same cost. Hence, we heuristically expect that the speed-up of radical isogenies in CTIDH decreases from 3.3%, and may even become a slow-down.

9.8 Concluding Remarks and Future Research

We have implemented, improved and analyzed radical isogeny formulas in constant-time and have optimized their integration into CSIDH with our hybrid strategy. We have evaluated their performance against state-of-the-art CSIDH implementations in constant-time. We show that fully projective radical isogenies are almost twice as fast as affine radical isogenies in constant-time. But, when integrated into CSIDH, both CSURF and radical isogenies provide only a minimal speed-up: about 2.53% and 2.15% respectively, compared to state-of-the-art CSIDH-512. Furthermore, larger (dummy-based) implementations of CSURF and CRADS become less competitive to CSIDH as radical isogenies scale worse than Vélu or $\sqrt{\text{élu}}$ isogenies. In such instances ($\log(p) \geq 1792$ bits) the use of constant-time radical isogenies even has a negative impact on performance.

Our hybrid strategy improves this performance somewhat, giving better results for primes of size 512 and 1024, increasing the speed-up to 3.3%, and making radical 9-isogenies cost effective. For larger primes however, our hybrid approach shows that $\sqrt{\text{élu}}$-isogenies quickly outperformed radical isogenies, as we get 'trivial' parameters for our hybrid strategy. We therefore conclude that, for larger primes, expanding 'horizontally' (in the degrees ℓ_i) is more efficient than expanding 'vertically' (in the bounds m_i).

Due to the large cost of a single exponentiation in large prime fields, which is required to compute radicals, it is unlikely that (affine or projective) radical isogenies can bring any (significant) speed-up. However, similar applications of modular curves in isogeny-based cryptography could bring improvements to current methods. Radical isogenies show that such applications do exist and are interesting; they might be more effective in isogeny-based cryptography in other situations than CSIDH. For example with differently shaped prime number p or perhaps when used in Verifiable Delay Functions (VDFs).

Chapter 10

Higher Security CSIDH

10.1 Placeholder

The paper will go here.

10.2 Introduction

The commutative isogeny-based key exchange protocol (CSIDH) was proposed by Castryck, Lange, Martindale, Panny, and Renes [84] at Asiacrypt 2018. Although it was proposed too late to be included as a candidate in the NIST post-quantum standardization effort [**nist˙pqc**], it has since received significant attention from the post-quantum-crypto research community.

From a crypto-engineering point of view, this attention can be explained by two unique features of CSIDH: Firstly, with the originally proposed parameters, CSIDH has remarkably small bandwidth requirements. Specifically, CSIDH-512, the parameter set targeting security equivalent to AES-128, needs to transmit only 64 bytes each way, more than 10 times less than `Kyber-512`, the KEM chosen for standardization by NIST [**ML-KEM**]. Secondly—and more importantly—CSIDH is so far the only realistic option for post-quantum *non-interactive key exchange* (NIKE), meaning it can be used as a post-quantum drop-in replacement for Diffie–Hellman (DH) key exchange in protocols that combine ephemeral and static DH key shares non-interactively. Such protocols include the Signal X3DH handshake [**X3DH**] and early proposals for TLS 1.3 [**EuroSP:KraWee16**], known as OPTLS. The OPTLS authentication

mechanism is still under consideration as an extension [**ietf-tls-semistatic-dh-01**]. CSIDH is the only post-quantum NIKE that might enable these use cases, except for the recently-proposed Swoosh algorithm [149] (which has too-large public keys for use in TLS).

Unfortunately, quite soon after CSIDH was proposed, several security analyses called into question the claimed concrete security against quantum attacks achieved by the proposed parameters [57, 90, 241]. The gist of these analyses seems troublesome; Peikert [241] states that *"the cost of CSIDH-512 key recovery is only about* 2^{16} *quantum evaluations using* 2^{40} *bits of quantumly accessible classical memory (plus relatively small other resources)"*. Similarly, Bonnetain and Schrottenloher [57] claim a cost of 2^{19} quantum evaluations for attacking the same instance, and propose a quantum circuit requiring only $2^{52.6}$ T-gates per evaluation, which means the security would still be insufficient. Upon exploring the quantum cost of attacking larger instances but ignoring the cost per CSIDH quantum evaluation, instances may require 2048 to 4096 bit keys to achieve the security level originally claimed by CSIDH-512 [90].

Interestingly, although some of these concerns were raised as early as May 2018 (i.e., at a time when [84] was available only as preprint), most research on efficient implementations [22, 86, 211, 235], side-channel attacks [**EPRINT:CMRS22:ournot** and fault attacks against CSIDH [**CKM+20**, **DBLP:conf/eurocrypt/BanegasKLMPRS** 195] continued to work with the original parameters. This can probably partly be explained by the fact that the software implementation referenced in [84][1] implements only the smaller two of the three original parameter sets, i.e., CSIDH-512 and CSIDH-1024. However, another reason is that the concerns about the quantum security of CSIDH were (and to some extent still are) subject of debate. Most notably, Bernstein, Lange, Martindale, and Panny [46] point out that one issue with quantum attacks against CSIDH is the rather steep cost of implementing CSIDH on a quantum computer in the first place. They conclude that the cost of each query pushes the total attack cost above 2^{80}.

In this paper, we do not take any position in this ongoing debate but rather set out to answer the question of what it means for CSIDH performance and applicability if we choose more conservative parameters. This includes protection against physical attacks, which is often required for real-world applications. We call such instantiations *high-security CSIDH*.

[1] Available from `https://yx7.cc/code/csidh/csidh-latest.tar.xz`

Contributions of this paper.

The core contribution of this paper is an in-depth assessment of the real-world practicality of CSIDH.[2] On a high level, this assessment is divided into three parts. First, we instantiate CSIDH at high(er) security levels, suitable for real-world applications, and with protection against physical attacks; second, we optimize the efficiency of high-security CSIDH; third, we test the practicality of high-security CSIDH.

1. Efficient CSIDH instantiations, following two different approaches of implementing high-security CSIDH.
   (a) The first approach aims at protection against physical attacks, and is based on SQALE [90]. In this approach, we eliminate randomness requirements and the use of dummy operations in CSIDH by restricting the keyspace to $\{-1, 1\}^n$, as proposed by Cervantes-Vázquez, Chenu, Chi-Domínguez, De Feo, Rodríguez-Henríquez, and Smith [86]. We refer to this deterministic version of CSIDH as dCSIDH.
   (b) The second approach optimizes purely for performance and uses the CTIDH batching techniques introduced in [22]. We refer to this variant of CSIDH as CTIDH. In particular, we extend the implementation from [22] to larger parameter sets.
2. Optimized implementation of dCSIDH and CTIDH.
   (a) On a high level, we present faster key validation for large parameters, and add a small number of bits to public keys to improve shared-key generation in dCSIDH.
   (b) On a low level, we improve the finite field arithmetic. Our implementations use curves over large prime fields $\mathbb{F}_p$, where p ranges from 2048 to 9216 bits. We optimize arithmetic in these fields for 64-bit Intel processors, specifically the Skylake microarchitecture, using three different options for the underlying field arithmetic.
      - an approach based on the GNU Multiple Precision Library (GMP)
      - MULX-based multiplier using the schoolbook approach (OpScan)
      - MULX-based multiplier using the Karatsuba approach (Karatsuba)

[2] All our work follows the "constant-time" paradigm for cryptographic implementations and thus protects against timing attacks by avoiding secret-dependent branch conditions and memory indices.

3. Practicality benchmark of dCSIDH and CTIDH.

   (a) As a standalone primitive, we benchmark our optimized C/assembly implementations. Our dCSIDH implementation outperforms previous implementations by a factor up to 2.53×. Our CTIDH implementation is the first using large parameters, and, dropping determinism, is thrice as fast as dCSIDH.

   (b) As a real-world use case, we benchmark both dCSIDH and CTIDH in real-world network protocols. We extend the Rustls library [**rustls**] to support OPTLS [**EuroSP:KraWee16**]. OPTLS is a variant of the TLS 1.3 handshake that heavily relies on NIKE for authentication, and avoids handshake signatures (which are especially large (Dilithium [200]) or hard to implement (Falcon [245]) in the post-quantum setting). We compare the performance of the resulting post-quantum OPTLS to post-quantum KEMTLS [261], which is an OPTLS-inspired protocol that uses KEMs for authentication to avoid handshake signatures (but requires significant changes to the handshake protocol). Our results show that dCSIDH and CTIDH are too slow for general-purpose use, as a fully CSIDH-instantiated handshake protocol, though smaller in bandwidth requirements, is orders of magnitude slower than an equivalent based on signatures or KEMs. This implies that current NIKE-based protocols will require changes to transition to post-quantum security, if they are sensitive to latency.

Related work.

The impact of the CSIDH proposal on the cryptographic community can be assessed by the many papers that have been produced around this protocol. Since Castryck, Lange, Martindale, Panny, and Renes [84] left open the problem of implementing CSIDH in constant-time, several papers have proposed different strategies for achieving this property.

The first constant-time implementation of CSIDH, by Bernstein, Lange, Martindale, and Panny [46], focused on assessing the quantum security level provided by CSIDH. For this purpose, the authors strive for producing not only a constant-time CSIDH instantiation but also a randomness-free implementation of it. Meyer, Campos, and Reith [211] (see also [213]) present a more efficient constant-time instantiation of CSIDH for practical purposes. They introduce several algorithmic tricks, including the SIMBA technique, and sampling secret keys from varying intervals, which are further improved by Onuki, Aikawa, Ya-

mazaki, and Takagi [235], who propose to keep track of two points to evaluate the action of an ideal: one in $E(\mathbb{F}_p)$, and one in $E(\mathbb{F}_{p^2})$ with its x-coordinate in $\mathbb{F}_p$. Moreover, Moriya, Onuki, and Takagi [222], and Cervantes-Vázquez *et al.* [86], perform more efficient CSIDH isogeny computations using the twisted Edwards model of elliptic curves. The authors of [86] propose a more computationally demanding dummy-free variant of CSIDH. In exchange, this variant is arguably better suited to resist physical attacks from stronger adversaries, such as fault attacks.

A second wave of studies around CSIDH improve several crucial building blocks. First, a framework that adapts optimal strategies à la SIDH/SIKE to the context of CSIDH [**AMC:ChiRod20**, 172]. Second, improving the computation of large-degree isogenies using an improved version of Vélu's formulas known as $\sqrt{\text{élu}}$ [43]. Several later variants of CSIDH [**JCEng:ACDRR22:UPDATED**, 22] use these improvements. Other works [79, 81, 93]. explore even more variants of CSIDH.

The introduction of CTIDH by Banegas, Bernstein, Campos, Chou, Lange, Meyer, Smith, and Sotáková [22] achieved a breakthrough in the performance of constant-time CSIDH, resulting in an almost twofold speedup, using a new key space and accompanying constant-time algorithms to exploit the idea of batching isogeny degrees. However, they restrict their performance evaluation to primes of 512 and 1024 bits. In contrast, Chávez-Saab, Chi-Domínguez, Jaques, and Rodríguez-Henríquez [90] presented SQALE, the first CSIDH implementation at higher security levels, using primes of size 2000 bits up to 9000 bits. The software we present here starts from their analysis and parameter sizes to reach NIST security levels 1 (equivalent AES-128) and 3 (equivalent AES-192) under different assumptions about the efficiency of quantum attacks, yet goes much further in optimizing the parameters and implementation techniques than [90].

CSIDH is not the only attempt at building a post-quantum NIKE. Although the SIDH protocol [107, 177] was known to be insecure in the static-static scenario [154], Azarderakhsh, Jao, and Leonardi [18] suggested that a NIKE can still be obtained at the cost of many parallel executions of SIDH. However, recent attacks [76, 203, 251] completely break SIDH/SIKE, making this path to a NIKE unfeasible. The only post-quantum NIKE *not* based on isogenies is based on (R/M)LWE and, according to Lyubashevsky, goes back to "folkore" [**Lyu17**]. In 2018, Kock [189] first analyzed such a NIKE and the recently proposed Swoosh [149] is a more concrete instantiation of this approach. We discuss differences between CSIDH and Swoosh in more detail in section 10.8.

Availability of software.

We place our CSIDH software into the public domain (CC0). All software described in this paper and all measurement data from the TLS experiments are available at

`https://github.com/kemtls-secsidh/code`.

Organization of this paper.

Section 12.3 presents the necessary background on isogeny-based cryptography and introduces CSIDH and its CTIDH instantiation. Section 10.4 explains how we instantiate dCSIDH and CTIDH and choose parameters for our optimized implementations. Section 10.5 introduces algorithmic optimizations that apply to our instantiations of dCSIDH and CTIDH. Section 10.6 details our optimization techniques for finite field arithmetic, in particular the efficient Karatsuba-based field arithmetic, and presents benchmarking results for the group action evaluation for dCSIDH and CTIDH. Section 10.7 describes our integration of dCSIDH and CTIDH into OPTLS and presents handshake performance results. Finally, section 10.8 concludes the paper and sketches directions for future work.

10.3 Preliminaries

10.3.1 NIKEs vs. KEMs

We briefly recall the definitions of non-interactive key exchange (NIKE) and key-encapsulation mechanism (KEM) as follows:

Definition 10.1. A *non-interactive key exchange (NIKE)* is a collection of two algorithms, Keygen and Decaps, where

- Keygen is a probabilistic algorithm that on input 1^k, where k is a security parameter, outputs a keypair $(\mathsf{sk}, \mathsf{pk})$; and
- Decaps is a deterministic algorithm that on input a public key pk and a secret key sk outputs a shared key K.

A NIKE is correct if for any $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{Keygen}(1^k)$ and $(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{Keygen}(1^k)$ it holds that $\mathsf{Decaps}(\mathsf{pk}_1, \mathsf{sk}_2) = \mathsf{Decaps}(\mathsf{pk}_2, \mathsf{sk}_1)$.

Definition 10.2. A *key-encapsulation mechanism (KEM)* is a collection of three algorithms, Keygen, Encaps, and Decaps, where

- Keygen is a probabilistic algorithm that on input 1^k, where k is a security parameter, outputs a keypair $(\mathsf{sk}, \mathsf{pk})$; and
- Encaps is a probabilistic algorithm that on input a public key pk outputs a ciphertext ct and a shared key K.
- Decaps is a deterministic algorithm that on input a ciphertext ct and a secret key sk outputs a shared key K.

A KEM is correct if for any $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Keygen}(1^k)$ and $(\mathsf{ct}, K) \leftarrow \mathsf{Encaps}(\mathsf{pk})$ it holds that $\mathsf{Decaps}(\mathsf{ct}, \mathsf{sk}) = K$.

Both NIKEs and KEMs can be used for key exchange, but the non-interactive nature of a NIKE makes it more flexible than a KEM. In the context of their use in protocols, there are three different scenarios:

1. Some scenarios naturally use a KEM. Those scenarios can alternatively also use a NIKE, but they do not benefit in any way from the non-interactive nature of a NIKE. An example for this scenario is the ephemeral key exchange in TLS 1.3, which currently uses (EC)DH, but will easily migrate to post-quantum KEMs [**CECPQ1**, **CECPQ2**, **CECPQ2b**, **WR22**, 61].
2. Some protocols, most notably the X3DH protocol in Signal [**X3DH**] have to use a NIKE and cannot replace this NIKE by a KEM. The reason is that this protocol cannot assume communication partners to be online at the same time and critically relies on the non-interactive nature of a NIKE.
3. Some protocols are somewhat in between: they can be designed from KEMs only, but this comes at the cost of more communication rounds. This has been discussed in some detail in the design of post-quantum Noise [10] and also in the context of the NIKE-based OPTLS [**EuroSP:KraWee16**] vs. the KEM-based KEMTLS [261]. We will revisit the comparison of these two protocols in a post-quantum context in more detail in section 10.7.

10.3.2 The CSIDH NIKE

Background.

Let $\mathbb{F}_p$ be a finite field of prime order p, such that $p+1 = f \cdot \prod_{i=1}^{n} \ell_i$, where each ℓ_i is a small odd prime, and f is a cofactor of the form $2^k \cdot g$ with $k \geq 2$ and g

possibly 1, guaranteeing that p is prime. Now consider the set of supersingular elliptic curves over $\mathbb{F}_p$, i.e., the elliptic curves with $p+1$ $\mathbb{F}_p$-rational points. We will represent these curves in the Montgomery model, i.e., through an equation of the form

$$E_A \colon y^2 = x^3 + Ax^2 + x, \quad A \in \mathbb{F}_p. \tag{10.1}$$

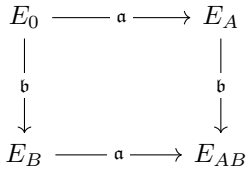
This is possible since the group order $(p+1)$ is a multiple of 4. In the context of CSIDH we are interested in *isogeny graphs of degree* N, denoted $\mathcal{G}_N(\mathbb{F}_p)$. The vertices of such graphs are precisely the supersingular curves over $\mathbb{F}_p$; the edges are $\mathbb{F}_p$-rational isogenies of degree N. CSIDH relies on the following property: for each small odd prime ℓ_i dividing $p+1$, a supersingular curve E_A has only two (supersingular) neighbors in the isogeny graph $\mathcal{G}_{\ell_i}(\mathbb{F}_p)$ (i.e., isogenies over $\mathbb{F}_p$ of degree ℓ_i). We can uniquely describe these isogenies by their kernels: The unique cyclic subgroup of order ℓ_i of $E_A(\mathbb{F}_p)$ defines the isogeny from E_A to one of these neighbors $E_{A'}$. This cyclic subgroup can be described by any of its generators, which in this case means that finding a point in $E_A(\mathbb{F}_p)$ of order ℓ_i is enough to describe an isogeny of degree ℓ_i. As $E_{A'}$ is again supersingular, $E_{A'}(\mathbb{F}_p)$ has order $p+1$ as well and hence a unique cyclic subgroup of order ℓ_i, which gives an isogeny to the unique neighbor that is not E_A. The general action of moving in this direction in this graph $\mathcal{G}_{\ell_i}(\mathbb{F}_p)$ using the unique subgroup of order ℓ_i is denoted by $\mathfrak{l}_i$, and the curve $E_{A'}$ that is reached from E_A by this action is denoted $\mathfrak{l}_i * E_A$. In short, $\mathfrak{l}_i$ represents one step in the isogeny graph $\mathcal{G}_{\ell_i}(\mathbb{F}_p)$, and each small odd prime ℓ_i dividing $p+1$ gives us such an $\mathfrak{l}_i$. Steps in $\mathcal{G}_{\ell_i}(\mathbb{F}_p)$, represented by $\mathfrak{l}_i$, are *commutative*, so that applying $\mathfrak{l}_i$ to $\mathfrak{l}_j * E_A$ is the same as applying $\mathfrak{l}_j$ to $\mathfrak{l}_i * E_A$ for different degrees ℓ_i and ℓ_j. We can also compute steps in the other direction, which is denoted by $\mathfrak{l}_i^{-1} * E_A$. The subgroup of points of order ℓ_i with x-coordinate in $\mathbb{F}_p$ and y-coordinate in $\mathbb{F}_{p^2} \backslash \mathbb{F}_p$ uniquely defines the corresponding isogeny kernels. Applying both $\mathfrak{l}_i$ and $\mathfrak{l}_i^{-1}$ effectively cancels out, i.e., we have $\mathfrak{l}_i * (\mathfrak{l}_i^{-1} * E) = \mathfrak{l}_i^{-1} * (\mathfrak{l}_i * E) = E$.

The CSIDH scheme.

The CSIDH scheme [84] unrolls naturally from the action described above: The secret key is a vector of n integers $(e_1, \ldots, e_n)$ defining the product $\mathfrak{a} = \prod_{i=1}^{n} \mathfrak{l}_i^{e_i}$. In the original proposal the integers e_i are chosen from $\{-m, \ldots, m\}$ for some $m \in \mathbb{N}$, which results in a key space of size $(2m+1)^n$. The public key is the supersingular curve E_A which corresponds to the secret key $\mathfrak{a}$ applied to a publicly known starting curve E_0:

$$E_A = \mathfrak{a} * E_0 = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_0. \tag{10.2}$$

This public key E_A can be encoded by the single value $A \in \mathbb{F}_p$ (see Equation (10.1)). Shared-key computation is the same as public-key computation, except that instead of the public parameter E_0 it uses a public key E_A as input curve. That is, Alice and Bob compute their shared secret by calculating $E_{AB} = \mathfrak{a} * E_B = (\mathfrak{a} \cdot \mathfrak{b}) * E_0$ and $E_{BA} = \mathfrak{b} * E_A = (\mathfrak{b} \cdot \mathfrak{a}) * E_0$, respectively, with $E_{AB} = E_{BA}$ thanks to the commutativity. This is summarized by the following diagram:



Computing the group action $\mathfrak{a} * E$.

Straightforward high-level pseudocode for the computation of the group action $\mathfrak{a} * E$ is given in algorithm 7. The dominating cost is the construction and evaluation of the ℓ_i-isogenies corresponding to the action of the $\mathfrak{l}_i$ (lines 5 and 7), which in turn decompose into a sequence of operations in $\mathbb{F}_p$. However, the high-level view also illustrates an additional complication for *secure* implementations of CSIDH, namely that the number of iterations of the inner loop (line 3) and the direction of the isogenies corresponding to the action of $\mathfrak{l}_i$ (line 4) depend on the secrets e_i and naive implementations thus leak secret information through timing.

Algorithm 7 High-level view of the CSIDH group action computation.

Input: $I \in \mathbb{F}_p$ defining a curve E_I
Input: secret key $(e_1, \ldots, e_n)$
Output: $R \in \mathbb{F}_p$ defining a curve $E_R = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_I$

```
1: E_R ← E_I
2: for i from 1 to n do
3:     for j from 1 to |e_i| do
4:         if e_i > 0 then
5:             E_R ← l_i * E_R
6:         else
7:             E_R ← l_i^{-1} * E_R
8: return R
```

For constant-time behavior, we need to be careful not to leak this information on e_i. Current implementations of CSIDH hide e_i by computing m isogenies per degree ℓ_i, while effectively performing $|e_i|$ isogenies, e.g., by using dummy computations or computations that effectively cancel each other such as $\mathfrak{l}_i * \mathfrak{l}_i^{-1} * E$.

For the sake of simplicity, Algorithm 7 omits the description of several underlying building blocks. For example, the computation of an isogeny of degree ℓ_i requires as input a point of order ℓ_i. Points of a prescribed order can be obtained probabilistically by sampling random points on the current curve. Any randomly sampled point T can generate exactly one isogeny of those degrees ℓ_i that divide the order of T, by pushing T through such isogenies to get a similar point T on the codomain curve. The order in which we perform such ℓ_i-isogenies giving a point T that can perform multiple of them influences the performance. Hence, different *strategies*, i.e. orderings of ℓ_i-isogenies, point evaluations, and point multiplications, can affect performance. Several efficient strategies are described in, e.g., [**AMC:ChiRod20**, 84]. We describe our choices for the CSIDH group action computation in more detail in section 10.4 and section 10.5.

Computing a single isogeny $E_R \leftarrow \mathfrak{l}_i * E_R$.

A single isogeny $E_R \leftarrow \mathfrak{l}_i * E_R$ can be computed in multiple ways: Traditionally, the formulas introduced by **velu** are used, at a cost of approximately 6ℓ field multiplications for an isogeny of degree ℓ. In 2020, [43] presented new formulas for constructing and evaluating isogenies of degree ℓ, at a combined cost of just $\tilde{O}(\sqrt{\ell})$ field multiplications, denoted as $\sqrt{\text{élu}}$. With respect to CSIDH, [**JCEng:ACDRR22:UPDATED**] reports that the $\sqrt{\text{élu}}$ formulas of [43] improve the traditional formulas for isogenies of degree $\ell \geq 89$, and concludes that constant-time CSIDH implementations using 511- and 1023-bit primes are moderately improved by the $\sqrt{\text{élu}}$ formulas. The authors from [79] presented a variant of CSIDH named CSURF, which essentially proposes using 2-isogenies by calculating radical computations (i.e., by performing exponentiation with a fixed exponent along with a field inversion). [82] extended the radical approach to compute isogenies for odd isogeny degrees less than 13. Both works suggest a modest savings in the running time of CSIDH and essentially CSURF can be considered CSIDH with radical isogenies of degree 2. On the one side, the authors from [93] improved the formulas from [79, 82] by presenting an inverse-free method to compute such radical isogenies at the cost of a single exponentiation. Conversely, the recent work from [81] provided some interesting improvements (in terms of field multiplication) to the results from [82]; they still require one

exponentiation by a fixed exponent and at least one field inversion, which are the bottleneck. Nevertheless, [93] additionally showed that such radical isogenies become too costly in large CSIDH parameters. On that basis, we will not make use of the radical isogenies, as the analysis from [93] shows that this is unfavorable when the base field $\mathbb{F}_p$ is larger than 1024 bits.

10.3.3 CTIDH

Banegas, Bernstein, Campos, Chou, Lange, Meyer, Smith, and Sotáková [22] proposed a new approach for constant-time CSIDH, named CTIDH. The main novelties are a different way of specifying the key spaces, and some algorithmic adaptions in order to obtain a constant-time algorithm.

CTIDH key spaces.

For defining CTIDH keyspaces, we organize the primes ℓ_i in N *batches*, such that each batch consists of consecutive primes. In particular, we choose a vector of batch sizes $N = (N_1, \ldots, N_B)$ with all $N_i > 0$, such that $\sum_{i=1}^{B} N_i = n$. Then we distribute the n prime degrees $\ell_1, \ldots, \ell_n$ among those B batches. That is, we define the first batch as $(\ell_{1,1}, \ldots, \ell_{1,N_1}) := (\ell_1, \ldots, \ell_{N_1})$, the second batch as $(\ell_{2,1}, \ldots, \ell_{2,N_2}) := (\ell_{N_1+1}, \ldots, \ell_{N_1+N_2})$, etc. Accordingly, we relabel the private key elements e_k as $e_{i,j}$.

Instead of directly sampling the key elements $e_{i,j}$ from some interval $[-m, m]$ as in CSIDH, CTIDH only limits the 1-norm of each key batch. That is, for the i-th batch $(\ell_{i,1}, \ldots, \ell_{i,N_i})$, we fix a bound m_i and sample corresponding key elements $e_{i,j}$ such that $\sum_{j=1}^{N_i} |e_{i,j}| \leq m_i$. This means that for each isogeny we compute for the i-th batch, its degree could be any of $\ell_{i,1}, \ldots, \ell_{i,N_i}$. This adds a combinatorial advantage, in the sense that the same number of isogenies as in CSIDH leads to a much larger key space size in CTIDH. In other words, CTIDH requires a smaller number of isogenies for reaching the same key space size. For example, the fastest previous constant-time implementation of CSIDH-512 with key space size 2^{256} required the computation of 438 isogenies, while the CTIDH parameters of [22] only requires 208 isogenies for the same key space size. For details, we refer to [22]. We note that as defined above, CSIDH is a special case of CTIDH using n batches of size 1.

CTIDH algorithm.

The main problem for constant-time implementations with this adapted key space lies in the fact that we must hide the degree of each isogeny from side channels. Given that the computational effort for an isogeny directly depends on its degree, a straightforward implementation of CTIDH would leak the degree of each isogeny. On the other hand, an attacker must not be able to observe to which degree out of $\{\ell_{i,1}, \ldots, \ell_{i,N_i}\}$ each isogeny for the i-th batch corresponds. [22] achieves this by using an observation from [46]. The usual isogeny formulas [**velu**, 43], have a *Matryoshka-doll structure*. That is, if $\ell_i < \ell_j$, then an ℓ_j-isogeny performs exactly the computations that an ℓ_i-isogeny would require, plus some extra operations. Therefore, we can easily compute an ℓ_i-isogeny at the cost of an ℓ_j-isogeny, by performing dummy operations for the extra steps. In CTIDH, we use this idea to compute each isogeny for the i-th batch $(\ell_{i,1}, \ldots, \ell_{i,N_i})$ at the cost of the most expensive degree, i.e., an ℓ_{i,N_i}-isogeny. In this way, the isogeny degrees do not leak via timing channels.

There are several other operations that require adjustments in CTIDH in order to obtain a constant-time implementation. For instance, this includes scalar multiplications that produce points of suitable order, or point rejections, which must occur independently of the required isogeny degree. For details on how these issues are resolved, we refer to [22].

Even though these algorithmic adjustments induce some computational overhead, CTIDH is almost twice as fast as its CSIDH counterpart for the CSIDH-512 and CSIDH-1024 parameter sets from [84] (see [22]).

10.3.4 Quantum security

While classical security imposes a restriction on the minimum key space size, quantum security usually poses more restrictive requirements. However, it is argued in [90] that for reasonable key spaces (that is, spaces large enough to achieve classical security), the quantum security of CSIDH relies only on the size of the prime p, regardless of the size of the actual key space being used. This is due to the fact that the most efficient quantum attack, Kuperberg's algorithm [**Kup13**], requires working over a set with a group structure. Since the entire group representing all possible isogenies is of size roughly $\sqrt{p}$,[3] this attack needs to search a space much larger than the keyspace itself, which only depends on n and the exponent bound m. For example, in the case of CSIDH-512, the element $\mathfrak{l}_3$ alone generates the entire group of size roughly 2^{257} [52].

[3] The $\mathfrak{l}_i$ represent elements of the *class group* $\mathcal{C}\ell(\mathbb{Z}[\sqrt{p}])$, which has size roughly $\sqrt{p}$.

It is expected that a handful of $\mathfrak{l}_i$ generate the entire group also for larger instances. In a nutshell, classical security is determined by the size of the key space, whereas quantum security is determined by the size of p, as long as the key space is not chosen particularly badly, e.g., as a small subgroup of the full class group.

10.4 Proposed instantiations of CSIDH

In this section, we describe how to instantiate and choose parameters for large-parameter CSIDH. We describe two different approaches to selecting parameters: dCSIDH targets a deterministic and dummy-operation-free implementation[4], whereas CTIDH optimizes for the batching strategies proposed in [22]. This reflects the two extreme choices one can make to either prioritize security against physical attacks or speed. We note that there are several choices in the middle ground, trading off physical security for speed. For comparability, both approaches share the choice of underlying finite fields $\mathbb{F}_p$, which we detail in section 10.4.1.

10.4.1 The choice of p

In this work, we take the conservative parameter suggestions from [90] at face value. In particular, we consider primes of 2048 and 4096 bits to target NIST security level 1, 5120 and 6144 bits to target NIST security level 2, and 8192 and 9216 bits to target NIST security level 3. Each pair of bitsizes represents a choice between more "aggressive" assumptions (with attacker circuit depth bounded by 2^{60}) or more "conservative" assumptions (attacker circuit depth bounded by 2^{80}). As stressed in [90], this choice of parameters does not take into account the cost of calls to the CSIDH evaluation oracle on a quantum computer and is likely to underestimate security. However, as discussed in Section 11.2, we merely aim at giving performance results for conservative parameters.

All our implementations use primes of the form $p = f \cdot \prod_{i=1}^{n} \ell_i - 1$, where ℓ_i are distinct odd primes, f is a large power of 2 and n denotes the number of such ℓ_i dividing $p+1$. For these sizes of p, it becomes natural to pick secret key exponents $e_i \in \{-1, +1\}$, as n can be chosen large enough to reach the desired

[4] Our implementation does not take the recent physical attacks [**EPRINT:CMRS22:ournote**, **DBLP:conf/eurocrypt/BanegasKLMPRST23**] into account, whose impact in the high-parameter range is unclear. Heuristically, countermeasures against both attacks should not impact performance by much.

Table 10.1: Parameters for reconstructing each prime $p = f \cdot \prod_{i=1}^{n} \ell_i - 1$. In each case the ℓ_i are assumed to be the first n odd primes, excluding some primes and including larger primes ℓ_i to ensure that p is prime. These are given in the Excluded and Included columns.

Prime bits	f	n	**Excluded**	**Included**	**Key Space**	**NIST level**
p2048	2^{64}	226	$\{1361\}$	–	2^{221}	1 (aggressive)
p4096	2^{1728}	262	$\{347\}$	$\{1699\}$	2^{256}	1 (conservative)
p5120	2^{2944}	244	$\{227\}$	$\{1601\}$	2^{234}	2 (aggressive)
p6144	2^{3776}	262	$\{283\}$	$\{1693, 1697, 1741\}$	2^{256}	2 (conservative)
p8192	2^{4992}	338	$\{401\}$	$\{2287, 2377\}$	2^{332}	3 (aggressive)
p9216	2^{5440}	389	$\{179\}$	$\{2689, 2719\}$	2^{384}	3 (conservative)

keyspace size [86, 90]. In particular, to achieve a keyspace of b bits in CSIDH we need to have at least $n = b$ of these ℓ_i in this case.

For conservative instances, we base the keyspace sizes on the classical meet-in-the-middle (MITM) attack considered in [84], requiring $b = 2\lambda$ for security parameter λ. That is, $b = 256,\ 256,\ 384$ for p4096, p6144, p9216, respectively[5]. On the other hand, for aggressive instances we based the keyspace size on the limited-memory van Oorschot-Wiener golden collision search [285] with the assumptions from [90], which leads to $b = 221,\ 234,\ 332$ for p2048, p5120, p8192, respectively.

Finally, we restrict to cofactors f for which the power of 2 is a multiple of 64, since the arithmetic optimizations discussed in Section 10.6 require this shape. Hence, to find optimal primes for our implementation, we let $\ell_1, \ldots, \ell_b$ be the b smallest odd primes and then compute the cofactor f as the largest power of 2^{64} that fits in the leftover bitlength. This still leaves us with a bitlength slightly smaller than the target, and hence the leftover bits can be used to search for additional factors ℓ_i (making $n > b$) that make $f \cdot \prod_{i=1}^{n} \ell_i - 1$ a prime number. These extra factors go unused for dCSIDH, where they are viewed as part of the cofactor, but are exploited by the batching strategies of CTIDH to increase performance. We set a minimum requirement of 5 additional ℓ_i factors (that is, $n \geq b + 5$), decreasing f by a single factor of 2^{64} when not enough bits were left over. The results of this search are shown in Table 10.1.

[5]Since conservative instances don't take memory restrictions into account, security levels 1 and 2 have the same key space requirements. They only differ in the size of the prime due to quantum security concerns as described in [90].

10.4.2 Parameters for dummy-free, deterministic dCSIDH

The restriction of exponents to $\{-1, +1\}$ makes it easier to make dCSIDH deterministic and dummy free [86, 90], as we always perform only one isogeny of each degree, with the only variable being the "direction" of each isogeny. Since isogenies in either direction require exactly the same operations, it is easy to obtain a constant-time implementation without using dummy operations.

Randomness appears in the traditional CSIDH implementation: it arises from the fact that performing isogenies of degree ℓ_i requires a point of order ℓ_i as input, and such a point is obtained by sampling random points on the current curve. Any random point can either be used for "positive" steps $\mathfrak{l}_i^{+1}$ or "negative" steps $\mathfrak{l}_i^{-1}$. Hence, a point of order ℓ_i can be used only once and only for a specific orientation. Doing more than one isogeny of each degree requires us, therefore, to sample new points midway. However, by restricting e_i to $\{-1, +1\}$, we have to compute only one isogeny per degree ℓ_i. This allows us to avoid random sampling by providing a pair of points T_+, T_- beforehand whose orders are divisble by all ℓ_i, where T_+ can be used for the positive steps $\mathfrak{l}_i$ with $e_i = 1$, and T_- for the negative steps $\mathfrak{l}_i^{-1}$, with $e_i = -1$. We refer to such points as *full-torsion* points, as they allow us to perform an isogeny of every degree ℓ_i by multiplying them by the right scalar. That is, to perform an ℓ_i-isogeny in the "plus" direction, we can use the point $[\frac{p+1}{\ell_i}]\, T_+$ of order ℓ_i.

Note that the probability for the order of a random point to contain the factor ℓ_i is given by $\frac{\ell_i - 1}{\ell_i}$. Thus, sampling for a pair of full-torsion points can be expensive when small factors ℓ_i are used, as they dominate the probability $\prod_i \frac{\ell_i - 1}{\ell_i}$ of sampling a full-torsion point. Since the primes we use always have additional ℓ_i factors that are unused in dCSIDH (see Section 10.4.1), we make point sampling more efficient by always discarding the smallest primes rather than the largest ones, increasing the odds to sample a full-torsion point. For example, the prime p4096 has 262 ℓ_i factors but only needs a keyspace of 2^{256}, hence we can discard 6 primes. By discarding the 6 *smallest* ones, the probability to sample a full-torsion point goes up from $\prod_{i=1}^{256} \frac{\ell_i - 1}{\ell_i} \approx 0.151$ to $\prod_{i=7}^{262} \frac{\ell_i - 1}{\ell_i} \approx 0.418$, making it more than 2.7 times as easy to sample full-torsion points T_+ and T_-. Such a shift in primes causes a trade-off in the rest of the protocol, as higher-degree isogenies are more expensive. However, due to the improvements in [43], the extra cost of using $\ell_{257}, \ldots, \ell_{262}$ instead of $\ell_1, \ldots, \ell_6$ is relatively small in comparison to the total cost of a group action computation. Thus, discarding the smallest ℓ_i is preferable as it significantly decreases the cost of sampling full-torsion points, and only increases the cost of computing $\mathfrak{a} * E$ by a marginal amount.

The points T_+, T_- on the starting curve E_0 can be precomputed and considered public parameters, but for the public-key curves they must be computed in real time. We include the computation of these points in the key generation, and include them in the public key, which makes the shared-secret derivation completely constant-time and deterministic. The key generation is then the only part that does not run in strictly constant wall-clock time (yet is implemented following the constant-time paradigm), but is still made deterministic by sampling points in a pre-defined order. As we describe in Section 10.5, these points can be represented in a very compact form, which increases public-key sizes by only a few bits. We further emphasize that in order to avoid active attacks, the shared-key computation must validate these transmitted points to be full-torsion points.

Following the SQALE implementation [90], we use the optimal strategy approach from [**AMC:ChiRod20**] to efficiently evaluate the class group action.

Choice of cofactor.

With the choice of cofactor $h = 4$ from [84], the prime p is fully determined from the bitsize. However, we propose to generalize the choice of cofactor to $h = 2^k$, which gives us the flexibility to reach the same desired bitsize of p with different combinations of n and k. As we will see in section 10.6, it is beneficial to choose k as large as possible. However, we need to choose the value n big enough to obtain a sufficiently large keyspace. **TODO:** Discuss here that we need $n = 2\lambda$ for λ bits of classical security with conservative estimates, but can choose n a bit smaller if we limit the attacker's memory.

We summarize the values of k, n and the resulting bitsize of p of our proposed parameter sets in table 10.2.

Factorization of $p + 1$.

The number n of small primes is determined by the classical security requirement. For λ bits of security, a first approach would be to use $n = 2\lambda$ having a MITM attack in mind, but such an attack also requires 2^λ cells of memory which we consider unrealistic. Instead, we base our security on Van Oorschot and Wiener's golden collision search [285] which takes total time

$$7.1 \times \frac{2^{\frac{3}{4}n}}{w^{1/2}}$$

when running with w cells of memory. We use a bound of $w = 2^{80}$ to derive the parameters shown in Table 10.2.

We stress that with the sizes of p that we have fixed in the previous section, the key space sizes required for classical security are easy to fit in. In particular, sampling exponents from $\{-1, 1\}$ is not restrictive and in fact even after including sufficient ℓ_i factors we still need to bloat the size of p even more.

We direct this bloating towards the cofactor, which for efficiency reasons (discussed in Section 10.6) contains an additional factor of $3 \cdot 5 \cdot 7$. The rest of the bloating is done such that $p = 2^k - c$ where k is derived from the required bit size and c is a 'small' positive integer. We say c is 'small' when $\log(c) \leq \frac{k}{2}$. Such a shape provides a speed-up in the reduction for the finite field arithmetic (see **??**). Finding these primes is relatively easy: let $A = 4 \cdot 3 \cdot 5 \cdot 7 \cdot \prod \ell_i$ be the structure that should divide $p + 1$, and let $B = \lfloor 2^k / A \rfloor$. Then we search for primes relatively close to 2^k by checking potential primes $p = (B - d) \cdot A - 1$ for $d = 0, 1, 2, \ldots$ until we find a prime. That is, the shape of our primes is

$$p = (B - d) \cdot 4 \cdot 3 \cdot 5 \cdot 7 \cdot \prod_{i=0}^{n} \ell_i - 1.$$

For such a prime, $p + 1 = (B - d) \cdot A = B \cdot A - d \cdot A \approx 2^k - d \cdot A$. Assuming d is very small, the distance to 2^k is therefore close to $\log(A)$. This implies that we can easily find primes of the form $p = 2^k - c$ with a suitable factorization of $p + 1$ and c 'small' as long as $\log(A) \leq \frac{k}{2}$. This is the case for parameter sets $(k = 4095, n = 221)$, $(k = 5119, n = 256)$, $(k = 6143, n = 306)$, $(k = 8191, n = 306)$ and $(k = 9215, n = 384)$. By slightly varying the ℓ_i in $A = 4 \cdot 3 \cdot 5 \cdot 7 \cdot \prod \ell_i$ we were able to find primes with $d = 0$ for all these parameter sets. So $p = B \cdot A - 1$ with A and B as defined above. The only exception was the 2048-bit prime, where it becomes more efficient to use a pure power of 2 as cofactor.

We expect that it is not easy to find primes much closer to 2^k than the primes we have found. The heuristic argument is that $p = B \cdot A - 1$ is the only possibility for a prime in the interval $[2^k - A, 2^k]$. The odds of this number p being prime are low, but different configurations of A increase these odds. The odds of such a prime being *much* closer to 2^k however are very low, as any uniformly random number in this interval is close to $\log(A)$ in distance to 2^k.

10.4.3 Parameters for CTIDH

As mentioned above, the instantiations of dCSIDH that we use are designed as dummy-free and deterministic algorithms, in order to avoid potential issues with randomness and dummy operations. However, these choices induce significant computational overhead. Therefore, we additionally give performance results for CTIDH [22], the fastest available constant-time implementation of CSIDH (allowing randomness and dummy operations), at the same security levels so that we can compare performance. Note that [22] only reports performance results for 512-bit and 1024-bit primes.

For the parameter sizes considered in this work, we thus use the same primes as in the dCSIDH case (see Table 10.1). This allows for a simple comparison of the two approaches, since both implementations use the same finite field arithmetic (see Section 10.6). On the other hand, it is unclear which parameters are optimal for CTIDH with the given prime sizes. A larger number of small prime factors ℓ_i in the factorization of $p+1$ can be beneficial, since the combinatorial advantage of CTIDH batching increases with the number of available prime degrees. On the other hand, this would mean that we have to include larger ℓ_i, and therefore compute more expensive large degree isogenies. Furthermore, the choice of CTIDH parameters, i.e., batches and norm bounds, becomes more challenging at larger prime sizes. We thus leave the exploration of optimal CTIDH parameters for large primes as future work.

For the given primes, we use the greedy algorithm from [22] for determining these additional parameters, adapted to the case of the cofactor $f > 4$. On input of the primes ℓ_i and a fixed number of batches, the algorithm searches for a locally optimal way of batching the primes, and according norm bounds, such that the expected number of field multiplications per group action evaluation is minimized. However, for the parameter sizes in this work, the greedy search becomes increasingly inefficient. We could thus only run searches for a small set of potential batch numbers, especially for the larger parameters. We obtained these potential inputs by extrapolating from the data of smaller parameter sizes from [22] and slightly beyond. For concrete parameter choices, we refer to our software. Note that the choice of a different number of batches could improve the results, but an exhaustive search using the greedy algorithm seems out of reach.

Apart from the parameters and batching setup, our CTIDH implementation uses the algorithms and strategies from [22]. We remark that CTIDH could in theory also be implemented in a dummy-free or deterministic way. [22] presents an algorithm that avoids dummy isogenies, but points out that the Matryoshka

isogenies require dummy operations by design. Thus, the current techniques do not allow for a dummy-free implementation of CTIDH. Further, the design of a deterministic variant of CTIDH requires some adaptions, such as computing multiple isogenies per batch in a single round. We leave the design and analysis of such an implementation for future work.

As mentioned in [84, Sec. 4], CSIDH is parameterized by a prime p and a range $\{-m, \ldots, m\}$, centered at zero, that the secret exponents are sampled from. In particular,

$$p = h \prod_{i=1}^{n} \ell_i - 1,$$

where ℓ_i are small odd primes, and the cofactor h is usually fixed to 4 as in [**MR18**, 84, 86, 94, 211, 236].

10.4.4 Design choices

Currently, constant-time implementations of CSIDH have different flavors: either dummy-based or dummy-free isogeny constructions, and either randomness-based or randomness-free group action evaluations.

As pointed out in **TODO:** described in an earlier section?, we decided on the usage of a dummy-free and randomness-free configuration, in order not to exclude devices without access to high-quality randomness, and to mitigate against certain side channel attacks (SCA). We briefly review the corresponding implications of our implementation.

Randomness-free configuration.

Roughly speaking, a CSIDH implementation samples a random point P on the current curve until $\left[\frac{p+1}{\ell_i}\right] P$ is different from the point at infinity, which plays as the kernel point generator of the required degree-ℓ_i isogeny. Although several improvements have been proposed, such as different strategies that compute multiple isogenies from each sampled point (see e.g. [**MR18**, 84, 86, 94, 211, 236]), most implementations rely on access to randomness for sampling these points. If this process is flawed, e.g. by instead proceeding deterministically, this could lead to a leakage of side-channel information.

However, Cervantes-Vázquez, Chenu, Chi-Domínguez, De Feo, Rodríguez-Henríquez, and Smith [86] proposed a version of CSIDH that refrains from assuming access to randomness, and does not introduce leakage of side-channel

information. In particular, let E be the input curve to the class group evaluation algorithm, and assume that two points $P_+ \in E[\pi - 1]$ and $P_- \in E[\pi + 1]$ of full order $\#E(\mathbb{F}_p) = p + 1$ are given. Then, it is possible to evaluate the full class group action for any private key $(e_1, \ldots, e_n)$ with exponents $e_i \in \{-1, 0, 1\}$, and without sampling additional points. Note that the key space is limited to a size of 3^n in this case, which means that $p + 1$ must contain enough distinct odd prime factors $\ell_1, \ldots, \ell_n$ to reach a large enough key space for a given security level. This will be detailed in section 10.4.6.

Dummy-free configuration.

Dummy-free constant-time implementations of CSIDH were also proposed in [86]. They achieve better security properties against fault injection attacks by design, since any fault or abort of the algorithm leads to a wrong output curve. This is in contrast to dummy-based versions, where fault injections can be used systematically to determine the total number of dummy isogenies, and therefore the weight of the respective private key [**CKM+20**].

On the other hand, [86] states that the application of dummy-free approaches comes at a steep performance cost for small instances of CSIDH, such as CSIDH-51. However, it "becomes natural" when targeting much larger parameters, which is our case study.

Together with the randomness-free setting, this implies that zero entries are not allowed in private keys, and the respective exponents are limited to $e_i \in \{-1, 1\}$. In other words, the keyspace size is reduced from 3^n to 2^n, thus we need to increase the number of available isogeny degrees n to achieve a sufficiently large keyspace.

10.4.5 Evaluating the class group action

Currently, the fastest constant-time instantiations of CSIDH either use the SIMBA approach (see [86, 211, 236]) or optimal strategies [94]; both procedures roughly lead to the same performance, but in our configuration, the use of optimal strategies is preferred. This is due to the fact that we want to compute all isogenies from only two initial points of full order, whereas the SIMBA approach assumes that we can sample fresh points at intermediate steps. Adapting SIMBA to our setting involves substantial overhead, and thus the instantiations used in this work make use of the optimal strategy approach as described in [94].

As mentioned above, the proposed CSIDH instantiation requires two full order points $P_+ \in E[\pi - 1]$ and $P_- \in E[\pi + 1]$, with E being a public curve. In other words, the class group action has as inputs the three public values E, P_+, and P_-. Now, this procedure will always construct an isogeny of degree $(\frac{p+1}{h})$, where h is the cofactor in the factorization of $p+1$. Thus, instead of full order points, we only require points of order $\frac{p+1}{h}$. Consequently, the CSIDH key generation will need to look for a pair of $\left(\frac{p+1}{h}\right)$-order points on the public output curve and include them in the public key, while the secret-sharing derivation will only focus on evaluating the class group action.

Isogeny formulas.

As illustrated by Bernstein, Feo, Leroux, and Smith [43], the fastest isogeny formulas becomes a hybrid between their new Vélu-sqrt formulas (for large degrees) and the traditional Vélu's formulas (for small degrees). We adopted the same isogeny degree cutoff $\ell \leq 83$ as suggested in [**JCEng:ACDRR22:UPDATED**].

For isogenies of degree $\ell \leq 13$, Castryck, Decru, and Vercauteren [82] proposed new formulas named 'radical isogenies'. We have not adopted these formulas for two reasons: Firstly, the degree-ℓ isogeny formula contains an ℓ-th root computation, which is a costly operation when the size of p is large. Secondly, the isogeny formula for odd degree ℓ requires sampling one initial point of order ℓ on a non-public curve to compute a chain of degree-ℓ isogenies, and then a dummy-free implementation of these radical isogeny formulas will require randomness.

10.4.6 Parameters

Specific primes used.

We analyse the performance of CSIDH for 6 specific primes.

- `p2048n221`: For NIST security level 1 with aggressive assumptions. This prime is of the form $p = 2^{108} \cdot 3 \cdot 5 \cdot 7 \cdot \prod_{i=1}^{221} \ell_i - 1$, where the ℓ_i are the odd primes up to 1409, excluding 389. Note that for this parameter set it is not more efficient to look for a prime $p = 2^{2047} - c$.

- `p4096n221`: For NIST security level 1 with moderate assumptions. This

prime is of the form

$$p = 2^{4095} - c = B \cdot 4 \cdot 3 \cdot 5 \cdot 7 \cdot \prod_{\substack{i=1 \\ \ell_i \notin S}}^{225} \ell_i - 1,$$

with $S = \{977, 1039, 1063, 1321\}$. The size of c is approximately 1938 bits.

- `p5120n256`: For NIST security level 1 with conservative assumptions. This prime is of the form

$$p = 2^{5119} - c = B \cdot 4 \cdot 3 \cdot 5 \cdot 7 \cdot \prod_{\substack{i=1 \\ \ell_i \notin S}}^{260} \ell_i - 1,$$

with $S = \{701, 719, 1097, 1217\}$. The size of c is approximately 2309 bits.

- `p6144n306`: For NIST security level 3 with aggressive assumptions. This prime is of the form

$$p = 2^{6143} - c = B \cdot 4 \cdot 3 \cdot 5 \cdot 7 \cdot \prod_{\substack{i=1 \\ \ell_i \notin S}}^{310} \ell_i - 1,$$

with $S = \{1213, 1249, 1627, 1907\}$. The size of c is approximately 2851 bits.

- `p8192n306`: For NIST security level 3 with moderate assumptions. This prime is of the form

$$p = 2^{8191} - c = B \cdot 4 \cdot 3 \cdot 5 \cdot 7 \cdot \prod_{\substack{i=1 \\ \ell_i \notin S}}^{310} \ell_i - 1,$$

with $S = \{619, 641, 1307, 1493\}$. The size of c is approximately 2854 bits.

- `p9216n384`: For NIST security level 3 with conservative assumptions. This prime is of the form

$$p = 2^{9215} - c = B \cdot 4 \cdot 3 \cdot 5 \cdot 7 \cdot \prod_{\substack{i=1 \\ \ell_i \notin S}}^{390} \ell_i - 1,$$

with $S = \{617, 1307, 1747, 2273, 2549, 2579\}$. The size of c is approximately 3726 bits.

Parameter set	Security	n	$\log_2 c$	$\log_2 p$
CSIDH-2048	NIST L1 (aggressive)	221	–	2047
CSIDH-4096	NIST L1	221	1938	4095
CSIDH-5120	NIST L1 (conservative)	256	2309	5119
CSIDH-6144	NIST L3 (aggressive)	306	2851	6143
CSIDH-8192	NIST L3	306	2854	8191
CSIDH-9216	NIST L3 (conservative)	384	3726	9215

Table 10.2: Proposed parameters for higher-security CSIDH

TODO: there was a begin comment end comment here, is the above text even true? check against eprint

10.5 Optimizing dCSIDH and CTIDH

Given the parameter choices from Section 10.4, we describe the high-level optimizations we apply for dCSIDH and CTIDH. Note that apart from the improved public key validation, we use the standard CTIDH implementation from [22] extended to the parameter sizes from Section 10.4.3. For dCSIDH, we present several improvements in Section 10.5.2.

10.5.1 Supersingularity verification

For the prime choices from Section 10.4.1, we need to adapt the supersingularity verification from [84]. In particular, given primes with cofactor $\log f > \frac{1}{2}\log p$, both algorithms discussed in [84, Alg. 1 and Alg. 3] to test supersingularity of a public key E_A do not work.

Note that these supersingular tests, verify whether $\#E_A(\mathbb{F}_p) = p+1$, by showing that there is a point P with large enough order $N \mid p+1$. Both algorithms start by sampling a random point P, followed by a multiplication by the cofactor $P \leftarrow [f]P$, and then by checking whether the resulting point has ℓ_i-torsion. This is done by calculating if $[\prod_{j\neq i} \ell_j]P \neq \mathcal{O}$ and $[\prod \ell_j]P = \mathcal{O}$. If the random point P has ℓ_i-torsion for enough ℓ_i such that their product $\prod \ell_i \geq 4\sqrt{p}$, then in the Hasse interval $p+1-2\sqrt{p} \leq \#E_A(\mathbb{F}_p) \leq p+1+2\sqrt{p}$, $p+1$ is

the only possible multiple of its order $\mathrm{ord}(P)$. This implies that $\#E_A(\mathbb{F}_p) = p + 1$. Unfortunately, this approach cannot be applied to our setting, because for primes where $\log f > \frac{1}{2}\log p$, even a point with ℓ_i-torsion for all i does not reach the threshold $4\sqrt{p}$, as $\log(\prod \ell_i) = \log p - \log f \leq \frac{1}{2}\log p$. We conclude that due to the large cofactors included in the primes targeted in this work, [84, Alg. 1 and Alg. 3] cannot perform a sound supersingularity test within our setting.

Luckily, in the primes as above, where $f = 2^k$, we can improve this algorithm to verify supersingularity: Instead of verifying that the order of a random point P has enough ℓ_i-torsion, we verify P has 2^k-torsion. When $\log f = k > \frac{1}{2}\log p$, verifying that P has 2^k-torsion implies that E_A must be supersingular by the same logic as above. Furthermore, for Montgomery curves E_A, we can sample P directly from $E_A(\mathbb{F}_p) \setminus [2]E_A$ by picking a point with rational non-square x-coordinate [105]. This ensures we always sample P with maximum 2^k-torsion. Using x-only arithmetic, we only have to keep track of x_P. We name this approach to verify supersingularity VeriFast, as described in Algorithm 8.

Algorithm 8 VeriFast: Supersingularity verification for primes with cofactor $2^k > 4\sqrt{p}$.

Input: $A \in \mathbb{F}_p$ defining a curve E_A
Output: `true` or `false`, verifying the supersingularity of E_A

$x_P \leftarrow 2$, $v_2 \leftarrow 1$
$x_P \leftarrow [\frac{p+1}{f}]x_P$
while $x_P \neq \mathcal{O}$ and $v_2 < k$ **do**
    $x_P \leftarrow \mathrm{xDBL}(x_P)$, $v_2 \leftarrow v_2 + 1$
if $v_2 > 1 + \log p/2$ and $x_P = \mathcal{O}$ **then**
    return `true`
return `false`

VeriFast can be performed deterministically or probabilistically: Given a point with rational non-square x-coordinate, the algorithm always returns $v_2 = k$ in case of supersingularity. Otherwise, any random point is likely to have v_2 close to k, and hence still verifies supersingularity if the cofactor is a few bits larger than $4\sqrt{p}$. For the probabilistic approach, we pick $x_P = 2 \in \mathbb{F}_p$, hence $P = (2, -)$, for all supersingularity checks. This has the advantage that multiplication by 2 can be performed as a simple addition, and hence, $x_P = 2$ optimizes the arithmetic in the computation of $x_P \leftarrow [\frac{p+1}{f}]x_P$. Furthermore, the bound $4\sqrt{p}$ can be improved to $2\sqrt{p}$ as this still implies $p + 1$ is the only

multiple in the Hasse interval. VeriFast is faster than any of the analyzed algorithms in [26], with a cost of $\mathcal{O}(\log p)$. More specifically, it requires a scalar multiplication by a scalar of $\log p - k$ bits and (at most) k point doublings, where $f = 2^k$ is the cofactor. In comparison to Doliskani's test [26, 131], also of complexity $\mathcal{O}(\log p)$, we have the advantage that we can stay over $\mathbb{F}_p$. The condition that $k > 1 + \frac{1}{2}\log p$ holds for our primes p5120 and beyond. More importantly, even with the probabilistic approach, for these primes the probability to sample a point that does *not* have large enough 2^z-torsion is lower than 2^{-256}. For the primes where $k \leq 1 + \frac{1}{2}\log p$, we can still use the 2^k-torsion, as in VeriFast, but we are required to also verify some ℓ_i-torsion to cross the bound $2\sqrt{p}$. A comparison of performance between VeriFast and previous methods is given in Table 10.3, showing VeriFast is 28 to 38 times as fast for large primes. The hybrid method for $k \leq 1 + \frac{1}{2}\log p$ still achieves a significant speedup.

Table 10.3: Benchmarking results for supersingularity verification using VeriFast for primes with cofactor $k > \frac{1}{2}\log p$, and hybrid (marked with *) when $k < \frac{1}{2}\log p$. Results of [90] added for comparison. Numbers are median clock cycles (in gigacycles) of 1024 runs on a Skylake CPU.

	p2048*	**p4096***	**p5120**	**p6144**	**p8192**	**p9216**
VeriFast	0.16	0.50	0.53	0.81	1.88	2.54
SQALE [90]	0.30	1.86	14.90	27.65	67.79	96.99

10.5.2 Optimized dCSIDH public keys

As described in Section 10.4.2, dCSIDH is dummy-free and deterministic by using secret key exponents $e_i \in \{-1, 1\}$, and public keys of the form (A, T_+, T_-). Recall, T_+ and T_- are full-torsion points that can be used to perform positive steps $\mathfrak{l}_i^{+1}$ and negative steps $\mathfrak{l}_i^{-1}$ respectively. For sampling suitable points T_+ and T_- for public keys during key generation, we use the Elligator map $(A, u) \mapsto (T_+, T_-)$ from [86], with Montgomery parameter $A \in \mathbb{F}_p$ and an Elligator seed $u \in \mathbb{F}_p$. The output of Elligator is exactly such a pair of points T'_+ and T'_-, although they might not be *full-torsion*, that is, their respective orders might not be divisible by *all* ℓ_i. Let P be either T_+ or T_-. To efficiently determine if P is a full-torsion point, we follow the usual product-tree approach that was also applied for public key validation in [84]. This requires us to compute $\left[\frac{p+1}{\ell_i}\right] P$

for each ℓ_i, and checking that these points are not equal to the point at infinity. In order to obtain a deterministic algorithm, we try Elligator seeds from a predefined sequence $(u_1, u_2, \ldots)$ until we find full-torsion points T_+ and T_-. To determine which of the points $T_\pm$ is T_+ resp. T_-, Elligator requires a Legendre symbol computation. In the case of our proposed dCSIDH configuration with public inputs A and u, we can use a fast non-constant-time algorithm for the Legendre symbol computation as the one presented in Hamburg [166].

Thus, a dCSIDH public key consists of an affine Montgomery coefficient $A \in \mathbb{F}_p$, and an Elligator seed $u \in \mathbb{F}_p$ such that elligator(A, u) returns two full-torsion points T_+ and T_- on E_A. We choose the fixed potential values for u small to get a public key (A, u) of only $\log_2(p) + \varepsilon$ bits for small $\varepsilon > 0$.

Finally, a user has to verify such a public key (A, u). For A, we verify E_A is supersingular as described in Section 11.5.2. For u, we verify that it generates two full-torsion points T_+ and T_-, by ensuring at the computation of each step $\mathfrak{l}_i^{\pm 1} * E$ that the correct multiple of *both* T_+ and T_- are not the point at infinity (i.e., both have order ℓ_i) regardless of which point we use to compute the step.

Remark 10.1. An alternative to finding and including an Elligator seed $u \in \mathbb{F}_p$ in the public key is to find and include small x-coordinates x_+ and x_- that define full-torsion points $T_+ = (x_+, -)$ and $T_- = (x_-, -)$. Information-theoretically, u and the pair (x_+, x_-) share similar probabilities (to generate full-torsion points) and hence their bitlengths should be comparatively small. One advantage of x_+ and x_- is that they can be found individually, which should speed up their search. We choose, however, the more succinct approach using u and Elligator.

10.6 Implementation

In this section, we describe the optimization steps at the level of field arithmetic to speed up both variants of CSIDH we consider. First and foremost, to enable a fair comparison, we implement a common code base for dCSIDH and CTIDH. Besides sharing the same field arithmetic, both instantiations of CSIDH share all the underlying functions required for computing the group action. However, some required parameters and the strategy within the group action strongly differ between dCSIDH and CTIDH. In the case of dCSIDH, the group action strategy and all the required parameters are based on the implementation provided by [90]. In the case of CTIDH, we generate the batching and other parameters using the methods provided by [22].

10.6.1 Low-level approaches for the field arithmetic layer

For the underlying field arithmetic, we implement three different approaches. They all share the representation of integers in radix 2^{64} and use Montgomery arithmetic for efficient reductions modulo p.

1. To establish a performance baseline, our first method uses the low-level functions for cryptography (`mpn_sec_`) of the GNU Multiple Precision Arithmetic Library (GMP). Modular multiplication uses a combination of `mpn_sec_mul` and `mpn_add_n` to implement Montgomery multiplication, i.e., interleaving multiplication with reduction. We refer to this first approach as **GMP**.

2. The second approach extends the optimized arithmetic from [84], using the MULX instruction, going from 512-bit and 1024-bit integers to the larger sizes we consider in this paper. Here, we also interleave multiplication with reduction; we generate code for all field sizes from a Python script. We refer to this second approach as **OpScan**.

3. Our third strategy uses Karatsuba multiplication [**Karatsuba:1963:MMN**] together with the MULX optimizations used in our second approach. We describe this strategy, and in particular an optimized reduction for primes of 5120 bits and above, in more detail in section 10.6.2. We refer to this third approach as **Karatsuba**.

We follow the earlier optimization efforts for CSIDH from [22, 84, 90] and focus on optimizing our code primarily on Intel's Skylake microarchitecture. More specifically, we perform all benchmarks on one core of an Intel Core E3-1260L (Skylake) CPU with hyperthreading and TurboBoost disabled. An overview of (modular) multiplication performance of the three approaches for the different field sizes is given in table 10.4. In the following, we will focus on describing the fastest of the three strategies mentioned above, i.e., **Karatsuba**, in more detail.

10.6.2 Optimized field arithmetic using MULX and Karatsuba

We present scripts to generate optimized code using the **Karatsuba** approach, based on the **OpScan** approach. More precisely, compared to the **OpScan** approach, we achieve speedups for multiplication, squaring, and reduction.

Table 10.4: Benchmarking results for multiplication and reduction. Numbers are median clock cycles of 100000 runs on a Skylake CPU. Note that for the **OpScan** and the **GMP** approach, we can only provide clock cycles for multiplication including reduction, due to the interleaved Montgomery reduction.

Prime	**GMP**	**OpScan**	**Karatsuba**		
	mult + redc	mult + redc	mult	redc	mult + redc
p2048	8662	4538	1442	2648	4090
p4096	34 030	20 318	4981	9777	14 758
p5120	51 671	33 676	8601	6528	15 129
p6144	74 338	53 746	10 210	9517	19 727
p8192	131 858	92 793	17 073	17 295	34 268
p9216	168 375	118 302	20 248	19 709	39 957

Multiplication.

The implementation of Karatsuba follows careful considerations to optimize performance. To improve efficiency, we select a breakout level into a MULX-based schoolbook multiplication with a maximum of 9×9 limbs. By choosing this threshold, the implementation aims to strike a balance between utilizing the benefits of Karatsuba's divide-and-conquer strategy and minimizing the overhead of stack operations. This leads to the following number of layers of Karatsuba: $2, 3, 4, 4, 4$, and 4 for the cases p2048, p4096, p5120, p6144, p8192, and p9216, respectively. To further enhance the speed of the implementation, the assembly code avoids function calls. By generating the assembly code dynamically, the implementation can adapt to different prime sizes and adjust the multiplication algorithm accordingly.

Squaring.

For squaring, we take advantage of the fact that some partial products ($a_i a_j$ such that $i \neq j$) only need to be calculated once, and then accumulated/used twice. On the lowest level of Karatsuba, where the schoolbook multiplication takes place, we implement a squaring function with the corresponding savings based on *lazy doubling* method [**DBLP:journals/jss/LeeKP13**] by adapting

the assembly code of the squaring function of the GMP library. For a given n, the implemented method achieves the lower bound of $\frac{n^2-n}{2} + n$ required multiplications. Furthermore, we save additions on the higher levels of Karatsuba by reusing calculated values. However, as shown in Table 10.5, due to the chosen breakout into schoolbook multiplication and the number of available registers, the effort for dealing with the carry chains only leads to a maximum speedup of 17%. Adding a layer of Karatsuba to reduce the number of limbs for the schoolbook multiplication leads to a speedup at this level. Overall, however, extra layers negate speed-ups gained from reducing limbs.

Table 10.5: Benchmarking results for multiplication and squaring for the **Karatsuba** approach. Numbers are median clock cycles of 100000 runs on a Skylake CPU.

Prime	**multiplication**	**squaring**
p2048	1442	1230
p4096	4981	4431
p5120	8601	7990
p6144	10 210	9120
p8192	17 073	15 050
p9216	20 248	19 197

Montgomery reduction.

For the cases $p \in \{\text{p5120, p6144, p8192, p9216}\}$, the reduction is calculated according to the *intermediate Montgomery reduction* [19]. For this, we use Montgomery-friendly primes of the form $p = f \cdot \prod_{i=1}^{n} \ell_i - 1$ with the cofactor $f = 2^{e_2}$ where $e_2 \geq \log_2(p)/2$. Table 10.1 shows the respective values for f and accordingly e_2 for all chosen prime numbers.

As shown in Algorithm 9, the basic idea of this reduction is to perform two Montgomery-reduction steps modulo 2^{e_2} instead of n steps modulo 2^w as in the standard Montgomery reduction. Based on this reduction approach, we can further apply the available Karatsuba-based multiplication when calculating $q_0 \times \alpha$ and $q_1 \times \alpha$ (see Line 2 and 4 in Algorithm 9), leading to further speedups.

For the cases $p \in \{\text{p2048, p4096}\}$, the respective primes cannot fulfill the

Algorithm 9 Intermediate Montgomery reduction for $p = 2^{e_2}\alpha - 1$ with $e_2 \geq \log_2(p)/2$

Input: $0 \leq a < 2^e p$
Output: $r = a2^{-2e_2} \mod p$ and $0 \leq r < p$
1: $q_0 \leftarrow a \mod 2^{e_2}$
2: $r_0 \leftarrow (a - q_0)/2^{e_2} + q_0 \times \alpha$ ▷ 1st reduction
3: $q_1 \leftarrow r_0 \mod 2^{e_2}$
4: $r \leftarrow (r_0 - q_1)/2^{e_2} + q_1 \times \alpha$ ▷ 2nd reduction
5: $r' \leftarrow r - p + 2^e$
6: **if** $r' \geq 2^e$ **then**
7: $\quad r \leftarrow r' \mod 2^e$
8: **return** r

described requirements. Hence, we implement the *word version of the Montgomery reduction* from [19] for these cases. The complexity of Algorithm 10 is dominated by multiplications by α in Line 4. Compared to the standard Montgomery reduction, this approach reduces the number of limbs to be multiplied depending on the value of e_2. We show the results for the corresponding reduction in Table 10.4.

Algorithm 10 Word version of the Montgomery reduction if $p = 2^{e_2}\alpha - 1$

Input: $0 \leq a < p\beta^n$
Output: $r = a\beta^{-n} \mod p$ and $0 \leq r < p$
1: $r \leftarrow a$
2: **for** $i = 0$ **to** $n - 1$ **do**
3: $\quad r_0 \leftarrow r \mod \beta$
4: $\quad r \leftarrow (r - r_0)/\beta + r_0 \times \alpha 2^{e_2 - w}$
5: $r' \leftarrow r + (\beta^n - p)$
6: **if** $r' \geq \beta^n$ **then**
7: $\quad r \leftarrow r' - \beta$
8: **return** r

10.6.3 Performance results

We demonstrate the performance increase due to the high-level improvements from Section 10.5 and the low-level improvements from Section 10.6.2 for dCSIDH

and CTIDH in Table 10.6. We compare our results to [90], the only other available implementation of CSIDH for similar parameters listing performance numbers. For parameter sizes above p5120, our implementation of dCSIDH is between 55% and 60% faster than SQALE (dummy-free), and CTIDH consistently achieves a speed-up of almost 75% compared to SQALE (OAYT). This excludes the significant speedup from VeriFast, as shown in Table 10.3.

Table 10.6: Benchmarking results for performing a group action for dCSIDH and CTIDH, excluding key validation. Results for the dummy-free and OAYT version of [90] added for comparison. Numbers are median clock cycles (in gigacycles) of 1024 executions on a Skylake CPU.

	p2048	**p4096**	**p5120**	**p6144**	**p8192**	**p9216**
dCSIDH	7.48	34.64	31.80	47.47	127.57	219.09
SQALE (dummy-free)	–	39.35	73.57	117.57	322.57	475.64
CTIDH	2.21	11.11	11.26	17.13	43.65	68.78
SQALE (OAYT)	–	23.21	44.56	74.88	199.15	292.41

In [**Lon22**], the authors proposed a novel approach for the computation of sums of products over large prime fields achieving a significant performance impact. However, since the primes in our work support very fast reductions, applying the approach from [**Lon22**] would not gain a significant advantage. Further, a comparison of the performance is unfortunately rather difficult due to the different underlying fields.

10.7 Non-Interactive Key Exchange in Protocols

Diffie–Hellman (DH) key exchange is probably the most well-known example of a NIKE protocol, even if it is often used as a "simple" interactive key exchange. One such example is TLS, where ephemeral DH key exchange is authenticated via a signature. This key exchange can be replaced with a KEM, as shown in [61]. Experiments by Google and Cloudflare [**CECPQ1**, **CECPQ2**, **CECPQ2b**] used the same approach.

However, in two scenarios the inherently interactive character of a KEM

creates issues for protocol designers. When used with long-term keys (and a suitable PKI), a NIKE allows a user Alice to send an authenticated ciphertext to an *offline* user Bob. Signal's X3DH handshake [**X3DH**] is a notable example using this feature of NIKEs. Indeed, [65] shows that a naive replacement of the DH operations by KEMs does not work.

In the early stages of the development of TLS 1.3, **EuroSP:KraWee16** proposed OPTLS [**EuroSP:KraWee16**], a variant that uses DH key exchange not only for ephemeral key exchange, but also for authentication. Many elements of this proposal, made it into the eventual RFC8446 [**RFC8446**]. Though the standard reverted to handshake signatures, the idea lives on in an Internet Draft [**ietf-tls-semistatic-dh-01**].

As **Kuh18** pointed out, OPTLS does use the non-interactive property of DH [**Kuh18**]. As part of the ephemeral key exchange, the client sends their ephemeral DH public key. For authentication, the server takes this ephemeral key share and combines it with their long-term DH key. The obtained shared secret is used to compute a MAC which is used in place of the signature in the `CertificateVerify` message. This computation proves the server's possession of the long-term secret key corresponding to the public key in the certificate. The client can compute the same shared secret by combining its ephemeral secret DH key with the certified public key, and thus verify the MAC.

10.7.1 Post-Quantum TLS without signatures

In a naive instantiation of an OPTLS-like protocol with KEMs, we require an additional round-trip. To compute the authentication message, the server needs to first receive the ciphertext that was encapsulated against the long-term public key held in its certificate—which the client can not send before having received it from the server. The KEMTLS proposal by Schwabe, Stebila, and Wiggers avoids this issue partially by letting the client already transmit data immediately after computing and sending the ciphertext to the server [261]. This relies on the fact that any keys derived from the shared secret encapsulated to the server's long term key are *implicitly authenticated*. KEMTLS has the advantage of not having to compute any typically expensive and/or large post-quantum signatures during the handshake protocol. Only the variant that assumes the client already has the server's public key, for example through caching, can achieve a protocol flow that is similar to OPTLS and TLS 1.3 [259]. In that flow, the server can send authenticated data immediately on their first response to the client.

However, as CSIDH does provide post-quantum NIKE we can use it to in-

stantiate post-quantum OPTLS and avoid any online post-quantum signatures. Because OPTLS immediately confirms the server's authenticity, its handshake has the same number of transmissions of messages as in TLS 1.3 and there is no need to rely on implicit authentication.

Integrating our implementations in OPTLS gives us an understanding of how CSIDH affects the performance of real-world network protocols, which will typically feature similar cryptographic operations and transmissions.

10.7.2 Benchmarking set-up

Integration into Rustls.

To investigate the performance of OPTLS with CSIDH, we integrate our optimized implementations into the implementation and the measurement framework of the authors of KEMTLS. As a side effect of this work, we also provide a Rust wrapper around our C implementations. We add OPTLS to the same modified version of Rustls [**rustls**] used to implement KEMTLS. This allows us to straightforwardly compare to KEMTLS and TLS 1.3 handshakes instantiated with post-quantum primitives.

Group operations and caching ephemeral key generation.

An OPTLS handshake requires a large number of group operations in each handshake, namely:

1. Generation of the ephemeral key of the client;
2. Generation of the ephemeral key of the server;
3. The server's computation of the ephemeral shared secret;
4. The client's computation of the ephemeral shared secret;
5. The server's computation of the authentication shared secret; and
6. The client's computation of the authentication shared secret.

Unfortunately, due to the order of the handshake messages and the requirements for handshake encryption, most of these computations need to be done in-order and can not really be parallelized. However, we can avoid the cost of CSIDH key generation by implementing caching of ephemeral keys. This

reduces the forward secrecy; but it emulates a best-case scenario for CSIDH-based OPTLS in which the keys are generated "offline", outside the handshake context. We exclude all first TLS handshakes from clients and servers from our measurements, to exclude this key generation time: in the *pregen* OPTLS instances, all subsequent handshakes use the same public key material. In the *ephemeral* OPTLS instances, we generate ephemeral keys in each handshake.

Note that because OPTLS combines the ephemeral and static keys, all need to use the same algorithm, and we can not use a faster KEM for ephemeral key exchange.

Measurement setup.

We run all TLS handshake experiments on a server with two Intel Xeon Gold 6230 CPUs, each featuring 20 physical cores. This gives us 80 hyperthreaded cores in total. For these experiments, we do *not* disable hyperthreading or frequency scaling, as these features would also be enabled in production scenarios. We run 80 servers and clients in parallel, as each pair of client and server roughly interleave their execution. We collect 8000 measurements per experiment. Every 11 handshakes, we restart the client and server, so that we measure many ephemeral keys even in the scenarios that use ephemeral-key caching. We exclude the first handshake from the measurements to allow for cache warm-up and ephemeral-key generation in the caching scenario.

As in the KEMTLS papers [259, 261], we measure the performance of the different TLS handshakes when run over two network environments: a low-latency 30.9TODO FIXTODO FIX round-trip time (RTT), 1000Mbps and a high-latency 195.5TODO FIX RTT, 10Mbps network connection. The latency of the former represents a continental, high-bandwidth connection, while the latter represents a transatlantic connection.

10.7.3 Benchmarking results

In table 10.7, we compare OPTLS with dCSIDH and CTIDH with the performance of instantiations of KEMTLS and TLS 1.3.

Comparing the sizes of the handshakes, OPTLS requires fewer bytes on the wire, as it only needs to transmit two ephemeral public keys and one static public key (and the CA signature). KEMTLS requires an additional ciphertext, and TLS an additional signature.

In OPTLS, like in TLS 1.3, the client receives the server's handshake completion message `ServerFinished` (SFIN) first and then sends the `ClientFinished`

Table 10.7: Public key cryptography transmission sizes in bytes and time in seconds until client receives and sends `Finished` messages for OPTLS, TLS 1.3 and KEMTLS.

				Handshake latencies (RTT, link speed)			
		Transmission		**30.9 ms, 1000 Mbps**		**195.5 ms, 10 Mbps**	
		KEX	**Auth**	**SFIN recv**	**CFIN sent**	**SFIN recv**	**CFIN sent**
OPTLS (pregen)	dCSIDH p2048	544	938	24.468	24.468	24.288	24.288
	CTIDH p2048	512	922	7.346	7.346	7.203	7.203
	CTIDH p4096	1024	1178	36.321	36.321	36.299	36.299
	CTIDH p5120	1280	1306	28.701	28.701	28.580	28.580
OPTLS (ephemeral)	dCSIDH p2048	544	938	43.642	43.642	43.486	43.486
	CTIDH p2048	512	922	10.042	10.042	9.882	9.882
	CTIDH p4096	1024	1178	50.039	50.039	49.951	49.951
	CTIDH p5120	1280	1306	42.383	42.383	42.163	42.163
TLS	Kyber512–Falcon512	1568	2229	0.064	0.064	0.428	0.428
	Kyber512–Dilithium2	1568	4398	0.063	0.063	0.519	0.519
	Kyber768–Falcon1024	2272	3739	0.065	0.065	0.497	0.497
KEMTLS	Kyber512	1568	2234	0.094	0.063	0.593	0.396
	Kyber768	2272	2938	0.094	0.063	0.597	0.400

All instantiations use Falcon-512 for the certificate authority; the CA public key is not transmitted. Bytes necessary for authentication includes 666 bytes for the Falcon-512 CA signature on the server's certificate.

(CFIN) message (and its request) immediately after. In KEMTLS, SFIN and full server authentication is received a full round-trip after CFIN is received. However, it is clear that the runtime requirements of dCSIDH are almost insurmountable, even for the smallest parameters (p2048). Even CTIDH, which is much more efficient, is orders of magnitude slower than the KEMTLS and OPTLS instances. If the more conservative p4096 prime is required for NIST level 1 security, even CTIDH handshakes do not complete in under 30 seconds. Due to a better reduction algorithm, the p5120 prime performs roughly on par with p4096, while providing NIST level 2 security in the aggressive analysis.

As discussed for (KEM)TLS in [**SPACE:GonzalezWiggers22**], for constrained environments, such as 46kbps IoT networks, in certain scenarios the transmission size can become the dominant factor instead of computation time. However, with the results shown here, we expect the environments in which CSIDH-based OPTLS instances are competitive to be very niche. To overcome

7 seconds of computational latency, the network needs to take more than 7 seconds to transmit the additional data required for e.g. TLS 1.3 with Dilithium. This suggests link speeds of less than 1 kilobyte per second. Additionally, these environments often rely on microcontrollers that are much less performant than the Intel CPUs on which we run our implementations.

Interestingly, the CSIDH experiments run on the high-latency, low-bandwidth networks show slightly lower latencies than those on the high-bandwidth, low-latency network. We suspect that this is due to an interaction with the TCP congestion control algorithm's transmission windows.

10.8 Conclusion and future work

In this paper, we presented low-level and high-level optimizations for CSIDH at larger parameter sets, focusing on deterministic and dummy-free behavior in dCSIDH, and on speed in CTIDH. These optimizations achieve impressive results on their own; dCSIDH is almost twice as fast as the state-of-the-art, and CTIDH, dropping determinism, is again three times as fast as dCSIDH. Further optimizations of the field arithmetic, i.e., by utilizing the vector processing capabilities of modern processors, might lead to additional speed-ups.

Nevertheless, when integrated into the latency-sensitive TLS variant OPTLS, both implementations still have too-large handshake latency in comparison to TLS or KEMTLS using lattice-based KEMs. We conclude that the reduced number of roundtrips, through the non-interactive nature of CSIDH, does not make up for the performance hit.

However, for truly non-interactive, latency-insensitive settings that cannot replace NIKEs by KEMs, the performance of CSIDH may be sufficient even at high-security levels. This includes, for example, using CSIDH in X3DH [**X3DH**] for post-quantum Signal, as it would incur a delay of seconds only when sending the first message to another user (who might be offline, thus ruling out KEM-based interactive approaches).

Unless significant performance improvements occur for CSIDH in large parameter sets, or the quantum-security debate shifts in favor of 512- to 1024-bits parameter sets, we conclude that CSIDH is unlikely to be practical in real-world applications, outside of those that specifically require NIKEs.

It will be interesting to investigate how CSIDH and Swoosh—the only two current proposals for a post-quantum NIKE—compare in a protocol context. There is no full implementation of Swoosh, yet; the cycle counts reported in [149] are for the passively-secure core component only. Based on the avail-

able figures it seems likely that SWOOSH outperforms CSIDH with the large parameters we consider in this paper *computationally*, but that key sizes are much smaller for CSIDH.

Chapter 11

Effective Pairings in Isogeny-based Cryptography

11.1 Placeholder

The paper will go here.

11.2 Introduction

In the event that quantum computers break current cryptography, post-quantum cryptography will provide the primitives required for digital security. Isogeny-based cryptography is a field with promising quantum-secure schemes, offering small public keys in key exchange (CSIDH [84]), and small signatures (SQISign [120]). The main drawback of isogeny-based cryptography is speed, as it requires heavy mathematical machinery in comparison to other areas of post-quantum cryptography. In particular, to ensure security against real-world side-channel analysis, the requirements for constant-time and leakage-free implementations cause a significant slowdown. Trends in current research in isogenies are, therefore, looking at new ideas to improve constant-time performance [**meyer2018faster**, **meyer2019lions**, **cervantes2019stronger**, **hutchinson2020further**, **AMC:ChiRod20**, **SQALE**, 22, 25, 68, 235], and analyzing side-channel threats

[**campos2020trouble**, **legrow2021short**, **ournote**, 28].

Surprisingly, although pairings were initially considered in SIDH and SIKE to improve the cost of key compression [106, 107], they have received little attention for optimizing CSIDH or later isogeny-based protocols. There are clear obstructions that heavily affect the performance of pairings: we have no control over the Hamming weight of p for the base fields (in CSIDH or SQISign), we are likely to compute pairings of highly-composite degree, and many optimizations in the pairing-based literature require different curve models than the ones we consider in isogeny-based cryptography. Nevertheless, the field of pairing-based cryptography is rich in ideas and altogether many small improvements can make pairings efficient even for unpractical curves. As a general technique, we can use pairings to analyze certain properties of *points on elliptic curves* by a pairing evaluation as *elements of finite fields*. In this way, a single pairing can be used to solve a curve-theoretical problem with only field arithmetic, which is much more efficient. Hence, even a relatively expensive pairing computation can become cost-effective if the resulting problem is much faster to solve "in the field" than "on the curve".

Pairings have been used constructively since the 2000s [167, 181]. The literature is rich, but the main focus has mostly been on pairings of prime degree. Although proposals using composite degree pairings have been analyzed, analysis such as Guillevic [165] shows that prime degrees are favorable. Composite degree pairings have thus received little attention compared to prime degree pairings.

Apart from the issues mentioned, CSIDH is well-suited for pairings, as it mostly works with points of order $N \mid p+1$ on supersingular curves $E/\mathbb{F}_p$ whose x-coordinate lives in $\mathbb{F}_p$. This allows for fast x-only arithmetic on the Kummer line $\mathbb{P}(\mathbb{F}_p)$, but also implies an embedding degree of 2 for pairings of degree $N \mid p+1$. The result $\zeta = e_N(P, Q)$ of a pairing is thus a norm-1 element in $\mathbb{F}_{p^2}$, and ζ contains useful information on P and Q, which is precisely the information that CSIDH often requires to perform certain isogenies. Hence, pairings are a natural tool to study properties of the pair (P, Q). Norm-1 elements in $\mathbb{F}_{p^2}$ allow for very fast $\mathbb{F}_p$-arithmetic (compared to $\mathbb{F}_{p^2}$-arithmetic). A final advantage is that for any supersingular curve E_A we choose over $\mathbb{F}_p$, the result of a pairing always ends up in $\mathbb{F}_{p^2}$. This allows for techniques that require fixing some public value in $\mathbb{F}_{p^2}$ which would not generalize to curves; it would require fixing a value for *all* supersingular curves E_A, independent of $A \in \mathbb{F}_p$.

Our contributions.

Our main contribution is combining an optimized pairing with highly-efficient arithmetic in $\mu_r \subseteq \mathbb{F}_{p^2}^*$ to solve isogeny problems faster. To achieve this, we first optimize the pairing and then apply this low-cost pairing to move specific problems from curves to finite fields. Specifically,

1. we optimize pairings on supersingular curves: in Miller'sAlgorithm, we first reduce the cost of subroutines Dbl and Add and then reduce the total number of subroutines using non-adjacent forms and windowing techniques.

2. we analyze the asymptotic and concrete cost of single and multi-pairings, in particular for supersingular curves over p512 (the prime used in CSIDH-512).

3. we apply these low-cost pairings to develop alternative algorithms for supersingularity verification, verifying full-torsion points, and finding full-torsion points, using highly-efficient arithmetic available for pairing evaluations.

4. we discuss the natural role these algorithms have when designing 'real-world' isogeny-based protocols, in particular, CSIDH-variants that are deterministic and secure against side-channel attacks.

5. we provide a full implementation of most of these algorithms in Rust, following the "constant-time" paradigm, that shows such algorithms can immediately be used in practice to speed up deterministic variants of CSIDH.

   Our implementation is available at:

   https://github.com/Krijn-math/EPIC

Related work.

This work can partly be viewed as a natural follow-up to [**cervantes2019stronger**, **SQALE**, 68], works that analyze CSIDH as a real-world protocol, that is, removing randomness and dummy operations. Independent work by **cryptoeprint:2023/753** applies pairings to improve the performance of SQISign [120]. This shows the potential of pairings in isogeny-based cryptography and we believe this work can contribute to improving performance even further.

Organization of the paper.

Section 12.3 introduces the mathematical tools in pairing-based and isogeny-based cryptography required in the rest of the paper. Section 11.4 analyzes optimization of pairings to the setting used in isogeny-based cryptography, which allows us to apply pairings to optimize general problems in Section 11.5. In Section 11.6, we show that these pairing-based algorithms speed up current variants of deterministic, dummy-free CSIDH and can be used to construct ideas beyond those in use now.

11.3 Preliminaries

Notation.

Throughout, p denotes a large prime used with the base field $\mathbb{F}_p$, and quadratic extension $\mathbb{F}_{p^2}$, realized as $\mathbb{F}_p(i)$ with $i^2 = -1$. Both ℓ and ℓ_i denote a small odd prime that divides $p+1$. A (supersingular) elliptic curve E_A is assumed to be in Montgomery form

$$E_A : y^2 = x^3 + Ax^2 + x, \quad A \in \mathbb{F}_p,$$

although our work also applies to other curve forms (most notably Edwards). μ_r denotes the set of r-th roots in $\mathbb{F}_{p^2}^*$. In particular, μ_{p+1} can be seen as $\mathbb{F}_{p^2}^*/\mathbb{F}_p^*$, the elements in $\mathbb{F}_{p^2}$ of norm 1, and μ_r for $r \mid p+1$ is a subgroup of μ_{p+1}.

Finite field operations are denoted as $\mathbf{M}$ for multiplications, $\mathbf{S}$ for squarings, and $\mathbf{A}$ for additions. Inversions ($\mathbf{I}$) and exponentiation ($\mathbf{E}$) are expressed in $\mathbf{M}$, $\mathbf{S}$ and $\mathbf{A}$ as far as possible. We use a cost model of $1\mathbf{S} = 0.8\mathbf{M}$ and $1\mathbf{A} = 0.01\mathbf{M}$ to compare performance in terms of finite field multiplications.

11.3.1 Isogeny-based cryptography

This work deals with specific problems in isogeny-based cryptography. We assume a basic familiarity with elliptic curve arithmetic, e.g. Montgomery ladders and addition chains. A great introduction is given by Costello and Smith [110].

The prime p.

We specifically look at supersingular elliptic curves over $\mathbb{F}_p$, where $p = h \cdot \prod_{i=1}^{n} \ell_i - 1$, with h is a suitable cofactor and the ℓ_i are small odd primes. We refer to these ℓ_i as *Elkies primes* [84]. We denote the set of Elkies primes as

L_χ[1] and write $\ell_\chi = \prod_{\ell_i \in L_\chi} \ell_i$. Hence, $\log p = \log h + \log \ell_\chi$. If h is large, the difference in bit-size between ℓ_χ and $p+1$ can be significant, and this can impact performance whenever an algorithm takes either $\log \ell_\chi$ or $\log p$ steps. For p512, h is only 4 and so we do not differentiate between the two.

Torsion points.

Let E be a supersingular elliptic curve over $\mathbb{F}_p$, then E has $p+1$ rational points. Such points $P \in E(\mathbb{F}_p)$ therefore have order $N \mid p+1$. When $\ell_i \mid N$, we say P has ℓ_i-torsion. When P is of order $p+1$, we say P is a *full-torsion* point. The twist of E over $\mathbb{F}_p$ is denoted by E^t, and E^t is also supersingular. Rational points of E^t can also be seen as $\mathbb{F}_{p^2}$-points in $E[p+1]$ of the form (x, iy) for $x, y \in \mathbb{F}_p$. Using x-only arithmetic, we can do arithmetic on *both* $E(\mathbb{F}_p)$ and $E^t(\mathbb{F}_p)$ using only these rational x-coordinates.

CSIDH.

We briefly revisit CSIDH [84] to show where full-torsion points appear, and refer to [**meyer2018faster**, 84, 235] for more details. CSIDH applies the class group action of $\mathcal{Cl}(\mathcal{O})$ on supersingular elliptic curves E_A over $\mathbb{F}_p$ whose rational endomorphism ring $\mathrm{End}_p(E_A) \xrightarrow{\sim} \mathcal{O}$ to create a non-interactive key exchange. Given a starting curve E_0, Alice's private key is an ideal class $[\mathfrak{a}] \in \mathcal{Cl}(\mathcal{O})$ and her public key is $E_A := \mathfrak{a} * E_0$, and equivalent for Bob with $[\mathfrak{b}]$ and $E_B := \mathfrak{b} * E_0$. Both can derive the shared secret $E_{AB} := \mathfrak{a} * E_B = \mathfrak{b} * E_A$, given only the other's public key. In reality, we cannot sample random ideal classes $[\mathfrak{a}] \in \mathcal{Cl}(\mathcal{O})$. Instead, we generate $\mathfrak{a}$ as a product of small ideals $\mathfrak{l}_i = (\ell_i, \pi - 1)$ and $\mathfrak{l}_i^{-1} = (\ell_i, \pi + 1)$ (the decomposition of (ℓ_i) into prime ideals), e.g., $\mathfrak{a} := \prod \mathfrak{l}_i^{e_i}$, where the e_i are secret.

Evaluating $\mathfrak{a} * E$ is done by the factorization of $\mathfrak{a}$ into $\mathfrak{l}_i$, where each $\mathfrak{l}_i^{\pm 1}$ can be evaluated using Vélu's formulas [**velu**] if we have a point $P \in \ker(\pi \pm 1) \cap E[\ell_i]$. This requirement comes down to P being a rational point of order ℓ_i, with $\pi P = P$, hence $P \in E(\mathbb{F}_p)$ and $\mathfrak{l}_i^{+1} * E$ is evaluated as $E \to E/\langle P \rangle$, or $\pi P = -P$, hence P lives on the twist E^t and $\mathfrak{l}_i^{-1} * E$ is evaluated as $E^t \to E^t/\langle P \rangle$.

By sampling random points $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$ of order ℓ_i, we can use the right scalar multiple of either P or Q to compute the action of $\mathfrak{l}_i$ resp. $\mathfrak{l}_i^{-1}$ whenever ℓ_i divides $\mathrm{Ord}(P)$ resp. $\mathrm{Ord}(Q)$. The original points P and Q can then be pulled through the isogeny and used again as a new set of points on the co-domain [**meyer2018faster**, 84, 235]. By repeating this procedure and sampling new points P, Q when necessary, we compute the full action of $\mathfrak{a} * E$.

[1] pronounced "ell-kie"

As P and Q are sampled randomly, they have probability $\frac{\ell_i - 1}{\ell_i}$ that ℓ_i divides their order.

Deterministic CSIDH.

The probabilistic nature of the evaluation of $\mathfrak{a} * E$, stemming from the random sampling of points P, Q, causes several issues, as randomness makes constant-time implementations difficult [**SQALE**, 22, 68], leaks secret information through physical attacks [28], and requires a good source of entropy, which can be expensive or difficult on certain devices.

One way to avoid this random nature of $\mathfrak{a} * E$ is to ensure that both P and Q are *full-torsion points*, e.g., we have $\mathrm{Ord}(P) = \mathrm{Ord}(Q) = p+1$. For a point P that is not a full-torsion point, we say P *misses* some torsion ℓ_i and we denote the *missing torsion* for P by $\mathsf{Miss}(P)$. Note that $\mathsf{Miss}(P) \cdot \mathsf{Ord}(P) = p + 1$. CSIDH strategies that require only two full-torsion points $T_+ \in E(\mathbb{F}_p)$ and $T_- \in E^t(\mathbb{F}_p)$ were discussed in [**cervantes2019stronger**] and implemented and improved by [**SQALE**, 68]. These restrict to coefficients $e_i \in \{-1, +1\}$ to remove randomness and dummy operations.

11.3.2 Building blocks in isogeny-based cryptography.

We list several general routines in isogeny-based cryptography that will be analyzed in more detail in later sections. These routines are posed as general problems, with their role in CSIDH specified afterward.

1. Finding the order of a point: Given $P \in E(\mathbb{F}_p)$, find the order $\mathrm{Ord}(P)$.
2. Verifying supersingularity: Given $A \in \mathbb{F}_p$, verify E_A is supersingular.
3. Verifying full-torsion points: Given two points $P \in E(\mathbb{F}_p)$, $Q \in E^t(\mathbb{F}_p)$ verify P and Q are full-torsion points.
4. Finding full-torsion points: Given a curve E_A, find two full-torsion points $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$.

It is easy to see that these problems are related. For example, verifying supersingularity is usually done by verifying that E_A has order $p+1$, by showing that there is a $\mathbb{F}_p$-rational point of order $N \geq 4\sqrt{p}$. This implies $N \mid \#E_A(\mathbb{F}_p)$ and hence we must have $\#E(\mathbb{F}_p) = p+1$ as $p+1$ is the only possible remaining value in the Hasse interval.

All variants of CSIDH use a supersingularity verification in order to ensure a public key E_A is valid. dCSIDH [68] includes full-torsion points (P, Q) in the public key to speed up the shared-secret computation. This requires finding full-torsion points in key generation and, given the public key, verifying such points (P, Q) are full-torsion before deriving the shared secret. Without this verification, a user is vulnerable to side-channel attacks.

11.3.3 Pairing-based cryptography

One of the goals in pairing-based cryptography is to minimize the cost of computing a Weil or Tate pairing. We assume a basic familiarity with pairings up to the level of Costello's tutorial [103]. Other great resources are Galbraith [153] and Scott [265]. We focus only on the reduced Tate pairing, as it is more efficient for our purposes. We build on top of the fundamental works [34, 35, 217, 288].

The reduced Tate pairing.

In this work, we are specifically focused on the reduced Tate pairing of degree r for supersingular elliptic curves with embedding degree $k = 2$, which can be seen as a bilinear pairing

$$e_r : E[r] \times E(\mathbb{F}_{p^2})/rE(\mathbb{F}_{p^2}) \to \mathbb{F}_{p^2}^*/(\mathbb{F}_{p^2}^*)^r.$$

In the *reduced* Tate pairing, the result $\zeta = e_r(P, Q)$ is raised to the power $k = (p^2 - 1)/r$, which ensures ζ^k is an r-th root of unity in μ_r. In this work, we want to evaluate the Tate pairing on points $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$ of order r. For supersingular curves over $\mathbb{F}_p$ and $r \mid p+1$, such points generate all of $E[r]$. From the point of view of pairings, $E(\mathbb{F}_p)$ is the *base-field subgroup* and $E^t(\mathbb{F}_p)$ is the *trace-zero subgroup* of $E[p+1]$. Using the bilinear properties of the Tate pairing, we can compute $e_r(P, Q)$ from its restriction to $E(\mathbb{F}_p) \times E^t(\mathbb{F}_p)$.

Computing the Tate pairing.

There are multiple ways to compute the Tate pairing [**robert1**, 199, 217, 274]. Most implementations evaluate e_r in three steps.

1. compute the Miller function f_{rP}, satisfying $\mathrm{div}(f_{rP}) = r(P) - r(\mathcal{O})$,
2. evaluate f_{rP} on an appropriate divisor D_Q,

3. raise $f_{rP}(D_Q)$ to the appropriate power, $\frac{p^2-1}{r}$, i.e., $e_r(P,Q) = f_{rP}(D_Q)^{p-1}$.

In practice, f_{rP} is a function in x and y of degree r, where r is cryptographically large, and therefore infeasible to store or evaluate. Miller's solution is a bitwise computation and direct evaluation of f_{rP} on D_Q to compute $f_{rP}(D_Q)$ in $\log(r)$ steps. By the work of Barreto, Kim, Lynn, and Scott [35, Theorem 1], we are in the fortunate situation that we can choose $D_Q = Q$. The Hamming weight of r is a large factor in the cost of computing $f_{rP}(Q)$ as a single step in Miller's loop takes close to twice the computational cost if the bit is 1. For our purposes, $r = p+1$ or $r = \ell_\chi$, and thus we have little control over the Hamming weight of r. The last step is also known as *the final exponentiation*. Algorithm 11 describes Miller'sAlgorithm before applying any optimizations, where $l_{T,T}$ and $l_{T,P}$ denote the required line functions (see [103, § 5.3]). We refer to the specific subroutines in Line 3 as Dbl and Line 5 as Add.

Algorithm 11 Miller'sAlgorithm

Input: $P \in E(\mathbb{F}_p)$, $Q \in E^t(\mathbb{F}_p)$, r of embedding degree $k = 2$, with $r = \sum_{i=0}^{t} t_i \cdot 2^i$
Output: The reduced Tate pairing $e_r(P,Q) \in \mu_r$
1: $T \leftarrow P$, $f \leftarrow 1$
2: **for** i from $t-1$ to 0 **do**
3: $\quad T \leftarrow 2T$, $f \leftarrow f^2 \cdot l_{T,T}(Q)$ ▷ Dbl
4: $\quad$ **if** $t_i = 1$ **then**
5: $\quad\quad T \leftarrow T + P$, $f \leftarrow f \cdot l_{T,P}(Q)$ ▷ Add
6: **return** f^{p-1}

More generally, the value f is updated according to the formula

$$f_{(n+m)P} = f_{nP} \cdot f_{mP} \cdot \frac{l}{v} \tag{11.1}$$

where l and v are the lines that arise in the addition of nP and mP. Miller'sAlgorithm uses $n = m$ to double $T = nP$, or $m = 1$ to add $T = nP$ and P.

11.3.4 Field arithmetic

The result of the reduced Tate pairing is a value $\zeta \in \mu_r \subset \mathbb{F}_{p^2}^*$ of norm 1, as it is an r-th root of unity. We require two useful algorithms from finite field arithmetic: Lucas exponentiation and Gauss'sAlgorithm to find primitive roots.

Gauss's algorithm.

An algorithm attributed to Gauss [207, p. 38] to find primitive roots of a certain order in a finite field is given in Algorithm 12, specialized to the case of finding a generator α for a finite field $\mathbb{F}_q$. It assumes a subroutine Ord computing the order of any element in the finite field.

Algorithm 12 Gauss'sAlgorithm.

Input: A prime power $q = p^k$.
Output: A generator α for $\mathbb{F}_q^*$.

1: $\alpha \xleftarrow{\$} \mathbb{F}_q^*$, $t \leftarrow \mathsf{Ord}(\alpha)$
2: **while** $t \neq q - 1$ **do**
3: $\quad \beta \xleftarrow{\$} \mathbb{F}_q^*$, $s \leftarrow \mathsf{Ord}(\beta)$
4: $\quad$ **if** $s = q - 1$ **then return** β
5: $\quad$ **else**
6: $\quad\quad$ Find $d \mid t$ and $e \mid s$ with $\gcd(d, e) = 1$ and $d \cdot e = \mathrm{lcm}(t, s)$
7: $\quad\quad$ Set $\alpha \leftarrow \alpha^{t/d} \cdot \beta^{s/e}$, $t \leftarrow d \cdot e$
8: **return** α

Gauss'sAlgorithm is easy to implement and finds generators quickly. The main cost is computing the orders. We can adapt Gauss'sAlgorithm to elliptic curves to find generators for $E(\mathbb{F}_p)$, simply by replacing the rôles of α, β by rational points P, P' until P reaches $\mathsf{Ord}(P) = p + 1$. Intuitively, one could say we "add" the torsion that P is missing using the right multiple of P'.

Lucas exponentiation.

Lucas exponentiation provides fast exponentiation for $\zeta \in \mathbb{F}_{q^2}$ of norm 1. They are used in cryptography since 1996 [182, 183], and specifically applied to pairings by Scott and Barreto [266]. We follow their notation.

Let $\zeta = a + bi \in \mathbb{F}_{p^2}$ be an element of norm 1, i.e. $a^2 + b^2 = 1$, then ζ^k can be efficiently computed using only $a \in \mathbb{F}_p$ for every $k \in \mathbb{N}$ using Lucas sequences, based on simple laddering algorithms. We denote these sequences by $V_k(a)$ and $U_k(a)$ but often drop a for clarity. The central observation is

$$\zeta^k = (a + bi)^k = V_k(2a)/2 + U_k(2a) \cdot bi, \quad \text{for } \zeta \in \mu_{p+1},$$

where

$$V_0 = 2, \quad V_1 = a, \quad V_{k+1} = a \cdot V_k - V_{k-1},$$
$$U_0 = 0, \quad U_1 = 1, \quad U_{k+1} = a \cdot U_k - U_{k-1}.$$

Given V_k, we can compute U_k by $(a \cdot V_k - 2 \cdot V_{k-1})/(a^2 - 4)$. An algorithmic description is given in [266, App. A]. For this work, we only require the value of $V_k(2a)$. As such, exponentiation of norm-1 elements is much more efficient than general exponentiation in $\mathbb{F}_{p^2}^*$: the former requires $1\mathbf{S}+1\mathbf{M}$ per bit of k, whereas the latter requires roughly $2\mathbf{S}+\frac{5}{2}\mathbf{M}$ per bit of k, assuming the Hamming weight of k is $\log(k)/2$. In our cost model, this is an almost 60% improvement. We denote the cost of exponentiation per bit for norm-1 elements by C_{Lucas}.

This arithmetic speed-up is key to the applications in this work: as the required pairings have evaluations of norm 1, we can apply Lucas exponentiation to the results to get very fast arithmetic. In comparison to x-only arithmetic on the curve, we are between five and six times faster per bit. Hence, if the cost of the pairing is low enough, the difference in cost between curve arithmetic and Lucas exponentiation is so large that it makes up for the cost of the pairing.

11.4 Optimizing pairings for composite order

In this section, we apply several techniques to decrease the cost of Miller'sAlgorithm, specifically for pairings of degree $r \mid p+1$, and points $P \in E(\mathbb{F}_p)$, $Q \in E^t(\mathbb{F}_p)$, with E supersingular and $p = h \cdot \ell_\chi - 1$. This is a different scenario than pairing-based literature usually considers: we have no control over the Hamming weight of $p+1$, and we compute pairings of composite degree.

We first give an abstract view and then start optimizing Miller'sAlgorithm. In Section 11.4.2, we decrease the cost per subroutine Dbl/Add with known optimizations that fit our scenario perfectly. In Section 11.4.3, we decrease the number of subroutines Dbl/Add using non-adjacent forms and (sliding) window techniques, inspired by finite field exponentiation and elliptic curves scalar multiplication.

11.4.1 An abstract view on pairings

Silverman [272], views the reduced Tate pairing as a threefold composition

$$E[r] \to \text{Hom}(E[r], \mu_r) \to \mathbb{F}_{p^2}^*/\mathbb{F}_{p^2}^{*,r} \xrightarrow{z \mapsto z^{(p^2-1)/r}} \mu_r(\mathbb{F}_{p^2}) \tag{11.2}$$

similar to the one described in Section 11.3.3. Namely, for $r = p + 1$, we can reduce the first map to $E(\mathbb{F}_p) \to \mathrm{Hom}(E^t(\mathbb{F}_p), \mu_r)$ to get

$$\Psi : E(\mathbb{F}_p) \to \mathrm{Hom}(E^t(\mathbb{F}_p), \mu_r), \quad P \mapsto e_r(P, -),$$

which can be made concrete as the Miller function $P \mapsto f_{rP}$. By composing with its evaluation on Q, we get $f_{rP}(Q) = e_r(P, Q)$ (unreduced). To $f_{rP}(Q)$, we apply the final exponentiation $z \mapsto z^{(p^2-1)/r}$. In the case of $r = p + 1$, we thus get the reduced Tate pairing $e_{p+1}(P, Q)$ as $\zeta = f_{(p+1)P}(Q)^{p-1}$.

From this point of view, identifying full-torsion points $P \in E(\mathbb{F}_p)$ is equivalent to finding points P that map to isomorphisms $f_{rP} \in \mathrm{Hom}(E^t(\mathbb{F}_p), \mu_r)$. We make this precise in the following lemma.

Lemma 11.1. Let E be a supersingular curve over $\mathbb{F}_p$. Let $P \in E(\mathbb{F}_p)$ and $r = p + 1$. Then f_{rP} as a function $E^t(\mathbb{F}_p) \to \mu_r$ has kernel

$$\ker f_{rP} = \{Q \in E^t(\mathbb{F}_p) \mid \mathsf{Ord}(P) \text{ divides } \mathsf{Miss}(Q)\}.$$

Hence, $|\ker f_{rP}| = \mathsf{Miss}(P)$. Thus, if P generates $E(\mathbb{F}_p)$, the kernel is trivial.

Proof. Recall that $\mathsf{Ord}(P) \cdot \mathsf{Miss}(P) = p + 1$. The reduced Tate pairing maps precisely to an exact $p + 1$-th root of unity if and only if P and Q are a torsion basis for $E[p + 1]$ [272]. Note that if $P \in E(\mathbb{F}_p)$ does not have order $p + 1$, we can write $P = [\mathsf{Miss}(P)]T_+$ for some specific point $T_+ \in E(\mathbb{F}_p)$ oforder $p + 1$, and similarly $Q = [\mathsf{Miss}(Q)]T_-$ for some $T_- \in E^t(\mathbb{F}_p)$. Hence $e_{p+1}(T_+, T_-)$ is an exact $p + 1$-th root of unity, and

$$\zeta = e_{p+1}(P, Q) = e_{p+1}(T_+, T_-)^{\mathsf{Miss}(P)\cdot\mathsf{Miss}(Q)}$$

is a $p + 1$-th root of unity with $\mathsf{Miss}(\zeta) = \mathrm{lcm}(\mathsf{Miss}(P), \mathsf{Miss}(Q))$. Whenever $\mathsf{Ord}(P)$ divides $\mathsf{Miss}(Q)$, we know that $p + 1$ divides $\mathsf{Miss}(P) \cdot \mathsf{Miss}(Q)$ and so we must have $\mathrm{lcm}(\mathsf{Miss}(P), \mathsf{Miss}(Q)) = p + 1$, i.e. $\zeta = 1$. As $\mathsf{Ord}(P) \mid \mathsf{Miss}(Q)$ implies $\mathsf{Ord}(Q) \mid \mathsf{Miss}(P)$, we can generate all such points Q by a single point R of order $\mathsf{Miss}(P)$, giving us $\ker f_{rP} = \langle R \rangle$ of order $\mathsf{Miss}(P)$. □ □

Note that, given a full-torsion point $P \in E(\mathbb{F}_p)$, we can thus identify full-torsion points $Q \in E^t(\mathbb{F}_p)$ as points where $e_r(P, Q)$ is a primitive root in μ_r. In light of Lemma 11.1, we can try to tackle the routines sketched in Section 11.3.2 using properties of f_{rP}, $f_{rP}(Q)$ and $\zeta = f_{rP}(Q)^{p-1}$. For example, we can find $\ker f_{rP}$ by evaluating f_{rp} on multiple points Q_i, and finding the orders of the resulting elements ζ_i. In the language of pairing-based cryptography, we compute multiple pairings $e_r(P, Q_i)$ for the same point P. Hence, we need to minimize the cost of several evaluations of the Tate pairing for fixed P but different points Q_i.

11.4.2 Reducing the cost per subroutine of Miller's loop

We now optimize the cost per Dbl and Add in Miller'sAlgorithm. We assume that $P \in E(\mathbb{F}_p)$ is given by $\mathbb{F}_p$-coordinates $x_P, y_P \in \mathbb{F}_p$, and $Q \in E^t(\mathbb{F}_p)$ can be given by $\mathbb{F}_p$-coordinates $x_Q, y_Q \in \mathbb{F}_p$ (we implicitly think of Q as $(x_Q, i \cdot y_Q)$).

Some of these techniques were used before in SIDH and SIKE [106, 107], in a different situation: in SIDH and SIKE, these pairings were specifically applied for $p = 2^{e_2} \cdot 3^{e_3} - 1$, whereas we assume $p = h \cdot \ell_\chi - 1$. Thus, we have much more different $\ell_i \mid p+1$ to work with, and we cannot apply most of their techniques.

Representations.

For T, we use projective coordinates to avoid costly inversions when doubling T, adding P and computing $\ell_{T,T}$ and $\ell_{T,P}$. For Q, as we only evaluate Q in $\ell_{T,T}$ and $\ell_{T,P}$, we leave Q affine. $f = a + bi$ is an $\mathbb{F}_{p^2}$-value represented projectively as $(a : b : c)$, with $a, b, c \in \mathbb{F}_p$ and c as the denominator. Although x-only pairings exist [151], they seem unfit for this specific scenario.

The final exponentiation.

As established before, after computing $f_{rP}(Q) \in \mathbb{F}_{p^2}$ we perform a final exponentiation by $p-1$. This is beneficial for two reasons:

1.) Raising to the power p is precisely applying Frobenius $\pi : z \mapsto z^p$ in $\mathbb{F}_{p^2}$, and so $\pi : a + bi \mapsto a - bi$. Hence we can compute $z \mapsto z^{p-1}$ as $z \mapsto \frac{\pi(z)}{z}$. The Frobenius part is 'free' in terms of computational cost. In $\mathbb{F}_{p^2}$, $z \mapsto z^{-1}$ is simply $(a+bi)^{-1} = \frac{(a-bi)}{a^2+b^2}$. Hence, the $\mathbb{F}_p$-inversion of $a^2 + b^2$ is the dominating cost of the final exponentiation. In constant-time, this costs about $\log p$ multiplications. When a and b are public, we use faster non-constant-time inversions. In comparison to prime pairings, this final exponentiation is surprisingly efficient.

2.) Raising z to the power $p-1$ for $\mathbb{F}_{p^2}$-values gives the same as $\alpha \cdot z$ for $\alpha \in \mathbb{F}_p^*$, as $(\alpha \cdot z)^{p-1} = \alpha^{p-1} \cdot z^{p-1} = z^{p-1}$. Hence, when we compute $f_{rP}(Q) \in \mathbb{F}_{p^2}$, we can ignore or multiply by $\mathbb{F}_p$-values [35], named "denominator elimination" in pairing literature. That is, we ignore the denominator c in the representation $(a : b : c)$ of f in the Miller loop, and similarly in evaluating $l_{T,T}(Q)$ and $l_{T,P}(Q)$, saving several $\mathbb{F}_p$-operations in Dbl/Add [88, 263].

Reusing intermediate values.

In Dbl, computing $T \leftarrow 2T$ shares many values with the computation of $l_{T,T}$ and in Add computing $T \leftarrow T + P$ shares values with $l_{T,P}$. Reusing such values saves again several $\mathbb{F}_p$-operations in Dbl/Add.

Improved doubling formulas.

As shown in [106, §4.1], the subroutine Dbl in Miller'sAlgorithm is more efficient when using a projective representation of $T \in E(\mathbb{F}_p)$ as (X^2, XZ, Z^2, YZ), although this requires a slight adjustment of the formulas used in Add. Overall, this reduces the cost for Dbl to $5\mathbf{S}+15\mathbf{M}$ and the cost for Add becomes $4\mathbf{S}+20\mathbf{M}$, for an average of $7\mathbf{S} + 25\mathbf{M}$ per bit.

11.4.3 Reducing the number of subroutines in the Miller loop

Next to reducing the cost for a single Dbl/Add, we apply techniques to reduce the total number of Adds. Usually in pairing-based cryptography, we do so by using primes p of low Hamming weight. Here we do not have this freedom, thus we resort to techniques inspired by exponentiation in finite fields.

Non-Adjacent Form.

With no control over the Hamming weight of $p + 1$, we assume half of the bits are 1. However, in Miller'sAlgorithm, it is as easy to add $T \leftarrow T + P$ as it is to subtract $T \leftarrow T - P$ (which we denote Sub), with the only difference being a negation of y_P. Hence, we use *non-adjacent forms* (NAFs [249]) to reduce the number of Add/Subs. A NAF representation of $p + 1$ as

$$p + 1 = \sum_{i=0}^{n} t_i \cdot 2^i, \quad t_i \in \{-1, 0, 1\},$$

reduces the Hamming weight from $\log(p)/2$ to $\log(p)/3$, and thus decrease the number of expensive Add/Subs in Miller'sAlgorithm by $\log(p)/6$. We get an average cost of $6\frac{1}{3}\mathbf{S} + 21\frac{2}{3}\mathbf{M}$ per bit, a saving of about 10%.

Algorithm 13 gives a high-level overview of the Miller loop with the improvements so far, for general p. Note that, as the output $\zeta \in \mathbb{F}_{p^2}$ will have norm 1, we only require the real part of ζ. See Section 11.A for the specific algorithms Dbl, Add and Sub, which are implemented in `bignafmiller.rs`.

Algorithm 13 Miller's algorithm, using NAFs

Input: $x_P, y_p, x_Q, y_Q \in \mathbb{F}_p$, $p+1 = \sum_{i=0}^{t} t_i \cdot 2^i$
Output: The real part of $e_r(P,Q) \in \mu_{p+1}$

$T = (X^2, XZ, Z^2, YZ) \leftarrow (x_P^2, x_P, 1, y_P)$
$f \leftarrow (1, 0)$
for i from $t-1$ to 0 **do**
    $(T, f) \leftarrow \mathsf{Dbl}(T, f, x_Q, y_Q)$
    if $t_i = 1$ **then**
        $(T, f) \leftarrow \mathsf{Add}(T, f, x_P, y_P, x_Q, y_Q)$
    if $t_i = -1$ **then**
        $(T, f) \leftarrow \mathsf{Sub}(T, f, x_P, y_P, x_Q, y_Q)$
$a \leftarrow f[0]$, $b \leftarrow f[1]$
$\zeta \leftarrow \frac{a^2-b^2}{a^2+b^2}$ ▷ The final exponentiation
return ζ

For specific primes.

For specific primes p, such as `p512`, we can improve on the NAF representation by using windowing techniques [185, 186]. This allows us to decrease even further the times we need to perform Add or Sub, at the cost of a precomputation of several values.

In short, windowing techniques allow us to not only add or subtract P but also multiples of P during the loop. To do so, we are required to precompute several values, namely $iP, -iP, f_{iP}$ and f_{-iP}. We need the multiples $\pm iP$ to perform $T \leftarrow T \pm iP$, and the line values f_{iP} to set $f \leftarrow f \cdot f_{\pm iP} \cdot \ell_{T,iP}(Q)$ in Add/Sub. We first precompute the required iP in projective form, and we keep track of f_{iP}. We use Montgomery's trick to return the points iP in affine form at the cost of a single inversion and some multiplications. Using affine form decreases the cost of $T \leftarrow T \pm iP$ during Add/Sub.

We note that iP gives $-iP$ for free, simply by negating y_{iP}. Furthermore, from $f_{iP} = a + bi$ we can obtain f_{-iP} as $f_{iP}^{-1} = \frac{a-bi}{a^2+b^2}$. However, as $a^2 + b^2 \in \mathbb{F}_p^*$, we can ignore these (thanks to the final exponentiation) and simply set $f_{-iP} = a - bi$. Altogether, these sliding-window techniques reduce the number of Add/Subs from $\log(p)/3$ down to about $\log(p)/(w+1)$. See `bigwindowmiller.rs` for the implementation of this algorithm.

For the prime `p512`, we found the optimum at a window size $w = 5$. This requires a precomputation of $\{P, 3P, \ldots, 21P\}$. Beyond $w = 5$, the cost of

additional computation does not outweigh the decrease in Add/Subs. Altogether this gives another saving of close to 10% for this prime.

Remark 11.1. A reader who is familiar with curve arithmetic might be tempted to suggest (differential) addition chains at this point, specialized for $p+1$, to decrease even further the number of Add/Subs. However, an addition chain requires either keeping track of the different points in the chain in projective form, hence performing projective additions/subtractions, or, converting them to affine points and performing affine additions/subtractions. Both approaches are not cost-effective; the amount of additions/subtractions hardly decreases, and the cost per addition/subtraction increases significantly. The crucial difference is that the precomputed points can be mapped into affine form using a single shared inversion and a few multiplications.

11.4.4 Multiple pairing evaluations.

The previous sections attempt to optimize a single pairing $e_r(P,Q)$. However, in many scenarios, including the ones in Section 11.5, it is beneficial to optimize the cost of multiple pairings, in particular multiple pairings $e_r(P,Q_1),\ldots e_r(P,Q_k)$ for the same point P. This is known as the "one more pairing" problem in pairing literature. We quickly sketch two methods to do so. Firstly, assuming the set $\{Q_1,\ldots,Q_k\}$ is known in advance and we minimize the overall cost of these k pairings. Secondly, assuming we want to compute additional pairings $e_r(P,Q_{k+1})$ after having already computed $e_r(P,Q_1),\ldots,e_r(P,Q_k)$.

Evaluating a fixed set $Q_1,Q_2,\ldots,Q_k$.

Optimizing k pairings $e_r(P,Q_i)$ for an already known set $Q_1,\ldots,Q_k$ is easy: only f depends on Q_i, hence we can easily adapt Algorithm 13 for an array of points $[Q_1,\ldots,Q_k]$ to keep track of a value $f^{(i)}$ per Q_i, and we return an array $\zeta^{(1)},\ldots,\zeta^{(k)}$. All evaluations share (per bit) the computations of T, $l_{T,T}$ and $l_{T,P}$. Our additional cost per extra point Q_i thus comes down to the evaluations $l_{T,T}(Q_i)$, $l_{T,P}(Q_i)$ and updating $f^{(i)}$. In total, this is $7\mathbf{M}$ per Dbl, and $5\mathbf{M}$ per Add/Sub, plus $2\mathbf{S}+\mathbf{I}$ to compute $\zeta^{(i)}$ given $f^{(i)}$ (the final exponentiation) per point Q_i. See `bigmultimiller.rs` for the implementation of this algorithm.

Evaluating additional points $Q_1,Q_2,\ldots$

It is more difficult when we want to compute $e_r(P,Q')$ *after* the computation of $e_r(P,Q)$. I.e., in some applications we compute $e_r(P,Q)$ first, and, based on

this evaluation, compute another $e_r(P, Q')$. In practice, this seems to require another full pairing computation.

Scott [264] observed that one can achieve a time/memory trade-off, by dividing Miller'sAlgorithm into three distinct subalgorithms: one to compute f_{rP}, one to evaluate $f_{rP}(Q_i)$ per Q_i and one for the final exponentiation. Paradoxically, this brings us back to the original three-step process from Section 11.3.3, where we argued that the degree of f_{rP} is too large to store f_{rP} in full. However, Scott notes, $f_{rP}(Q)$ can be computed from the set of all line functions $l_{T,T}$ and $l_{T,P}$ and Q. Up to $\mathbb{F}_p$-invariance, all such line functions l can be written as

$$l(x, y) = \lambda_x \cdot x + \lambda_y \cdot y + \lambda_0, \quad \lambda_i \in \mathbb{F}_p,$$

and we get a line function per Dbl, Add, and Sub. Thus, at a memory cost of

$$\underbrace{(\log(p) + 1/3\log(p))}_{\#\text{ subroutines}} \cdot \underbrace{3 \cdot \log(p)}_{\text{bits per } l} = 4 \cdot \log(p)^2$$

(the factor $1/3$ can be decreased using windowing) we can store a representation of f_{rP} as an array of line functions. Hence, we can split up Algorithm 13 into three subroutines Construct, `xEVAL`, and Exponentiate, which coincide precisely with the decomposition of the Tate pairing given in Equation (11.2). We refer to the composition of these subalgorithms as Scott-Miller's algorithm. See Section 11.B for an algorithmic description.

11.4.5 Summary of costs

We summarize Section 11.4 in terms of $\mathbb{F}_p$-operations for pairings of degree $r = p + 1$.

General primes.

Miller'sAlgorithm has $\log p$ steps and each step performs 1 Dbl. Using the techniques from Section 11.4.3 decreases the number of Add/Subs[2]:

1. each Dbl costs $15\mathbf{M} + 5\mathbf{S} + 7\mathbf{A}$, we always perform $\log p$ Dbls
2. each Add or Sub costs $20\mathbf{M} + 4\mathbf{S} + 9\mathbf{A}$,
   (a) in a naïve approach, we perform $\frac{1}{2}\log p$ Adds and Subs

[2]These techniques are inspired by finite field exponentiation and scalar multiplication, and have been analyzed previously for pairing-based cryptography [185, 186].

(b) using NAFs, we perform $\frac{1}{3}\log p$ Adds and Subs

(c) using windowing, we perform $\frac{1}{w+1}\log p$ Adds and Subs

For CSIDH-512.

For p512, Table 11.1 gives the number of $\mathbb{F}_p$-operations to compute a pairing, and shows the effectiveness of the optimizations: a reduction of 40% compared to unoptimized pairings.

	M	**S**	**A**	Total
Original Miller'sAlgorithm	28 498	2621	39 207	30 987
Optimized step (Sec 11.4.2)	12 740	3569	12 230	15 717
Using NAFs (Alg. 13)	11 152	3254	11 125	13 866
Using windows ($w = 5$)	9963	2960	10 592	12 436
Additional pairing	4410	2	5704	4468

Table 11.1: Concrete cost of the Miller loop for p512 for a pairing of degree $r = p+1$. 'Total' gives the number of $\mathbb{F}_p$-operations, with cost model $1\mathbf{S} = 0.8\mathbf{M}$ and $1\mathbf{A} = 0.01\mathbf{M}$.

For an additional pairing, if the points are known beforehand, we require only slightly more memory cost for each additional pairing. If we need to compute a multipairing for variable points, this takes $4 \cdot \log(p)^2$ bits of memory to store the representation of f_{rP}, using Scott-Miller's algorithm (Section 11.4.4).

Remark 11.2. In Table 11.1, we consider the cost for a pairing of degree $r = p + 1$. An alternative for primes $p = h \cdot \ell_\chi - 1$ with large cofactor h is to consider pairings of degree $r = \ell_\chi$ on $E[\ell_\chi]$. In many applications, such as [68], this contains all the necessary torsion information needed. The loop, then, has $\log \ell_\chi = \log p - \log h$ steps. The cost of such a pairing can be deduced from the given estimates.

11.5 Applications of pairings to isogeny problems

In this section, we apply the optimized pairing from Section 11.4 to the isogeny problems described in Section 11.3.1. The core design idea is clear: the pairing is now cheap enough to move isogeny problems from curves to finite fields,

where we have highly efficient Lucas exponentiation. Lemma 11.1 captures what information about the original points remains after the pairing.

11.5.1 Verification of full-torsion points.

We start with the following problem: Given $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$, verify both points have full-torsion, for a supersingular curve E over $\mathbb{F}_p$ [3].

Current methods.

Current methods to verify full-torsion points compute $[\frac{p+1}{\ell_i}]P \neq \mathcal{O}$ to conclude p has ℓ_i-torsion for every $\ell_i \mid p+1$ and hence is a full-torsion point. In a naïve way, this can be done per ℓ_i for the cost of a scalar multiplication of size $\log p$, using either Montgomery ladders or differential addition chains, at a total complexity of $\mathcal{O}(n \log p)$. Concretely, this comes down to a cost of $C_{\text{curve}} \cdot n \log p$ where C_{curve} is the cost of curve arithmetic per bit.

Using product trees this drops down to $\mathcal{O}(\log n \log p)$, although taking $\mathcal{O}(\log n)$ space. Product-tree-based order verification is currently the method used in state-of-the-art deterministic and dummy-free implementations [68] to verify a given basis (P, Q). Hence, doing this for both P and Q comes down to approximately $2 \cdot C_{\text{curve}} \cdot \log n \log p$ operations in $\mathbb{F}_p$.

Torsion bases.

We can improve on the previous verification matter by two easy observations. Firstly, whenever a pair of points $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$ generates all of $E[p+1] \subset E(\mathbb{F}_{p^2})$, the pair (P, Q) is a torsion basis for the $(p+1)$-torsion. As proposed in SIDH/SIKE [106, 107], we can verify such a torsion basis using the result that $\zeta = e_{p+1}(P, Q) \in \mu_{p+1}$ must be a $(p+1)$-th primitive root in $\mathbb{F}_{p^2}$. Secondly, this situation is ideal for our pairing: P has both rational coordinates, and Q has rational x and purely imaginary y-coordinate. As noted before, ζ is an element of norm 1 in $\mathbb{F}_{p^2}$, which allows us to apply fast Lucas exponentiation to compute ζ^k. The following lemma is our key building block.

Lemma 11.2. Let $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$. Let $\zeta = e_{p+1}(P, Q) \in \mu_{p+1}$. Then

$$\zeta^{\frac{p+1}{\ell_i}} \neq 1 \Leftrightarrow [\frac{p+1}{\ell_i}]P \neq \mathcal{O} \text{ and } [\frac{p+1}{\ell_i}]Q \neq \mathcal{O}.$$

[3] We treat *finding* full-torsion points in Section 11.5.3.

Proof. This is a direct application of Lemma 11.1. □ □

Hence, instead of verifying that both P and Q have ℓ_i-torsion, we verify that the powers $\zeta^{\frac{p+1}{\ell_i}}$ do not vanish. Furthermore, as ζ and its powers have norm 1, we simply verify that $\mathrm{Re}(\zeta^k) \neq 1$ which implies $\zeta^k \neq 1$. In terms of Lucas sequences, this is equal to $V_k(2a) \neq 2$ for $\zeta = a + bi$. Per bit, Lucas exponentiation is much more efficient than curve arithmetic, which allows us to compute every $\zeta^{\frac{p+1}{\ell_i}}$ (again using product trees) very efficiently at a similar complexity $\mathcal{O}(\log n \log p)$, and a concrete cost of $1 \cdot C_{\mathrm{Lucas}} \cdot \log n \log p$ [4].

Algorithm 14 summarizes this approach, where Order is a product-tree based function that computes the order of ζ (equivalently, verifies for which ℓ_i we have $\zeta^{\frac{p+1}{\ell_i}} \neq 1$) using Lucas exponentiation. Order is the field analogue of algorithms such as OrderRec [25] and validate_rec in CTIDH [22]. See `bigpairingfo.rs` for an implementation of Algorithm 14.

Algorithm 14 Verification of torsion basis

Input: $P \in E(\mathbb{F}_p)$, $Q \in E^t(\mathbb{F}_p)$
Output: **True** or **False**
1: $\zeta \leftarrow \mathrm{Re}(e_{p+1}(P, Q))$
2: $m \leftarrow \mathsf{Order}(\zeta)$
3: **if** $m = p + 1$ **then**
4: **return True**
5: **else**
6: **return False**

Further improvements.

As stated before, working in μ_{p+1} has the added benefit that we work in same subgroup of $\mathbb{F}_{p^2}$, independent of the curve E_A. This can be used to speed up the cost of torsion basis verification even more, at the cost of $\log p$ additional bits next to the pair (P, Q), as follows.

The scalars $\Lambda := \mathbb{Z}_{p+1}^*$ of invertible elements in $\mathbb{Z}_{p+1}$ act faithfully and free on both the group of full-torsion points of E_A, as well as on exact primitive roots in $\mu_{p+1} \subset \mathbb{F}_{p^2}$. This means that we can write any full-torsion point T relative to another full-torsion point T' as $T = [\lambda]T'$ with $\lambda \in \Lambda$. Similarly for

[4] To compute using Lucas exponentiation we use a constant-time laddering approach. Interesting future work would be to use (differential) addition chains to reduce costs.

primitive roots, this implies we can pick a system parameter ζ_0 as the "standard" primitive root, and can find $\lambda \in \Lambda$ such that for $\zeta = e_{p+1}(P, Q)$ we get $\zeta = \zeta_0^\lambda$.

By including λ next to (P, Q), we do not have to verify the complete order of ζ. Instead, we simply verify that $\lambda \in \Lambda$, compute $\zeta = e_{p+1}(P, Q)$ and verify that $\zeta = \zeta_0^\lambda$ which implies that ζ is an exact $(p+1)$-th root of unity. Compared to the algorithm sketched before this means that instead of $\mathcal{O}(\log n \log p)$ to verify the order of ζ, we use a single Lucas exponentiation $\mathcal{O}(\log p)$ to verify ζ.

Remark 11.3. The addition of the discrete log λ such that $\zeta = e_{p+1}(P, Q) = \zeta_0^\lambda$ might be unnecessary depending on the specific application. Namely, for a pair (P, Q), we get another pair $(P, \lambda^{-1}Q)$ that is a torsion basis, with

$$e_{p+1}(P, \lambda^{-1}Q) = e_{p+1}(P, Q)^{\lambda^{-1}} = \zeta^{\lambda^{-1}} = \zeta_0.$$

As ζ_0 is a public parameter, verification requires no extra λ. However, the choice of P and Q might have been performed carefully, e.g. to make sure both have small x-coordinates to reduce communication cost. It thus depends per application if a modified torsion basis $(P, \lambda^{-1}Q)$ reduces communication cost.

Remark 11.4. The above algorithm not only verifies that P and Q are full-torsion points, but includes the supersingularity verification of E_A, as it shows E_A has points of order $p + 1$. This can be useful for applications, e.g. those in [68].

11.5.2 Pairing-based supersingularity verification

Supersingularity verification asks us to verify that E_A is a supersingular curve. A sound analysis of the performance of different algorithms was made by Banegas, Gilchrist, and Smith [25]. They examine

a. a product-tree-based approach to find a point of order $N \geq 4\sqrt{p}$,

b. Sutherland's algorithm [276] based on isogeny volcanoes, and

c. Doliskani's test [131] based on division polynomials.

They conclude that Doliskani's test is best for Montgomery models over $\mathbb{F}_p$, as it requires only a single scalar multiplication over $\mathbb{F}_{p^2}$ of length $\log p$ followed by $\log p$ squarings in $\mathbb{F}_{p^2}$. The algorithms we propose resemble the product-tree approach, but move the computation of the orders of points from the curve to the field. There are two ways to apply the pairing approach here:

Aproach 1. Sample, using Elligator [45], random points $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$, compute $\zeta = e_r(P, Q)$ for $r = p + 1$, and compute $\mathsf{Ord}(\zeta)$ using a product-tree up until we have verified $\mathsf{Ord}(\zeta) \geq 4\sqrt{p}$.
Aproach 2. Divide L_χ into two lists L_1 and L_2 such that $\ell^{(1)} := \prod_{\ell_i \in L_1} \ell_i$ is slightly larger than $4\sqrt{p}$. Sample again two random points $P \in E(\mathbb{F}_p)$ and $Q \in E^t(\mathbb{F}_p)$, and multiply both by $\frac{p+1}{\ell^{(1)}} = h \cdot \ell^{(2)}$ so that $P, Q \in E[\ell^{(1)}]$. Then, compute the pairing of degree $r = \ell^{(1)}$ and verify that $\zeta \in \mu_r$ has $\mathsf{Ord}(\zeta) \geq 4\sqrt{p}$.

Approach 1 is essentially Algorithm 14, where we cut off the computation of $\mathsf{Ord}(\zeta)$ early whenever we have enough torsion. Approach 2 uses the fact that we do not have to work in all of μ_{p+1} to verify supersingularity. This reduces the number of steps in the Miller loop by half, $\frac{1}{2}\log p$ compared to $\log p$, but requires two Montgomery ladders of $\log(p)/2$ bits to kill the $\ell^{(2)}$-torsion of P and Q. Note that we must take L_1 a few bits larger than $4\sqrt{p}$ to ensure with high probability that random points P, Q have enough torsion to verify or falsify supersingularity. In practice, the fastest approach is highly dependent on the prime p and the number of factors ℓ_i, as well as the size of the cofactor 2^k. Approach 2 is summarized in Algorithm 15. See the folder `supersingularity` for the implementations of these algorithms.

Algorithm 15 Verification of supersingularity

Input: $A \in \mathbb{F}_p$, where $p = h \cdot \ell^{(1)} \cdot \ell^{(2)} - 1$
Output: **True** or **False**

1: $(P, Q) \xleftarrow{\$} E(\mathbb{F}_p) \times E^t(\mathbb{F}_p)$
2: $P \leftarrow [4 \cdot \ell^{(2)}]P$, $Q \leftarrow [h \cdot \ell^{(2)}]Q$
3: $\zeta \leftarrow \mathrm{Re}(e_{\ell^{(1)}}(P, Q))$
4: $m \leftarrow \mathsf{Order}(\zeta)$
5: **if** $m \geq 4\sqrt{p}$ **then**
6: **return True**
7: **else**
8: **else** repeat

Remark 11.5. Line 4 of Algorithm 15 computes the order of ζ using a product-tree approach to verify **(a)** $\zeta' := \zeta^{\frac{p+1}{\ell_i}} \neq 1$ and **(b)** $\zeta'^{\ell_i} = 1$. If at any moment, **(a)** holds but **(b)** does not, this implies the order of ζ does not divide $p + 1$, which implies the curve is *ordinary*. In such a case, as in the original algorithm [84, Alg. 1], we return **False** as we know the curve is not supersingular.

Remark 11.6. Note that in an approach where torsion points (P, Q) are given, together with a discrete log λ, so that $e_{p+1}(P, Q) = \zeta_0^\lambda$, or even the variant where $(P, \lambda^{-1}Q)$ is given as in Remark 11.3, the cost of supersingularity verification is essentially that of a single pairing computation, or that of a pairing computation together with a Lucas exponentiation of length $\log \lambda \approx \log p$. For CSIDH-512, this beats Doliskani's test as we will see in Section 11.5.4

11.5.3 Finding full-torsion points.

Finding full-torsion points is more tricky. Current implementations [**SQALE**, 68] simply sample random points and compute their order until they find a full-torsion point. Although the probability of finding a full-torsion point is not low, this approach is inefficient as the cost of computing the order per point is similar to the one sketched in Section 11.5.1. The curve-equivalent of Gauss'sAlgorithm can be given as follows: Given a point $P \in E(\mathbb{F}_p)$ with $\mathsf{Ord}(P) \neq p+1$, sample a random point $P' \in E(\mathbb{F}_p)$. Set $P' \leftarrow [\mathsf{Ord}(P)]P'$ and compute $\mathsf{Ord}(P')$. Set $P \leftarrow P + P'$ and $\mathsf{Ord}(P) \leftarrow \mathsf{Ord}(P) \cdot \mathsf{Ord}(P')$. Repeat until P is a full-torsion point. This already improves on the naïve approach,

yet still requires a lot of curve arithmetic. We apply pairings again to improve performance. We specifically need Scott-Miller's algorithm (Section 11.4.4) to compute a variable number of pairings for a given P.

Abstract point-of-view.

From the abstract point of view, sketched in Section 11.4.1, we want to identify a full-torsion point P as an isomorphism $f_{rP} : E^t(\mathbb{F}_p) \to \mu_r$, using Lemma 11.1. Scott-Miller's algorithm allows us to compute a representation of f_{rP} and to evaluate f_{rP} efficiently on points $Q \in E^t(\mathbb{F}_p)$.

Starting from random points $P_1 \in E(\mathbb{F}_p)$ and $Q_1 \in E^t(\mathbb{F}_p)$, we compute $\zeta_1 := f_{rP_1}(P_1, Q_1)^{p-1}$ and $\mathsf{Ord}(\zeta_1)$. The missing torsion $m_1 = \mathsf{Miss}(\zeta_1)$ is then equal to $\mathrm{lcm}(\mathsf{Miss}(P), \mathsf{Miss}(Q)))$. If $m_1 = 1$, then we know both P_1 and Q_1 are full-torsion points. If $m_1 \neq 1$, we continue with a second point Q_2. Compute ζ_2 and $m_2 = \mathsf{Miss}(\zeta_2)$. Let $d = \gcd(m_1, m_2)$. If $d = 1$, that is, m_1 and m_2 are co-prime, then P_1 is a full-torsion point, and we can apply Gauss'sAlgorithm to compute a full-order point Q, given Q_1 and Q_2.

For $d > 1$, it is most likely that $d = |\ker f_{rP_1}| = \mathsf{Miss}(P_1)$, or, if unlucky, both Q_1 and Q_2 miss d-torsion. The probability that both Q_1 and Q_2 miss d-torsion is $\frac{1}{d^2}$. Hence, if d is small, this is unlikely but possible. If d is a large prime, we are almost certain P_1 misses d-torsion. In the former case, we sample

a third point Q_3 and repeat the same procedure. In the latter case, we use Q_1 and Q_2 to compute a full-torsion Q. Using Q, we compute f_{rQ} and apply the same procedure to points P_i to create a full-torsion point P (reusing P_1).

Distinguishing between these cases is highly dependent on the value of d, which in turn depends on L_χ and p. We leave these case-dependent details to the reader. See `bigfastfinding.rs` for the implementation of this algorithm.

Remark 11.7. To distinguish between the cases dependent on d, it is favorable to have no small factors ℓ_i in $p+1$. This coincides with independent analysis [**SQALE**, 22] that small $\ell_i \mid p+1$ might not be optimal for deterministic variants of CSIDH.

Remark 11.8. One can improve on randomly sampling P_1 and Q_1, as we can sample points directly in $E \setminus [2]E$ [106] by ensuring the x-coordinates are not quadratic residues in $\mathbb{F}_p$. Similar techniques from p-descent might also apply for $\ell_i > 2$.

Remark 11.9. The above approach is very efficient to find a second full-torsion point $Q \in E^t(\mathbb{F}_p)$ given a first full-torsion point $P \in E(\mathbb{F}_p)$, which may be of independent interest in other situations.

11.5.4 Concrete cost for CSIDH-512

We have implemented and evaluated the performance of the algorithms in Section 11.5 for `p512`, the prime used in CSIDH-512. Table 11.2 shows the performance in $\mathbb{F}_p$-operations, compared to well-known or state-of-the-art algorithms.

	Source	**M**	**S**	**A**	Total
Product-tree torsion verif.	[68]	51 318	29 388	73 396	75 562
Pairing-based torsion verif.	Alg. 14	13 693	6838	18 424	19 293
Pairing-based (given λ)	Sec. 11.5.1	10 472	3471	11 616	13 364
CSIDH-Supersingularity verif.	[84]	13 324	7628	19 052	19 617
Doliskani's test	[25, 131]	13 789	2	30 642	14 097
Approach 1 (pairing-based)	Sec. 11.5.2	11 081	4112	12 914	14 500
Approach 2 (pairing-based)	Alg. 15	9589	5801	10 434	14 334

Table 11.2: Concrete cost of the algorithms in this section using the prime in CSIDH-512. 'Total' gives the number of $\mathbb{F}_p$-operations, with cost model $1\mathbf{S} = 0.8\mathbf{M}$ and $1\mathbf{A} = 0.01\mathbf{M}$.

Verifying torsion points.

We find that Algorithm 14 specifically for p512 takes about 19293 operations, with 12426 operations taken up by the pairing, hence order verification of ζ using Lucas exponentiation requires only 6867 operations, closely matching the predicted cost $C_{\text{Lucas}} \cdot \log n \cdot \log p$. In comparison, the currently-used method [68] to verify full-torsion points requires 75562 operations, hence we achieve a speed-up of 75%, due to the difference in cost per bit between C_{curve} and C_{Lucas}. If we include a system parameter ζ_0 and a discrete log λ, our cost drops down to 13364 operations, increasing the speedup from 75% to 82%.

Supersingularity verification.

We find that Doliskani's test is still slightly faster, but our algorithms come within 2% of performance. Saving a single M or S in Dbl or Add would push Algorithm 15 below Doliskani's test for p512. When we include λ, as mentioned above, we outperform Doliskani's test by 6%.

Finding torsion points.

Although the cost of this algorithm depends highly on divisors ℓ_i of $p + 1$, heuristics for p512 show that we usually only require 2 points $P_1, P_2 \in E(\mathbb{F}_p)$ and two points $Q_1, Q_2 \in E^t(\mathbb{F}_p)$, together with pairing computations $e_r(P_1, Q_1)$ and $e_R(P_1, Q_2)$ to find full-torsion points P and Q. We leave out concrete performance numbers, as this varies too much per case.

11.6 Applications of pairing-based algorithms

The pairing-based algorithms from Section 11.5 are of independent interest, but also find natural applications in (deterministic) variants of CSIDH.

Applying pairing-based algorithms.

In all versions of CSIDH, supersingularity verification is required on public keys. We estimate that, depending on the shape and size of the prime p, either Doliskani's test or one of the pairing-based algorithms (Section 11.5.2) is optimal.

For deterministic variants of CSIDH [**SQALE**, 68], including (an Elligator seed for) a torsion basis of E_A in the public key is only natural, and this is exactly

what is proposed for the dCSIDH variant of [68]. This requires verification of such a torsion basis. For this verification, Algorithm 14 clearly outperforms curve-based approaches. Furthermore, the verification of such a torsion basis also verifies the supersingularity of E_A, which would otherwise have cost an additional $\mathcal{O}(\log p)$ operations, using either Doliskani's test or one of our pairing-based algorithms.

Including torsion-point information in the public key also requires a party to *find* such a torsion basis in key generation. The pairing-based approach described in Section 11.5.3 heuristically beats current approaches based on random sampling.

Remark 11.10. One might think that including a torsion basis (P, Q) resulting from the pairing-based approach in a public key would cost $2 \log p$ bits, as it requires a description of the x-coordinates of both points. However, it is possible to use points P_i and Q_i with very small x-coordinates, and to describe the construction of P (resp. Q) as a combination of the P_i (resp. Q_i) in a few bits.

Constant-time versions.

Both Algorithms 14 and 15 are easy to implement in constant-time, given constant-time curve and field arithmetic. However, constant-time verification is usually not required, as both E_A and (P, Q) should be public.

For a constant-time approach to finding full-torsion points, a major roadblock is finding a constant-time version of Gauss′sAlgorithm. This is both mathematically interesting as well as cryptographically useful, but seems to require a better understanding of the distribution of the x-coordinates of full-torsion points for curves, or the $(p+1)$-th primitive roots for fields.

Beyond current implementations.

Current deterministic variants of CSIDH [**cervantes2019stronger**, **SQALE**, 68] are limited to exponents $e_i \in \{-1, 0, +1\}$. Going beyond such exponents requires sampling new points during the class group action evaluation on an intermediate curve E'. To not leak any information on E' in a deterministic implementation, therefore, requires a constant-time torsion-basis algorithm as sketched above. This would allow approaches with $e_i \geq 1$ for multiple small ℓ_i to reduce the number of ℓ_i-isogenies for large ℓ_i, which is deemed favorable in constant-time probabilistic approaches [**meyer2019lions**, **AMC:ChiRod20**, 22].

For ordinary CSIDH and CTIDH, using full-torsion points in every round would have the further improvement that the number of rounds is constant, and we have no trial-and-error approaches in the group action computation, providing a stronger defence against certain side-channel attacks [28]. In particular for CTIDH, we can use full-torsion points to remove the "coin flip" that decides if a batch is performed. This improves performance and design properties. We leave a full analysis for future work.

A final remark should be made about the use of theta functions to compute pairings [**robert1**, 199]. Such an approach might yield speed-ups in comparison to the methods used in this work and the use of theta functions could provide a more general framework for higher-dimensional isogeny-based cryptography.

11.A Subalgorithms of Miller's algorithm

The following algorithms give an algorithmic description of the subroutines Dbl, Add and Sub as used in the optimizations of the Miller loop.

Algorithm 16 Subroutine Dbl in the Miller loop

Input: $T \in \mathbb{F}_p^4, f \in \mathbb{F}_p^2, x_Q, y_Q \in \mathbb{F}_p, A \in \mathbb{F}_p$
Output: (T, f) corresponding to the doubling $T \leftarrow T + T$, $f \leftarrow f^2 \cdot \ell_{T,T}(Q)$
1: $(T, \ell) \leftarrow \mathsf{DblAndLine}(T, A)$ ▷ Doubles T and computes $\ell_{T,T}$
2: $(\alpha, \beta) \leftarrow \mathsf{Eval}(\ell, x_Q, y_Q)$ ▷ Evaluates $\ell_{T,T}(Q)$
3: $f \leftarrow \mathsf{FP2SQR}(f)$
4: $f \leftarrow \mathsf{FP2MUL}(f, (\alpha, \beta))$
5: **return** T, f

Algorithm 17 Subroutine Add in the Miller loop

Input: $T \in \mathbb{F}_p^4, f \in \mathbb{F}_p^2, x_P, y_P, x_Q, y_Q \in \mathbb{F}_p, A \in \mathbb{F}_p$
Output: (T, f) corresponding to the addition $T \leftarrow T + P$, $f \leftarrow f \cdot \ell_{T,P}(Q)$
1: $(T, \ell) \leftarrow \mathsf{AddAndLine}(T, x_P, y_P, A)$ ▷ Adds $T + P$ and computes $\ell_{T,P}$
2: $(\alpha, \beta) \leftarrow \mathsf{Eval}(\ell, x_Q, y_Q)$ ▷ Evaluates $\ell_{T,P}(Q)$
3: $f \leftarrow \mathsf{FP2MUL}(f, (\alpha, \beta))$
4: **return** T, f

Algorithm 18 Subroutine Sub in the Miller loop

Input: $T \in \mathbb{F}_p^4, f \in \mathbb{F}_p^2, x_P, y_P, x_Q, y_Q \in \mathbb{F}_p, A \in \mathbb{F}_p$
Output: (T, f) corresponding to the substraction $T \leftarrow T - P$, $f \leftarrow f \cdot \ell_{T,-P}(Q)$
1: $(T, f) \leftarrow \mathsf{Add}(T, f, x_P, y_P, x_Q, y_Q, A)$
2: **return** T, f

11.B Subalgorithms of Scott-Miller's algorithm

We describe here Scott-Miller's subalgorithms Construct, `xEVAL`and Exponentiate.

Algorithm 19 Scott-Miller's subalgorithm Construct

Input: $x_P, y_p \in \mathbb{F}_p$, $p+1 = \sum_{i=0}^{t} t_i \cdot 2^i$
Output: A representation of f_{rP} as an array of line functions

$T = (X^2, XZ, Z^2, YZ) \leftarrow (x_P^2, x_P, 1, y_P)$
$f \leftarrow [\,]$
for i from $t-1$ to 0 **do**
    $T, l \leftarrow \mathsf{Dbl}(T)$
    Append l to f
    if $t_i = 1$ **then**
        $T, l \leftarrow \mathsf{Add}(T, f, x_P, y_P)$
        Append l to f
    if $t_i = -1$ **then**
        $T, l \leftarrow \mathsf{Sub}(T, f, x_P, y_P)$
        Append l to f
return f

Algorithm 20 Scott-Miller's subalgorithm Evaluate

Input: A representation of f_{rP} as an array of line functions, and a point $x_Q, y_Q \in \mathbb{F}_p$
Output: The unreduced Tate evaluation $f_{rP}(Q) \in \mathbb{F}_{p^2}$

$f_0 \leftarrow (1, 0)$
for l in f **do**
    if l is a doubling **then** $f_0 \leftarrow f_0^2$
    $f_0 \leftarrow f_0 \cdot \mathsf{Eval}(l, x_Q, y_Q)$
return f_0

Algorithm 21 Scott-Miller's subalgorithm Exponentiate

Input: The unreduced Tate pairing $f_0 = a + bi$ as a pair $f = (a, b)$
Output: The reduced Tate pairing $\zeta \in \mu_r$

1: $a \leftarrow f[0]$, $b \leftarrow f[1]$
2: $\zeta \leftarrow \frac{a^2-b^2}{a^2+b^2}$
3: **return** ζ

The composition of these three algorithms is referred to as Scott-Miller's algorithm.

Considerations

Some considerations on future research on CSIDH, including ideas we worked on but did not publish.

Part III

Isogenies (SQIsign)

Intro

Some introduction on SQIsign

Chapter 12

AprèsSQI

12.1 Placeholder

The paper will go here.

12.2 Introduction

Research has shown that large-scale quantum computers will break current public-key cryptography, such as RSA or ECC, whose security relies on the hardness of integer factorization or the discrete logarithm, respectively [269]. Post-quantum cryptography seeks to thwart the threat of quantum computers by developing cryptographic primitives based on alternative mathematical problems that cannot be solved efficiently by quantum computers. In recent years, lattice-based cryptography has developed successful post-quantum schemes for essential primitives such as key encapsulation mechanisms (KEMs) and digital signatures that will be standardized by NIST. Lattice-based signatures are able to provide fast signing and verification, but have to resort to larger key and signature sizes than were previously acceptable in pre-quantum signatures. For applications where the amount of data transmitted is crucial, these lattice-based schemes may not be a practical option. NIST is therefore looking for other digital signature schemes with properties such as smaller combined public key and signature sizes to ensure a smooth transition to a post-quantum world [273].

A potential solution to this problem is provided by the sole isogeny-based candidate in NIST's new call for signatures – SQIsign [121] – as it is currently

the candidate that comes closest to the data sizes transmitted (i.e. the combined size of the signature and the public key) in pre-quantum elliptic curve signatures [179, 180]. SQIsign is most interesting in scenarios that require small signature sizes and fast verification, particularly in those applications where the performance of signing is not the main concern. A few common examples include long-term signatures, specifically public-key certificates, code updates for small devices, authenticated communication with embedded devices or other microcontrollers that solely run verification, and smart cards. For such use cases it is imperative to bring down the cost of verification as much as possible.

Performance bottlenecks in SQIsign.

The bottleneck of verification in SQIsign is the computation of an isogeny of fixed degree 2^e with $e \approx (15/4)\log(p)$, where p denotes the prime one is working over, e.g. $\log(p) \approx 256$ for NIST Level I security. However, the rational 2-power torsion, from here on denoted as the $2^\bullet$-torsion, is limited, since we work with supersingular elliptic curves over $\mathbb{F}_{p^2}$ of order $(p+1)^2$ and $(p-1)^2$. This sets a theoretical limit of $2^{\log p}$ for the $2^\bullet$-torsion. Therefore, the verifier needs to perform several *blocks* of degree $2^\bullet$ to complete the full 2^e-isogeny, where each of these blocks involves costly steps such as computing a $2^\bullet$-torsion basis or isogeny kernel generator. Hence, in general, a smaller number of blocks improves the performance of verification.

On the other hand, the bottleneck in signing is the computation of several T-isogenies for odd smooth $T \approx p^{5/4}$. Current implementations of SQIsign therefore require $T \mid (p-1)(p+1)$, such that $\mathbb{F}_{p^2}$-rational points are available for efficient T-isogeny computations. The performance of this step is dominated by the smoothness of T, i.e., its largest prime factor.

While this additional divisibility requirement theoretically limits the maximal $2^\bullet$-torsion to roughly $p^{3/4}$, current techniques for finding SQIsign-friendly primes suggest that achieving this with acceptable smoothness of T is infeasible [67, 89, 101, 109, 121]. In particular, the NIST submission of SQIsign uses a prime with rational 2^{75}-torsion and 1973 as largest factor of T. Since $e \approx (15/4) \cdot 256 = 960$, this means that the verifier has to perform $\lceil e/75 \rceil = 13$ costly isogeny blocks. Increasing the $2^\bullet$-torsion further is difficult as it decreases the probability of finding a smooth and large enough T for current implementations of SQIsign.

Our contributions.

In this work, we deploy a range of techniques to increase the $2^\bullet$-torsion and push the SQIsign verification cost far below the state of the art. Alongside these technical contributions, we aim to give an accessible description of SQIsign, focusing primarily on verification, which solely uses elliptic curves and isogenies and does not require knowledge of quaternion algebras.

Even though we target faster verification, our main contribution is signing with field extensions. From this, we get a much weaker requirement on the prime p, which in turn enables us to increase the size of the $2^\bullet$-torsion.

Focusing on NIST Level I security, we study the range of possible $2^\bullet$-torsion to its theoretical maximum, and measure how its size correlates to verification time through an implementation that uses an equivalent to the number of field multiplications as cost metric. Compared to the state of the art, increasing the $2^\bullet$-torsion alone makes verification almost 1.7 times faster. Further, we implement the new signing procedure as proof-of-concept in SageMath and show that signing times when signing with field extensions are in the same order of magnitude as when signing only using operations in $\mathbb{F}_{p^2}$.

For verification, in addition to implementing some known general techniques for improvements compared to the reference implementation provided in the NIST submission of SQIsign, we show that increasing the $2^\bullet$-torsion also opens up a range of optimisations that were previously not possible. For instance, large $2^\bullet$-torsion allows for an improved challenge-isogeny computation and improved basis and kernel generation. Furthermore, we show that size-speed trade-offs as first proposed by De Feo, Kohel, Leroux, Petit, and Wesolowski [121] become especially worthwhile for large $2^\bullet$-torsion. When pushing the $2^\bullet$-torsion to its theoretical maximum, this even allows for uncompressed signatures, leading to significant speed-ups at the cost of roughly doubling the signature sizes.

For two specific primes with varying values of $2^\bullet$-torsion, we combine all these speed-ups, and measure the performance of verification. Compared to the implementation of the SQIsign NIST submission [89], we reach a speed-up up to a factor 2.70 at NIST Level I when keeping the signature size of 177 bytes. When using our size-speed trade-offs, we reach a speed-up by a factor 3.11 for signatures of 187 bytes, or a factor 4.46 for uncompressed signatures of 322 bytes. Compared to the state of the art [198], these speed-ups are factors 2.07, 2.38 and 3.41 respectively.

Related work.

De Feo, Kohel, Leroux, Petit, and Wesolowski [121] published the first SQIsign implementation, superseded by the work of De Feo, Leroux, Longa, and Wesolowski [122]. Subsequently, Lin, Wang, Xu, and Zhao [198] introduced several improvements for this implementation. The NIST submission of SQIsign [89] features a new implementation that does not rely on any external libraries. Since this is the latest and best documented implementation, we will use it as a baseline for performance comparison, and refer to it as SQIsign (NIST). Since the implementation by Lin, Wang, Xu, and Zhao [198] is not publicly available, we included their main improvement for verification in SQIsign (NIST), and refer to this as SQIsign (LWXZ).

Dartois, Leroux, Robert, and Wesolowski [116] recently introduced SQIsignHD, which massively improves the signing time in SQIsign, in addition to a number of other benefits, but at the cost of a still unknown slowdown in verification. This could make SQIsignHD an interesting candidate for applications that prioritise the combined cost of signing and verification over the sole cost of verification.

Recent work by Eriksen, Panny, Sotáková, and Veroni [135] explored the feasibility of computing the Deuring correspondence (see Section 12.3.2) for *general* primes p via using higher extension fields. We apply the same techniques and tailor them to *specialised* primes for use in the signing procedure of SQIsign.

Overview.

The rest of the paper is organised as follows. Section 12.3 recalls the necessary background, including a high-level overview of SQIsign. Section 12.4 describes how using field extensions in signing affects the cost and relaxes requirements on the prime. Section 12.5 analyses how the size of the $2^\bullet$-torsion correlates to verification time. Section 12.6 presents optimisations enabled by the increased $2^\bullet$-torsion, while Section 12.7 gives further optimisations enabled by increased signature sizes. Finally, Section 12.8 gives some example parameters, and measures their performance compared to the state of the art.

Availability of software.

We make our Python and SageMath software publically available under the MIT licence at

`https://github.com/TheSICQ/ApresSQI`.

12.3 Preliminaries

Throughout this paper, p denotes a prime number and $\mathbb{F}_{p^k}$ the finite field with p^k elements, where $k \in \mathbb{Z}_{>0}$.

12.3.1 Elliptic curves and their endomorphism rings.

We first give the necessary geometric background to understand the SQIsign signature scheme. For a more general exposition we refer to Silverman [272].

Isogenies.

An isogeny $\varphi : E_1 \to E_2$ between two elliptic curves E_1, E_2 is a non-constant morphism that sends the identity of E_1 to the identity of E_2. The degree $d = \deg(\varphi)$ of an isogeny is its degree as a rational map. If the degree d of an isogeny φ has the prime factorisation $d = \prod_{i=1}^{n} \ell_i^{e_i}$, we can decompose φ into the composition of e_i isogenies of degree ℓ_i for $i = 1$ to n. For every isogeny $\varphi : E_1 \to E_2$, there is a (unique) *dual* isogeny $\widehat{\varphi} : E_2 \to E_1$ that satisfies $\widehat{\varphi} \circ \varphi = [\deg(\varphi)]$, the multiplication-by-$\deg(\varphi)$ map on E_1. Similarly, $\varphi \circ \widehat{\varphi}$ is the multiplication by $\deg(\varphi)$ on E_2.

A *separable* isogeny is described, up to isomorphism, by its kernel, a group of order d. Given a kernel G of prime order d, we can compute the corresponding isogeny $\phi : E \to E/G$ using Vélu's formulas [287] in $\widetilde{O}(d)$. Bernstein, De Feo, Leroux, and Smith [42] showed that this can be asymptotically reduced to $\widetilde{O}(\sqrt{d})$ using $\sqrt{\text{élu}}$ formulas. In Section 12.3.5, we return to the topic of computing isogenies and give a more detailed discussion.

Endomorphism rings.

An isogeny from a curve E to itself is called an *endomorphism*. For example, for any integer n, the multiplication-by-n map is an endomorphism. Another, not necessarily distinct, example for elliptic curves defined over $\mathbb{F}_q$ is the *Frobenius endomorphism* $\pi : (x, y) \mapsto (x^q, y^q)$.

The set of endomorphisms $\mathrm{End}(E)$ of an elliptic curve E forms a ring under (pointwise) addition and composition of isogenies. The endomorphism ring of $E/\overline{\mathbb{F}}_p$ is either isomorphic to an imaginary quadratic order, or to a maximal order in a quaternion algebra ramified at p and ∞ (which will be defined in Section 12.3.2). In the latter case, we say that E is *supersingular*, and from this point forward, E will denote a supersingular curve.

Supersingular elliptic curves and their isomorphism classes.

We will mostly consider supersingular elliptic curves *up to isomorphism*, and thus work with isomorphism classes of these curves. Throughout, we will exploit the fact that every isomorphism class of supersingular curves has a model over $\mathbb{F}_{p^2}$, such that the p^2-power Frobenius π is equal to the multiplication-by-$(-p)$ map. Such curves E are $\mathbb{F}_{p^2}$-isogenous to curves defined over $\mathbb{F}_p$, and satisfy

$$E(\mathbb{F}_{p^{2k}}) = E\left[p^k - (-1)^k\right] \cong \mathbb{Z}/\left(p^k - (-1)^k\right)\mathbb{Z} \oplus \mathbb{Z}/\left(p^k - (-1)^k\right)\mathbb{Z}, \quad (12.1)$$

while their quadratic twist over $\mathbb{F}_{p^{2k}}$, which we will denote E_k^t, satisfies

$$E_k^t(\mathbb{F}_{p^{2k}}) = E\left[p^k + (-1)^k\right] \cong \mathbb{Z}/\left(p^k + (-1)^k\right)\mathbb{Z} \oplus \mathbb{Z}/\left(p^k + (-1)^k\right)\mathbb{Z}. \quad (12.2)$$

For such curves, for any positive integer $N \mid p^k \pm 1$, the full N-torsion group $E[N]$ is defined over $\mathbb{F}_{p^{2k}}$, either on the curve itself, or on its twist.

The isogeny problem.

The fundamental hard problem underlying the security of all isogeny-based primitives is the following: given two elliptic curves E_1, E_2 defined over $\mathbb{F}_{p^2}$ find an isogeny $\phi : E_1 \to E_2$. The best classical attack against this problem is due to Delfs and Galbraith [126] which runs in time $\widetilde{O}(\sqrt{p})$, and quantum attack due to Biasse, Jao, and Sankar [55] that runs in $\widetilde{O}(\sqrt[4]{p})$. A related problem, which will be useful in the context of SQIsign, is the *endomorphism ring problem*, which asks, given a supersingular elliptic curve $E/\mathbb{F}_{p^2}$, to find the endomorphism ring $\mathrm{End}(E)$. Wesolowski [291] showed that this is equivalent to the isogeny problem under reductions of polynomial expected time, assuming the generalised Riemann hypothesis, and further, Page and Wesolowski [238] recently showed that the endomorphism ring problem is equivalent to the problem of computing one endomorphism.

12.3.2 Quaternion algebras and the Deuring correspondence

We give the necessary arithmetic background to understand the signing procedure of SQIsign at a high level.[1] The heart of the signing procedure in SQIsign

[1] This section is only necessary for Section 12.3.3 and Section 12.4, as all other sections are concerned only with SQIsign verification, which will only use well-known isogeny terminology. In contrast, signing heavily relies on the arithmetic of quaternion algebras.

lies in the Deuring correspondence, which connects the geometric world of supersingular curves from Section 12.3.1 to the arithmetic world of quaternion algebras. For more details on quaternion algebras, we refer to Voight [289].

Quaternion algebras, orders and ideals.

Quaternion algebras are four-dimensional $\mathbb{Q}$-algebras, generated by four elements $1, i, j, k$ following certain multiplication rules. For SQIsign, we work in the quaternion algebra *ramified* at p and ∞. When $p \equiv 3 \pmod 4$, one representation of such a quaternion algebra is given by $\mathcal{B}_{p,\infty} = \mathbb{Q} + i\mathbb{Q} + j\mathbb{Q} + k\mathbb{Q}$ with multiplication rules

$$i^2 = -1,\ j^2 = -p,\ ij = -ji = k.$$

For an element $\alpha = x + yi + zj + wk \in \mathcal{B}_{p,\infty}$ with $x, y, z, w \in \mathbb{Q}$, we define its *conjugate* to be $\bar{\alpha} = x - yi - zj - wk$, and its *reduced norm* to be $n(\alpha) = \alpha\bar{\alpha}$.

We are mainly interested in *lattices* in $\mathcal{B}_{p,\infty}$, defined as full-rank $\mathbb{Z}$-modules contained in $\mathcal{B}_{p,\infty}$, i.e., abelian groups of the form

$$\alpha_1\mathbb{Z} + \alpha_2\mathbb{Z} + \alpha_3\mathbb{Z} + \alpha_4\mathbb{Z},$$

where $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \in \mathcal{B}_{p,\infty}$ are linearly independent. If a lattice $\mathcal{O} \subset \mathcal{B}_{p,\infty}$ is also a subring of $\mathcal{B}_{p,\infty}$, i.e., it contains 1 and is closed under multiplication, then $\mathcal{O}$ is called an *order*. Orders that are not strictly contained in any other order are called *maximal* orders. From this point on, we only consider maximal orders.

A lattice I that is closed under multiplication by an order $\mathcal{O}$ on the left is called a *left* (resp. *right*) $\mathcal{O}$-*ideal*. We refer to $\mathcal{O}$ as the left (resp. right) order of I. When $\mathcal{O}$ is the left order of I and $\mathcal{O}'$ the right order of I, we define I to be a *connecting* $(\mathcal{O}, \mathcal{O}')$-*ideal*.[2] A left $\mathcal{O}$-ideal I that is also contained in $\mathcal{O}$ is called an *integral* ideal. From this point on, we only deal with integral left ideals and simply refer to them as ideals.

The *norm* of an ideal I is the greatest common divisor of the reduced norms of the elements of I, whereas the *conjugate* $\bar{I}$ of an ideal I is the ideal consisting of the conjugates of the elements of I. Two ideals I and J are said to be equivalent if $I = J\alpha$ for some $\alpha \in \mathcal{B}_{p,\infty}^{\times}$ and is denoted $I \sim J$. Equivalent ideals have equal left orders and isomorphic right orders.

[2] Note that $\mathcal{O}$ and $\mathcal{O}'$ need not be distinct.

The Deuring correspondence.

Given an elliptic curve E with $\mathrm{End}(E) \cong \mathcal{O}$, there is a one-to-one correspondence between separable isogenies from E and left $\mathcal{O}$-ideals I of norm coprime to p. Given an isogeny φ, we denote the corresponding ideal I_φ, and conversely, given an ideal I, we denote the corresponding isogeny φ_I. The Deuring correspondence acts like a dictionary: a given isogeny $\varphi : E \to E'$ corresponds to an ideal I_φ with left order $\mathcal{O} \cong \mathrm{End}(E)$ and right order $\mathcal{O}' \cong \mathrm{End}(E')$. Furthermore, the degree of φ is equal to the norm of I_φ and the dual isogeny $\widehat{\varphi}$ corresponds to the conjugate $\overline{I_\varphi} = I_{\widehat{\varphi}}$. Equivalent ideals I, J have isomorphic right orders and the corresponding isogenies φ_I, φ_J have isomorphic codomains.

The (generalised) KLPT algorithm.

The KLPT algorithm, introduced by Kohel, Lauter, Petit, and Tignol [190], is a purely quaternionic algorithm, but has seen a variety of applications in isogeny-based cryptography due to the Deuring correspondence. Given an ideal I, KLPT finds an equivalent ideal J of prescribed norm. The drawback is that the norm of the output J will be comparatively large.

For example, the KLPT algorithm is used to compute isogenies between two curves of known endomorphism ring. Given two maximal orders $\mathcal{O}, \mathcal{O}'$, translating the standard choice[3] of connecting ideal I to its corresponding isogeny is hard. However, by processing I through KLPT first, we can find an equivalent ideal J of smooth norm, allowing us to compute φ_J. This is essential for computing the response in SQIsign.

The original KLPT algorithm only works for $\mathcal{O}_0$-ideals, where $\mathcal{O}_0$ is a maximal order of a special form.[4] This was generalised by De Feo, Kohel, Leroux, Petit, and Wesolowski [121] to work for arbitrary orders $\mathcal{O}$, albeit at the cost of an even larger norm bound for the output. Note that SQIsign utilizes both versions.

12.3.3 SQIsign

Next, we give a high-level description of signing and verification in SQIsign. SQIsign is a signature scheme based on an underlying Sigma protocol that proves knowledge of a *secret* (non-scalar) endomorphism $\alpha \in \mathrm{End}(E_A)$ for some *public*

[3] $I = N\mathcal{O}\mathcal{O}'$, where N is the smallest integer making I integral.

[4] Specifically, it only works for special, p-extremal orders. An example of such an order when $p \equiv 3 \pmod 4$ is $\mathrm{End}(E_0)$ where $j(E_0) = 1728$.

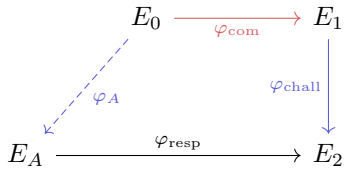


Figure 12.1: The SQIsign protocol with three phases: commitment φ_{com}, challenge φ_{chall}, and response φ_{resp}.

curve E_A. At its core, the prover shows this knowledge by being able to compute an isogeny φ from E_A to some random curve E_2.

A high-level description of the SQIsign Sigma protocol is given below (see also Figure 12.1).

Setup: Fix a prime number p and supersingular elliptic curve $E_0/\mathbb{F}_{p^2}$ with known endomorphism ring.

Key generation: Compute a secret key $\varphi_A : E_0 \to E_A$, giving the prover knowledge of $\text{End}(E_A)$, with corresponding public verification key E_A.

Commit: The prover generates a random commitment isogeny $\varphi_{\text{com}} : E_0 \to E_1$, and sends E_1 to the verifier.

Challenge: The verifier computes a random challenge isogeny $\varphi_{\text{chall}} : E_1 \to E_2$, and sends φ_{chall} to the prover.

Response: The prover uses the knowledge of φ_{com} and φ_{chall} to compute $\text{End}(E_2)$, allowing the prover to compute the response isogeny $\varphi_{\text{resp}} : E_A \to E_2$, by translating an ideal computed using the generalised KLPT algorithm, as described at the end of Section 12.3.2.

The verifier needs to check that φ_{resp} is an isogeny from E_A to E_2.[5] Assuming the hardness of the endomorphism ring problem, the protocol is sound: if the prover is able to respond to two different challenges φ_{chall}, φ'_{chall} with φ_{resp} and φ'_{resp}, the prover knows an endomorphism of the public key E_A, namely $\widehat{\varphi'_{\text{resp}}} \circ \varphi'_{\text{chall}} \circ \widehat{\varphi_{\text{chall}}} \circ \varphi_{\text{resp}}$. Proving zero-knowledge is harder and relies on

[5] Additionally, $\widehat{\varphi_{\text{chall}}} \circ \varphi_{\text{resp}}$ needs to be cyclic. Observe that otherwise, the soundness proof might return a scalar endomorphism.

the output distribution of the generalised KLPT algorithm. Note that KLPT is needed for computing the response:[6] while setting $\varphi_{\text{resp}} = \varphi_{\text{chall}} \circ \varphi_{\text{com}} \circ \widehat{\varphi_A}$ gives an isogeny from E_A to E_2, this leaks the secret φ_A.[7] For a further discussion on zero-knowledge, we refer to the original SQIsign articles [121, 122].

Remark 12.1. The best-known attacks against SQIsign are the generic attacks against the endomorphism ring problem. As discussed in Section 12.3.1, their run time depends only on the size of p and, with high probability, do not recover the original secret isogeny, but rather a different isogeny between the same curves. Therefore, their complexity should be unaffected by the changes we introduce to the SQIsign protocol in Section 12.4, as for these attacks it does not matter whether the original secret isogeny had kernel points defined over a larger extension field. In short, the changes in this work do not affect the security of SQIsign.

Verification.

Using the Fiat–Shamir heuristic, the SQIsign Sigma protocol is transformed into a signature scheme. This means that a signature on the message msg is of the form $\sigma = (\varphi_{\text{resp}}, \mathsf{msg}, E_1)$. For efficiency, φ_{resp} is compressed, and E_1 is replaced by a description of φ_{chall}. Thus, given the signature σ and public key E_A, the verifier recomputes the response isogeny $\varphi_{\text{resp}} : E_A \to E_2$ and the (dual of the) challenge isogeny $\widehat{\varphi_{\text{chall}}} : E_2 \to E_1$, and then verifies that the hash $H(\mathsf{msg}, E_1)$ indeed generates φ_{chall}.

The isogeny φ_{resp} is of degree 2^e with $e = \lceil \frac{15}{4} \log(p) \rceil + 25$ [89, §7.2.3], where 2^e corresponds to the output size of the generalised KLPT algorithm. The bottleneck in verification is the (re)computation of φ_{resp} in $\lceil e/f \rceil$ steps of size 2^f. Accelerating this will be the focus of this paper.

12.3.4 SQIsign-friendly primes

Next, we give more details on the parameter requirements in SQIsign. We refer to the original SQIsign works [89, 121, 122] for a detailed description of their origins.

[6] Alternatively, one can replace the connecting ideal with the shortest equivalent ideal, and translate it by embedding it in an isogeny between higher-dimensional abelian varieties, as shown in SQIsignHD [116]

[7] Further, this is not a valid response, since the composition with $\widehat{\varphi_{\text{chall}}}$ is not cyclic.

SQIsign prime requirements.

The main bottleneck of SQIsign is the computation of isogenies. Recall from Equations (12.1) and (12.2) that, when working with supersingular elliptic curves $E/\mathbb{F}_{p^2}$, we have $E(\mathbb{F}_{p^2}) = E[p \pm 1]$. Thus, to use x-only arithmetic over $\mathbb{F}_{p^2}$, SQIsign restricts to computing isogenies of *smooth* degree $N \mid (p^2 - 1)$. Finding SQIsign-friendly primes reduces to finding primes p, with $p^2 - 1$ divisible by a large, smooth number. More explicitly, for a security level λ, the following parameters are needed:

- A prime p of bitsize $\log_2(p) \approx 2\lambda$ with $p \equiv 3 \mod 4$.
- The torsion group $E[2^f]$ as large as possible, that is $2^f \mid p + 1$.
- A smooth odd factor $T \mid (p^2 - 1)$ of size roughly $p^{5/4}$.
- The degree of φ_{com}, $D_{\text{com}} \mid T$, of size roughly $2^{2\lambda} \approx p$.
- The degree of φ_{chall}, $D_{\text{chall}} \mid 2^f T$, of size roughly $2^\lambda \approx p^{1/2}$.
- Coprimality between D_{com} and D_{chall}.

To achieve NIST Level I, III, and V security, we set the security parameter as $\lambda = 128, 192, 256$, respectively. Concretely, this means that, for each of these security parameters, we have $\log p \approx 256, 384, 512$, and $\log T \approx 320, 480, 640$, with f as large as possible given the above restrictions. The smoothness of T directly impacts the signing time, and the problem of finding primes p with a large enough T that is reasonably smooth is difficult. We refer to recent work on this problem for techniques to find suitable primes [67, 89, 101, 109, 121, 122].

The crucial observation for this work is that T occupies space in $p^2 - 1$ that limits the size of f, hence current SQIsign primes balance the smoothness of T with the size of f.

Remark 12.2. SQIsign (NIST) further requires $3^g \mid p + 1$ such that $D_{\text{chall}} = 2^f \cdot 3^g \geq p^{1/2}$ and $D_{\text{chall}} \mid p + 1$. While this is not a strict requirement in the theoretical sense, it facilitates efficiency of computing φ_{chall}. From this point on, we ensure that this requirement is always fulfilled.

Remark 12.3. Since the curves E and their twists E^t satisfy

$$E(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(p \pm 1)\mathbb{Z} \oplus \mathbb{Z}/(p \pm 1)\mathbb{Z},$$

and we work with both simultaneously, choosing T and f is often incorrectly described as choosing divisors of $p^2 - 1$. There is a subtle issue here: even if 2^f divides $p^2 - 1$, $E[2^f]$ may not exist as a subgroup of $\langle E(\mathbb{F}_{p^2}), \rho^{-1}(E^t(\mathbb{F}_{p^2}))\rangle \subseteq E(\mathbb{F}_{p^4})$, where $\rho : E \to E^t$ is the twisting isomorphism. While this does not usually matter in the case of SQIsign (we pick 2^f as a divisor of $p + 1$, and T is odd), this becomes a problem when working over multiple extension fields. In Section 12.4.2, we make this precise and reconcile it using Theorem 12.2.

12.3.5 Computing rational isogenies from irrational generators

Finally, to facilitate signing with field extensions, we recall the techniques for computing $\mathbb{F}_{p^2}$-rational isogenies, i.e., isogenies defined over $\mathbb{F}_{p^2}$, generated by *irrational* kernel points, that is, not defined over $\mathbb{F}_{p^2}$. In the context of this paper, we again stress that such isogenies will only be computed by the signer; the verifier will only work with points in $\mathbb{F}_{p^2}$.

The main computational task of most isogeny-based cryptosystems (including SQIsign) lies in evaluating isogenies given the generators of their kernels. Explicitly, given an elliptic curve $E/\mathbb{F}_q$, a point $K \in E(\mathbb{F}_{q^k})$ such that $\langle K \rangle$ is *defined over* $\mathbb{F}_q$,[8] and a list of points $(P_1, P_2, \ldots, P_n)$ in E, we wish to compute the list of points $(\varphi(P_1), \varphi(P_2), \ldots, \varphi(P_n))$, where φ is the separable isogeny with $\ker \varphi = \langle K \rangle$. Since we work with curves E whose p^2-Frobenius π is equal to the multiplication-by-$(-p)$ map (see Section 12.3.1), *every* subgroup of E is closed under the action of $\mathrm{Gal}(\bar{\mathbb{F}}_{p^2}/\mathbb{F}_{p^2})$, hence every isogeny from E can be made $\mathbb{F}_{p^2}$-rational, by composing with the appropriate isomorphism.

Computing isogenies of smooth degree.

Recall from Section 12.3.1 that the isogeny factors as a composition of small prime degree isogenies, which we compute using Vélu-style algorithms. For simplicity, for the rest of the section, we therefore assume that $\langle K \rangle$ is a subgroup of order $\ell > 2$, where ℓ is a small prime.

At the heart of these Vélu-style isogeny formulas is evaluating the kernel polynomial. Pick any subset $S \subseteq \langle K \rangle$ such that $\langle K \rangle = S \sqcup -S \sqcup \{\infty\}$. Then the kernel polynomial can be written as

$$f_S(X) = \prod_{P \in S} (X - x(P)). \tag{12.3}$$

[8] That is, the group $\langle K \rangle$ is closed under the action of $\mathrm{Gal}(\bar{\mathbb{F}}_q/\mathbb{F}_q)$.

Here, the generator K can be either a rational point, i.e., lying in $E(\mathbb{F}_q)$, or an irrational point, i.e., lying in $E(\mathbb{F}_{q^k})$ for $k > 1$, but whose group $\langle K \rangle$ is defined over $\mathbb{F}_q$. Next, we discuss how to solve the problem efficiently in the latter case.

Irrational generators.

For $K \notin E(\mathbb{F}_q)$ of order ℓ, we can speed up the computation of the kernel polynomial using the action of Frobenius. This was used in two recent works [24, 135], though the general idea was used even earlier [282].

As $\langle K \rangle$ is defined over $\mathbb{F}_q$, we know that the q-power Frobenius π acts as an endomorphism on $\langle K \rangle \subseteq E(\mathbb{F}_{p^k})$ and thus maps K to a multiple $[\gamma]K$ for some $\gamma \in \mathbb{Z}$. This fully determines the action on $\langle K \rangle$, i.e., $\pi|_{\langle K \rangle}$ acts as $P \mapsto [\gamma]P$ for all $P \in \langle K \rangle$. For the set S as chosen above, this action descends to an action on its x-coordinates $X_S = \{x(P) \in \mathbb{F}_{q^k} \mid P \in S\}$ and thus partitions X_S into orbits $\{x(P), x([\gamma]P), x([\gamma^2]P), \ldots\}$ of size equal to the order of γ in $(\mathbb{Z}/\ell\mathbb{Z})^\times/\{1, -1\}$.

If we pick one representative $P \in S$ per orbit, and call this set of points S_0, we can compute the kernel polynomial (12.3) as a product of the minimal polynomials $\mu_{x(P)}$ of the $x(P) \in \mathbb{F}_{q^k}$ for these $P \in S_0$, with each $\mu_{x(P)}$ defined over $\mathbb{F}_q$, as

$$f_S(X) = \prod_{P \in S_0} \mu_{x(P)}(X), \tag{12.4}$$

where μ_β denotes the minimal polynomial of β over $\mathbb{F}_q$.

To compute $f_S(\alpha)$ for $\alpha \in \mathbb{F}_q$, we only require the smaller polynomial $f_{S_0}(X)$ and compute $\mathrm{Norm}_{\mathbb{F}_{q^k}/\mathbb{F}_q}(f_{S_0}(\alpha))$, as

$$\mathrm{Norm}_{\mathbb{F}_{q^k}/\mathbb{F}_q}(f_{S_0}(\alpha)) = \prod_{\pi \in G} \pi(f_{S_0}(\alpha)) = \prod_{P \in S_0} \prod_{\pi \in G} (\alpha - \pi(x(P))) = \prod_{P \in S_0} \mu_{x(P)}(\alpha),$$

where $G = \mathrm{Gal}(\mathbb{F}_{q^k}/\mathbb{F}_q)$, as per Banegas, Gilchrist, Le Dévéhat, and Smith [24]. This allows us to compute the image under f_S of x-values of points in $E(\mathbb{F}_q)$, but only works for values in $\mathbb{F}_q$. To evaluate $f_S(\alpha)$ for general $\alpha \in \overline{\mathbb{F}}_p$, i.e. to compute the image of a point in $E(\overline{\mathbb{F}}_p)$, we instead use the larger polynomial $f_S(X)$, which we compute, as in Equation (12.4), as a product of the minimal polynomials $\mu_{x(P)}$, where we use Shoup's algorithm [270] to compute each $\mu_{x(P)}$ given $x(P)$. Computing $f_S(X)$ requires a total of $O(\ell k) + \widetilde{O}(\ell)$ operations, with k such that each $x(P) \in \mathbb{F}_{q^k}$. Evaluation f_S at α takes $\widetilde{O}(\ell k')$ operations, with k' the smallest value such that $\alpha \in \mathbb{F}_{q^{k'}}$ [135, Section 4.3].

Remark 12.4. The biggest drawback to using this technique is that $\sqrt{\text{élu}}$ is no longer effective, as we would need to work in the smallest field where both the isogeny generator and the x-value of the point we are evaluating are defined in.

12.4 Signing with extension fields

By allowing torsion T from extension fields, we enable more flexibility in choosing SQIsign primes p, thus enabling a larger $2^\bullet$-torsion. Such torsion T requires us to compute rational isogenies with kernel points in extension fields $\mathbb{F}_{p^{2k}}$. This section describes how to adapt SQIsign's signing procedure to enable such isogenies, and the increased cost this incurs. In particular, we describe two approaches for T: allowing torsion T from a particular extension field $\mathbb{F}_{p^{2k}}$, or from all extension fields $\mathbb{F}_{p^{2n}}$ for $1 \leq n \leq k$. The first approach means that we can look for T dividing an integer of bit size $\Theta(k \log p)$, and the second approach allows for $\Theta(k^2 \log p)$. In Section 12.5, we explore how increased $2^\bullet$-torsion affects verification.

Throughout this section, we will reuse the notation from Section 12.3.4 to describe the various parameters related to SQIsign.

12.4.1 Changes in the signing procedure

Recall from Section 12.3.3 that the signing operation in SQIsign requires us to work with both elliptic curves and quaternion algebras, and to translate back and forth between these worlds. Note that the subroutines that work solely with objects in the quaternion algebra $\mathcal{B}_{p,\infty}$, including all operations in KLPT and its derivatives, are indifferent to what extension fields the relevant torsion groups lie in. Hence, a large part of signing is unaffected by torsion from extension fields.

In fact, the only subroutines that are affected by moving to extension fields are those relying on Algorithm 22, $\mathsf{IdealToIsogeny}_D$, which translates $\mathcal{O}_0$-ideals I of norm dividing D to their corresponding isogenies φ_I. $\mathsf{IdealToIsogeny}_D$ is not used during verification, and is used only in the following parts of signing:

Commitment: The signer translates a random ideal of norm D_{com} to its corresponding isogeny, using one execution of $\mathsf{IdealToIsogeny}_{D_{\text{com}}}$.

Response: The signer translates an ideal of norm 2^e to its corresponding

Algorithm 22 $\mathsf{IdealToIsogeny}_D(I)$

Input: I a left $\mathcal{O}_0$-ideal of norm dividing D
Output: φ_I

1: Compute α such that $I = \mathcal{O}_0\langle \alpha, \mathrm{nrd}(I)\rangle$
2: Let $\mathbf{A} = [1, i, \frac{i+j}{2}, \frac{1+k}{2}]$ denote a basis of $\mathcal{O}_0$
3: Compute $\mathbf{v}_{\bar{\alpha}} := [x_1, x_2, x_3, x_4]^T \in \mathbb{Z}^4$ such that $\mathbf{A}\mathbf{v}_{\bar{\alpha}} = \bar{\alpha}$
4: **for** $\ell^e \mid\mid D$ **do**
5: $\quad \bar{\alpha}|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle} := x_1\mathbf{I} + x_2(i|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle}) + x_3(\frac{i+j}{2}|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle}) + x_4(\frac{1+k}{2}|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle})$
6: $\quad$ Let a, b, c, d be integers such that $\bar{\alpha}|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$
7: $\quad K_{\ell^e} := [a]P_{\ell^e} + [c]Q_{\ell^e}$
8: $\quad$ **if** $\mathrm{ord}(K_{\ell^e}) < \ell^e$ **then**
9: $\quad\quad K_{\ell^e} = [b]P_{\ell^e} + [d]Q_{\ell^e}$
10: Set φ_I to be the isogeny generated by the points K_{ℓ^e}. **return** φ_I

isogeny, requiring $2 \cdot \lceil e/f \rceil$ executions of $\mathsf{IdealToIsogeny}_T$.[9]

Remark 12.5. We will choose parameters such that $2^f \mid p+1$ and $D_{\mathrm{chall}} \mid p+1$, so that $E[2^f]$ and $E[D_{\mathrm{chall}}]$ are defined over $\mathbb{F}_{p^2}$. As a result, the verifier only works in $\mathbb{F}_{p^2}$ and the added complexity of extension fields applies only to the signer.

Adapting ideal-to-isogeny translations to field extensions.

To facilitate signing with field extensions, we slightly adapt $\mathsf{IdealToIsogeny}_D$ so that it works with prime powers separately. Note that the additional cost of this is negligible compared to the cost of computing the isogeny from the generators because finding the action of the relevant endomorphisms consists of simple linear algebra. See Algorithm 22 for details.

In Line 5 of Algorithm 22, the notation $\beta|_{\langle P_{\ell^e}, Q_{\ell^e}\rangle}$ refers to the action of an endomorphism β on a fixed basis P_{ℓ^e}, Q_{ℓ^e} of $E[\ell^e]$. This action is described by a matrix in $M_2(\mathbb{Z}/\ell^e\mathbb{Z})$. These matrices can be precomputed, hence the only operations in which the field of definition of $E[\ell^e]$ matters are the point additions in Lines 7 and 9, and isogenies generated by each K_{ℓ^e} in Line 10.

[9] The technical details are given by De Feo, Leroux, Longa, and Wesolowski [122].

12.4.2 Increased torsion availability from extension fields

Next, we detail the two approaches to allow torsion groups from extension fields, which permits more flexibility in choosing the final prime p.

Working with a single field extension of $\mathbb{F}_{p^2}$.

Although the choice of solely working in $\mathbb{F}_{p^2}$ occurs naturally,[10] there is no reason *a priori* that this choice is optimal. Instead, we can choose to work in the field $\mathbb{F}_{p^{2k}}$. We emphasise that this does *not* affect signature sizes; the only drawback is that we now perform more expensive $\mathbb{F}_{p^{2k}}$-operations during signing in IdealToIsogeny. The upside, however, is a relaxed prime requirement: we are no longer bound to $E[T] \subseteq \langle E(\mathbb{F}_{p^2}), \rho^{-1}(E^t(\mathbb{F}_{p^2}))\rangle$ and can instead use

$$E[T] \subseteq \langle E(\mathbb{F}_{p^{2k}}), \rho^{-1}(E^t(\mathbb{F}_{p^{2k}}))\rangle.$$

By Equations (12.1) and (12.2), we have $E(\mathbb{F}_{p^{2k}}) \cong E[p^k \pm 1]$ and $E^t(\mathbb{F}_{p^{2k}}) \cong E[p^k \mp 1]$, thus we simply get

$$E[T] \subseteq E\left[\frac{p^{2k}-1}{2}\right],$$

since $\langle E[A], E[B]\rangle = E[\mathrm{lcm}(A, B)]$. Hence, by using torsion from $E(\mathbb{F}_{p^{2k}})$, we increase $T \mid (p^2-1)/2$ to $T \mid (p^{2k}-1)/2$. This implies there are $2(k-1)\log p$ more bits available to find T with adequate smoothness.

Working with multiple field extensions of $\mathbb{F}_{p^2}$.

Instead of fixing a single higher extension field $\mathbb{F}_{p^{2k}}$, we can also choose to work with multiple field extensions, in particular all fields $\mathbb{F}_{p^{2n}}$, where $1 \leq n \leq k$. The torsion group we can access by this relaxed requirement is described by the following definition.

Definition 12.1. Let E be a supersingular elliptic curve over $\mathbb{F}_{p^2}$ and let E_n^t denote an arbitrary quadratic twist of E over $\mathbb{F}_{p^{2n}}$ with respect to the twisting isomorphism $\rho_n : E \to E_n^t$. We define the *k-available torsion* of E to be the group generated by $E(\mathbb{F}_{p^{2n}})$ and $\rho_n^{-1}(E_n^t(\mathbb{F}_{p^{2n}}))$ for $1 \leq n \leq k$.

[10] It is the smallest field over which every isomorphism class of supersingular elliptic curves has a model.

Any point P in the k-available torsion can thus be written as a sum

$$P = \sum_{i=1}^{k} (P_i + \rho_n^{-1}(P_i^t))$$

of points $P_i \in E(\mathbb{F}_{p^{2n}})$ and $P_i^t \in E_n^t(\mathbb{F}_{p^{2n}})$. Since the twisting isomorphism keeps the x-coodinate fixed, the computation of this isomorphism can be ignored when using x-only arithmetic, and we simply obtain a sum of points whose x-coordinates lie in $\mathbb{F}_{p^{2n}}$ for $1 \leq n \leq k$. This justifies the name k-available torsion, as we do not have to go beyond $\mathbb{F}_{p^{2k}}$ to do arithmetic with P by working with the summands separately.

The structure of the k-available torsion is completely captured by the following result.

Theorem 12.2. Let $p > 2$ be a prime, and let $E/\mathbb{F}_{p^2}$ be a supersingular curve with $\mathrm{tr}(\pi) = \pm 2p$, where π is the Frobenius endomorphism. Then the k-available torsion is precisely the group $E[N]$ with

$$N = \prod_{n=1}^{k} \Phi_n(p^2)/2,$$

where Φ_n denotes the n-th cyclotomic polynomial.

Lemma 12.3. For any integer $m \geq 2$, we have the following identity

$$\mathrm{lcm}\left(\{m^n - 1\}_{n=1}^{k}\right) = \prod_{n=1}^{k} \Phi_n(m)$$

where Φ_n denotes the n-th cyclotomic polynomial.

Proof. We denote the left-hand side and right-hand side of the equation in the statement of the lemma by LHS and RHS, respectively. We show that any prime power dividing one side, also divides the other.

For any prime ℓ and $e > 0$, if ℓ^e divides the LHS, then, by definition, it divides $m^i - 1 = \prod_{d|i} \Phi_d(m)$ for some $1 \leq i \leq k$. Hence, it also divides the RHS. Conversely, if ℓ^e divides the RHS, then ℓ^e also divides the LHS. To show this, we need to know when $\Phi_i(m)$ and $\Phi_j(m)$ are coprime. We note that

$$\gcd(\Phi_i(m), \Phi_j(m)) \mid R$$

where R is the resultant of $\Phi_i(X)$ and $\Phi_j(X)$, and a classic result by Apostol [11, Theorem 4.], tells us that

$$\mathrm{Res}(\Phi_i(X), \Phi_j(X)) > 1 \Rightarrow i = jm$$

for $i > j$ and some integer m.

Using this, if ℓ^e divides the RHS, then it will also divide the product

$$\prod_{n=1}^{\lfloor k/d \rfloor} \Phi_{dn}(m),$$

for some integer d, and this product divides the LHS, as it divides $m^{d\lfloor k/d \rfloor} - 1$. □

□

We can now conclude the proof of Theorem 12.2.

Proof. From the structure of $E(\mathbb{F}_{p^{2n}})$ (see Remark 12.3), where E is as in the statement, the k-available torsion can be seen as the group generated by the full torsion groups

$$E[p^n \pm 1]$$

for $1 \leq n \leq k$. Using the fact that

$$\langle E[A], E[B] \rangle = E[\mathrm{lcm}(A, B)],$$

we see that the k-available torsion is $E[N]$ where

$$N = \mathrm{lcm}\left(\{p^n - 1\}_{n=1}^{k} \cup \{p^n + 1\}_{n=1}^{k}\right) = \mathrm{lcm}\left(\{p^{2n} - 1\}_{n=1}^{k}\right)/2,$$

where the last equality only holds for $p > 2$. Applying Lemma 12.3 with $m = p^2$, we obtain

$$N = \prod_{k=1}^{n} \Phi_k(p^2)/2,$$

□ □

We find that using all extension fields $\mathbb{F}_{p^{2n}}$, for $1 \leq n \leq k$, increases $T \mid p^2 - 1$ to $T \mid N$, with N as given by Theorem 12.2. Given that

$$\log(N) = \sum_{n=1}^{k} \log(\Phi_n(p^2)/2) \approx 2 \sum_{n=1}^{k} \phi(n) \log(p),$$

and the fact that $\sum_{n=1}^{k} \phi(n)$ is in the order of $\Theta(k^2)$, where ϕ denotes Euler's totient function, we find that $T \mid N$ gives roughly $k^2 \log(p)$ more bits to find T with adequate smoothness, compared to the $\log(p)$ bits in the classical case of working over $\mathbb{F}_{p^2}$, and $k \log(p)$ bits in the case of working over $\mathbb{F}_{p^{2k}}$. Due to this, we only consider working in multiple field extensions from this point on.

12.4.3 Cost of signing using extension fields

In SQIsign, operations over $\mathbb{F}_{p^2}$ make up the majority of the cost during signing [122, Section 5.1]. Hence, we can roughly estimate the cost of signing by ignoring purely quaternionic operations, in which case the bottleneck of the signing procedure becomes running $\mathsf{IdealToIsogeny}_T$ as many times as required by the IdealToIsogenyEichler algorithm [122, Algorithm 5] in the response phase. In other words, we estimate the total signing cost from the following parameters:

- f, such that $2^f \mid p+1$.
- T, the chosen torsion to work with.
- For each $\ell_i^{e_i} \mid T$, the smallest k_i such that $E[\ell_i^{e_i}]$ is defined over $\mathbb{F}_{p^{2k_i}}$.

Since Algorithm 22 works with prime powers separately, we can estimate the cost of a single execution by considering the cost per prime power.

Cost per prime power.

For each $\ell_i^{e_i} \mid T$, let k_i denote the smallest integer so that $E[\ell_i^{e_i}] \subseteq E(\mathbb{F}_{p^{2k_i}})$, and let $M(k_i)$ denote the cost of operations in $\mathbb{F}_{p^{2k_i}}$ in terms of $\mathbb{F}_{p^2}$-operations. Computing the generator $K_{\ell_i^{e_i}}$ consists of a few point additions in $E[\ell_i^{e_i}]$, hence is $O(M(k) \cdot e \log \ell)$, while the cost of computing the isogeny generated by $K_{\ell_i^{e_i}}$ comes from computing e isogenies of degree ℓ at a cost of $O(\ell k) + \tilde{O}(\ell)$, using the techniques from Section 12.3.5.

To compute the whole isogeny, we need to push the remaining generators $K_{\ell_j^{e_j}}$, through this isogeny. To minimize the total cost, we pick the greedy strategy of always computing the smaller ℓ first. This bounds the cost of evaluating K_{ℓ^e} in *other* isogenies by $O(M(k) \cdot \ell)$.

Total cost of signing.

Based on the analysis above, we let

$$\textsc{Cost}_p(\ell_i^{e_i}) = c_1 M(k_i) e \log \ell + c_2 e_i \ell_i k_i + c_3 e_i \ell_i \log(\ell_i) + c_4 M(k_i) \ell$$

where k_i, and $M(k)$ are as before, and c_i are constants corresponding to the differences in the asymptotic complexities. Since we can estimate the total cost of executing $\mathsf{IdealToIsogeny}_T$ by summing the cost of each maximal prime power divisor of T, and observing that signing consists of executing $\mathsf{IdealToIsogeny}_{D_{\mathrm{com}}}$ one time, and $\mathsf{IdealToIsogeny}_T$ a total of $2 \cdot \lceil e/f \rceil$ times, we get a rough cost estimate of signing as

$$\textsc{SigningCost}_p(T) = (2 \cdot \lceil e/f \rceil + 1) \cdot \sum_{\ell_i^{e_i} | T} \textsc{Cost}_p(\ell_i^{e_i}).$$

In Section 12.8, we use this function to pick p and T minimising this cost. While this cost metric is very rough, we show that our implementation roughly matches the times predicted by this function. Further, we show that this cost metric suggests that going to extension fields gives signing times within the same order of magnitude as staying over $\mathbb{F}_{p^2}$, even when considering the additional benefit of using $\sqrt{\text{élu}}$ to compute isogenies in the latter case.

12.5 Effect of increased $2^\bullet$-torsion on verification

In Section 12.4, we showed that signing with extension fields gives us more flexibility in choosing the prime p, and, in particular, allows us to find primes with rational 2^f-torsion for larger f. In this section, we analyse how such an increase in $2^\bullet$-torsion affects the cost of SQIsign verification, e.g., computing φ_{resp} and $\widehat{\varphi_{\mathrm{chall}}}$, in terms of $\mathbb{F}_p$-multiplications,[11] taking the SQIsign (NIST) implementation (with no further optimisations) as the baseline for comparison.

12.5.1 Detailed description of verification

Before giving an in-depth analysis of verification performance, we give a detailed description of how verification is executed. Recall that a SQIsign signature σ for

[11] As standard, we denote multiplications by **M**, squarings by **S**, and additions by **a**.

a message msg created by a signer with secret signing key $\varphi_A : E_0 \to E_A$ proves knowledge of an endomorphism on E_A by describing an isogeny $\varphi_{\text{resp}} : E_A \to E_2$ of degree 2^e(see Figure 12.1). A given message msg is hashed on E_1 to a point K_{chall} of order D_{chall}, hence represents an isogeny $\varphi_{\text{chall}} : E_1 \to E_2$. A signature is valid if the composition of φ_{resp} with $\widehat{\varphi_{\text{chall}}}$ is cyclic of degree $2^e \cdot D_{\text{chall}}$.

Thus, to verify a signature σ, the verifier must **(a)** recompute φ_{resp}, **(b)** compute the dual of φ_{chall}, to confirm that both are well-formed, and finally **(c)** recompute the hash of the message msg to confirm the validity of the signature.

In SQIsign, the size of the sample space for φ_{chall} impacts soundness, a key security property for signature schemes. In SQIsign (NIST), to obtain negligible soundness error (in the security parameter λ) the message is hashed to an isogeny of degree $D_{\text{chall}} = 2^f \cdot 3^g$ so that the size of cyclic isogenies of degree D_{chall} is larger than 2^λ. In contrast, when $f \geq \lambda$, we can simply set $D_{\text{chall}} = 2^\lambda$.

The signature σ consists of a compressed description of the isogenies φ_{resp} and $\widehat{\varphi_{\text{chall}}}$. For $f < \lambda$ and $D_{\text{chall}} = 2^f \cdot 3^g$ it is of the form

$$\sigma = (b, s^{(1)}, \ldots, s^{(n)}, r, b_2, s_2, b_3, s_3)$$

with $s^{(j)}, s_2 \in \mathbb{Z}/2^f\mathbb{Z}$, $s_3 \in \mathbb{Z}/3^g\mathbb{Z}$, $r \in \mathbb{Z}/2^f 3^g\mathbb{Z}$, and $b, b_2, b_3 \in \{0, 1\}$. If $f \geq \lambda$, we set $D_{\text{chall}} = 2^f$ and have $s_2 \in \mathbb{Z}/2^\lambda\mathbb{Z}$ and $r \in \mathbb{Z}/2^f\mathbb{Z}$, while b_3, s_3 are omitted. Algorithmically, the verification process mostly requires three subroutines.

FindBasis: Given a curve E, find a deterministic basis (P, Q) of $E[2^f]$.

FindKernel: Given a curve E with basis (P, Q) for $E[2^f]$ and $s \in \mathbb{Z}/2^f\mathbb{Z}$, compute the kernel generator $K = P + [s]Q$.

ComputeIsogeny: Given a curve E and a kernel generator K, compute the isogeny $\varphi : E \to E/\langle K \rangle$ and $\varphi(Q)$ for some $Q \in E$.

Below we detail each of the three verification steps **(a)**-**(c)**.

Step (a).

Computing φ_{resp} is split up into $n - 1$ *blocks* $\varphi^{(j)} : E^{(j)} \to E^{(j+1)}$ of size 2^f, and a last block of size 2^{f_0}, where $f_0 = e - (n-1) \cdot f$. For every $\varphi^{(j)}$, the kernel $\langle K^{(j)} \rangle$ is given by the generator $K^{(j)} = P^{(j)} + [s^{(j)}]Q^{(j)}$ for a deterministic basis $(P^{(j)}, Q^{(j)})$ of $E^{(j)}[2^f]$.

In the first block, after sampling $(P^{(1)}, Q^{(1)})$ via FindBasis, the bit b indicates whether $P^{(1)}$ and $Q^{(1)}$ have to be swapped before running FindKernel. For the following blocks, the verifier pushes $Q^{(j)}$ through the isogeny $\varphi^{(j)}$ to get

a point $Q^{(j+1)} \leftarrow \varphi^{(j)}(Q^{(j)})$ on $E^{(j+1)}$ of order 2^f above $(0,0)$.[12] Hence, for $j > 1$ FindBasis only needs to find a suitable point $P^{(j)}$ to complete the basis $(P^{(j)}, Q^{(j)})$. Furthermore, $K^{(j)}$ is never above $(0,0)$ for $j > 1$, which ensures cyclicity when composing $\varphi^{(j)}$ with $\varphi^{(i-1)}$. In all cases we use $s^{(j)}$ from σ to compute the kernel generator $K^{(j)}$ via FindKernel and $\varphi^{(j)}$ via ComputeIsogeny.

The last block of degree 2^{f_0} uses $Q^{(n)} \leftarrow [2^{f-f_0}]\varphi^{(n-1)}(Q^{(n-1)})$ and samples another point $P^{(n)}$ as basis of $E^{(n)}[2^{f_0}]$. In the following, we will often assume $f_0 = f$ for the sake of simplicity.[13] An algorithmic description of a single block of SQIsign (NIST) is given in Algorithm 23 in Section 12.B.

Step (b).

Computing $\widehat{\varphi_{\text{chall}}}$ requires a single isogeny of smooth degree $D_{\text{chall}} \approx 2^\lambda$. For the primes given in SQIsign (NIST), we have $E_2[D_{\text{chall}}] \subseteq E_2(\mathbb{F}_{p^2})$. Thus, we compute φ_{chall} by deterministically computing a basis (P, Q) for $E_2[D_{\text{chall}}]$ and finding the kernel $\langle K \rangle$ for $\widehat{\varphi_{\text{chall}}} : E_2 \to E_1$. For $f < \lambda$, we have $D_{\text{chall}} = 2^f \cdot 3^g$, and split this process into two parts.

Given the basis (P, Q) for $E_2[D_{\text{chall}}]$, we compute $(P_2, Q_2) = ([3^g]P, [3^g]Q)$ as basis of $E_2[2^f]$, and use $K_2 = P_2 + [s_2]Q_2$, where b_2 indicates whether P_2 and Q_2 have to be swapped prior to computing K_2. We compute $\varphi_2 : E_2 \to E_2'$ with kernel $\langle K_2 \rangle$, and $P_3 = [2^f]\varphi_2(P)$ and $Q_3 = [2^f]\varphi_3(Q)$ form a basis of $E_2'[3^g]$. Then b_3 indicates a potential swap of P_3 and Q_3, while $K_3 = P_3 + [s_3]Q_3$ is the kernel generator of the isogeny $\varphi_3 : E_2' \to E_1$. Thus, we have $\widehat{\varphi_{\text{chall}}} = \varphi_3 \circ \varphi_2$. If $f \geq \lambda$, we require only the first step.

We furthermore verify that the composition of φ_{resp} and $\widehat{\varphi_{\text{chall}}}$ is cyclic, by checking that the first 2-isogeny step of φ_2 does not revert the last 2-isogeny step of $\varphi^{(n)}$. This guarantees that $\widehat{\varphi_{\text{chall}}} \circ \varphi_{\text{resp}}$ is non-backtracking, hence cyclic.

Step (c).

On E_1, the verifier uses the point $Q' \leftarrow \widehat{\varphi_{\text{chall}}}(Q')$, where Q' is some (deterministically generated) point, linearly independent from the generator of $\widehat{\varphi_{\text{chall}}}$, and r (given in σ) to compute $[r]Q'$, and checks if $[r]Q'$ matches the hashed point $K_{\text{chall}} = H(\mathsf{msg}, E_1)$ with hash function H.

[12] A point P is said to be *above* a point R if $[k]P = R$ for some $k \in \mathbb{N}$.

[13] In contrast to earlier versions, SQIsign (NIST) fixes $f_0 = f$. However, our analysis benefits from allowing $f_0 < f$.

12.5.2 Impact of large f on verification

The techniques of Section 12.4 extend the possible range of f to any size below $\log(p)$. This gives two benefits to the cost of verification, especially when $f \geq \lambda$.

Number of blocks in φ_{resp}.

The larger f is, the fewer blocks of size 2^f are performed in **Step (a)**. Per block, the dominating part of the cost are FindBasis and FindKernel as we first need to complete the torsion basis $(P^{(j)}, Q^{(j)})$ for $E^{(j)}[2^f]$ (given $Q^{(j)}$ if $j > 1$), followed by computing $K^{(j)} = P^{(j)} + [s^{(j)}]Q^{(j)}$. By minimizing the number of blocks n, we minimize the amount of times we perform FindBasis and FindKernel, and the cost of each individual FindKernel only mildly increases, as $s^{(j)}$ increases in size. The overall cost of ComputeIsogeny, that is, performing the n isogenies of degree 2^f given their kernels $K^{(j)}$, only moderately increases with growing f.

We further note that larger f requires fewer T-isogeny computations for the signer, hence signing performance also benefits from smaller n.

Challenge isogeny.

When $f \geq \lambda$, we can simply set $D_{\text{chall}} = 2^\lambda$, which has two main benefits.

- The cost of FindBasis for this step is reduced as finding a basis for $E[2^\lambda]$ is much easier than a basis search for $E[2^f \cdot 3^g]$.
- The cost for ComputeIsogeny for φ_{chall} decreases as we only have to compute a chain of 2-isogenies instead of additional 3-isogenies.

12.5.3 Implementation and benchmark of cost in $\mathbb{F}_p$-multiplications

To measure the influence of the size of f on the performance, we implemented SQIsign verification for the NIST Level I security parameter set in Python, closely following SQIsign (NIST). As is standard in isogeny-based schemes, we use x-only arithmetic and represent points and curve coefficients projectively. The benchmark counts $\mathbb{F}_p$-operations and uses a cost metric that allows us to estimate the runtime of real-world implementations for 256-bit primes $p^{(f)}$, where $p^{(f)}$ denotes a prime such that 2^f divides $p^{(f)} + 1$. We benchmark primes $p^{(f)}$ for all values $50 \leq f \leq 250$. These results serve as a baseline to which we compare the optimisations that we introduce in Sections 12.6 and 12.7. We briefly outline how SQIsign (NIST) implements the three main subroutines FindBasis, FindKernel, and ComputeIsogeny.

FindBasis.

We search for points of order 2^f by sampling x-coordinates in a specified order,[14] and check if the corresponding point P lies on E (and not on its twist E^t). We then compute $P \leftarrow [\frac{p+1}{2^f}]P$ and verify that $[2^{f-1}]P \neq \infty$. Given two points $P, Q \in E$ of order 2^f, we verify linear independence by checking that $[2^{f-1}]P \neq [2^{f-1}]Q$, and discard and re-sample the second point otherwise.

FindKernel.

Given a basis (P, Q), FindKernel computes $K = P + [s]Q$ via the `3ptLadder` algorithm as used in SIKE [176]. In addition to the x-coordinates x_P and x_Q of P and Q, it requires the x-coordinate x_{P-Q} of $P - Q$. Hence, after running FindBasis, we further compute x_{P-Q} as described in SQIsign (NIST) [89].

ComputeIsogeny.

Given a kernel generator K of order 2^f, ComputeIsogeny follows the approach of SIKE [176], and computes the 2^f-isogeny $\varphi^{(j)}$ as a chain of 4-isogenies for efficiency reasons. If f is odd, we further compute a single 2-isogeny. Following SQIsign (NIST), ComputeIsogeny proceeds as follows:

1. Compute $R = [2^{f-2}]K$ and the corresponding 4-isogeny φ with kernel $\langle R \rangle$. Note that the point $(0, 0)$ might be contained in $\langle R \rangle$ for the first block in φ_{resp}, which requires a special 4-isogeny formula. Thus, we check if this is the case and call the suitable 4-isogeny function. We set $K \leftarrow \varphi(K)$.

2. If f is odd, we compute $R = [2^{f-3}]K$, the 2-isogeny φ with kernel $\langle R \rangle$, and $K \leftarrow \varphi(K)$.

3. Compute the remaining isogeny of degree $2^{f'}$ with even f' as a chain of 4-isogenies, where $(0, 0)$ is guaranteed not to lie in any of the kernels.

In the last step, SQIsign (NIST) uses *optimal strategies* as in SIKE [176] to compute a chain of 4-isogenies. Naive multiplicative strategies would compute $R = [2^{f'-2j}]K$, the 4-isogeny φ with kernel $\langle R \rangle$, and $K \leftarrow \varphi(K)$ for $j = 1, \ldots, f'/2$. However, this strategy is dominated by costly doublings. Instead, we can save intermediate multiples of K during the computation of $R = [2^{f'-2j}]K$, and push them through isogenies to save multiplicative effort

[14] SQIsign (NIST) fixes the sequence $x_k = 1 + k \cdot i$ with $i \in \mathbb{F}_{p^2}$ such that $i^2 = -1$ and picks the smallest k for which we find a suitable point.

in following iterations. Optimal strategies that determine which multiples are pushed through isogenies and minimise the cost can be found efficiently [**SIDH**, 176].

We note that for $f < \lambda$ the computation of $\widehat{\varphi_{\text{chall}}}$ requires small adaptations to these algorithms to allow for finding a basis of $E[D_{\text{chall}}]$ and computing 3-isogenies. Most notably, SQIsign (NIST) does *not* use optimised formulas or optimal strategies for 3-isogenies from SIKE [176], but uses a multiplicative strategy and general odd-degree isogeny formulas [104, 214]. We slightly deviate from SQIsign (NIST) by implementing optimised 3-isogeny formulas, but note that the performance difference is minor and in favor of SQIsign (NIST).

Cost metric.

In implementations, $\mathbb{F}_{p^2}$-operations usually call underlying $\mathbb{F}_p$-operations. We follow this approach and use the total number of $\mathbb{F}_p$-operations in our benchmarks. As cost metric, we express these operations in terms of $\mathbb{F}_p$-multiplications, with $\mathbf{S} = 0.8 \cdot \mathbf{M}$, ignoring $\mathbb{F}_p$-additions and subtractions due to their small impact on performance. $\mathbb{F}_p$-inversions, $\mathbb{F}_p$-square roots, and Legendre symbols over $\mathbb{F}_p$ require exponentiations by an exponent in the range of p, hence we count their cost as $\log p$ $\mathbb{F}_p$-multiplications. In contrast to measuring clock cycles of an optimised implementation, our cost metric eliminates the dependence on the level of optimisation of finite field arithmetic and the specific device running SQIsign, hence, can be considered more general.

Benchmark results.

Figure 12.2 shows the verification cost for the NIST Level I-sized primes $p^{(f)}$ for $50 \leq f \leq 250$, fixing $e = 975$, using our cost metric. For more efficient benchmarking, we sample random public key curves and signatures σ of the correct form instead of signatures generated by the SQIsign signing procedure.

The graph shows the improvement for $f \geq 128$. Furthermore, we can detect when the number of blocks n decreases solely from the graph (e.g. $f = 122, 140, 163, 195, 244$). The cost of sampling a 2^f-torsion basis is highly variable between different runs for the same prime, which is visible from the oscillations of the graph. The performance for odd f is worse in general due to the inefficient integration of the 2-isogeny, which explains the zigzag-shaped graph.

From the above observations, we conclude that $f \geq \lambda$ is significantly faster for verification, with local optima found at $f = 195$ and $f = 244$, due to those

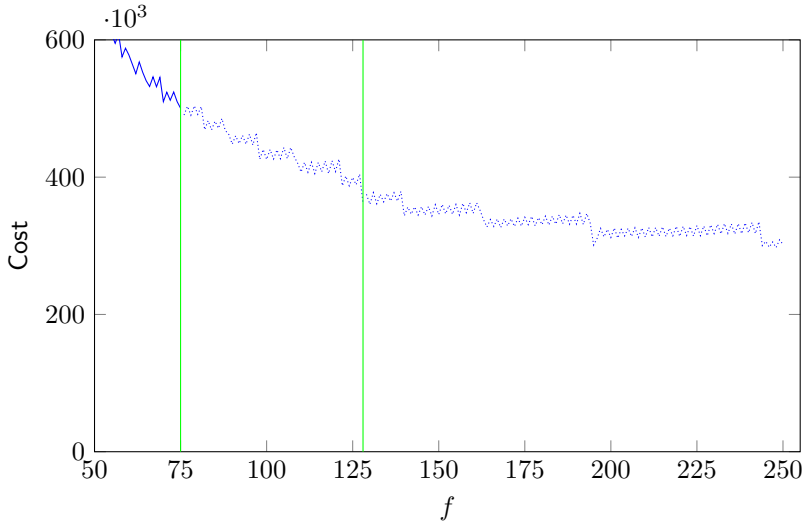


Figure 12.2: Cost in $\mathbb{F}_p$-multiplications for verification at NIST Level I security, for varying f and $p^{(f)}$, averaged over 1024 runs per prime. The green vertical lines mark $f = 75$ as used in SQIsign (NIST) for signing without extension fields, and $f = \lambda = 128$, beyond which we can set $D_{\text{chall}} = 2^\lambda$. The dotted graph beyond $f = 75$ is only accessible when signing with extension fields.

being (almost) exact divisors of the signing length $e = 975$.

Remark 12.6. The average cost of FindBasis differs significantly between primes p even if they share the same 2^f-torsion. This happens because SQIsign (NIST) finds basis points from a pre-determined sequence $[x_1, x_2, x_3, \ldots]$ with $x_j \in \mathbb{F}_{p^2}$. As we will see in Section 12.6, these x_j values can not be considered random: some values x_j are certain to be above a point of order 2^f, while others are certain not to be, for any supersingular curve over p.

12.6 Optimisations for verification

In this section, we show how the improvements from Section 12.4 that increase f beyond λ together with the analysis in Section 12.5 allow several other optimisations that improve the verification time of SQIsign in practice. Whereas the techniques in Section 12.4 allow us to decrease the *number* of blocks, in this

section, we focus on the operations occurring *within blocks*. We optimise the cost of FindBasis, FindKernel and ComputeIsogeny.

We first analyse the properties of points that have full 2^f-torsion, and use these properties to improve FindBasis and FindKernel for general f. We then describe several techniques specifically for $f \geq \lambda$. Altogether, these optimisations significantly change the implementation of verification in comparison to SQIsign (NIST). We remark that the implementation of the signing procedure must be altered accordingly, as exhibited by our implementation.

Notation.

As we mostly focus on the subroutines *within* a specific block $E^{(j)} \to E^{(j+1)}$, we will omit the superscripts in $E^{(j)}, K^{(j)}, P^{(j)}, \ldots$ and write $E, K, P, \ldots$ to simplify notation.

For reference throughout this section, the pseudocode for a single block in the verification procedure of SQIsign (NIST) and of our optimised variant is in Section 12.B as Algorithm 23 and Algorithm 24, respectively.

12.6.1 Basis generation for full 2-power torsion

We first give a general result on points having full 2^f-torsion that we will use throughout this section. This theorem generalises previous results [**CostelloJLNRU17**, 198] and will set the scene for easier and more efficient basis generation for $E[2^f]$.

Theorem 12.4. Let $E : y^2 = (x-\lambda_1)(x-\lambda_2)(x-\lambda_3)$ be an elliptic curve over $\mathbb{F}_{p^2}$ with $E[2^f] \subseteq E(\mathbb{F}_{p^2})$ the full 2-power torsion. Let $L_i = (\lambda_i, 0)$ denote the points of order 2 and $[2]E$ denote the image of E under $[2] : P \mapsto P + P$ so that $E \setminus [2]E$ are the points with full 2^f-torsion. Then

$$Q \in [2]E \text{ if and only if } x_Q - \lambda_i \text{ is square for } i = 1, 2, 3.$$

More specifically, for $Q \in E \setminus [2]E$, Q is above L_i if and only if $x_Q - \lambda_i$ is square and $x_Q - \lambda_j$ is non-square for $j \neq i$.

Proof. It is well-known that $Q = (x, y) \in [2]E$ if and only if $x - \lambda_1$, $x - \lambda_2$ and $x - \lambda_3$ are all three squares [171, Ch. 1, Thm. 4.1]. Thus, for $Q \in E \setminus [2]E$, one of these three values must be a square, and the others non-squares (as their product must be y^2, hence square). We proceed similarly as the proof of [198, Thm. 3]. Namely, let P_1, P_2 and P_3 denote points of order 2^f above $L_1 = (\lambda_1, 0)$, $L_2 = (\lambda_2, 0)$ and $L_3 = (\lambda_3, 0)$, respectively, of order 2. A point

$Q \in E \setminus [2]E$ must lie above one of the L_i. Therefore, the reduced Tate pairing of degree 2^f of P_i and Q gives a primitive 2^f-th root of unity if and only if Q is not above L_i. Let $\zeta_i = e_{2^f}(P_i, Q)$, then by [152, Thm. IX.9] we have

$$\zeta_i^{2^{f-1}} = e_2(L_i, Q).$$

We can compute $e_2(L_i, Q)$ by evaluating a Miller function f_{2,L_i} in Q, where $\operatorname{div} f_{2,L_i} = 2(L_i) - 2(\mathcal{O})$. The simplest option is the line that doubles L_i, that is, $f_{2,L_i}(x, y) = x - \lambda_i$, hence

$$e_2(L_i, Q) = (x_Q - \lambda_i)^{\frac{p^2-1}{2}}.$$

Applying Euler's criterion to this last term, we get that if $x_Q - \lambda_i$ is square, then ζ_i is not a primitive 2^f-th root and hence Q must be above L_i, whereas if $x_Q - \lambda_i$ is non-square, then ζ_i is a primitive 2^f-th root and hence Q is not above L_i. □ □

Note that for Montgomery curves $y^2 = x^3 + Ax^2 + x = x(x - \alpha)(x - 1/\alpha)$, the theorem above tells us that non-squareness of x_Q for $Q \in E(\mathbb{F}_{p^2})$ is enough to imply Q has full 2^f-torsion and is not above $(0, 0)$ [198, Thm. 3].

Finding points with 2^f-torsion above $(0, 0)$.

We describe two methods to efficiently sample Q above $(0, 0)$, based on Theorem 12.4.

1. **Direct x sampling.** By deterministically sampling $x_Q \in \mathbb{F}_p$, we ensure that x_Q is square in $\mathbb{F}_{p^2}$. Hence, if Q lies on E and $x_Q - \alpha \in \mathbb{F}_{p^2}$ is non-square, where α is a root of $x^2 + Ax + 1$, then Theorem 12.4 ensures that $Q \in E \setminus [2]E$ and above $(0, 0)$.

2. **Smart x sampling.** We can improve this using the fact that α is always square [16, 102]. Hence, if we find $z \in \mathbb{F}_{p^2}$ such that z is square and $z-1$ is non-square, we can choose $x_Q = z\alpha$ square and in turn $x_Q - \alpha = (z-1)\alpha$ non-square. Again, by Theorem 12.4 if Q is on E, this ensures Q is above $(0, 0)$ and contains full 2^f-torsion. Hence, we prepare a list $[z_1, z_2, \ldots]$ of such values z for a given prime, and try $x_j = z_j\alpha$ until x_j is on E.

Both methods require computing α, dominated by one $\mathbb{F}_{p^2}$-square root. Direct sampling computes a Legendre symbol of $x^3 + Ax^2 + x$ per x to check if the corresponding point lies on E. If so, we check if $x - \alpha$ is non-square via the

Legendre symbol. On average, this requires four samplings of x and six Legendre symbols to find a suitable x_Q with $Q \in E(\mathbb{F}_{p^2})$, and, given that we can choose x_Q to be small, we can use fast scalar multiplication on x_Q (see Section 12.A).

In addition to computing α, smart sampling requires the Legendre symbol computation of $x^3 + Ax^2 + x$ per x. On average, we require two samplings of an x to find a suitable x_Q, hence saving four Legendre symbols in comparison to direct sampling. However, we can no longer choose x_Q small, which means that improved scalar multiplication for small x_Q is not available.

Finding points with 2^f-torsion *not* above $(0,0)$.

As shown in [198], we find a point P with full 2^f-torsion *not* above $(0,0)$ by selecting a point on the curve with non-square x-coordinate. Non-squareness depends only on p, not on E, so a list of small non-square values can be precomputed. In this way, finding such a point P simply becomes finding the first value x_P in this list such that the point $(x_P, -)$ lies on $E(\mathbb{F}_{p^2})$, that is, $x_P^3 + Ax_P^2 + x_P$ is square. On average, this requires two samplings of x, hence two Legendre symbol computations.

12.6.2 General improvements to verification

In this section, we describe improvements to SQIsign verification and present new optimisations, decreasing the cost of the three main subroutines of verification.

Known techniques from literature.

There are several state-of-the-art techniques in the literature on efficient implementations of elliptic curve or isogeny-based schemes that allow for general improvements to verification, but are not included in SQIsign (NIST). We implemented such methods, e.g., to improve scalar multiplication $P \mapsto [n]P$ and square roots. The details are described in Section 12.A.

In particular, we use that $P \mapsto [n]P$ is faster when x_P is small.

Improving the subroutine FindBasis.

In SQIsign (NIST), to find a complete basis for $E[2^f]$ we are given a point $Q \in E[2^f]$ lying above $(0,0)$ and need to find another point $P \in E(\mathbb{F}_{p^2})$ of order 2^f not lying above $(0,0)$. We sample P directly using x_P non-square, as described above and demonstrated by [198], and in particular can choose x_P small. We

then compute $P \leftarrow [\frac{p+1}{2^f}]P$ via fast scalar multiplication to complete the torsion basis (P, Q).

Improved strategies for ComputeIsogeny**.**

Recall that ComputeIsogeny follows three steps in SQIsign (NIST): it first computes a 4-isogeny that may contain $(0, 0)$ in the kernel, and a 2-isogeny if f is odd, before entering an optimal strategy for computing the remaining chain of 4-isogenies. However, the first two steps include many costly doublings. We improve this by adding these first two steps in the optimal strategy. If f is even, this is straightforward, with a simple check for $(0, 0)$ in the kernel in the first step. For odd f, we add the additional 2-isogeny in this first step.[15] For simplicity of the implementation, we determine optimal strategies as in SIKE [176], thus we assume that only 4-isogenies are used.

Note that techniques for strategies with variable isogeny degrees are available from the literature on CSIDH implementations [96]. However, the performance difference is very small, hence our simplified approach appears to be preferable.

In addition to optimising 4-isogeny chains, we implemented optimised 3-isogeny chains from SIKE [176] for the computation of $\widehat{\varphi_{\text{chall}}}$ when $f < 128$.

12.6.3 To push, or not to push[16]

In SQIsign (NIST), the point Q is pushed through φ so that we easily get the basis point above $(0, 0)$ on the image curve, and we can then use Theorem 12.4 to sample the second basis point P. Instead of pushing Q, one can also use Theorem 12.4 to efficiently sample this basis point Q above $(0, 0)$. Although pushing Q seems very efficient, for larger f we are pushing Q through increasingly larger isogeny chains, whereas sampling becomes increasingly more efficient as multiplication cost by $\frac{p+1}{2^f}$ decreases. Furthermore, sampling *both* P and Q allows us to use those points as an *implicit basis* for $E[2^f]$, even if their orders are multiples of 2^f, as described in more detail below. We observe experimentally that this makes sampling Q, instead of pushing Q, more efficient for $f > 128$.

[15] In particular, we compute $R' = [2^{f-3}]K$ and $R = [2]R'$, a 4-isogeny with kernel $\langle R \rangle$, push R' through, and compute a 2-isogeny with kernel $\langle R' \rangle$.

[16] –that is, the Q.

Using implicit bases.

Using Theorem 12.4, it is possible to find points P and Q efficiently so that both have full 2^f-torsion. The pair (P, Q) is not an *explicit basis* for $E[2^f]$, as the orders of these points are likely to be multiples of 2^f. However, instead of multiplying both points by the cofactor to find an explicit basis, we can use these points implicitly, as if they were a basis for $E[2^f]$. This allows us to compute $K = P + [s]Q$ first, and only then multiply K by the cofactor. This saves a full scalar multiplication by the cofactor $\frac{p+1}{2^f}$. We refer to such a pair (P, Q) as an *implicit basis* of $E[2^f]$. Algorithmically, implicit bases combine FindBasis and FindKernel into a single routine FindBasisAndKernel.

12.6.4 Improved challenge for $f \geq \lambda$

Recall from Section 12.5.2 that when $f \geq \lambda$, we can simply set $D_{\text{chall}} = 2^\lambda$. This decreases the cost of FindBasis for the challenge computation considerably, as we can now use Theorem 12.4 to find a basis for $E[2^\lambda]$.

Improving FindBasis for the challenge isogeny when $f \geq \lambda$.

We use Theorem 12.4 twice, first to find P not above $(0, 0)$ having full 2^f-torsion and then to find Q above $(0, 0)$ having full 2^f-torsion. We choose x_P and x_Q small such that faster scalar multiplication is available. We find the basis for $E[2^\lambda]$ by $P \leftarrow [\frac{p+1}{2^f}]P$ followed by $f - \lambda$ doublings, and $Q \leftarrow [\frac{p+1}{2^f}]Q$ followed by $f - \lambda$ doublings.[17] Alternatively, if Q is pushed through isogenies, we can reuse $Q \leftarrow \varphi^{(n)}(Q^{(n)}) \in E[2^f]$ from the computation of the last step of φ_{resp}, so that we get a basis point for $E[2^\lambda]$ by $f - \lambda$ doublings of Q. Reusing this point Q also guarantees cyclicity of $\widehat{\varphi_{\text{chall}}} \circ \varphi_{\text{resp}}$.

Remark 12.7. For SQIsign without extension fields, obtaining $f \geq \lambda$ seems infeasible, hence the degree D of φ_{chall} is $2^f \cdot 3^g$. Nevertheless, some optimizations are possible in the computation of φ_{chall} in this case. FindBasis for $E[2^f \cdot 3^g]$ benefits from similar techniques as previously used in SIDH/SIKE, as we can apply known methods to improve generating a torsion basis for $E[3^g]$ coming from 3-descent [**CostelloJLNRU17**]. Such methods are an analogue to generating a basis for $E[2^f]$ as described in Theorem 12.4 and [198, Thm. 3].

[17] Algorithmically, this is faster than a single scalar multiplication by $2^{f-\lambda} \cdot \frac{p+1}{2^f}$.

12.7 Size-speed trade-offs in SQIsign signatures

The increase in f also enables several size-speed trade-offs by adding further information in the signature or by using uncompressed signatures. Some trade-offs were already present in earlier versions of SQIsign [121], however, by using large f and the improvements from Section 12.6, they become especially worthwhile.

We take a slightly different stance from previous work on SQIsign as for many use cases the main road block to using SQIsign is the efficiency of verification in cycles. In contrast, in several applications the precise size of a signature is less important as long as it is below a certain threshold.[18] For example, many applications can handle the combined public key and signature size of RSA-2048 of 528 bytes, while SQIsign (NIST) features a combined size of only 241 bytes. In this section, we take the 528 bytes of RSA-2048 as a baseline, and explore size-speed trade-offs for SQIsign verification with data sizes up to this range.

We note that the larger signatures in this section encode the same information as standard SQIsign signatures, hence have no impact on the security.

12.7.1 Adding seeds for the torsion basis in the signature

We revisit an idea that was previously present in SQIsign verification [121] (but no longer in [89] or [122]), and highlight its particular merits whenever $f \geq \lambda$, as enabled by signing with extension fields. So far, we have assumed that completing or sampling a basis for $E[2^f]$ is done by deterministically sampling points. Recall from Section 12.6.1 that sampling x_P resp. x_Q (when not pushing Q) on average requires the computation of several Legendre symbols resp. square roots. We instead suggest using a seed to find x_P (when pushing Q) or x_P and x_Q (otherwise), which we include in the signature, so that the verifier saves all of the above cost for finding x_P, resp. x_Q. Finding these seeds adds negligible overhead for the signer, while verification performance improves. Signer and verifier are assumed to agree upon all precomputed values.

Seeding a point *not* above $(0, 0)$.

For x_P *not* above $(0, 0)$, we fix a large enough $k > 0$ and precompute the 2^k smallest values $u_j \in \mathbb{F}_p$ such that $u_j + i \in \mathbb{F}_{p^2}$ is non-square (where i is the same as in Section 12.6). During signing, we pick the smallest u_j such that $x_P = u_j + i$ is the x-coordinate of a point $P \in E(\mathbb{F}_{p^2})$, and add the index j to the signature as a seed for x_P. Theorem 12.4 ensures that any $P \in E(\mathbb{F}_{p^2})$ for

[18] See https://blog.cloudflare.com/sizing-up-post-quantum-signatures/.

non-square x_P is a point with full 2^f-torsion not above $(0,0)$. This furthermore has the advantage of fast scalar multiplication for x_P as the x-coordinate is very small.

Seeding a point *above* $(0,0)$.

As noted above, when f is large, it is faster to deterministically compute a point of order 2^f above $(0,0)$ than to push Q through φ. We propose a similar seed here for fixed large enough $k > 0$, using Theorem 12.4 and the "direct sampling" approach from Section 12.6.1. During signing, we pick the smallest $j \leq 2^k$ such that $x_Q = j$ is the x-coordinate of a point $Q \in E(\mathbb{F}_{p^2})$ and $x_Q - \alpha$ is non-square. We add $x_Q = j$ to the signature as seed.

Note that when using both seeding techniques, we do not explicitly compute $[\frac{p+1}{2^f}]P$ or $[\frac{p+1}{2^f}]Q$, but rather use the seeded points P and Q as an implicit basis, as described in Section 12.6.3.

Size of seeds.

Per seeded point, we add k bits to the signature size. Thus, we must balance k such that signatures are not becoming too large, while failure probabilities for not finding a suitable seed are small enough. In particular, seeding x_P resp. x_Q via direct sampling has a failure probability of $\frac{1}{2}$ resp. $\frac{3}{4}$ per precomputed value. For the sake of simplicity, we set $k = 8$ for both seeds, such that every seed can be encoded as a separate byte.[19] This means that the failure rate for seeding Q is $(\frac{3}{4})^{256} \approx 2^{-106.25}$ for our choice, while for P it is 2^{-256}. Theoretically it is still possible that seeding failures occur. In such a case, we simply recompute KLPT. We furthermore include similar seeds for the torsion basis on E_A and E_2, giving a size increase of $(n+1) \cdot 2$ bytes.

The synergy with large f now becomes apparent. The larger f gets, the fewer blocks n are required, hence adding fewer seeds overall. For $f = 75$, the seeds require an additional 28 bytes when seeding both P and Q. For $f = 122, 140, 163, 195, 244$ this drops to 18, 16, 14, 12, and 10 additional bytes, respectively, to the overall signature size of 177 bytes for NIST Level I security.

Remark 12.8. Instead of using direct sampling for Q with failure probability $\frac{3}{4}$, we can reduce it to $\frac{1}{2}$ via "smart sampling" (see Section 12.6.1). However, this requires the verifier to compute α via a square root to set $x_Q = z\alpha$ with

[19] Note that for equal failing rates the number of possible seeds for P can be chosen smaller than for Q, hence slightly decreasing the additional data sizes.

seeded z. We thus prefer direct sampling for seeded Q, which incurs no such extra cost.

12.7.2 Uncompressed signatures

In cases where f is very large, and hence the number of blocks is small, in certain cases it is worthwhile to replace the value s in the signature by the full x-coordinate of $K = P + [s]Q$. In essence, this is the uncompressed version of the SQIsign signature σ, and we thus refer to this variant as *uncompressed* SQIsign.

Speed of uncompressed signatures.

Adding the precise kernel point K removes the need for both FindBasis and FindKernel, leaving ComputeIsogeny as the sole remaining cost. This speed-up is significant, and leaves little room for improvement beyond optimizing the cost of computing isogenies. The cost of verification in this case is relatively constant, as computing an 2^e-isogeny given the kernels is only slightly affected by the size of f, as is visible in the black dashed line in Figure 12.3. This makes uncompressed SQIsign an attractive alternative in cases where the signature size, up to a certain bound, is less relevant.

Size of uncompressed signatures.

Per step, this increases the size from $\log(s) \approx f$ to $2 \cdot \log(p)$ bits, which is still relatively size efficient when f is close to $\log(p)$. For recomputing φ_{chall}, we take a slightly different approach than before. We add the Montgomery coefficient of E_1 to the signature, and seeds for a basis of $E[2^f]$. From this, the verifier can compute the kernel generator of φ_{chall}, and verify that the j-invariant of its codomain matches E_2. Hence this adds $2 \cdot \log(p)$ bits for E_1 and two bytes for seeds to the signature, for a total of $(n+1) \cdot (\log p/4) + 2$ bytes.

For $f = 244$, this approach less than doubles the signature size from 177 bytes to 322 bytes for NIST Level I security, for $f = 145$, the signature becomes approximately 514 bytes, while for the current NIST Level I prime with $f = 75$, the size would become 898 bytes. When adding the public key size of 64 bytes, especially the first two cases still appear to be reasonable alternatives to RSA-2048's combined data size of 528 bytes.

Remark 12.9. Uncompressed signatures significantly simplify verification, as many functionalities required for compressed signatures are not necessary. Hence,

this allows for a much more compact code base, which might be important for use cases featuring embedded devices with strict memory requirements.

12.8 Primes and Performance

In this section we show the performance of verification for varying f, using the optimisations from the previous sections. Further, we find specific primes with suitable f for $n = 4$ and $n = 7$, and report their signing performance using our SageMath implementation, comparing it with the current SQIsign (NIST) prime.

12.8.1 Performance of optimised verification

To compare the verification performance of our optimised variants with compressed signatures to SQIsign (NIST) and SQIsign (LWXZ),[20] we run benchmarks in the same setting as in Section 12.5.3. In particular, Figure 12.3 shows the cost of verification for the NIST Level I primes $p^{(f)}$ for $50 \leq f \leq 250$. As before, we sample random public key curves and signatures σ of the correct form instead of using signatures generated by the SQIsign signing procedure.

For the sake of simplicity, Figure 12.3 displays only the fastest compressed `AprèsSQI` variant, namely the version that does not push Q through isogenies and uses seeds to sample P and Q. This variant significantly outperforms both SQIsign (NIST) and SQIsign (LWXZ) already at $f = 75$, at the cost of slightly larger signatures. A detailed description and comparison of all four compressed variants is in Section 12.C, which shows that our unseeded variants achieve similar large speed-ups with no increase in signature size. Lastly, the uncompressed variant achieves the fastest speed, although at a significant increase in signature size.

12.8.2 Finding specific primes

We now give two example primes, one prime optimal for 4-block verification, as well as the best we found for 7-block verification. The "quality" of a prime p is measured using the cost metric $\textsc{SigningCost}_p$ defined in Section 12.4.3.

[20] Our implementation of SQIsign (LWXZ) [198] is identical to SQIsign (NIST) except for the improved sampling of P described in Section 12.6.1.

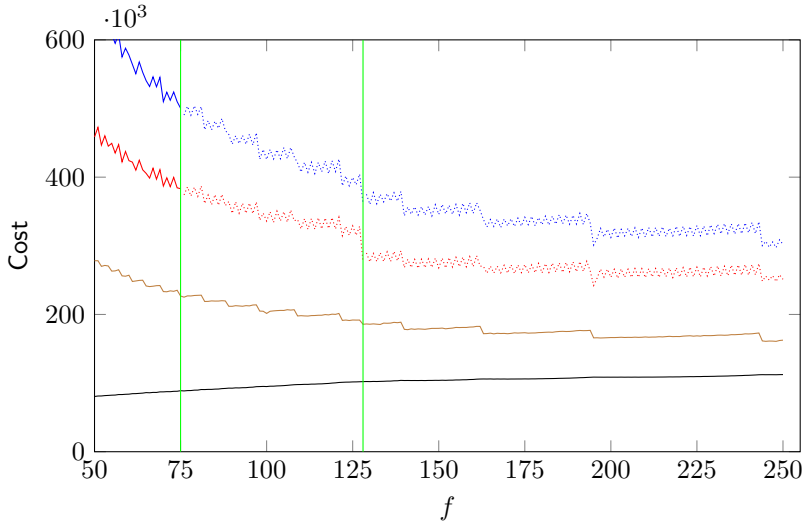


Figure 12.3: Extended version of Figure 12.2 showing the cost in $\mathbb{F}_p$-multiplications for verification at NIST Level I security, for varying f and $p^{(f)}$, averaged over 1024 runs per prime. In addition to SQIsign (NIST) in blue, it shows the performance of SQIsign (LWXZ) in red, our fastest compressed `AprèsSQI` variant in brown, and uncompressed `AprèsSQI` in black.

Optimal 4-block primes.

For 4-block primes, taking $e = 975$ as a baseline, we need f bigger than 244. In other words, we are searching for primes of the form

$$p = 2^{244}N - 1$$

where $N \in [2^4, 2^{12}]$ (accepting primes between 250 and 256 bits). This search space is quickly exhausted. For each prime of this form, we find the optimal torsion T to use, minimising $\textsc{SigningCost}_p(T)$. The prime with the lowest total cost in this metric, which we denote p_4, is

$$p_4 = 2^{246} \cdot 3 \cdot 67 - 1$$

Balanced primes.

Additionally, we look for primes that get above the significant $f > 128$ line, while minimizing $\text{SigningCost}_p(T)$. To do this, we adopt the "sieve-and-boost" technique used to find the current SQIsign primes [89, §5.2.1]. However, instead of looking for divisors of $p^2 - 1$, we follow Theorem 12.2 and look for divisors of

$$\prod_{n=1}^{k} \Phi_n(p^2)/2$$

to find a list of good candidate primes. This list is then sorted with respect to their signing cost according to SigningCost_p. The prime with the lowest signing cost we could find, which we call p_7, is

$$p_7 = 2^{145} \cdot 3^9 \cdot 59^3 \cdot 311^3 \cdot 317^3 \cdot 503^3 - 1.$$

Remark 12.10. This method of searching for primes is optimised for looking for divisors of $p^2 - 1$, hence it might be suboptimal in the case of allowing torsion in higher extension fields. We leave it as future work to find methods which further take advantage of added flexibility in the prime search.

12.8.3 Performance for specific primes

We now compare the performance of the specific primes p_4, p_7, as well as the current NIST Level I prime p_{1973} used in SQIsign (NIST).

Signing performance.

We give a summary of the estimated signing costs in Table 12.1. For p_{1973}, we include the metric "Adjusted Cost", which we compute as SigningCost with the isogeny computations scaling as $\sqrt{\ell}\log\ell$ to (rather optimistically) account for the benefit of $\sqrt{\text{élu}}$. Further, we ran our proof-of-concept SageMath implementation on the three primes, using SageMath 9.8, on a laptop with an Intel-Core i5-1038NG7 processor, averaged over five runs. An optimised C implementation will be orders of magnitude faster; we use these timings simply for comparison.

We note that the SigningCost-metric correctly predicts the ordering of the primes, though the performance difference is smaller than predicted. A possible explanation for this is that the SigningCost-metric ignores all overhead, such as quaternion operations, which roughly adds similar amounts of cost per prime.

Table 12.1: Comparison between estimated cost of signing for three different primes.

p	largest $\ell \mid T$	largest $\mathbb{F}_{p^{2k}}$	$\textsc{SigningCost}_p(T)$	Adj. Cost	Timing
p_{1973}	1973	$k = 1$	8371.7	1956.5	11m, 32s
p_7	997	$k = 23$	4137.9	-	9m, 20s
p_4	2293	$k = 53$	9632.7	-	15m, 52s

Our implementation uses $\sqrt{\text{élu}}$ whenever the kernel generator is defined over $\mathbb{F}_{p^2}$ and ℓ is bigger than a certain crossover point. This mainly benefits p_{1973}, as this prime only uses kernel generators defined over $\mathbb{F}_{p^2}$. The crossover point is experimentally found to be around $\ell > 300$ in our implementation, which is not optimal, compared to an optimised C implementation.[21] Nevertheless, we believe that these timings, together with the cost metrics, provide sufficient evidence that extension field signing in an optimised implementation stays in the same order of magnitude for signing time as staying over $\mathbb{F}_{p^2}$.

Verification performance.

In Table 12.2, we summarise the performance of verification for p_{1973}, p_7, and p_4, both in terms of speed, and signature sizes.

Two highlights of this work lie in using p_7, both with and without seeds, having (almost) the same signature sizes as the current SQIsign signatures, but achieving a speed-up of factor 2.37 resp. 2.80 in comparison to SQIsign (NIST) and 1.82 resp. 2.15 in comparison to SQIsign (LWXZ), using p_{1973}. Another interesting alternative is using uncompressed p_4, at the cost of roughly double signature sizes, giving a speed-up of factor 4.46 in comparison to SQIsign (NIST) and 3.41 in comparison to SQIsign (LWXZ).

Remark 12.11. We analyse and optimise the cost of verification with respect to $\mathbb{F}_p$-operations. However, primes of the form $p = 2^f \cdot c - 1$ are considered to be particularly suitable for fast optimised finite field arithmetic, especially when f is large [20]. Hence, we expect primes like p_4 to improve significantly more in comparison to p_{1973} in low-level field arithmetic, leading to a larger speed-up than predicted in Table 12.2. Furthermore, other low-level improvements, such as fast non-constant time GCD for inversions or Legendre symbols, will improve

[21] For instance, work by **DBLP:journals/jce/AdjCR23** gives the crossover point at $\ell > 89$, although for isogenies defined over $\mathbb{F}_p$.

Table 12.2: Comparison between verification cost for different variants and different primes, with cost given in terms of 10^3 $\mathbb{F}_p$-multiplications, using $\mathbf{S} = 0.8 \cdot \mathbf{M}$.

p	f	Implementation	Variant	Verif. cost	Sig. size
p_{1973}	75	SQIsign (NIST) [89]	-	500.4	177 B
		SQIsign (LWXZ) [198]	-	383.1	177 B
		`AprèsSQI`	unseeded	276.1	177 B
		`AprèsSQI`	seeded	226.8	195 B
p_7	145	`AprèsSQI`	unseeded	211.0	177 B
		`AprèsSQI`	seeded	178.6	193 B
		`AprèsSQI`	uncompressed	103.7	514 B
p_4	246	`AprèsSQI`	unseeded	185.2	177 B
		`AprèsSQI`	seeded	160.8	187 B
		`AprèsSQI`	uncompressed	112.2	322 B

the performance of primes in terms of cycles, which is unaccounted for by our cost metric.

TODO: CHECK: ARE THE PROOFS CORRECT WITH APPENDIX C COMMENTED OUT?

12.A Curve arithmetic

In this section we describe in detail the known techniques from literature that allow for general improvements to verification, but that are not included in SQIsign (NIST).

We use $\texttt{xDBL}(x_P)$ to denote x-only point doubling of a point P and similarly $\texttt{xADD}(x_P, x_Q, x_{P-Q})$ to denote x-only differential addition of points P and Q. We use $\texttt{xMUL}(x_P, m)$ to denote x-only scalar multiplication of a point P by the scalar m.

12.A.1 Faster scalar multiplications

We describe three improvements to the performance of `xMUL`, that can be applied in different situations during verification.

1. **Affine A.** Throughout verification and specifically in FindBasis and FindKernel, we work with the Montgomery coordinate A in projective form. However, some operations, such as computing the point difference x_{P-Q} given x_P and x_Q require A in affine form. Having an affine A allows an additional speed-up, as `xDBL` requires one $\mathbb{F}_{p^2}$-multiplication less in this case. Thus, `xMUL` with affine A is cheaper by 3 **M** per bit of the scalar.

2. **Affine points.** Using batched inversion, whenever we require A in affine form we can get x_P and x_Q in affine form for almost no extra cost. An `xMUL` with affine x_P or x_Q saves another $\mathbb{F}_{p^2}$-multiplication, hence again 3 **M** per bit of the scalar.

3. **Small x-coordinate.** For a point P with $x_P = a + bi$ with small a and b, we can replace an $\mathbb{F}_{p^2}$-multiplication by x_P with $a + b$ additions. This, in turn, saves almost 3 **M** per bit of the scalar in any `xMUL` of x_P.

As we can force P and Q to have $b \in \{0, 1\}$ and small a when sampling them in FindBasis, these points are affine and have small x-coordinates. Together with the affine A, this saves almost 9 **M** per bit for such scalar multiplications, saving roughly 27% per `xMUL`. We call such an `xMUL` a *fast* `xMUL`. Whenever `xMUL` uses 2 of these optimisations, we call it *semi-fast*.

Whenever possible, we use differential addition chains [41] to improve scalar multiplications by certain system parameters, such as $\frac{p+1}{2^f}$. In particular, we will only need to multiply by a few, predetermined scalars, and therefore we follow the method described by Cervantes-Vázquez, Chenu, Chi-Domínguez, Feo, Rodríguez-Henríquez, and Smith [87, §4.2]. Our optimal differential addition chains were precomputed using the CTIDH software [21].

12.A.2 Faster square roots

We apply several techniques from the literature to further optimise low-level arithmetic in all of verification. The most significant of these is implementing faster square roots in $\mathbb{F}_{p^2}$ [262, §5.3], which decreases the cost of finding square roots to two $\mathbb{F}_p$-exponentiations and a few multiplications.

12.A.3 Projective point difference

The implementation of SQIsign (NIST) switches between affine and projective representations for x_P, x_Q and A within each block. It does so to be able to derive the point difference x_{P-Q} from x_P and x_Q in order to complete the basis P, Q in terms of x-coordinates. However, it is possible to compute the point difference entirely projectively using Proposition 3 from [250]. This allows us to stay projective during the SQIsign (NIST) verification until we reach E_2, where we do normalization of the curve. This saves costly inversions during verification and has the additional benefit of improved elegance for SQIsign (NIST).

However, in our variant of verification, we make no use of projective point difference, as the improvements of Section 12.6 seem to outperform this already.

12.B Algorithms

The bottleneck of SQIsign verification is the computation of an isogeny of fixed degree 2^e, which is computed as $\lceil e/f \rceil$ isogenies of degree 2^f, where $f \leq e$. Each such 2^f-isogeny is called a *block*. In this section, we present algorithms for the computation of a single block in verification of SQIsign (NIST) (see Algorithm 23) and the computation using the improvements described in Sections 12.6 and 12.7 (see Algorithm 24).

Algorithm 23 Single block in verification of SQIsign (NIST)

Input: Affine coeff. $A \in \mathbb{F}_p$, a basis x_P, x_Q, x_{P-Q} for $E_A[2^f]$ with Q above $(0,0)$ and $s \in \mathbb{Z}/2^f\mathbb{Z}$ defining a kernel
Output: Affine coeff. $A' \in \mathbb{F}_p$ as the codomain of $E_A \to E_{A'}$ of degree 2^f, with a basis x_P, x_Q, x_{P-Q} for $E'_A[2^f]$ with Q above $(0,0)$

1: $K \leftarrow \texttt{3ptLadder}(x_P, x_Q, x_{P-Q}, s, A)$
2: $A_{\text{proj.}}, x_Q \leftarrow \texttt{FourIsogenyChain}(K, x_Q, A)$
3: $A, x_Q \leftarrow \texttt{ProjectiveToAffine}(A_{\text{proj.}}, x_Q)$
4: $x_P, x_{P-Q} \leftarrow \texttt{CompleteBasis}_{2^f}(x_Q, A)$
return A, x_P, x_Q, x_{P-Q}

Algorithm 24 Single block in verification using improvements of Section 12.6

Input: Projective coeff. $A \in \mathbb{F}_p$, a seed (n, m) and $s \in \mathbb{Z}/2^f\mathbb{Z}$ defining a kernel
Output: Affine coeff. $A' \in \mathbb{F}_p$ as the codomain of $E_A \to E_{A'}$ of degree 2^f

1: $x_P \leftarrow \texttt{SmallNonSquare}(m)$, $x_Q \leftarrow n$
2: $x_{P-Q} \leftarrow \texttt{PointDifference}(x_P, x_Q, A)$ ▷ implicit basis x_P, x_Q, x_{P-Q}
3: $K \leftarrow \texttt{3ptLadder}(x_P, x_Q, x_{P-Q}, s, A)$
4: $K \leftarrow \texttt{xMUL}(x_K, \frac{p+1}{2^f}, A)$ ▷ semi-fast `xMUL`
5: $A_{\text{proj.}} \leftarrow \texttt{FourIsogenyChain}(K, A)$
6: $A \leftarrow \texttt{ProjectiveToAffine}(A_{\text{proj.}})$ **return** A

12.C Performance of optimised verification

The optimisations for compressed variants from Section 12.6 and Section 12.7 allow for several variants of verification, depending on using seeds and pushing Q through isogenies. We summarise the four resulting approaches and measure their performance.

12.C.1 Four approaches for verification

To obtain our measurements, we combine our optimisations to give four different approaches to perform SQIsign verification, specifically optimised for $f \geq \lambda$. Firstly, we either push Q through φ in every block, or sample Q. Secondly, we either sample the basis or seed it.

Pushing Q, sampling P without seed.

This variant is closest to the original SQIsign (NIST) and SQIsign (LWXZ) implementations. It is the optimal version for non-seeded verification for $f \leq 128$, using the general optimisations from Section 12.6.2 and the challenge optimisations from Section 12.6.4.

Not pushing Q, sampling both P and Q without seed.

This variant competes with the previous version in terms of signature size. Due to Section 12.6.3, sampling a new Q is more efficient than pushing Q for large f.[22] This is the optimal version for non-seeded verification for $f > 128$, and additionally uses the optimisations from Section 12.6.3.

Pushing Q, sampling P with seed.

This variant only adds seeds to the signature to describe x_P. As such, it lies between the other three variants in terms of both signature size and speed. The signature is 1 byte per block larger than the unseeded variants, and 1 byte per block smaller than the variant where x_Q is seeded too. In terms of speed, it is faster than the variants where P is unseeded, but slower than the variant where Q is seeded too. It uses the optimisations from Sections 12.6.2 and 12.6.4, but cannot benefit from the kernel computation via implicit bases from Section 12.6.3.

[22] Based on benchmarking results, we sample Q with $x = n\alpha$ for $f < 200$ and directly for $f \geq 200$.

Not pushing Q and sampling both P and Q with seed.

This is the fastest compressed version that we present in this work. Although it adds 2 bytes per block, the small number of blocks n for large f makes the total increase in signature size small. All the optimisations from Section 12.6 now apply: we additionally have fast `xMUL` for Q, as well as the optimised implicit basis method to compute the kernel and optimised challenge. An algorithmic description of a single block in this version is given in Algorithm 24.

12.C.2 Performance benchmark

We benchmarked these four approaches according to our cost metric by taking the average over 1024 random signatures. The results are given in Figure 12.4 showing the significant increase in performance compared to SQIsign (NIST) and SQIsign (LWXZ), as well as the additional performance gained from seeding. For comparison, we also show the performance when using uncompressed signatures, serving as a lower bound for the cost.

12.D Detailed information on primes

We give more details on the specific primes used in Section 12.8.

12.D.1 Details on p_7

The prime p_7 is used for a verification with $n = 7$ blocks. It achieves $f = 145$, with T given as below.

$$
\begin{aligned}
p_7 &= \texttt{0x309c04bcaedbb0134cca8373e439ffffffffffffffffffffffffffffffffffff} \\
T &= 3^7 \cdot 5^4 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19^2 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53^2 \cdot 59^3 \cdot 61 \cdot 67 \\
&\quad \cdot 71 \cdot 73 \cdot 79 \cdot 109 \cdot 113 \cdot 131 \cdot 157 \cdot 181 \cdot 193 \cdot 223 \cdot 239 \cdot 241 \cdot 271 \cdot 283 \cdot 311^3 \\
&\quad \cdot 317^3 \cdot 331 \cdot 349 \cdot 503^2 \cdot 859 \cdot 997 \\
\textsc{SigningCost}_{p_7}(T) &= 4137.91235
\end{aligned}
$$

The field of definition for the various torsion groups we work with can be found in Table 12.3.

Table 12.3: Torsion groups $E[N]$ and their minimal field $E(\mathbb{F}_{p^{2k}})$ for the prime p_7

k	N
1	$3^7, 53^2, 59^3, 61, 79, 283, 311^3, 317^3, 349, 503^2, 859, 997$
3	$13, 109, 223, 331$
4	17
5	$11, 31, 71, 241, 271$
6	157
7	$7^2, 29, 43, 239$
8	113
9	19^2
10	$5^4, 41$
11	$23, 67$
12	193
13	131
15	181
18	$37, 73$
23	47

12.D.2 Details on p_4

The prime p_4 is used for a verification with $n = 4$ blocks. It achieves $f = 246$, with T given as below.

$p_4 =$ `0x323fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff`

$T = 3^3 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53 \cdot 59 \cdot 61 \cdot 67 \cdot 71$
$\cdot 73 \cdot 79 \cdot 83 \cdot 89 \cdot 97 \cdot 101 \cdot 103 \cdot 107 \cdot 109 \cdot 113 \cdot 127 \cdot 149 \cdot 151 \cdot 157 \cdot 163 \cdot 181$
$\cdot 197 \cdot 211 \cdot 229 \cdot 241 \cdot 271 \cdot 317 \cdot 397 \cdot 577 \cdot 593 \cdot 641 \cdot 661 \cdot 757 \cdot 1069 \cdot 2293$

$\textsc{SigningCost}_{p_4}(T) = 9632.7307$

The field of definition for the various torsion groups can be found in Table 12.4.

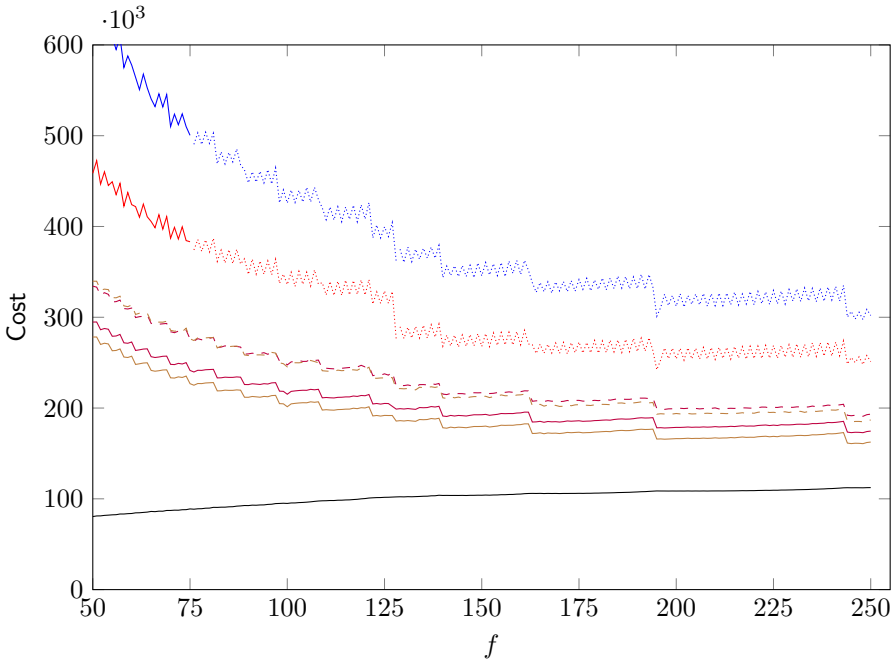


Figure 12.4: Extended version of Figure 12.3 showing the cost in $\mathbb{F}_p$-multiplications for verification at NIST-I security level, for varying f and $p^{(f)}$, averaged over 1024 runs per prime. In addition to SQIsign (NIST) in blue, and SQIsign (LWXZ) in red, it shows all `AprèsSQI` variants: In purple is the performance of `AprèsSQI` when pushing Q, with dashed blue when not seeding P. In brown is the performance of `AprèsSQI` when not pushing Q, with dashed brown when not seeding P, Q. The performance of uncompressed `AprèsSQI` is shown in black.

Table 12.4: Torsion groups $E[N]$ and their minimal field $E(\mathbb{F}_{p^{2k}})$ for the prime p_4

k	N
1	$67, 73, 757$
2	$317, 2293$
3	$37, 127, 1069$
4	593
5	$11, 31, 71, 661$
6	13
7	43
8	$17, 113$
9	$3^3, 19, 181, 577$
10	$5^2, 61, 641$
11	$23, 89$
14	$29, 197$
18	397
19	229
20	41
21	7^2
23	47
25	151
26	53
27	$109, 163, 271$
29	59
30	241
35	211
37	149
39	$79, 157$
41	83
48	97
50	101
51	103
53	107

Chapter 13

SQIsign on M4

13.1 Placeholder

The paper will go here.

Chapter 14

Return of the Kummer

14.1 Placeholder

The paper will go here.

Considerations

Some considerations on future research on SQIsign, given the ideas in Apres, Kummer and current state-of-the-art.

The remainder is not part of part:measuring

add some additional spacing

Part IV

General

Chapter 15

Guide to the design of signature schemes.

Contents of the SOK, e.g. guide to the design.

Bibliography

[1] Gora Adj, Jesús-Javier Chi-Domínguez, and Francisco Rodríguez-Henríquez. "Karatsuba-based square-root Vélu's formulas applied to two isogeny-based protocols". In: *IACR Cryptol. ePrint Arch.* online first (2020), p. 1109. DOI: 10.1007/s13389-022-00293-y. URL: https://eprint.iacr.org/2020/1109 (cit. on pp. 141, 197, 198).

[2] Gora Adj, Jesús-Javier Chi-Domıénguez, Vıéctor Mateu, and Francisco Rodrıéguez-Henrıéquez. "Faulty isogenies: a new kind of leakage". In: *CoRR* abs/2202.04896 (2022). arXiv: 2202.04896. URL: https://arxiv.org/abs/2202.04896 (cit. on pp. 137, 158, 164, 165).

[3] Gora Adj, Jesús-Javier Chi-Domıénguez, and Francisco Rodrıéguez-Henrıéquez. "Karatsuba-based square-root Vélu's formulas applied to two isogeny-based protocols". In: *IACR Cryptol. ePrint Arch.* (2020), p. 1109. URL: https://eprint.iacr.org/2020/1109 (cit. on p. 227).

[4] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and Jurjen Bos. *HQC*. NIST PQC Submission. 2020 (cit. on p. 83).

[5] Carlos Aguilar-Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. *The Return of the SDitH*. Cryptology ePrint Archive, Paper 2022/1645. To appear at Eurocrypt 2023. 2022. URL: https://eprint.iacr.org/2022/1645 (cit. on pp. 119, 125).

[6] Toru Akishita and Tsuyoshi Takagi. "Zero-Value Point Attacks on Elliptic Curve Cryptosystem". In: *Information Security, 6th International Conference, ISC 2003, Bristol, UK, October 1-3, 2003, Proceedings*. Ed. by Colin Boyd and Wenbo Mao. Vol. 2851. Lecture Notes in Computer

Science. Springer, 2003, pp. 218–233. URL: https://doi.org/10.1007/10958513%5C_17 (cit. on p. 137).

[7] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. "Cryptographic group actions and applications". In: *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. LNCS. Springer. 2020, pp. 411–439 (cit. on p. 15).

[8] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. *Classic McEliece*. NIST PQC Submission. 2020 (cit. on pp. 83, 115).

[9] Gianira N Alfarano, Francisco Javier Lobillo, Alessandro Neri, and Antonia Wachter-Zeh. "Sum-rank product codes and bounds on the minimum distance". In: *Finite Fields and Their Applications* 80 (2022), p. 102013 (cit. on p. 62).

[10] Yawning Angel, Benjamin Dowling, Andreas Hülsing, Peter Schwabe, and Fiona Johanna Weber. "Post Quantum Noise". In: 2022, pp. 97–109. DOI: 10.1145/3548606.3560577 (cit. on p. 245).

[11] Tom M. Apostol. "Resultants of cyclotomic polynomials". In: *Proceedings of the American Mathematical Society* 24.3 (1970), pp. 457–462 (cit. on p. 330).

[12] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zémor, Valentin Vasseur, and Santosh Ghosh. *BIKE*. NIST PQC Submission. 2020 (cit. on p. 83).

[13] Nicolas Aragon, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, Olivier Ruatta, Jean-Pierre Tillich, Gilles Zemor, Carlos Aguilar Melchor, Slim Bettaieb, Loic Bidoux, Magali Bardet, and Ayoub Otmani. *ROLLO (Rank-Ouroboros, LAKE and LOCKER)*. 2019. URL: https://csrc.nist.gov/projects/post-quantum-cryptography/%20round-2-submissions (cit. on p. 46).

[14] Nicolas Aragon, Philippe Gaborit, Adrien Hauteville, Olivier Ruatta, and Gilles Zémor. "Low Rank Parity Check Codes: New Decoding Algorithms and Applications to Cryptography". In: *IEEE Transactions on Information Theory* 65 (2019), pp. 7697–7717 (cit. on p. 46).

[15] V. Arlazarov, E. Dinic, M. Kronrod, and I. Faradžev. "On Economical Construction of the Transitive Closure of an Oriented Graph". In: *Doklady Akademii Nauk*. Vol. 194. Russian Academy of Sciences. 1970, pp. 487–488 (cit. on p. 113).

[16] Roland Auer and Jaap Top. "Legendre Elliptic Curves over Finite Fields". In: *Journal of Number Theory* 95.2 (2002), pp. 303–312. ISSN: 0022-314X. DOI: https://doi.org/10.1006/jnth.2001.2760. URL: https://www.sciencedirect.com/science/article/pii/S0022314X0192760X (cit. on p. 340).

[17] Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. "Supersingular Isogeny Key Encapsulation". In: *Third round candidate of the NIST's post-quantum cryptography standardization process, 2020.* (2020). URL: https://sike.org/ (cit. on p. 213).

[18] Reza Azarderakhsh, David Jao, and Christopher Leonardi. "Post-Quantum Static-Static Key Agreement Using Multiple Protocol Instances". In: 2017, pp. 45–63. DOI: 10.1007/978-3-319-72565-9_3 (cit. on p. 243).

[19] Jean-Claude Bajard and Sylvain Duquesne. "Montgomery-friendly primes and applications to cryptography". In: 11.4 (Nov. 2021), pp. 399–415. DOI: 10.1007/s13389-021-00260-z (cit. on pp. 267, 268).

[20] Jean-Claude Bajard and Sylvain Duquesne. "Montgomery-friendly primes and applications to cryptography". In: *J. Cryptogr. Eng.* 11.4 (2021), pp. 399–415. DOI: 10.1007/s13389-021-00260-z. URL: https://doi.org/10.1007/s13389-021-00260-z (cit. on p. 350).

[21] Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. "CTIDH: faster constant-time CSIDH". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.4 (2021), pp. 351–387. DOI: 10.46586/tches.v2021.i4.351-387. URL: https://doi.org/10.46586/tches.v2021.i4.351-387 (cit. on pp. 137, 140, 141, 152, 153, 214, 219, 238, 353).

[22] Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. "CTIDH: faster constant-time CSIDH". In: 2021.4 (2021), pp. 351–387. DOI: 10.46586/tches.v2021.i4.351-387. URL: https://tches.iacr.org/index.php/TCHES/article/view/9069 (cit. on pp. 240, 241, 243, 249–251, 256, 261, 264, 265, 277, 282, 295, 299, 301).

[23] Gustavo Banegas, Thomas Debris-Alazard, Milena Nedeljković, and Benjamin Smith. *Wavelet: Code-based postquantum signatures with fast verification on microcontrollers*. Cryptology ePrint Archive, Paper 2021/1432. 2021. URL: https://eprint.iacr.org/2021/1432 (cit. on pp. 119, 125).

[24] Gustavo Banegas, Valerie Gilchrist, Anaëlle Le Dévéhat, and Benjamin Smith. "Fast and Frobenius: Rational Isogeny Evaluation over Finite Fields". In: *LATINCRYPT 2023 - 8th International Conference on Cryptology and Information Security in Latin America*. Springer. 2023, pp. 129–148 (cit. on p. 325).

[25] Gustavo Banegas, Valerie Gilchrist, and Benjamin Smith. "Efficient supersingularity testing over $\mathbb{F}_p$ and CSIDH key validation". In: *Mathematical Cryptology* 2.1 (2022), pp. 21–35 (cit. on pp. 277, 295, 296, 299).

[26] Gustavo Banegas, Valerie Gilchrist, and Benjamin Smith. *Efficient supersingularity testing over $\mathbb{F}_p$ and CSIDH key validation*. Cryptology ePrint Archive, Report 2022/880. 2022. URL: https://eprint.iacr.org/2022/880 (cit. on p. 263).

[27] Gustavo Banegas, Valerie Gilchrist, and Benjamin Smith. "Efficient supersingularity testing over GF(p) and CSIDH key validation". In: *Mathematical Cryptology* 2.1 (Oct. 2022), pp. 21–35. URL: https://journals.flvc.org/mathcryptology/article/view/132125 (cit. on p. 208).

[28] Gustavo Banegas, Juliane Krämer, Tanja Lange, Michael Meyer, Lorenz Panny, Krijn Reijnders, Jana Sotáková, and Monika Trimoska. "Disorientation faults in CSIDH". In: *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*. Springer. 2023, pp. 310–342 (cit. on pp. 278, 282, 302).

[29] Feng Bao, Robert H. Deng, and Huafei Zhu. "Variations of Diffie-Hellman Problem". In: *ICICS 2003*. Ed. by Sihan Qing, Dieter Gollmann, and Jianying Zhou. Vol. 2836. LNCS. Springer, 2003, pp. 301–312 (cit. on p. 91).

[30] Magali Bardet, Maxime Bros, Daniel Cabarcas, Philippe Gaborit, Ray A. Perlner, Daniel Smith-Tone, Jean-Pierre Tillich, and Javier A. Verbel. "Improvements of Algebraic Attacks for Solving the Rank Decoding and MinRank Problems". In: *ASIACRYPT 2020*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. LNCS. Springer, 2020, pp. 507–536. DOI: 10.1007/978-3-030-64837-4_17. URL: https://doi.org/10.1007/978-3-030-64837-4%5C_17 (cit. on pp. 84, 107, 109).

[31] Alessandro Barenghi, Jean-Francois Biasse, Edoardo Persichetti, and Paolo Santini. *On the Computational Hardness of the Code Equivalence Problem in Cryptography*. Cryptology ePrint Archive, Paper 2022/967. https://eprint.iacr.org/2022/967. 2022. URL: https://eprint.iacr.org/2022/967 (cit. on p. 51).

[32] Alessandro Barenghi, Jean-François Biasse, Tran Ngo, Edoardo Persichetti, and Paolo Santini. "Advanced signature functionalities from the code equivalence problem". In: *Int. J. Comput. Math. Comput. Syst. Theory* 7.2 (2022), pp. 112–128 (cit. on pp. 84, 118).

[33] Alessandro Barenghi, Jean-François Biasse, Edoardo Persichetti, and Paolo Santini. "LESS-FM: fine-tuning signatures from the code equivalence problem". In: *Post-Quantum Cryptography: 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20–22, 2021, Proceedings 12*. Ed. by Jung Hee Cheon and Jean-Pierre Tillich. Vol. 12841. LNCS. Springer. Springer, 2021, pp. 23–43 (cit. on pp. 45, 51, 61, 84, 91, 99, 100, 119, 125).

[34] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. "Efficient Implementation of Pairing-Based Cryptosystems". In: 17.4 (Sept. 2004), pp. 321–334. DOI: 10.1007/s00145-004-0311-z (cit. on p. 283).

[35] Paulo SLM Barreto, Hae Y Kim, Ben Lynn, and Michael Scott. "Efficient algorithms for pairing-based cryptosystems". In: *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22*. Springer. 2002, pp. 354–369 (cit. on pp. 283, 284, 288).

[36] Paul Barrett. “Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor”. In: *CRYPTO’ 86*. Ed. by Andrew M. Odlyzko. Springer Berlin Heidelberg, 1987, pp. 311–323 (cit. on p. 113).

[37] Aurélie Bauer, Éliane Jaulmes, Emmanuel Prouff, and Justine Wild. “Horizontal Collision Correlation Attack on Elliptic Curves”. In: *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*. Ed. by Tanja Lange, Kristin E. Lauter, and Petr Lisonek. Vol. 8282. Lecture Notes in Computer Science. Springer, 2013, pp. 553–570. URL: https://doi.org/10.1007/978-3-662-43414-7%5C_28 (cit. on p. 137).

[38] Genrich R. Belitskii, Vyacheslav Futorny, Mikhail Muzychuk, and Vladimir V. Sergeichuk. “Congruence of matrix spaces, matrix tuples, and multilinear maps”. In: *Linear Algebra and its Applications* 609 (2021), pp. 317–331. ISSN: 0024-3795. DOI: https://doi.org/10.1016/j.laa.2020.09.018. URL: https://www.sciencedirect.com/science/article/pii/S0024379520304407 (cit. on pp. 47, 58).

[39] Emanuele Bellini, Florian Caullery, Philippe Gaborit, Marcos Manzano, and Víctor Mateu. “Improved Veron Identification and Signature Schemes in the Rank Metric”. In: *2019 IEEE International Symposium on Information Theory (ISIT)* (2019), pp. 1872–1876 (cit. on p. 46).

[40] Thierry P. Berger. “Isometries for rank distance and permutation group of Gabidulin codes”. In: *IEEE Trans. Inf. Theory* 49 (2003), pp. 3016–3019 (cit. on p. 20).

[41] Daniel J. Bernstein. *Differential addition chains*. 2006. URL: http://cr.yp.to/ecdh/diffchain-20060219.pdf (cit. on p. 353).

[42] Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. “Faster computation of isogenies of large prime degree”. In: *Open Book Series* 4.1 (2020), pp. 39–55 (cit. on p. 317).

[43] Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. “Faster computation of isogenies of large prime degree”. In: *CoRR* abs/2003.10118 (2020). Ed. by Steven D. Galbraith. https://msp.org/obs/2020/4-1/obs-v4-n1-p04-p.pdf, pp. 39–55. DOI: 10.2140/obs.2020.4.39. arXiv: 2003.10118. URL: https://arxiv.org/abs/2003.10118 (cit. on pp. 141, 175, 243, 248, 250, 253, 259).

[44] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. "Elligator: elliptic-curve points indistinguishable from uniform random strings". In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM, 2013, pp. 967–980. DOI: 10.1145/2508859.2516734. URL: https://ia.cr/2013/325 (cit. on pp. 175, 178).

[45] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. "Elligator: elliptic-curve points indistinguishable from uniform random strings". In: 2013, pp. 967–980. DOI: 10.1145/2508859.2516734 (cit. on p. 297).

[46] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. "Quantum Circuits for the CSIDH: Optimizing Quantum Evaluation of Isogenies". In: 2019, pp. 409–441. DOI: 10.1007/978-3-030-17656-3_15 (cit. on pp. 240, 242, 250).

[47] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. "Quantum Circuits for the CSIDH: Optimizing Quantum Evaluation of Isogenies". In: *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. Lecture Notes in Computer Science. Springer, 2019, pp. 409–441. DOI: 10.1007/978-3-030-17656-3_15. URL: https://doi.org/10.1007/978-3-030-17656-3%5C_15 (cit. on pp. 139, 140, 142, 172, 173, 177).

[48] Daniel J. Bernstein and Bo-Yin Yang. "Fast constant-time gcd computation and modular inversion". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.3 (2019), pp. 340–398. DOI: 10.13154/tches.v2019.i3.340-398. URL: https://doi.org/10.13154/tches.v2019.i3.340-398 (cit. on p. 226).

[49] Ward Beullens. "Not enough LESS: An improved algorithm for solving code equivalence problems over $\mathbb{F}_q$". In: *International Conference on Selected Areas in Cryptography*. Ed. by Orr Dunkelman, Michael J. Jacobson, and Colin O'Flynn. Vol. 12804. LNCS. https://ia.cr/2020/801. Springer. Springer, 2020, pp. 387–403 (cit. on p. 46).

[50] Ward Beullens, Ming-Shing Chen, Shih-Hao Hung, Matthias J. Kannwischer, Bo-Yuan Peng, Cheng-Jhih Shih, and Bo-Yin Yang. *Oil and Vinegar: Modern Parameters and Implementations*. Cryptology ePrint

Archive, Paper 2023/059. 2023. URL: https://eprint.iacr.org/2023/059 (cit. on pp. 100, 119, 125).

[51] Ward Beullens, Shuichi Katsumata, and Federico Pintore. "Calamari and Falafl: Logarithmic (Linkable) Ring Signatures from Isogenies and Lattices". In: *ASIACRYPT 2020*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. LNCS. Springer, 2020, pp. 464–492 (cit. on pp. 84, 89, 96, 99, 102, 118).

[52] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. "CSI-FiSh: Efficient Isogeny Based Signatures Through Class Group Computations". In: 2019, pp. 227–247. DOI: 10.1007/978-3-030-34578-5_9 (cit. on p. 250).

[53] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. "CSI-FiSh: Efficient Isogeny Based Signatures Through Class Group Computations". In: *ASIACRYPT 2019*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11921. LNCS. Springer, 2019, pp. 227–247. DOI: 10.1007/978-3-030-34578-5_9. URL: https://ia.cr/2019/498 (cit. on pp. 99, 100, 171).

[54] Ward Beullens and Bart Preneel. "Field Lifting for Smaller UOV Public Keys". In: *Progress in Cryptology – INDOCRYPT 2017*. Ed. by Arpita Patra and Nigel P. Smart. Cham: Springer International Publishing, 2017, pp. 227–246. ISBN: 978-3-319-71667-1 (cit. on p. 45).

[55] Jean-François Biasse, David Jao, and Anirudh Sankar. "A quantum algorithm for computing isogenies between supersingular elliptic curves". In: *International Conference on Cryptology in India*. Springer. 2014, pp. 428–442 (cit. on p. 318).

[56] Jean-François Biasse, Giacomo Micheli, Edoardo Persichetti, and Paolo Santini. "LESS is More: Code-Based Signatures Without Syndromes". In: *Progress in Cryptology - AFRICACRYPT 2020*. Ed. by Abderrahmane Nitaj and Amr Youssef. Vol. 12174. LNCS. Cham: Springer International Publishing, 2020, pp. 45–65. ISBN: 978-3-030-51938-4 (cit. on pp. 45, 51, 84).

[57] Xavier Bonnetain and André Schrottenloher. "Quantum Security Analysis of CSIDH". In: *EUROCRYPT 2020*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. LNCS. https://eprint.iacr.org/2018/537. Springer, 2020, pp. 493–522. ISBN: 978-3-030-45723-5. DOI: 10.1007/978-3-030-45724-2_17. URL: https://doi.org/10.1007/978-3-030-45724-2%5C_17 (cit. on pp. 86, 240).

[58] Giacomo Borin, Edoardo Persichetti, Paolo Santini, Federico Pintore, and Krijn Reijnders. *A Guide to the Design of Digital Signatures based on Cryptographic Group Actions*. Cryptology ePrint Archive, Paper 2023/718. https://eprint.iacr.org/2023/718. 2023. URL: https://eprint.iacr.org/2023/718 (cit. on p. 13).

[59] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. "CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM". In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2018, pp. 353–367 (cit. on p. 10).

[60] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. "Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem". In: *2015 IEEE Symposium on Security and Privacy*. 2015, pp. 553–570. DOI: 10.1109/SP.2015.40 (cit. on p. 10).

[61] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. "Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem". In: 2015, pp. 553–570. DOI: 10.1109/SP.2015.40 (cit. on pp. 245, 269).

[62] C. Bouillaguet, J.-C. Faugère, P.A. Fouque, and L. Perret. "Practical Cryptanalysis of the Identification Scheme Based on the Isomorphism of Polynomial With One Secret Problem". In: *Public Key Cryptography – PKC 2011*. Vol. 6571. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 441–458 (cit. on pp. 54, 60, 76, 77).

[63] Charles Bouillaguet. "Algorithms for some hard problems and cryptographic attacks against specific cryptographic primitives. (Études d'hypothèses algorithmiques et attaques de primitives cryptographiques)". PhD thesis. Paris Diderot University, France, 2011. URL: https://tel.archives-ouvertes.fr/tel-03630843 (cit. on pp. 47, 48, 54, 68, 75, 77, 78, 80, 105).

[64] Charles Bouillaguet, Pierre-Alain Fouque, and Amandine Véber. "Graph-Theoretic Algorithms for the "Isomorphism of Polynomials" Problem". In: *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. Lecture Notes in Computer Science. Springer, 2013, pp. 211–227. DOI: 10.1007/978-

3-642-38348-9_13. URL: https://doi.org/10.1007/978-3-642-38348-9%5C_13 (cit. on pp. 47, 48, 54, 65–69, 75–77, 80, 104, 105).

[65] Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. "Towards Post-Quantum Security for Signal's X3DH Handshake". In: 2020, pp. 404–430. DOI: 10.1007/978-3-030-81652-0_16 (cit. on p. 270).

[66] Peter Bruin. "The Tate pairing for abelian varieties over finite fields". In: *Journal de theorie des nombres de Bordeaux* 23.2 (2011), pp. 323–328 (cit. on p. 37).

[67] Giacomo Bruno, Maria Corte-Real Santos, Craig Costello, Jonathan Komada Eriksen, Michael Meyer, Michael Naehrig, and Bruno Sterner. "Cryptographic Smooth Neighbors". In: *IACR Cryptol. ePrint Arch.* (2022), p. 1439. URL: https://eprint.iacr.org/2022/1439 (cit. on pp. 314, 323).

[68] Fabio Campos, Jorge Chavez-Saab, Jesús-Javier Chi-Domínguez, Michael Meyer, Krijn Reijnders, Francisco Rodríguez-Henríquez, Peter Schwabe, and Thom Wiggers. *On the Practicality of Post-Quantum TLS Using Large-Parameter CSIDH*. Cryptology ePrint Archive, Paper 2023/793. 2023. URL: https://eprint.iacr.org/2023/793 (cit. on pp. 277, 279, 282, 283, 293, 294, 296, 298–301).

[69] Fabio Campos, Matthias J. Kannwischer, Michael Meyer, Hiroshi Onuki, and Marc Stöttinger. "Trouble at the CSIDH: Protecting CSIDH with Dummy-Operations Against Fault Injection Attacks". In: *17th Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2020, Milan, Italy, September 13, 2020*. IEEE. IEEE, 2020, pp. 57–65. DOI: 10.1109/FDTC51366.2020.00015. URL: https://doi.org/10.1109/FDTC51366.2020.00015 (cit. on pp. 137, 174, 204–206, 208).

[70] Fabio Campos, Juliane Krämer, and Marcel Müller. "Safe-Error Attacks on SIKE and CSIDH". In: *Security, Privacy, and Applied Cryptography Engineering - 11th International Conference, SPACE 2021, Kolkata, India, December 10-13, 2021, Proceedings*. Ed. by Lejla Batina, Stjepan Picek, and Mainack Mondal. Vol. 13162. Lecture Notes in Computer Science. Springer, 2021, pp. 104–125. DOI: 10.1007/978-3-030-95085-9_6. URL: https://doi.org/10.1007/978-3-030-95085-9%5C_6 (cit. on p. 137).

[71] Fabio Campos, Michael Meyer, Krijn Reijnders, and Marc Stöttinger. *Patient Zero and Patient Six: Zero-Value and Correlation Attacks on CSIDH and SIKE*. Cryptology ePrint Archive, Paper 2022/904. To appear in SAC 2022. 2022. URL: https://ia.cr/2022/904 (cit. on p. 172).

[72] Anne Canteaut and Yuval Ishai, eds. *Advances in Cryptology – EUROCRYPT 2020 – 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*. Vol. 12106. Lecture Notes in Computer Science. Springer, 2020. ISBN: 978-3-030-45723-5. DOI: 10.1007/978-3-030-45724-2.

[73] David G Cantor. "On arithmetical algorithms over finite fields". In: *Journal of Combinatorial Theory, Series A* 50.2 (1989), pp. 285–300 (cit. on p. 24).

[74] Antoine Casanova, Jean-Charles Faugère, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. "GeMSS: A Great Multivariate Short Signature". In: 2017 (cit. on p. 45).

[75] J. W. S. Cassels and E. V. Flynn. *Prolegomena to a Middlebrow Arithmetic of Curves of Genus 2*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1996 (cit. on p. 24).

[76] Wouter Castryck and Thomas Decru. "An Efficient Key Recovery Attack on SIDH". In: 2023, pp. 423–447. DOI: 10.1007/978-3-031-30589-4_15 (cit. on p. 243).

[77] Wouter Castryck and Thomas Decru. "An Efficient Key Recovery Attack on SIDH". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. Lecture Notes in Computer Science. Springer, 2023, pp. 423–447. DOI: 10.1007/978-3-031-30589-4_15. URL: https://doi.org/10.1007/978-3-031-30589-4%5C_15 (cit. on p. 33).

[78] Wouter Castryck and Thomas Decru. *An efficient key recovery attack on SIDH (preliminary version)*. Cryptology ePrint Archive, Paper 2022/975. 2022. URL: https://ia.cr/2022/975 (cit. on p. 173).

[79] Wouter Castryck and Thomas Decru. "CSIDH on the Surface". In: 2020, pp. 111–129. DOI: 10.1007/978-3-030-44223-1_7 (cit. on pp. 243, 248).

[80] Wouter Castryck and Thomas Decru. "CSIDH on the Surface". In: *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, Paris, France, April 15-17, 2020, Proceedings*. Ed. by Jintai Ding and Jean-Pierre Tillich. Vol. 12100. Lecture Notes in Computer Science. Springer, 2020, pp. 111–129. DOI: 10.1007/978-3-030-44223-1_7. URL: https://doi.org/10.1007/978-3-030-44223-1%5C_7 (cit. on pp. 213, 216, 219, 231, 234, 235).

[81] Wouter Castryck, Thomas Decru, Marc Houben, and Frederik Vercauteren. "Horizontal Racewalking Using Radical Isogenies". In: 2022, pp. 67–96. DOI: 10.1007/978-3-031-22966-4_3 (cit. on pp. 243, 248).

[82] Wouter Castryck, Thomas Decru, and Frederik Vercauteren. "Radical Isogenies". In: 2020, pp. 493–519. DOI: 10.1007/978-3-030-64834-3_17 (cit. on pp. 248, 259).

[83] Wouter Castryck, Thomas Decru, and Frederik Vercauteren. "Radical Isogenies". In: *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. Lecture Notes in Computer Science. Springer, 2020, pp. 493–519. DOI: 10.1007/978-3-030-64834-3_17. URL: https://doi.org/10.1007/978-3-030-64834-3%5C_17 (cit. on pp. 169, 213, 214, 216, 219–221, 223–225, 230, 231, 234, 235).

[84] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. "CSIDH: An Efficient Post-Quantum Commutative Group Action". In: 2018, pp. 395–427. DOI: 10.1007/978-3-030-03332-3_15 (cit. on pp. 239, 240, 242, 246, 248, 250, 252, 254, 257, 261–263, 265, 277, 280, 281, 297, 299).

[85] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. "CSIDH: an efficient post-quantum commutative group action". In: *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III 24*. Ed. by Thomas Peyrin and Steven D. Galbraith. Vol. 11274. LNCS. Springer. Springer, 2018, pp. 395–427. DOI: 10.1007/978-3-030-03332-3_15. URL: https://doi.org/10.1007/978-3-030-03332-3%5C_15 (cit. on pp. 9, 29).

[86] 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: 2019, pp. 173–193. DOI: 10.1007/978-3-030-30530-7_9 (cit. on pp. 240, 241, 243, 252, 253, 257, 258, 263).

[87] Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domiénguez, Luca De Feo, Francisco Rodrıéguez-Henrıéquez, and Benjamin Smith. "Stronger and Faster Side-Channel Protections for CSIDH". In: *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*. Ed. by Peter Schwabe and Nicolas Thériault. Vol. 11774. Lecture Notes in Computer Science. Springer. Springer, 2019, pp. 173–193. DOI: 10.1007/978-3-030-30530-7_9. URL: https://doi.org/10.1007/978-3-030-30530-7%5C_9 (cit. on p. 353).

[88] Sanjit Chatterjee, Palash Sarkar, and Rana Barua. "Efficient Computation of Tate Pairing in Projective Coordinate over General Characteristic Fields". In: 2005, pp. 168–181. DOI: 10.1007/11496618_13 (cit. on p. 288).

[89] Jorge Chavez-Saab, Maria Corte-Real Santos, Luca De Feo, Jonathan Komada Eriksen, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Michael Meyer, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Francisco Rodrıéguez-Henrıéquez, Sina Schaeffler, and Benjamin Wesolowski. *SQIsign: Algorithm specifications and supporting documentation*. National Institute of Standards and Technology. 2023. URL: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/sqisign-spec-web.pdf (cit. on pp. 32, 314–316, 322, 323, 336, 344, 349, 351).

[90] Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. "The SQALE of CSIDH: sublinear Vélu quantum-resistant isogeny action with low exponents". In: 12.3 (Sept. 2022), pp. 349–368. DOI: 10.1007/s13389-021-00271-w (cit. on pp. 240, 241, 243, 250–254, 263–265, 269).

[91] Jorge Chávez-Saab, Jesús-Javier Chi-Domiénguez, Samuel Jaques, and Francisco Rodrıéguez-Henrıéquez. "The SQALE of CSIDH: Square-root vélu Quantum-resistant isogeny Action with Low Exponents". In: *IACR*

Cryptol. ePrint Arch. 2020 (2020), p. 1520. URL: https://eprint.iacr.org/2020/1520 (cit. on pp. 137, 139–141, 150, 177, 197, 198).

[92] Jorge Chávez-Saab, Jesús-Javier Chi-Domıénguez, Samuel Jaques, and Francisco Rodrıéguez-Henrıéquez. "The SQALE of CSIDH: sublinear Vélu quantum-resistant isogeny action with low exponents". In: *Journal of Cryptographic Engineering* (2021). DOI: 10.1007/s13389-021-00271-w (cit. on p. 229).

[93] Jesús-Javier Chi-Domínguez and Krijn Reijnders. "Fully Projective Radical Isogenies in Constant-Time". In: 2022, pp. 73–95. DOI: 10.1007/978-3-030-95312-6_4 (cit. on pp. 243, 248, 249).

[94] Jesús-Javier Chi-Domínguez and Francisco Rodríguez-Henríquez. *Optimal strategies for CSIDH*. Cryptology ePrint Archive, Report 2020/417. 2020. URL: https://eprint.iacr.org/2020/417 (cit. on pp. 257, 258).

[95] Jesús-Javier Chi-Domıénguez and Francisco Rodrıéguez-Henrıéquez. "Optimal strategies for CSIDH". In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 417 (cit. on pp. 216, 219, 228, 234, 235).

[96] Jesús-Javier Chi-Domıénguez and Francisco Rodrıéguez-Henrıéquez. "Optimal strategies for CSIDH". In: *Adv. Math. Commun.* 16.2 (2022), pp. 383–411. DOI: 10.3934/amc.2020116. URL: https://doi.org/10.3934/amc.2020116 (cit. on p. 342).

[97] Andrew M. Childs, David Jao, and Vladimir Soukharev. "Constructing elliptic curve isogenies in quantum subexponential time". In: *J. Mathematical Cryptology* 8.1 (2014), pp. 1–29. DOI: 10.1515/jmc-2012-0016. URL: https://arxiv.org/abs/1012.4019 (cit. on p. 173).

[98] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of elliptic and hyperelliptic curve cryptography*. CRC press, 2005 (cit. on p. 37).

[99] John H. Conway and Neil J. A. Sloane. "Low dimensional lattices VII: Coordination sequences". In: *Proceedings of the Royal Society of London, Series A 453*. 1997, pp. 2369–2389 (cit. on p. 199).

[100] Maria Corte-Real Santos, Craig Costello, and Benjamin Smith. *Efficient (3,3)-isogenies on fast Kummer surfaces*. Cryptology ePrint Archive, Paper 2024/144. https://eprint.iacr.org/2024/144. 2024. URL: https://eprint.iacr.org/2024/144 (cit. on p. 34).

[101] Craig Costello. "B-SIDH: Supersingular Isogeny Diffie-Hellman Using Twisted Torsion". In: *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. Lecture Notes in Computer Science. Springer, 2020, pp. 440–463. DOI: 10.1007/978-3-030-64834-3_15. URL: https://doi.org/10.1007/978-3-030-64834-3%5C_15 (cit. on pp. 314, 323).

[102] Craig Costello. "Computing Supersingular Isogenies on Kummer Surfaces". In: *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*. Ed. by Thomas Peyrin and Steven D. Galbraith. Vol. 11274. Lecture Notes in Computer Science. Springer, 2018, pp. 428–456. DOI: 10.1007/978-3-030-03332-3_16. URL: https://doi.org/10.1007/978-3-030-03332-3%5C_16 (cit. on p. 340).

[103] Craig Costello. *Pairings for beginners*. 2015. URL: https://www.craigcostello.com.au/ (cit. on pp. 283, 284).

[104] Craig Costello and Hüseyin Hisil. "A Simple and Compact Algorithm for SIDH with Arbitrary Degree Isogenies". In: *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. Lecture Notes in Computer Science. Springer, 2017, pp. 303–329. DOI: 10.1007/978-3-319-70697-9_11. URL: https://doi.org/10.1007/978-3-319-70697-9%5C_11 (cit. on p. 337).

[105] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. "Efficient Compression of SIDH Public Keys". In: 2017, pp. 679–706. DOI: 10.1007/978-3-319-56620-7_24 (cit. on p. 262).

[106] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. "Efficient Compression of SIDH Public Keys". In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10210. Lecture Notes

in Computer Science. Springer. 2017, pp. 679–706. DOI: 10.1007/978-3-319-56620-7_24 (cit. on pp. 278, 288, 289, 294, 299).

[107] Craig Costello, Patrick Longa, and Michael Naehrig. "Efficient Algorithms for Supersingular Isogeny Diffie-Hellman". In: 2016, pp. 572–601. DOI: 10.1007/978-3-662-53018-4_21 (cit. on pp. 243, 278, 288, 294).

[108] Craig Costello, Patrick Longa, and Michael Naehrig. "Efficient Algorithms for Supersingular Isogeny Diffie-Hellman". In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Springer, 2016, pp. 572–601. DOI: 10.1007/978-3-662-53018-4_21. URL: https://doi.org/10.1007/978-3-662-53018-4%5C_21 (cit. on pp. 136, 158).

[109] Craig Costello, Michael Meyer, and Michael Naehrig. "Sieving for Twin Smooth Integers with Solutions to the Prouhet-Tarry-Escott Problem". In: *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. Lecture Notes in Computer Science. Springer, 2021, pp. 272–301. DOI: 10.1007/978-3-030-77870-5_10. URL: https://doi.org/10.1007/978-3-030-77870-5%5C_10 (cit. on pp. 314, 323).

[110] Craig Costello and Benjamin Smith. "Montgomery curves and their arithmetic - The case of large characteristic fields". In: 8.3 (Sept. 2018), pp. 227–240. DOI: 10.1007/s13389-017-0157-6 (cit. on p. 280).

[111] Nicolas Courtois, Er Klimov, Jacques Patarin, and Adi Shamir. "Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations". In: *In Advances in Cryptology, Eurocrypt'00, LNCS 1807*. Ed. by Bart Preneel. Vol. 1807. EUROCRYPT'00. Bruges, Belgium: Springer-Verlag, 2000, pp. 392–407. ISBN: 3-540-67517-5 (cit. on p. 107).

[112] Nicolas Tadeusz Courtois. "Efficient zero-knowledge authentication based on a linear algebra problem MinRank". In: *Advances in Cryptology – ASIACRYPT 2001*. Vol. 2248. LNCS. Springer, 2001, pp. 402–421 (cit. on p. 109).

[113] Jean-Marc Couveignes. *Hard Homogeneous Spaces*. Cryptology ePrint Archive, Paper 2006/291. http://eprint.iacr.org/2006/291. 2006. URL: https://ia.cr/2006/291 (cit. on p. 86).

[114] Alain Couvreur, Thomas Debris-Alazard, and Philippe Gaborit. *On the hardness of code equivalence problems in rank metric*. 2021. arXiv: 2011.04611 [cs.IT]. URL: https://arxiv.org/abs/2011.04611 (cit. on pp. 20, 46, 49, 51, 52, 61, 65, 84, 92, 103).

[115] Ronald Cramer. "Modular design of secure yet practical cryptographic protocols". In: *Ph. D.-thesis, CWI and U. of Amsterdam* 2 (1996) (cit. on p. 12).

[116] Pierrick Dartois, Antonin Leroux, Damien Robert, and Benjamin Wesolowski. "SQISignHD: new dimensions in cryptography". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2024, pp. 3–32 (cit. on pp. 37, 316, 322).

[117] Luca De Feo and Steven D. Galbraith. "SeaSign: Compact Isogeny Signatures from Class Group Actions". In: *Advances in Cryptology – EUROCRYPT 2019*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11478. LNCS. Cham: Springer International Publishing, 2019, pp. 759–789. ISBN: 978-3-030-17659-4 (cit. on pp. 45, 84, 99, 100).

[118] Luca De Feo and Steven D. Galbraith. "SeaSign: Compact Isogeny Signatures from Class Group Actions". In: *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11478. Lecture Notes in Computer Science. Springer, 2019, pp. 759–789. DOI: 10.1007/978-3-030-17659-4_26. URL: https://ia.cr/2018/824 (cit. on p. 171).

[119] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. "SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies". In: *Advances in Cryptology – ASIACRYPT 2020*. Ed. by Shiho Moriai and Huaxiong Wang. Cham: Springer International Publishing, 2020, pp. 64–93. ISBN: 978-3-030-64837-4 (cit. on p. 45).

[120] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. "SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies". In: 2020, pp. 64–93. DOI: 10.1007/978-3-030-64837-4_3 (cit. on pp. 277, 279).

[121] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. "SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies". In: *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. Lecture Notes in Computer Science. Springer, 2020, pp. 64–93. DOI: 10.1007/978-3-030-64837-4_3. URL: https://doi.org/10.1007/978-3-030-64837-4%5C_3 (cit. on pp. 32, 313–316, 320, 322, 323, 344).

[122] Luca De Feo, Antonin Leroux, Patrick Longa, and Benjamin Wesolowski. "New Algorithms for the Deuring Correspondence - Towards Practical and Secure SQISign Signatures". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. Lecture Notes in Computer Science. Springer, 2023, pp. 659–690. DOI: 10.1007/978-3-031-30589-4_23. URL: https://doi.org/10.1007/978-3-031-30589-4%5C_23 (cit. on pp. 32, 316, 322, 323, 327, 331, 344).

[123] Luca De Feo and Michael Meyer. "Threshold Schemes from Isogeny Assumptions". In: *Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II*. Ed. by Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas. Vol. 12111. Lecture Notes in Computer Science. Springer, 2020, pp. 187–212. DOI: 10.1007/978-3-030-45388-6_7. URL: https://ia.cr/2019/1288 (cit. on p. 171).

[124] Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. "Wave: A New Family of Trapdoor One-Way Preimage Sampleable Functions Based on Codes". In: *Advances in Cryptology – ASIACRYPT 2019*. Ed. by Steven D. Galbraith and Shiho Moriai. Cham: Springer International Publishing, 2019, pp. 21–51. ISBN: 978-3-030-34578-5 (cit. on p. 45).

[125] Christina Delfs and Steven D. Galbraith. "Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$". In: *Des. Codes Cryptography* 78.2 (2016), pp. 425–440. DOI: 10.1007/s10623-014-0010-1. URL: https://arxiv.org/abs/1310.7789 (cit. on p. 173).

[126] Christina Delfs and Steven D. Galbraith. "Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$". In: *Designs, Codes and Cryptography* 78 (2016), pp. 425–440 (cit. on p. 318).

[127] Max Deuring. "Die typen der multiplikatorenringe elliptischer funktionenkörper". In: *Abhandlungen aus dem mathematischen Seminar der Universität Hamburg*. Vol. 14. 1. Springer Berlin/Heidelberg. 1941, pp. 197–272 (cit. on p. 30).

[128] Whitfield Diffie and Martin E Hellman. "New directions in cryptography". In: *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*. 2022, pp. 365–390 (cit. on p. 8).

[129] Jintai Ding, Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, Bo-Yin Yang, Matthias Kannwischer, and Jacques Patarin. *Rainbow*. Tech. rep. National Institute of Standards and Technology, 2020. URL: https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions (cit. on p. 100).

[130] Jintai Ding and Dieter Schmidt. "Rainbow, a New Multivariable Polynomial Signature Scheme". In: *ACNS*. Ed. by John Ioannidis, Angelos D. Keromytis, and Moti Yung. Vol. 3531. Lecture Notes in Computer Science. 2005, pp. 164–175. ISBN: 3-540-26223-7 (cit. on p. 45).

[131] Javad Doliskani. "On division polynomial PIT and supersingularity". In: *Applicable Algebra in Engineering, Communication and Computing* 29.5 (2018), pp. 393–407 (cit. on pp. 263, 296, 299).

[132] Vivien Dubois, Louis Granboulan, and Jacques Stern. "An efficient provable distinguisher for HFE". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2006, pp. 156–167 (cit. on pp. 53, 69).

[133] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. "Crystals-dilithium: A lattice-based digital signature scheme". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), pp. 238–268 (cit. on pp. 83, 119, 125).

[134] Léo Ducas and Wessel van Woerden. "On the Lattice Isomorphism Problem, Quadratic Forms, Remarkable Lattices, and Cryptography". In: *Advances in Cryptology – EUROCRYPT 2022*. Ed. by Orr Dunkelman and Stefan Dziembowski. Cham: Springer International Publishing, 2022, pp. 643–673. ISBN: 978-3-031-07082-2 (cit. on p. 45).

[135] Jonathan Komada Eriksen, Lorenz Panny, Jana Sotáková, and Mattia Veroni. "Deuring for the People: Supersingular Elliptic Curves with Prescribed Endomorphism Ring in General Characteristic". In: *IACR Cryptol. ePrint Arch.* (2023), p. 106. URL: https://eprint.iacr.org/2023/106 (cit. on pp. 316, 325).

[136] Jean-Charles Faugère, Mohab Safey El Din, and Pierre-Jean Spaenlehauer. "Gröbner bases of bihomogeneous ideals generated by polynomials of bidegree (1, 1): Algorithms and complexity". In: *J. Symb. Comput.* 46.4 (2011), pp. 406–437 (cit. on p. 107).

[137] Jean-Charles Faugère, Françoise Levy-dit-Vehel, and Ludovic Perret. "Cryptanalysis of MinRank". In: *CRYPTO*. Ed. by David A. Wagner. Vol. 5157. Lecture Notes in Computer Science. Springer, 2008, pp. 280–296. ISBN: 978-3-540-85173-8 (cit. on pp. 84, 107, 109).

[138] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, Frédéric de Portzamparc, and Jean-Pierre Tillich. "Folding Alternant and Goppa Codes With Non-Trivial Automorphism Groups". In: *IEEE Trans. Inf. Theory* 62.1 (2016), pp. 184–198. DOI: 10.1109/TIT.2015.2493539. URL: https://doi.org/10.1109/TIT.2015.2493539 (cit. on p. 58).

[139] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, Frédéric Portzamparc, and Jean-Pierre Tillich. "Structural Cryptanalysis of McEliece Schemes with Compact Keys". In: *Des. Codes Cryptography* 79.1 (Apr. 2016), pp. 87–112. ISSN: 0925-1022. DOI: 10.1007/s10623-015-0036-z. URL: https://doi.org/10.1007/s10623-015-0036-z (cit. on p. 58).

[140] Jean-Charles Faugère and Ludovic Perret. "Polynomial Equivalence Problems: Algorithmic and Theoretical Aspects". In: *EUROCRYPT '06*. Ed. by Serge Vaudenay. Vol. 4004. Lecture Notes in Computer Science. Springer, July 5, 2006, pp. 30–47. ISBN: 3-540-34546-9 (cit. on pp. 53, 54, 65, 67, 75–77).

[141] Joël Felderhoff. *Hard Homogenous Spaces and Commutative Supersingular Isogeny based Diffie-Hellman*. Internship report. LIX, Ecole polytechnique and ENS de Lyon, Aug. 2019. URL: https://hal.archives-ouvertes.fr/hal-02373179 (cit. on p. 91).

[142] Luca De Feo, Tako Boris Fouotsa, Péter Kutas, Antonin Leroux, Simon-Philipp Merz, Lorenz Panny, and Benjamin Wesolowski. "SCALLOP: scaling the CSI-FiSh". In: *IACR International Conference on Public-Key Cryptography*. Springer. 2023, pp. 345–375 (cit. on p. 9).

[143] Luca De Feo, Nadia El Mrabet, Aymeric Genêt, Novak Kaluderovic, Natacha Linard de Guertechin, Simon Pontié, and Élise Tasso. "SIKE Channels". In: *IACR Cryptol. ePrint Arch.* (2022), p. 54. URL: https://eprint.iacr.org/2022/054 (cit. on pp. 135–137, 145, 146, 159, 161, 164).

[144] Amos Fiat and Adi Shamir. "How to prove yourself: Practical solutions to identification and signature problems". In: *Conference on the theory and application of cryptographic techniques*. Ed. by Andrew M. Odlyzko. Vol. 263. LNCS. Springer. Santa Barbara, California, United States: Springer-Verlag, 1986, pp. 186–194 (cit. on p. 12).

[145] Pierre-Alain Fouque, Louis Granboulan, and Jacques Stern. "Differential Cryptanalysis for Multivariate Schemes". In: *Advances in Cryptology – EUROCRYPT 2005*. Ed. by Ronald Cramer. Vol. 3494. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 341–353. ISBN: 978-3-540-25910-7 (cit. on p. 53).

[146] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, Zhenfei Zhang, et al. "Falcon: Fast-Fourier lattice-based compact signatures over NTRU". In: *Submission to the NIST's post-quantum cryptography standardization process* 36.5 (2018), pp. 1–75 (cit. on pp. 83, 119, 125).

[147] Vyacheslav Futorny, Joshua A. Grochow, and Vladimir V. Sergeichuk. "Wildness for tensors". In: *Linear Algebra and its Applications* 566 (2019), pp. 212–244. ISSN: 0024-3795. DOI: https://doi.org/10.1016/j.laa.2018.12.022. URL: https://www.sciencedirect.com/science/article/pii/S0024379518305937 (cit. on p. 47).

[148] Phillip Gajland, Bor de Kock, Miguel Quaresma, Giulio Malavolta, and Peter Schwabe. *Swoosh: Efficient Lattice-Based Non-Interactive Key Exchange*. 2024 (cit. on p. 9).

[149] Phillip Gajland, Bor de Kock, Miguel Quaresma, Giulio Malavolta, and Peter Schwabe. *Swoosh: Practical Lattice-Based Non-Interactive Key Exchange*. Cryptology ePrint Archive, Report 2023/271. 2023. URL: https://eprint.iacr.org/2023/271 (cit. on pp. 240, 243, 274).

[150] Steven D Galbraith, Florian Hess, and Frederik Vercauteren. "Hyperelliptic pairings". In: *International Conference on Pairing-Based Cryptography*. Springer. 2007, pp. 108–131 (cit. on p. 37).

[151] Steven D Galbraith and Xibin Lin. "Computing pairings using x-coordinates only". In: *Designs, Codes and Cryptography* 50.3 (2009), pp. 305–324 (cit. on p. 288).

[152] Steven D. Galbraith. *Advances in Elliptic Curve Cryptography, Chapter IX*. Ed. by Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart. Vol. 317. Cambridge University Press, 2005 (cit. on pp. 37–39, 340).

[153] Steven D. Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, 2012 (cit. on pp. 7, 21, 283).

[154] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. "On the Security of Supersingular Isogeny Cryptosystems". In: 2016, pp. 63–91. DOI: 10.1007/978-3-662-53887-6_3 (cit. on p. 243).

[155] Theodoulos Garefalakis. "The generalized Weil pairing and the discrete logarithm problem on elliptic curves". In: *LATIN 2002: Theoretical Informatics: 5th Latin American Symposium Cancun, Mexico, April 3–6, 2002 Proceedings 5*. Springer. 2002, pp. 118–130 (cit. on p. 37).

[156] Alexandre Gélin and Benjamin Wesolowski. "Loop-Abort Faults on Supersingular Isogeny Cryptosystems". In: *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*. Ed. by Tanja Lange and Tsuyoshi Takagi. Vol. 10346. Lecture Notes in Computer Science. Springer, 2017, pp. 93–106. DOI: 10.1007/978-3-319-59879-6_6. URL: https://doi.org/10.1007/978-3-319-59879-6%5C_6 (cit. on p. 137).

[157] Aymeric Genêt, Natacha Linard de Guertechin, and Novak Kaluderovic. "Full Key Recovery Side-Channel Attack Against Ephemeral SIKE on the Cortex-M4". In: *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings*. Ed. by Shivam Bhasin and Fabrizio De Santis. Vol. 12910. Lecture Notes in Computer Science. Springer, 2021, pp. 228–254. URL: https://doi.org/10.1007/978-3-030-89915-8%5C_11 (cit. on p. 137).

[158] Marc Girault. "A (Non-Practical) Three-Pass Identification Protocol Using Coding Theory". In: *Proceedings of the International Conference on Cryptology on Advances in Cryptology*. AUSCRYPT '90. Sydney, Australia: Springer-Verlag, 1990, pp. 265–272. ISBN: 0387530002 (cit. on p. 45).

[159] Elisa Gorla. “Rank-metric codes”. In: *CoRR* abs/1902.02650 (2019). arXiv: 1902.02650. URL: http://arxiv.org/abs/1902.02650 (cit. on p. 17).

[160] Elisa Gorla and Flavio Salizzoni. “MacWilliams’ Extension Theorem for rank-metric codes”. In: *Journal of Symbolic Computation* 122 (2024), p. 102263 (cit. on p. 18).

[161] Louis Goubin. “A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems”. In: *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*. Ed. by Yvo Desmedt. Vol. 2567. Lecture Notes in Computer Science. Springer, 2003, pp. 199–210. URL: https://doi.org/10.1007/3-540-36288-6%5C_15 (cit. on p. 137).

[162] Torbjörn Granlund and Peter L. Montgomery. “Division by Invariant Integers Using Multiplication”. In: *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation*. PLDI ’94. Orlando, Florida, USA: Association for Computing Machinery, 1994, pp. 61–72 (cit. on p. 113).

[163] Joshua A. Grochow and Youming Qiao. *Isomorphism problems for tensors, groups, and cubic forms: completeness and reductions*. 2019. DOI: 10.48550/ARXIV.1907.00309. URL: https://arxiv.org/abs/1907.00309 (cit. on pp. 46, 47).

[164] Shay Gueron, Edoardo Persichetti, and Paolo Santini. “Designing a Practical Code-Based Signature Scheme from Zero-Knowledge Proofs with Trusted Setup”. In: *Cryptography* 6.1 (2022), p. 5 (cit. on p. 102).

[165] Aurore Guillevic. “Comparing the Pairing Efficiency over Composite-Order and Prime-Order Elliptic Curves”. In: 2013, pp. 357–372. DOI: 10.1007/978-3-642-38980-1_22 (cit. on p. 278).

[166] Mike Hamburg. *Computing the Jacobi symbol using Bernstein-Yang*. Cryptology ePrint Archive, Report 2021/1271. 2021. URL: https://eprint.iacr.org/2021/1271 (cit. on p. 264).

[167] Ryuichi Harasawa, Junji Shikata, Joe Suzuki, and Hideki Imai. “Comparing the MOV and FR reductions in elliptic curve cryptography”. In: *Advances in Cryptology—EUROCRYPT’99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*. Springer. 1999, pp. 190–205 (cit. on p. 278).

[168] Ishay Haviv and Oded Regev. “On the lattice isomorphism problem”. In: *SODA 2014*. Ed. by Chandra Chekuri. ACM SIAM, 2014, pp. 391–404 (cit. on p. 84).

[169] Loo-Keng Hua. “A theorem on matrices over a sfield and its applications”. In: *Bulletin of the American Mathematical Society*. Vol. 55. 11. 1949, pp. 1046–1046 (cit. on p. 50).

[170] Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. *SPHINCS+*. NIST PQC Submission. 2020 (cit. on pp. 83, 119, 125).

[171] Dale Husemöller. *Elliptic Curves, 2nd edition*. Springer, 2004 (cit. on pp. 39, 339).

[172] Aaron Hutchinson, Jason T. LeGrow, Brian Koziel, and Reza Azarderakhsh. “Further Optimizations of CSIDH: A Systematic Approach to Efficient Strategies, Permutations, and Bound Vectors”. In: 2020, pp. 481–501. DOI: 10.1007/978-3-030-57808-4_24 (cit. on p. 243).

[173] Aaron Hutchinson, Jason T. LeGrow, Brian Koziel, and Reza Azarderakhsh. “Further Optimizations of CSIDH: A Systematic Approach to Efficient Strategies, Permutations, and Bound Vectors”. In: *Applied Cryptography and Network Security – 18th International Conference, ACNS 2020, Rome, Italy, October 19-22, 2020, Proceedings, Part I*. Ed. by Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi. Vol. 12146. Lecture Notes in Computer Science. https://ia.cr/2019/1121. Springer. Springer, 2020, pp. 481–501. DOI: 10.1007/978-3-030-57808-4_24. URL: https://ia.cr/2019/1121 (cit. on p. 172).

[174] Tetsuya Izu and Tsuyoshi Takagi. “Exceptional Procedure Attack on Elliptic Curve Cryptosystems”. In: *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*. Ed. by Yvo Desmedt. Vol. 2567. Lecture Notes in Computer Science. Springer, 2003, pp. 224–239. URL: https://doi.org/10.1007/3-540-36288-6%5C_17 (cit. on p. 137).

[175] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, and Geovandro Pereira. *SIKE–Supersingular Isogeny Key Encapsulation*. https://sike.org/. 2017 (cit. on pp. 135–137).

[176] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hutchinson. *SIKE*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions. National Institute of Standards and Technology, 2022 (cit. on pp. 33, 336, 337, 342).

[177] David Jao and Luca De Feo. "Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies". In: 2011, pp. 19–34. DOI: 10.1007/978-3-642-25405-5_2 (cit. on p. 243).

[178] David Jao and Luca De Feo. "Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies". In: *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*. Ed. by Bo-Yin Yang. Vol. 7071. Lecture Notes in Computer Science. Springer, 2011, pp. 19–34. DOI: 10.1007/978-3-642-25405-5_2. URL: https://doi.org/10.1007/978-3-642-25405-5%5C_2 (cit. on p. 135).

[179] Don Johnson, Alfred Menezes, and Scott A. Vanstone. "The Elliptic Curve Digital Signature Algorithm (ECDSA)". In: *Int. J. Inf. Sec.* 1.1 (2001), pp. 36–63. DOI: 10.1007/s102070100002. URL: https://doi.org/10.1007/s102070100002 (cit. on p. 314).

[180] Simon Josefsson and Ilari Liusvaara. "Edwards-Curve Digital Signature Algorithm (EdDSA)". In: *RFC* 8032 (2017), pp. 1–60. DOI: 10.17487/RFC8032. URL: https://doi.org/10.17487/RFC8032 (cit. on p. 314).

[181] Antoine Joux. "A one round protocol for tripartite Diffie–Hellman". In: *Journal of cryptology* 17.4 (2004), pp. 263–276 (cit. on p. 278).

[182] Marc Joye and Jean-Jacques Quisquater. "On the Importance of Securing Your Bins: The Garbage-man-in-the-middle Attack". In: 1997, pp. 135–141. DOI: 10.1145/266420.266449 (cit. on p. 285).

[183] Marc Joye and Sung-Ming Yen. "The Montgomery Powering Ladder". In: 2003, pp. 291–302. DOI: 10.1007/3-540-36400-5_22 (cit. on p. 285).

[184] Ernst Kani. “The number of curves of genus two with elliptic differentials.” In: (1997) (cit. on pp. 33, 34).

[185] Yutaro Kiyomura and Tsuyoshi Takagi. “Efficient algorithm for Tate pairing of composite order”. In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 97.10 (2014), pp. 2055–2063 (cit. on pp. 290, 292).

[186] Tetsutaro Kobayashi, Kazumaro Aoki, and Hideki Imai. “Efficient algorithms for Tate pairing”. In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 89.1 (2006), pp. 134–143 (cit. on pp. 290, 292).

[187] Neal Koblitz. “Elliptic curve cryptosystems”. In: *Mathematics of computation* 48.177 (1987), pp. 203–209 (cit. on p. 22).

[188] Neal Koblitz. “Hyperelliptic cryptosystems”. In: *Journal of cryptology* 1 (1989), pp. 139–150 (cit. on p. 23).

[189] Bor de Kock. “A non-interactive key exchange based on ring-learning with errors”. PhD thesis. Master’s thesis. Master’s thesis, Eindhoven University of Technology, 2018 (cit. on pp. 9, 243).

[190] David Kohel, Kristin Lauter, Christophe Petit, and Jean-Pierre Tignol. “On the quaternion-isogeny path problem”. In: *LMS Journal of Computation and Mathematics* 17.A (2014), pp. 418–432 (cit. on pp. 31, 320).

[191] Brian Koziel, Reza Azarderakhsh, and David Jao. “Side-Channel Attacks on Quantum-Resistant Supersingular Isogeny Diffie-Hellman”. In: *Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers*. Ed. by Carlisle Adams and Jan Camenisch. Vol. 10719. Lecture Notes in Computer Science. Springer, 2017, pp. 64–81. URL: https://doi.org/10.1007/978-3-319-72565-9%5C_4 (cit. on pp. 136, 137, 165, 166).

[192] Greg Kuperberg. “Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem”. In: *TQC 2013*. Ed. by Simone Severini and Fernando G. S. L. Brandão. Vol. 22. LIPIcs. Schloss Dagstuhl, 2013, pp. 20–34 (cit. on p. 92).

[193] Yi-Fu Lai, Steven D. Galbraith, and Cyprien Delpech de Saint Guilhem. "Compact, Efficient and UC-Secure Isogeny-Based Oblivious Transfer". In: *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. Lecture Notes in Computer Science. Springer, 2021, pp. 213–241. DOI: 10.1007/978-3-030-77870-5_8. URL: https://ia.cr/2020/1012 (cit. on p. 171).

[194] Georg Landsberg. "Ueber eine Anzahlbestimmung und eine damit zusammenhängende Reihe." In: (1893) (cit. on p. 69).

[195] Jason LeGrow and Aaron Hutchinson. *An Analysis of Fault Attacks on CSIDH*. Cryptology ePrint Archive, Report 2020/1006. 2020. URL: https://eprint.iacr.org/2020/1006 (cit. on p. 240).

[196] Jason T. LeGrow and Aaron Hutchinson. "(Short Paper) Analysis of a Strong Fault Attack on Static/Ephemeral CSIDH". In: *Advances in Information and Computer Security - 16th International Workshop on Security, IWSEC 2021, Virtual Event, September 8-10, 2021, Proceedings*. Ed. by Toru Nakanishi and Ryo Nojima. Vol. 12835. Lecture Notes in Computer Science. Springer. Springer, 2021, pp. 216–226. DOI: 10.1007/978-3-030-85987-9_12. URL: https://doi.org/10.1007/978-3-030-85987-9%5C_12 (cit. on pp. 137, 165, 174, 204, 205).

[197] Jeffrey Leon. "Computing automorphism groups of error-correcting codes". In: *IEEE Transactions on Information Theory* 28.3 (1982), pp. 496–511 (cit. on p. 46).

[198] Kaizhan Lin, Weize Wang, Zheng Xu, and Chang-An Zhao. *A Faster Software Implementation of SQISign*. Cryptology ePrint Archive, Paper 2023/753. https://eprint.iacr.org/2023/753. 2023. URL: https://eprint.iacr.org/2023/753 (cit. on pp. 315, 316, 339–341, 343, 347, 351).

[199] David Lubicz and Damien Robert. "A generalisation of Miller's algorithm and applications to pairing computations on abelian varieties". In: *Journal of Symbolic Computation* 67 (2015), pp. 68–92 (cit. on pp. 283, 302).

[200] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. *CRYSTALS-DILITHIUM*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-

2022. National Institute of Standards and Technology, 2022 (cit. on p. 242).

[201] Florence Jessie MacWilliams. "Combinatorial problems of elementary abelian groups". PhD thesis. 1962 (cit. on p. 18).

[202] Luciano Maino and Chloe Martindale. *An attack on SIDH with arbitrary starting curve*. Cryptology ePrint Archive, Paper 2022/1026. 2022. URL: https://ia.cr/2022/1026 (cit. on p. 173).

[203] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. "A Direct Key Recovery Attack on SIDH". In: 2023, pp. 448–471. DOI: 10.1007/978-3-031-30589-4_16 (cit. on p. 243).

[204] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. "A Direct Key Recovery Attack on SIDH". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. Lecture Notes in Computer Science. Springer, 2023, pp. 448–471. DOI: 10.1007/978-3-031-30589-4_16. URL: https://doi.org/10.1007/978-3-031-30589-4%5C_16 (cit. on p. 33).

[205] Moxie Marlinspike and Trevor Perrin. *The X3DH Key Agreement Protocol*. Signal Specifications. https://signal.org/docs/specifications/x3dh/(accessed2022-01-04). 2016 (cit. on p. 11).

[206] R. J. McEliece. "A Public-Key System Based on Algebraic Coding Theory". In: *Jet Propulsion Laboratory, California Institute of Technology* (1978). DSN Progress Report 44, pp. 114–116 (cit. on pp. 45, 83).

[207] Robert J McEliece. *Finite fields for computer scientists and engineers*. Vol. 23. Springer Science & Business Media, 2012 (cit. on p. 285).

[208] Michael B. McLoughlinn. *addchain: Cryptographic Addition Chain Generation in Go*. Github Repository. Version 90573bb1d9cf. 2020. URL: https://github.com/mmcloughlin/addchain (cit. on p. 234).

[209] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Gilles Zemor, Alain Couvreur, and Adrien Hauteville. *RQC*. 2019. URL: https://csrc.nist.gov/projects/post-quantum-cryptography/%20round-2-submissions (cit. on p. 46).

[210] Ralph C Merkle. “Secure communications over insecure channels”. In: *Communications of the ACM* 21.4 (1978), pp. 294–299 (cit. on p. 8).

[211] Michael Meyer, Fabio Campos, and Steffen Reith. “On Lions and Elligators: An Efficient Constant-Time Implementation of CSIDH”. In: 2019, pp. 307–325. DOI: 10.1007/978-3-030-25510-7_17 (cit. on pp. 240, 242, 257, 258).

[212] Michael Meyer, Fabio Campos, and Steffen Reith. “On Lions and Elligators: An Efficient Constant-Time Implementation of CSIDH”. In: *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019, Chongqing, China, May 8-10, 2019 Revised Selected Papers*. Ed. by Jintai Ding and Rainer Steinwandt. Vol. 11505. Lecture Notes in Computer Science. Springer. Springer, 2019, pp. 307–325. DOI: 10.1007/978-3-030-25510-7_17. URL: https://doi.org/10.1007/978-3-030-25510-7%5C_17 (cit. on pp. 140, 172, 177, 197, 205).

[213] Michael Meyer and Steffen Reith. “A Faster Way to the CSIDH”. In: 2018, pp. 137–152. DOI: 10.1007/978-3-030-05378-9_8 (cit. on p. 242).

[214] Michael Meyer and Steffen Reith. “A Faster Way to the CSIDH”. In: *Progress in Cryptology - INDOCRYPT 2018 - 19th International Conference on Cryptology in India, New Delhi, India, December 9-12, 2018, Proceedings*. Ed. by Debrup Chakraborty and Tetsu Iwata. Vol. 11356. Lecture Notes in Computer Science. Springer. Springer, 2018, pp. 137–152. DOI: 10.1007/978-3-030-05378-9_8. URL: https://doi.org/10.1007/978-3-030-05378-9%5C_8 (cit. on p. 337).

[215] Victor S Miller. “The Weil pairing, and its efficient calculation”. In: *Journal of cryptology* 17.4 (2004), pp. 235–261 (cit. on p. 38).

[216] Victor S Miller. “Use of elliptic curves in cryptography”. In: *Conference on the theory and application of cryptographic techniques*. CRYPTO ’85. Springer. London, UK: Springer-Verlag, 1985, pp. 417–426. ISBN: 3-540-16463-4 (cit. on p. 22).

[217] Victor S. Miller. “The Weil Pairing, and Its Efficient Calculation”. In: 17.4 (Sept. 2004), pp. 235–261. DOI: 10.1007/s00145-004-0315-8 (cit. on p. 283).

[218] Peter L Montgomery. “Speeding the Pollard and elliptic curve methods of factorization”. In: *Mathematics of computation* 48.177 (1987), pp. 243–264 (cit. on p. 22).

[219] Peter L. Montgomery. “Modular multiplication without trial division”. In: *Mathematics of Computation* 44 (1985), pp. 519–521. ISSN: 0025–5718 (cit. on p. 113).

[220] Dustin Moody and Daniel Shumow. “Analogues of Vélu’s formulas for isogenies on alternate models of elliptic curves”. In: *Math. Comput.* 85.300 (2016), pp. 1929–1951. URL: https://doi.org/10.1090/mcom/3036 (cit. on p. 141).

[221] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. “Correlation-Enhanced Power Analysis Collision Attack”. In: *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*. Ed. by Stefan Mangard and François-Xavier Standaert. Vol. 6225. Lecture Notes in Computer Science. Springer, 2010, pp. 125–139. URL: https://doi.org/10.1007/978-3-642-15031-9%5C_9 (cit. on pp. 137, 159).

[222] Tomoki Moriya, Hiroshi Onuki, and Tsuyoshi Takagi. “How to Construct CSIDH on Edwards Curves”. In: 2020, pp. 512–537. DOI: 10.1007/978-3-030-40186-3_22 (cit. on p. 243).

[223] D. Mumford, C. Musili, M. Nori, E. Previato, M. Stillman, and H. Umemura. *Tata Lectures on Theta II: Jacobian theta functions and differential equations*. Modern Birkhäuser Classics. Birkhäuser Boston, 2012. ISBN: 9780817645786. URL: https://books.google.nl/books?id=xaNCAAAAQBAJ (cit. on p. 24).

[224] Kohei Nakagawa and Hiroshi Onuki. “QFESTA: Efficient algorithms and parameters for FESTA using quaternion algebras”. In: *Cryptology ePrint Archive* (2023) (cit. on p. 37).

[225] Kohei Nakagawa and Hiroshi Onuki. *SQIsign2D-East: A New Signature Scheme Using 2-dimensional Isogenies*. Cryptology ePrint Archive, Paper 2024/771. https://eprint.iacr.org/2024/771. 2024. URL: https://eprint.iacr.org/2024/771 (cit. on p. 37).

[226] Kohei Nakagawa, Hiroshi Onuki, Atsushi Takayasu, and Tsuyoshi Takagi. *L_1-Norm Ball for CSIDH: Optimal Strategy for Choosing the Secret Key Space*. Cryptology ePrint Archive, Report 2020/181. https://ia.cr/2020/181. 2020 (cit. on p. 219).

[227] Erick Nascimento and Lukasz Chmielewski. "Applying Horizontal Clustering Side-Channel Attacks on Embedded ECC Implementations". In: *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*. Ed. by Thomas Eisenbarth and Yannick Teglia. Vol. 10728. Lecture Notes in Computer Science. Springer, 2017, pp. 213–231. URL: https://doi.org/10.1007/978-3-319-75208-2%5C_13 (cit. on pp. 137, 159).

[228] National Institute for Standards and Technology. *NIST Workshop on Cybersecurity in a Post-Quantum World*. URL: http://www.nist.gov/itl/csd/ct/post-quantum-crypto-workshop-2015.cfm (cit. on p. 46).

[229] Alessandro Neri. "Twisted linearized Reed-Solomon codes: A skew polynomial framework". In: *arXiv preprint arXiv:2105.10451* (2021) (cit. on p. 62).

[230] P Nguyen and C Wolf. *International Workshop on Post-Quantum Cryptography*. 2006 (cit. on p. 83).

[231] H. Niederreiter. "Knapsack-type cryptosystems and algebraic coding theory." English. In: *Probl. Control Inf. Theory* 15 (1986), pp. 159–166 (cit. on p. 45).

[232] NIST. *Post-Quantum Cryptography Standardization*. URL: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography. 2017 (cit. on p. 83).

[233] Roberto W Nóbrega and Bartolomeu F Uchôa-Filho. "Multishot codes for network coding using rank-metric codes". In: *2010 Third IEEE International Workshop on Wireless Network Coding*. IEEE. 2010, pp. 1–6 (cit. on p. 62).

[234] Hiroshi Onuki. "On oriented supersingular elliptic curves". In: *Finite Fields and Their Applications* 69 (2021), p. 101777 (cit. on p. 29).

[235] Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. "(Short Paper) A Faster Constant-Time Algorithm of CSIDH Keeping Two Points". In: 2019, pp. 23–33. DOI: 10.1007/978-3-030-26834-3_2 (cit. on pp. 240, 242, 277, 281).

[236] Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. "(Short Paper) A Faster Constant-Time Algorithm of CSIDH Keeping Two Points". In: *Advances in Information and Computer Security - 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan, August 28-30, 2019, Proceedings*. Ed. by Nuttapong Attrapadung and Takeshi Yagi. Vol. 11689. Lecture Notes in Computer Science. Springer, 2019, pp. 23–33. DOI: 10.1007/978-3-030-26834-3_2. URL: https://doi.org/10.1007/978-3-030-26834-3%5C_2 (cit. on pp. 140, 177, 205, 257, 258).

[237] Hiroshi Onuki and Tomoki Moriya. "Radical Isogenies on Montgomery Curves". In: *IACR Cryptol. ePrint Arch.* 2021 (2021), p. 699. URL: https://eprint.iacr.org/2021/699 (cit. on pp. 220, 221).

[238] Aurel Page and Benjamin Wesolowski. "The supersingular Endomorphism Ring and One Endomorphism problems are equivalent". In: *CoRR* abs/2309.10432 (2023). DOI: 10.48550/arXiv.2309.10432. arXiv: 2309.10432. URL: https://doi.org/10.48550/arXiv.2309.10432 (cit. on p. 318).

[239] Jacques Patarin. "Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of asymmetric algorithms". In: *Advances in Cryptology – EUROCRYPT '96*. Ed. by Ueli M. Maurer. Vol. 1070. LNCS. Saragossa, Spain: Springer-Verlag, 1996, pp. 33–48. ISBN: 3-540-61186-X (cit. on pp. 83, 84).

[240] Jacques Patarin, Louis Goubin, and Nicolas Courtois. "Improved Algorithms for Isomorphisms of Polynomials". In: *EUROCRYPT '98*. Vol. 1403. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 1998, pp. 184–200 (cit. on pp. 49, 54).

[241] Chris Peikert. "He Gives C-Sieves on the CSIDH". In: *Advances in Cryptology – EUROCRYPT 2020 – 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. Lecture Notes in Computer Science. https://eprint.iacr.org/2019/725. Springer, 2020, pp. 463–492. ISBN: 978-3-030-45723-5. DOI: 10.1007/978-3-030-45724-2_16. URL: https://doi.org/10.1007/978-3-030-45724-2%5C_16 (cit. on p. 240).

[242] Ray Perlner and Daniel Smith-Tone. *Rainbow Band Separation is Better than we Thought*. Cryptology ePrint Archive, Paper 2020/702. 2020 (cit. on pp. 106–108).

[243] Christiane Peters. "Information-set decoding for linear codes over $\mathbb{F}_q$". In: *International Workshop on Post-Quantum Cryptography*. Springer. 2010, pp. 81–94 (cit. on p. 61).

[244] Eugene Prange. "The use of information sets in decoding cyclic codes". In: *IRE Trans. Information Theory* 8.5 (1962), pp. 5–9 (cit. on p. 104).

[245] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. *FALCON*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. National Institute of Standards and Technology, 2022 (cit. on p. 242).

[246] Dana Randall. *Efficient Generation of Random Nonsingular Matrices*. Tech. rep. UCB/CSD-91-658. EECS Department, UC Berkeley, 1991 (cit. on p. 114).

[247] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *Theory of computing*. Ed. by Harold N. Gabow and Ronald Fagin. ACM, 2005, pp. 84–93 (cit. on p. 83).

[248] Krijn Reijnders, Simona Samardjiska, and Monika Trimoska. *Hardness estimates of the Code Equivalence Problem in the Rank Metric*. **TODO:** FIX. 2022 (cit. on pp. 84, 92, 97, 103, 105, 106).

[249] George W Reitwiesner. "Binary arithmetic". In: *Advances in computers*. Vol. 1. Elsevier, 1960, pp. 231–308 (cit. on p. 289).

[250] Joost Renes and Benjamin Smith. "qDSA: small and secure digital signatures with curve-based Diffie–Hellman key pairs". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2017, pp. 273–302 (cit. on p. 353).

[251] Damien Robert. "Breaking SIDH in Polynomial Time". In: 2023, pp. 472–503. DOI: 10.1007/978-3-031-30589-4_17 (cit. on p. 243).

[252] Damien Robert. "Breaking SIDH in Polynomial Time". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. Lecture Notes in Computer Science. Springer, 2023, pp. 472–503. DOI: 10.1007/978-3-031-30589-4_17. URL: https://doi.org/10.1007/978-3-031-30589-4%5C_17 (cit. on p. 33).

[253] Damien Robert. “Evaluating isogenies in polylogarithmic time”. In: *IACR Cryptol. ePrint Arch.* (2022), p. 1068. URL: https://eprint.iacr.org/2022/1068 (cit. on pp. 35, 36).

[254] Damien Robert. *The geometric interpretation of the Tate pairing and its applications*. Cryptology ePrint Archive, Paper 2023/177. 2023. URL: https://eprint.iacr.org/2023/177 (cit. on p. 37).

[255] Georg Rosenhain. *Abhandlung über die Functionen zweier Variabler mit vier Perioden: welche die Inversen sind der ultra-elliptischen Integrale erster Klasse*. 65. W. Engelmann, 1895 (cit. on p. 23).

[256] Alexander Rostovtsev and Anton Stolbunov. *PUBLIC-KEY CRYPTOSYSTEM BASED ON ISOGENIES*. Cryptology ePrint Archive, Paper 2006/145. 2006. URL: https://ia.cr/2006/145 (cit. on p. 86).

[257] Tobias Schneider and Amir Moradi. “Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations”. In: *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*. Ed. by Tim Güneysu and Helena Handschuh. Vol. 9293. Lecture Notes in Computer Science. Springer, 2015, pp. 495–513. URL: https://doi.org/10.1007/978-3-662-48324-4%5C_25 (cit. on pp. 137, 159).

[258] Claus-Peter Schnorr. “Efficient identification and signatures for smart cards”. In: *Advances in Cryptology—CRYPTO’89 Proceedings 9*. Springer. 1990, pp. 239–252 (cit. on p. 11).

[259] Peter Schwabe, Douglas Stebila, and Thom Wiggers. “More Efficient Post-quantum KEMTLS with Pre-distributed Public Keys”. In: 2021, pp. 3–22. DOI: 10.1007/978-3-030-88418-5_1 (cit. on pp. 270, 272).

[260] Peter Schwabe, Douglas Stebila, and Thom Wiggers. “Post-Quantum TLS Without Handshake Signatures”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1461–1480. ISBN: 9781450370899. DOI: 10.1145/3372297.3423350. URL: https://doi.org/10.1145/3372297.3423350 (cit. on p. 10).

[261] Peter Schwabe, Douglas Stebila, and Thom Wiggers. “Post-Quantum TLS Without Handshake Signatures”. In: 2020, pp. 1461–1480. DOI: 10.1145/3372297.3423350 (cit. on pp. 242, 245, 270, 272).

[262] Michael Scott. “A note on the calculation of some functions in finite fields: Tricks of the trade”. In: *Cryptology ePrint Archive* (2020) (cit. on p. 353).

[263] Michael Scott. “Computing the Tate Pairing”. In: 2005, pp. 293–304. DOI: 10.1007/978-3-540-30574-3_20 (cit. on p. 288).

[264] Michael Scott. “Pairing implementation revisited”. In: *Cryptology ePrint Archive* (2019) (cit. on p. 292).

[265] Michael Scott. *Understanding the Tate pairing*. 2004. URL: http://www.computing.dcu.ie/~mike/tate.html (cit. on p. 283).

[266] Michael Scott and Paulo S. L. M. Barreto. “Compressed Pairings”. In: 2004, pp. 140–156. DOI: 10.1007/978-3-540-28628-8_9 (cit. on pp. 285, 286).

[267] Nicolas Sendrier. “Finding the permutation between equivalent linear codes: The support splitting algorithm”. In: *IEEE Trans. Inf. Theory* 46 (2000), pp. 1193–1203 (cit. on p. 46).

[268] V V Sergeichuk. “CLASSIFICATION PROBLEMS FOR SYSTEMS OF FORMS AND LINEAR MAPPINGS”. In: *Mathematics of the USSR-Izvestiya* 31.3 (June 1988), pp. 481–501. DOI: 10.1070/im1988v031n03abeh001086. URL: https://doi.org/10.1070/im1988v031n03abeh001086 (cit. on p. 58).

[269] Peter W. Shor. “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer”. In: *SIAM review* 41.2 (1999), pp. 303–332 (cit. on pp. 9, 25, 313).

[270] Victor Shoup. “Efficient computation of minimal polynomials in algebraic extensions of finite fields”. In: *Proceedings of the 1999 international symposium on Symbolic and algebraic computation*. 1999, pp. 53–58 (cit. on p. 325).

[271] Joseph H Silverman. “A survey of local and global pairings on elliptic curves and abelian varieties”. In: *Pairing-Based Cryptography-Pairing 2010: 4th International Conference, Yamanaka Hot Spring, Japan, December 2010. Proceedings 4*. Springer. 2010, pp. 377–396 (cit. on p. 37).

[272] Joseph H. Silverman. *The arithmetic of elliptic curves*. Vol. 106. Springer, 2009 (cit. on pp. 21, 286, 287, 317).

[273] National Institute of Standards and Technology (NIST). *Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process*. 2022. URL: https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf (cit. on p. 313).

[274] Katherine E Stange. "The Tate pairing via elliptic nets". In: *Pairing Based Cryptography*. Springer. 2007, pp. 329–348 (cit. on p. 283).

[275] Anton Stolbunov. "Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves". In: *Adv. in Math. of Comm.* 4.2 (2010), pp. 215–235 (cit. on p. 213).

[276] Andrew V Sutherland. "Identifying supersingular elliptic curves". In: *LMS Journal of Computation and Mathematics* 15 (2012), pp. 317–325 (cit. on p. 296).

[277] Gang Tang, Dung Hoang Duong, Antoine Joux, Thomas Plantard, Youming Qiao, and Willy Susilo. "Practical Post-Quantum Signature Schemes from Isomorphism Problems of Trilinear Forms". In: *Advances in Cryptology – EUROCRYPT 2022*. Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13277. LNCS. Cham: Springer International Publishing, 2022, pp. 582–612. ISBN: 978-3-031-07082-2 (cit. on p. 45).

[278] Élise Tasso, Luca De Feo, Nadia El Mrabet, and Simon Pontié. "Resistance of Isogeny-Based Cryptographic Implementations to a Fault Attack". In: *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings*. Ed. by Shivam Bhasin and Fabrizio De Santis. Vol. 12910. Lecture Notes in Computer Science. Springer, 2021, pp. 255–276. DOI: 10.1007/978-3-030-89915-8_12. URL: https://doi.org/10.1007/978-3-030-89915-8%5C_12 (cit. on pp. 137, 174).

[279] John Tate. "Endomorphisms of abelian varieties over finite fields". In: *Inventiones mathematicae* 2.2 (1966), pp. 134–144 (cit. on p. 26).

[280] "Theorie der quadratischen Formen in beliebigen Korpern". In: *J. Reine Angew. Math.* 176 (1937), pp. 31–44 (cit. on p. 55).

[281] Yan Bo Ti. "Fault Attack on Supersingular Isogeny Cryptosystems". In: *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*. Ed. by Tanja Lange and Tsuyoshi Takagi. Vol. 10346. Lecture Notes in Computer Science. Springer, 2017, pp. 107–122. DOI: 10.1007/978-3-319-

59879-6_7. URL: https://doi.org/10.1007/978-3-319-59879-6%5C_7 (cit. on pp. 137, 174).

[282] Kiminori Tsukazaki. "Explicit isogenies of elliptic curves". PhD thesis. University of Warwick, 2013 (cit. on p. 325).

[283] Aleksei Udovenko and Giuseppe Vitto. *Breaking the $IKEp182 Challenge*. IACR Cryptology ePrint Archive 2021/1421. 2021. URL: https://ia.cr/2021/1421 (cit. on p. 203).

[284] "Untersuchungen über quadratische Formen in Korpern der Charakteristik 2, I". In: *J. Reine Angew. Math.* 183 (1941), pp. 148–167 (cit. on p. 55).

[285] Paul C. van Oorschot and Michael J. Wiener. "Parallel Collision Search with Cryptanalytic Applications". In: 12.1 (Jan. 1999), pp. 1–28. DOI: 10.1007/PL00003816 (cit. on pp. 252, 254).

[286] Serge Vaudenay. *A Classical Introduction to Cryptography: Applications for Communications Security*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005 (cit. on p. 67).

[287] Jacques Vélu. "Isogénies entre courbes elliptiques". In: *Comptes-Rendus de l'Académie des Sciences* 273 (1971). https://gallica.bnf.fr/ark:/12148/cb34416987n/date, pp. 238–241. URL: https://gallica.bnf.fr/ark:/12148/cb34416987n/date (cit. on pp. 27, 317).

[288] Frederik Vercauteren. "Optimal pairings". In: *IEEE transactions on information theory* 56.1 (2009), pp. 455–461 (cit. on p. 283).

[289] John Voight. *Quaternion algebras*. Springer Nature, 2021 (cit. on pp. 21, 319).

[290] Zhe-Xian Wan. "A proof of the automorphisms of linear groups over a sfield of characteristic 2". In: *Sci. Sinica* 11 (1962), pp. 1183–1194 (cit. on p. 50).

[291] Benjamin Wesolowski. "The supersingular isogeny path and endomorphism ring problems are equivalent". In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2022, pp. 1100–1111 (cit. on p. 318).

[292] Fan Zhang, Bolin Yang, Xiaofei Dong, Sylvain Guilley, Zhe Liu, Wei He, Fangguo Zhang, and Kui Ren. "Side-Channel Analysis and Countermeasure Design on ARM-Based Quantum-Resistant SIKE". In: *IEEE Trans. Computers* 69.11 (2020), pp. 1681–1693. URL: https://doi.org/10.1109/TC.2020.3020407 (cit. on p. 137).