An Algorithmic Approach to (2, 2)-isogenies in the Theta Model and Applications to Isogeny-based Cryptography

Pierrick Dartois[1,2], Luciano Maino[3], Giacomo Pope[3,4], and Damien Robert[1,2]

[1] Univ. Bordeaux, CNRS, Bordeaux INP, IMB, UMR 5251, F-33400, Talence, France
[2] INRIA, IMB, UMR 5251, F-33400, Talence, France
[3] University of Bristol, Bristol, United Kingdom
[4] NCC Group, Cheltenham, United Kingdom

Abstract. In this paper, we describe an algorithm to compute chains of $(2, 2)$-isogenies between products of elliptic curves in the theta model. The description of the algorithm is split into various subroutines to allow for a precise field operation counting.
We present a constant time implementation of our algorithm in Rust and an alternative implementation in SageMath. Our work in SageMath runs ten times faster than a comparable implementation of an isogeny chain using the Richelot correspondence. The Rust implementation runs up to forty times faster than the equivalent isogeny in SageMath and has been designed to be portable for future research in higher-dimensional isogeny-based cryptography.

1 Introduction

The devastating attacks on SIDH [3,17,34] have highlighted the relevance of studying higher-dimension abelian varieties in isogeny-based cryptography. Following the attacks, it soon became evident that these new tools would have had applications beyond cryptanalysis. For instance, Robert leveraged these techniques both to give a representation of isogenies in polylogarithmic time [32] and to compute the endomorphism ring of ordinary elliptic curves in quantum polynomial time [33].

On a more cryptographic side, the attacks have been used to design new protocols. Basso, Maino and Pope utilise these cryptanalytic techniques to construct a trapdoor mechanism, and using standard transformations, this trapdoor is used to derive a public-key encryption protocol named FESTA [2]. Subsequently, two additional protocols employing similar ideas to FESTA have appeared [20,25]. One of the main building blocks underlying these protocols is the computation of chains of $(2, 2)$-isogenies between products of two elliptic curves. However, the cryptographic application of these isogeny chains extends beyond FESTA-based applications. For instance, they are at the core of computing the group action in SCALLOP-HD [4], as well as in novel constructions of isogeny-based weak verifiable delay functions [8] and verifiable random functions [15]. Therefore, improving algorithms to compute chains of $(2, 2)$-isogenies between elliptic products is of paramount importance to the progress of higher-dimensional isogeny-based protocols.

Prior to this work, the only method to compute $(2,2)$-isogenies between elliptic products relied on ad-hoc procedures for gluing and splitting, and the use of the Richelot correspondence to compute isogenies between Jacobians of genus two hyperelliptic curves [36,26]. This method can be considered satisfactory for cryptanalytic purposes, but it is definitely not efficient enough for constructive applications. Indeed, for the proof-of-concept implementations of [2,25], the two-dimensional isogenies are the bottleneck of the protocol. Richelot isogenies describe $(2,2)$-isogenies between Jacobians of genus two hyperelliptic curves in the Mumford model. Here, kernel elements are divisors, represented by a pair of univariate polynomials. The arithmetic of the group elements, as well as isogeny codomain computation and evaluation, require working in a univariate polynomial ring above the base field. This model makes doubling and evaluation of points expensive and the implementation of the isogeny chain itself is significantly more complicated than the more familiar isogeny chains between two elliptic curves, which use Vélu's formulae. A natural question is then to ask whether it could be possible to use different models that are more amenable to simple and efficient algorithmic descriptions.

In the literature, another model used to compute isogenies is already known: the *theta model*. Despite being suitable for isogenies between elliptic curves, the theta model has mainly been employed to compute isogenies in higher dimension due to the lack of alternatives. For instance, Cosset and Robert describe an algorithm for (ℓ,ℓ)-isogenies in the theta model for odd primes ℓ [5]. The case $\ell = 2$ has been briefly treated in [30, Proposition 6.3.5] and [31, Remarks 2.10.3, 2.10.7, 2.10.14] but never formalised. The isogeny formulae in the theta model can be seen as a natural generalisation of Vélu's formulae in higher dimension. In contrast, the Richelot correspondence is a very special relationship on hyperelliptic curves which does not seem to naturally generalise to arbitrary dimension. Nevertheless, the theta model has not received much attention as many have deemed the algorithms as impractical for efficient computation.

This paper is dedicated to disprove this claim of impracticality. We aim to demystify the hard algebraic geometry underpinning the theta model and make it accessible to cryptographers who want to employ isogenies between higher-dimensional abelian varieties within their protocols. The end result of our work is a set of concrete algorithms which describe the necessary pieces for computing isogenies between elliptic products; written to be particularly amenable to efficient and optimised implementations which are not all that different in appearance to the one-dimensional isogenies many are more familiar with.

1.1 Contributions

This paper has been written with the aim of being modular, using an algorithmic approach. All the formulae in the paper are mainly derived from the *duplication formula*. As a result, a reader uniquely interested in the computational results can assume the validity of the work in Section 2 and follow along the subsequent sections, which contain the explicit algorithms. From the duplication formula, we first re-obtain the addition formulae that have already been described in [12] and also give a precise operation counting in the base field.

The algorithm to compute chains of $(2,2)$-isogenies between elliptic products is split into various subroutines. Each subroutine is carefully described in algorithmic boxes; this allows for a precise field operation counting. The main advantage of this approach is that both reducible and irreducible abelian surfaces can be described in the same way. However, some extra care will be devoted to the splitting and gluing case.

The gluing case is the most delicate one, where zero-coordinates must be carefully handled during both arithmetic and isogeny computations. To recover the theta-model representation of an elliptic product, we build upon the formulae described in [1] which constructs theta structures on elliptic curves in the Montgomery model. We efficiently compute the theta-model representation of an elliptic product using only the dimension one representation of the theta structures and a few additional multiplications to recover the product structure. A summary of the costs of the algorithms described in this paper is shown in Table 1.

Table 1. Table of base field costs of doubling, codomain computation and evaluation for both generic and gluing isogenies in the theta model. We denote by $\mathbf{M}, \mathbf{S}, \mathbf{I}$ the costs of multiplication, squaring and inversion of an element in the base field and ignore the cost of additions. The generic codomain optimisation comes from reusing certain field elements also required for doubling. These are naturally available when computing long isogeny chains for a given kernel, and so the optimised cost is the expected cost for an isogeny between elliptic products.

Isogeny Type	Doubling	Codomain		Evaluation
		Generic	Optimised	
Generic	$8\mathbf{S} + 6\mathbf{M}$	$8\mathbf{S} + 29\mathbf{M} + 1\mathbf{I}$	$8\mathbf{S} + 9\mathbf{M} + 1\mathbf{I}$	$4\mathbf{S} + 3\mathbf{M}$
Gluing	$12\mathbf{S} + 12\mathbf{M}$	$8\mathbf{S} + 13\mathbf{M} + 1\mathbf{I}$		$8\mathbf{S} + 5\mathbf{M} + 1\mathbf{I}$

Note that unlike the case of the Richelot chain, which requires both a $(2,2)$-gluing and $(2,2)$-splitting isogeny, computing the elliptic product at the end of a chain of isogenies in the theta model is a case of simply converting from one model to another, which can be done efficiently.

Finally, we offer both a constant time implementation of an isogeny between elliptic products in the programming language Rust, as well as an alternative implementation for the computer algebra system SageMath [37]. Both are available at the following GitHub repository:

`https://github.com/ThetaIsogenies/two-isogenies`.

The Rust implementation has been written with cryptographic applications in mind and so has been built to run in constant time, with the appropriate finite field arithmetic and no secret-dependent conditional branching. It should also be easily portable to other projects in the future. The SageMath implementation has been designed to be a drop-in replacement for the work of [26]. As a result, all the protocols whose implementation relies on this work or the proof-of-concept of [2] can be upgraded to

$(2,2)$-isogenies in the theta model with minimal effort. As a use case, we show the benefit of these algorithms in FESTA in Section 5.3.

We give explicit timings of our implementations in Table 2. In SageMath, our implementation achieves a ten times speed up for the codomain computation and more than twenty times speed up for evaluation time compared to [26]. For smaller characteristic, the Rust code runs approximately forty times faster than the same algorithms written in SageMath, and more than two times as fast for very large characteristic. Concretely, on an Intel Core i7-9750H CPU with a clock-speed of 2.6 GHz with turbo-boost disabled, we compute an isogeny chain of length $n = 208$ between elliptic products over $\mathbb{F}_{p^2}$ with a 254 bit characteristic in only 2.85 ms.

Roadmap In Section 2, we give a concise summary of the algebraic theory of theta functions. The most important parts of this section are the duplication formula and change-of-basis algorithm; the reader willing to accept these two main building blocks can skip this section entirely. In Section 3, we derive addition formulae from the duplication formula. The isogeny formulae are described in Section 4. We discuss our implementation results in Section 5 and draw some conclusions in Section 6.

Notation Throughout the paper, $\mathbf{M}, \mathbf{S}, \mathbf{I}$ will represent the cost of multiplication, squaring and inversion of an element in the base field, respectively. In Section 2, we will introduce the Hadamard transform $\mathcal{H}$; in dimension two,

$$\mathcal{H}(x,y,z,w) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

We also define $(\tilde{\theta}_{00}(P) : \tilde{\theta}_{10}(P) : \tilde{\theta}_{01}(P) : \tilde{\theta}_{11}(P)) = \mathcal{H}\left(\theta_{00}(P), \theta_{10}(P), \theta_{01}(P), \theta_{11}(P)\right)$ to be the *dual coordinates* of P, and the $\star$ operator:

$$(x,y,z,w) \star (x',y',z',w') = (xx', yy', zz', ww').$$

Another useful operator we will introduce in Section 3 is the squaring operator $\mathcal{S}$:

$$\mathcal{S}(x,y,z,w) = (x^2, y^2, z^2, w^2).$$

When computing the cost of inverting k elements, we will use *batched inversions*. Batched inversions allow us to invert k elements at a cost of $3(k-1)$ multiplications and only one inversion [19, §10.3.1]. We refer to our implementation for an explicit description of the algorithm.

Acknowledgments. Huge thanks are given to Thomas Pornin for his advice and previous collaborations, both of which were instrumental in the design and implementation of the constant time Rust implementation. We also thank Sabrina Kunzweiler for fruitful discussion.

2 Preliminaries

We assume the reader has some familiarity with (N, N)-isogenies between principally polarised abelian surfaces; we refer to [17, §2] for a gentle introduction to the subject. Before giving an explicit description of the algorithm used to compute chains of $(2, 2)$-isogenies between products of two elliptic curves, we provide a concise and self-contained summary of the *algebraic theory of theta functions*. Using theta functions, it is possible to perform arithmetic on *principally polarised abelian varieties*. The reader willing to assume the validity of the *duplication formula* and our method to perform a *change of basis* can skip this section entirely and use Algorithm 2 as a black box.

For all the other readers, in what follows, we utilise the language of *Mumford's theory* to provide a summary of the algebraic theory of theta functions [21,22,23]. We will first briefly recapitulate Mumford's results, then recall the *Duplication formula*, which will allow us to describe isogeny formulae between principally polarised abelian surfaces, and finally provide a formula for the change of basis in the case of a product of two elliptic curves, which is the case analysed in the paper.

2.1 Mumford's Theory

We refer to [18] for background material on abelian varieties and line bundles. Let A be an abelian variety defined over a field k whose characteristic is different from 2, and let $\mathcal{L}$ be a line bundle on A; we further assume that $\mathcal{L}$ is a *totally symmetric* line bundle [21, §2]. We define $K(\mathcal{L})$ to be the set of points x such that the pullback of $\mathcal{L}$ under the translation-by-x map T_x is isomorphic to $\mathcal{L}$, i.e.

$$K(\mathcal{L}) := \{x \in A \,|\, T_x^*\mathcal{L} \simeq \mathcal{L}\}\,.$$

Given a point $x \in K(\mathcal{L})$, the isomorphism between $\mathcal{L}$ and $T_x^*\mathcal{L}$ is in general not unique. Also, given $x, y \in K(\mathcal{L})$, together with two isomorphisms $\phi_x \colon \mathcal{L} \xrightarrow{\sim} T_x^*\mathcal{L}$ and $\phi_y \colon \mathcal{L} \xrightarrow{\sim} T_y^*\mathcal{L}$, it is possible to define an isomorphism between $\mathcal{L}$ and $T_{x+y}^*\mathcal{L}$ as follows:

$$\mathcal{L} \xrightarrow{\phi_x} T_x^*\mathcal{L} \xrightarrow{T_x^*\phi_y} T_x^*(T_y^*\mathcal{L}) = T_{x+y}^*\mathcal{L}.$$

This observation led Mumford to define the group

$$\mathcal{G}(\mathcal{L}) := \{(x, \phi_x) \,|\, x \in K(\mathcal{L}),\, \phi_x \colon \mathcal{L} \xrightarrow{\sim} T_x^*\mathcal{L}\}\,,$$

where $(x, \phi_x) \cdot (y, \phi_y) = (x + y, T_x^*\phi_y \circ \phi_x)$. We will refer to $\mathcal{G}(\mathcal{L})$ as the *Mumford theta group*. The Mumford theta group fits in the exact sequence

$$0 \to \bar{k}^* \to \mathcal{G}(\mathcal{L}) \to K(\mathcal{L}) \to 0.$$

Let $f \colon A \to B$ be an isogeny between abelian varieties, let $\mathcal{L}$ be a line bundle on A and $\mathcal{M}$ a line bundle on B. Suppose there exists an isomorphism $\alpha \colon f^*\mathcal{M} \xrightarrow{\sim} \mathcal{L}$. Then, one can prove that $\mathsf{Ker}(f) \subset K(\mathcal{L})$. On the other hand, given $K \subset K(\mathcal{L})$, it is not generally true that the isogeny $f' \colon A \to B$ of kernel K generates an isomorphism between $\mathcal{L}$ and $(f')^*\mathcal{M}'$ for some line bundle $\mathcal{M}'$ on B. This is exactly true when K

admits a *level subgroup* $\tilde{K}$ of $\mathcal{G}(\mathcal{L})$; a subgroup $\tilde{K}$ of $\mathcal{G}(\mathcal{L})$ such that its image under the forgetful map $(x, \phi_x) \to x$ coincides with K.

To understand when a subgroup $K \subset K(\mathcal{L})$ admits a level subgroup, Mumford introduces a *pairing* on $K(\mathcal{L})$: define $e_{\mathcal{L}} \colon K(\mathcal{L}) \times K(\mathcal{L}) \to \bar{k}^*$ to be the pairing such that $e_{\mathcal{L}}(x, y) = \tilde{x} \cdot \tilde{y} \cdot \tilde{x}^{-1} \cdot \tilde{y}^{-1}$, where $\tilde{x}$ is any lift of x to $\mathcal{G}(\mathcal{L})$.[5] Then, there exists a level subgroup $\tilde{K}$ over K if and only if $e_{\mathcal{L}} \equiv 1$ on $K \times K$. Also, $(K(\mathcal{L}), e_{\mathcal{L}})$ is a symplectic space, which implies that there exists a decomposition into two maximal isotropic groups $K(\mathcal{L}) = K(\mathcal{L})_1 \oplus K(\mathcal{L})_2$. Let $\delta := (d_1, \ldots, d_g)$ be the elementary divisors of $K(\mathcal{L})_1 \simeq K(\mathcal{L})_2$. We say that $\mathcal{L}$ *is of type* δ. Define

$$K(\delta) := \bigoplus_{i=1}^{g} \mathbb{Z}/d_i\mathbb{Z}, \quad H(\delta) := K(\delta) \oplus \widehat{K}(\delta), \quad \mathcal{G}(\delta) := \bar{k}^* \times H(\delta),$$

where $\widehat{K}(\delta) := \mathsf{Hom}\left(K(\delta), \bar{k}^*\right)$. As $K(\mathcal{L})$, $H(\delta)$ can be equipped with a pairing deriving from the duality: for all $(x, \chi), (x', \chi') \in H(\delta)$

$$e_\delta((x, \chi), (x, \chi')) = \chi'(x)\chi(x')^{-1}.$$

The set $\mathcal{G}(\delta)$ is endowed with a group structure and is called *Heisenberg group*. This group structure is given explicitly by: for all $(\alpha, x, \chi), (\alpha', x', \chi') \in \mathcal{G}(\delta)$

$$(\alpha, x, \chi) \cdot (\alpha', x', \chi') := (\alpha\alpha'\chi'(x), x + x', \chi \cdot \chi').$$

The Heisenberg group $\mathcal{G}(\delta)$ has a unique irreducible representation which acts trivially with $\bar{k}^*$; such a representation is $V(\delta) := \left\{g \colon K(\delta) \to \bar{k}\right\}$. On the other hand, $\mathcal{G}(\mathcal{L})$ acts on the space of the global sections $\Gamma(A, \mathcal{L})$, and $\Gamma(A, \mathcal{L})$ is an irreducible representation for $\mathcal{G}(\mathcal{L})$. We denote the action of $\mathcal{G}(\mathcal{L})$ on $\Gamma(A, \mathcal{L})$ simply by "$\cdot$".

A *theta structure* $\Theta^{\mathcal{L}}$ of type δ is an isomorphism of central extensions from $\mathcal{G}(\delta)$ to $\mathcal{G}(\mathcal{L})$ fitting in the diagram:

$$\begin{array}{ccccccccc} 0 & \longrightarrow & \bar{k}^* & \longrightarrow & \mathcal{G}(\delta) & \longrightarrow & H(\delta) & \longrightarrow & 0 \\ & & \| & & \downarrow{\scriptstyle \Theta^{\mathcal{L}}} & & \downarrow{\scriptstyle \overline{\Theta}^{\mathcal{L}}} & & \\ 0 & \longrightarrow & \bar{k}^* & \longrightarrow & \mathcal{G}(\mathcal{L}) & \longrightarrow & K(\mathcal{L}) & \longrightarrow & 0 \end{array}$$

Considering the horizontal arrows as the natural maps, the isomorphism $\Theta^{\mathcal{L}}$ induces another isomorphism $\overline{\Theta}^{\mathcal{L}}$ between $H(\delta)$ and $K(\mathcal{L})$, which is symplectic for the pairings e_δ and $e_{\mathcal{L}}$. As a result, this isomorphism defines a symplectic decomposition on $K(\mathcal{L}) = K(\mathcal{L})_1 \oplus K(\mathcal{L})_2$, where $K(\mathcal{L})_1 := \overline{\Theta}^{\mathcal{L}}(K(\delta))$ and $K(\mathcal{L})_2 := \overline{\Theta}^{\mathcal{L}}(\hat{K}(\delta))$. Using such a decomposition, we will see below that it is possible to identify a basis $(\theta_x)_{x \in K(\mathcal{L})_1}$ for the space of global sections $\Gamma(A, \mathcal{L})$. Since $\Theta^{\mathcal{L}}$ restricts to the identity with $\bar{k}^*$, $\Gamma(A, \mathcal{L})$ is an irreducible representation of $\mathcal{G}(\delta)$, which acts trivially with $\bar{k}^*$. Also, since $V(\delta)$ is

[5] Technically, $e_{\mathcal{L}}$ maps onto $\mathcal{G}(\mathcal{L})$. However, the first component of $e_{\mathcal{L}}(x, y)$ is the identity on the abelian variety, whereas the second one is an automorphism of $\mathcal{L}$, and the group of such automorphisms can be identified with $\bar{k}^*$.

the unique irreducible representation of $H(\delta)$, there exists (up to constants) a unique isomorphism $\beta\colon \Gamma(A, L) \xrightarrow{\sim} V(\delta)$.

The space $V(\delta)$ admits a canonical basis given by the Kronecker delta functions $(\delta_i)_{i\in K(\delta)}$ on $K(\delta)$. Via β, it is possible to transfer that basis to a basis $(\theta_i)_{i\in K(\mathcal{L})_1}$ on $\Gamma(A, \mathcal{L})$; the functions θ_i are called *theta coordinates*, and, if $d_1 = \ldots = d_g = n$, these theta coordinates are called theta coordinates of *level* n. Using theta coordinates, it is possible to represent abelian varieties via an embedding into the projective space [24, Ch. II, Theorem 1.3]. In particular, the abelian variety A can be completely described in the projective space via the evaluation of theta coordinates at the identity using the *Riemann relations*; we call the projective point $\left(\theta_i(0^A)\right)_{i\in K(\mathcal{L})_1}$ *theta-null point*. Given a point $P \in A$ and $T \in K(\mathcal{L})$, we can efficiently represent $P + T$ in theta coordinates: if T corresponds to $(s, \chi) \in \mathcal{H}(\delta)$,

$$\left(\theta_i(P + T)\right)_i = \left(\chi(i)\theta_{i+s}(P)\right)_i . \tag{1}$$

This remark will allow us to compute $(2, 2)$-isogenies using projective coordinates.

Example 1. Let $(a : b : c : d)$ be a theta-null point of level 2 obtained from the theta structure $\Theta^{\mathcal{L}}$. Implicitly, we have a symplectic basis (S_1, S_2, T_1, T_2) of the 2-torsion. Let $i_1 = ([1], [0]) \in K(2, 2)$, $i_2 = ([1], [0]) \in K(2, 2)$, $\chi_1 \in \widehat{K}(2, 2)$ such that $\chi_1(i_1) = -1$ and $\chi_1(i_2) = 1$, and $\chi_2 \in \widehat{K}(2, 2)$ such that $\chi_2(i_1) = 1$ and $\chi_2(i_2) = -1$. Then, let us define $S_1 = \overline{\Theta}^{\mathcal{L}}(i_1)$, $S_2 = \overline{\Theta}^{\mathcal{L}}(i_2)$, $T_1 = \overline{\Theta}^{\mathcal{L}}(\chi_1)$ and $T_2 = \overline{\Theta}^{\mathcal{L}}(\chi_2)$.

If $P = (x : y : z : t)$, we have $S_1 = (b : a : d : c)$ and $P + S_1 = (y : x : t : z)$, $S_2 = (c : d : a : b)$ and $P + S_2 = (z : t : x : y)$, $T_1 = (a : -b : c : -d)$ and $P + T_1 = (x : -y : z : -t)$, $T_2 = (a : b : -c : -d)$ and $P + T_2 = (x : y : -z : -t)$.

Another fundamental ingredient is the change of theta structure given by *Heisenberg group automorphisms*. A Heisenberg group automorphism is an automorphism of $\mathcal{G}(\delta)$ acting as the identity on $\bar{k}^*$. In particular, such an automorphism induces a symplectic automorphism on $\mathcal{H}(\delta)$ with respect to its natural pairing e_δ. The most fundamental example is the *Hadamard Transform*, which is the automorphism that swaps $K(\delta)$ and $\widehat{K}(\delta)$.

Hadamard Transform Let $(\theta_i)_i$ be some theta coordinates on A. The action of the Hadamard transform on $(\theta_i)_i$ is described in [31, Eq. 2.4]. The resulting theta coordinates after this transform are called the *dual theta coordinates*; we will denote such coordinates by $(\tilde{\theta}_i)_i$. In what follows, we will use the Hadamard transform on level-2 theta coordinates. For the sake of clarity, we explicitly state the action of this symplectic automorphism in dimension one and two.

First, let us fix an ordering for theta coordinates. In dimension one, there are only two theta coordinates. Whenever we write $(x : y)$ to represent a point P in theta coordinates, we actually mean $(\theta_0(P) : \theta_1(P))$. Hence, specialising [31, Eq. 2.4], we obtain $(\tilde{\theta}_0(P) : \tilde{\theta}_1(P)) = (x + y : x - y)$. In dimension two, we represent a point P in theta coordinates by a tuple $(x : y : z : w)$, where we fix the ordering $(\theta_{00}(P) : \theta_{10}(P) :$

$\theta_{01}(P) : \theta_{11}(P))$. [6] Specialising [31, Eq. 2.4], we have

$$\begin{aligned}
\tilde{\theta}_{00}(P) &= x + y + z + w,\\
\tilde{\theta}_{10}(P) &= x - y + z - w,\\
\tilde{\theta}_{01}(P) &= x + y - z - w,\\
\tilde{\theta}_{11}(P) &= x - y - z + w.
\end{aligned}$$

Henceforth, we will use the operator $\mathcal{H}$ to refer to the action of the Hadamard transform on theta coordinates. Finally, we remark that $\mathcal{H}\left(\mathcal{H}\left((\theta_i^A)_i\right)\right) = (\theta_i^A)_i$.

2.2 Duplication Formula

Let $(\theta_i^A)_i$ be theta coordinates of level 2 on A. Implicitly, we have a symplectic decomposition of $A[2] = K(\mathcal{L}) = K(\mathcal{L})_1 \oplus K(\mathcal{L})_2$ — from now on, we will drop the dependence on the line bundle $\mathcal{L}$ and simply write $A[2] = K_1 \oplus K_2$. Let $\langle S_1, \ldots, S_g\rangle$ be the canonical basis induced by $\overline{\Theta}^{\mathcal{L}}(K(2, \ldots, 2))$ and $\langle T_1, \ldots, T_g\rangle$ the canonical basis induced by $\overline{\Theta}^{\mathcal{L}}(\widehat{K}(2, \ldots, 2))$, which means $K_1 = \langle S_1, \ldots, S_g\rangle$ and $K_2 = \langle T_1, \ldots, T_g\rangle$. Now, let us consider the isogeny $f\colon A \to B$, where $\ker(f) = K_2$. The abelian variety B is principally polarised and in turn can be endowed with a type-$(2, \ldots, 2)$ theta structure, whose theta coordinates are denoted by $(\theta_i^B)_i$. Also, let us define $\star$ to be the operator such that $(x_i)_i \star (y_i)_i = (x_i y_i)_i$. Then, a consequence of the isogeny theorem [21, Theorem 4, p. 302] (see also [30, Theorem 3.6.4]) and duplication formula [21, Equation A, p. 332] shows that:

$$\left(\theta_i^A(P+Q)\right)_i \star \left(\theta_i^A(P-Q)\right)_i = \mathcal{H}\left(\left(\tilde{\theta}_i^B(f(P))\right)_i \star \left(\tilde{\theta}_i^B(f(Q))\right)_i\right), \tag{2}$$

$$\mathcal{H}\left(\left(\theta_i^A(\tilde{f}(R))\right)_i \star \left(\theta_i^A(\tilde{f}(S))\right)_i\right) = \left(\tilde{\theta}_i^B(R+S)\right)_i \star \left(\tilde{\theta}_i^B(R-S)\right)_i, \tag{3}$$

where $\tilde{f}$ denotes the dual isogeny of f and $(\tilde{\theta}_i^B)_i$ are the dual theta coordinates of $(\theta_i^B)_i$.

2.3 Change of Basis

We now focus on products of two elliptic curves and explain how to endow these products with a theta structure of type $(2, 2)$. First, let us recall that every theta structure of type $(2, \ldots, 2)$ comes from a symplectic basis of the 4-torsion [21, Remark 4, p. 319]. Let E_1 and E_2 be two elliptic curves. The natural candidate for a theta structure on $E_1 \times E_2$ is the *product theta structure*, which is obtained via the combination of the theta structures on elliptic curves [7, Lemma F.3.1].

Proposition 2. *Let $(a_i : b_i)$ be a theta-null points on E_i induced by a symplectic 4-torsion basis $\langle e_i, f_i\rangle$, for $i = 1, 2$. Then, $(a_1a_2 : b_1a_2 : a_1b_2 : b_1b_2)$ is a theta-null point for $E_1 \times E_2$ induced by the symplectic 4-torsion basis $\langle (e_1, 0), (0, e_2)\rangle \oplus \langle (f_1, 0), (0, f_2)\rangle$.*

[6] The subscript ij refers to the pair $([i], [j]) \in K(2, 2)$.

However, in the next sections, we might need to work with theta structures associated to a different symplectic 4-torsion basis. The change-of-basis algorithm we describe here is a generalisation of the case described in [1], which uses a different approach based on the ramification of the Kummer line. We briefly describe the idea below and adapt it to the case of products of two elliptic curves.

Let E be an elliptic curve, and $\langle T_1', T_2' \rangle$ a basis of the 4-torsion. We explain how to compute the theta-null point associated to this basis. Given a point $T \in E[2]$, there are two symmetric elements $\pm\mathfrak{g}$ above T in the theta group $\mathcal{G}(\mathcal{L}(2(0_E)))$. We can fix a symmetric element via a point T' of 4-torsion above T. Let $T_1 = 2T_1'$, $T_2 = 2T_2'$, and let $\mathfrak{g}_1, \mathfrak{g}_2$ be these elements associated to T_1' and T_2', respectively. By definition of the compatible theta basis, the theta coordinate θ_0 must be invariant under the action of $\mathfrak{g}_2$, and $\theta_1 = \mathfrak{g}_1 \cdot \theta_0$. The coordinate θ_0 can be computed as the trace of a global section $s \in \Gamma(E, \mathcal{L}(2(0_E)))$, provided it is not equal to zero, i.e. $\theta_0 = \mathrm{id} \cdot s + \mathfrak{g}_2 \cdot s \neq 0$.

Working on a Montgomery curve in Weierstrass coordinates, we have a canonical point of 4-torsion $T' = (1 : 1)$ above $T = (0 : 1)$ that induces the canonical element $\mathfrak{g}$ of the theta group acting by $\mathfrak{g} \cdot (X, Z) = (Z, X)$. Indeed, translation by T is given by $(X : Z) \mapsto (Z : X)$, and the two symmetric elements above this translation act by $(X, Z) \mapsto (\pm Z, \pm X)$ since they have order 2. The element $\mathfrak{g}_{T'}$ in $\mathcal{G}(\mathcal{L}(2(0_E)))$ fixed by T' corresponds to $\pm\mathfrak{g}$. But since $\mathfrak{g}$ leaves invariant any affine lift of T', we have $\mathfrak{g}_{T'} = \mathfrak{g}$.

For a general elliptic curve, if $T' = (x, y, z)$ is a point of 4-torsion and $2T' = T = (u, v, w)$, we can map T' to the Montgomery point $(1 : 1)$ via the linear transformation (in the Kummer line): $M \colon (X : Z) \mapsto (X' : Z') = (zwX - zuZ : (xw - zu)Z)$. It follows that the action of $\mathfrak{g}_{T'}$ is given by

$$M^T U M^{T^{-1}} = \begin{pmatrix} uz/(wx - uz) & zw/(wx - uz) \\ (-wx^2 + 2uzx)/(-zwx + uz^2) & -uz/(wx - uz) \end{pmatrix},$$

with $M = \begin{pmatrix} wz & -zu \\ 0 & xw - uz \end{pmatrix}$, $U = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. This computation is the output of Algorithm 1.

Example 3. Using previous notation, on a Montgomery curve we have $T_2' = (-1 : 1)$, which acts by $\mathfrak{g}_2 \cdot (X, Z) = (-Z, -X)$. Taking the trace of X under this action we get $\theta_0 = \mathrm{id} \cdot X + \mathfrak{g}_2 \cdot X = X - Z$.

Let $T_1' = (a + b : a - b)$ be another point of 4-torsion; its double is then $(a^2 + b^2 : a^2 - b^2)$. Let $x = a + b$, $z = a - b$, $u = a^2 + b^2$, $w = a^2 - b^2$. We compute $\theta_1 = \mathfrak{g}_1 \cdot \theta_0 = \mathfrak{g}_1 \cdot (X - Z) = (uz - wxx/z + 2ux)/(wx - uz)X + z(w + u)/(wx - uz)Z = b/aX + b/aZ$. We recover the same conversion formula between Montgomery and theta coordinates as obtained in [1].

We can use the same strategy to compute the theta-null point associated to a symplectic basis of the 4-torsion on a product of elliptic curves. If $T' = (T_1', T_2') \in E_1 \times E_2$ is a point of 4-torsion, the associated element $\mathfrak{g}_{T'}$ is given by $\mathfrak{g}_{T'} = \mathfrak{g}_{T_1'} \otimes \mathfrak{g}_{T_2'}$. We can take θ_0 as the trace of $X_1 \otimes X_2$, i.e. $\theta_0 = \sum_i \mathfrak{g}_i \cdot X_1 \otimes X_2 = \sum_i \mathfrak{g}_{i,1} \cdot X_1 \otimes \mathfrak{g}_{i,2} \cdot X_2$, where the $\mathfrak{g}_i = \mathfrak{g}_{i,1} \otimes \mathfrak{g}_{i,2}$'s are the elements above K_2 fixed by the 4-torsion. The other theta coordinates are computed via the action of the elements above K_1 on θ_0.

Algorithm 1 Action by Translation

Input: A point P' in the 4-torsion of the Kummer line of an elliptic curve
Output: The 2×2 submatrix M with coefficients m_{ij} describing the action of $\mathfrak{g}_{P'}$ lifting the action by translation of $P = 2P'$.

1: $P \leftarrow [2]P'$ (▷) Cost: $3\mathbf{M} + 2\mathbf{S}$
2: Let $P' = (X : Z)$ and $(U : W) = P$
3: $WX, WZ, UX, UZ \leftarrow W \cdot X, W \cdot Z, U \cdot X, U \cdot Z$
4: $\delta \leftarrow WX - UZ$
5: Compute δ^{-1}, Z^{-1} via batched inversions
6: $m_{00} \leftarrow -UZ \cdot \delta^{-1}$
7: $m_{01} \leftarrow -WZ \cdot \delta^{-1}$
8: $m_{10} \leftarrow UX \cdot \delta^{-1} - X \cdot Z^{-1}$
9: $m_{11} \leftarrow -m_{00}$
10: **return** M (▷) Total cost: $14\mathbf{M} + 2\mathbf{S} + 1\mathbf{I}$

In practice, in the algorithm to compute the $(2^n, 2^n)$-isogeny $f : E_1 \times E_2 \to E_1' \times E_2'$, we only have access to $\ker(f)[4] = \langle T_1', T_2' \rangle$ and not to a complete symplectic torsion basis of $E_1 \times E_2[4]$; let $T_1' = (P_1, P_2)$ and $T_2' = (Q_1, Q_2)$. To bypass this problem, we define $S_1' = (0, Q_2)$ and $S_2' = (P_1, 0)$. Then, $K_1 = \langle S_1, S_2 \rangle$ and $K_2 = \langle T_1, T_2 \rangle$, where $S_i = [2]S_i'$ and $T_i = [2]T_i'$. We use the symplectic 4-torsion basis $\langle S_1', S_2', T_1', T_2' \rangle$ when endowing $E_1 \times E_2$ with a theta structure. We summarise this procedure in Algorithm 2. The output of this algorithm is a matrix N that allows for a change of coordinates as follows. If $R = (R_1, R_2) \in E_1 \times E_2$ is a point in Weierstrass coordinates for the Montgomery elliptic curves, where $R_i = (X_i : Z_i)$, the image of R in theta coordinates is given by

$$N \cdot \begin{pmatrix} X_1 \cdot X_2 \\ X_1 \cdot Z_2 \\ Z_1 \cdot X_2 \\ Z_1 \cdot Z_2 \end{pmatrix}.$$

Remark 4. In Algorithm 2, it is possible to optimise the computation of the four inversions required in the four calls to Algorithm 1 in lines 1, 2, 3 and 4 by using a unique batched inversion. We decided not to show this optimisation in Algorithm 2 for the sake of a cleaner exposition.

3 Addition Formulae

In this section, we derive addition formulae using the equations in Subsection 2.2. These formulae have already been described in dimension one [1] and dimension two [12]. However, we prefer to restate them in dimension two to highlight the connection with $(2, 2)$-isogenies and provide explicit operation counting. In what follows, we use the same notation as in Subsection 2.2.

Let $P, Q \in A$ and suppose we have $(\theta_i^A(P - Q))_i$. To compute $(\theta(P + Q)_i)_i$, we can use Equation 2, but first we need to recover $\left(\tilde{\theta}_i^B(f(P))\right)_i$ and $\left(\tilde{\theta}_i^B(f(Q))\right)_i$, which can

Algorithm 2 Change of Basis

Input: The points (P_1', P_2') and (Q_1', Q_2') in the four-torsion of $E_1 \times E_2$ below the kernel.
Output: The 4×4 change of basis matrix N.

$G_1 \leftarrow \mathsf{action\_by\_translation}(P_1')$ (▷) Algorithm 1: Cost: $56\mathbf{M} + 8\mathbf{S} + 4\mathbf{I}$
$G_2 \leftarrow \mathsf{action\_by\_translation}(P_2')$
$H_1 \leftarrow \mathsf{action\_by\_translation}(Q_1')$
$H_2 \leftarrow \mathsf{action\_by\_translation}(Q_2')$
$t_{00|1} \leftarrow g_{00|1} \cdot h_{00|1} + g_{01|1} \cdot h_{10|1}$ (▷) Compute the first column of $G_1 \times H_1$
$t_{10|1} \leftarrow g_{10|1} \cdot h_{00|1} + g_{11|1} \cdot h_{10|1}$
$t_{00|2} \leftarrow g_{00|2} \cdot h_{00|2} + g_{01|2} \cdot h_{10|2}$ (▷) Compute the first column of $G_2 \times H_2$
$t_{10|2} \leftarrow g_{10|2} \cdot h_{00|2} + g_{11|2} \cdot h_{10|2}$
$n_{00} \leftarrow g_{00|1} \cdot g_{00|2} + h_{00|1} \cdot h_{00|2} + t_{00|1} \cdot t_{00|2} + 1$ (▷) Compute the trace for the first row
$n_{01} \leftarrow g_{00|1} \cdot g_{10|2} + h_{00|1} \cdot h_{10|2} + t_{00|1} \cdot t_{10|2}$
$n_{02} \leftarrow g_{10|1} \cdot g_{00|2} + h_{10|1} \cdot h_{00|2} + t_{10|1} \cdot t_{00|2}$
$n_{03} \leftarrow g_{10|1} \cdot g_{10|2} + h_{10|1} \cdot h_{10|2} + t_{10|1} \cdot t_{10|2}$
$n_{10} \leftarrow h_{00|2} \cdot n_{00} + h_{01|2} \cdot n_{01}$ (▷) Compute the action of $(0, Q_2')$ for the second row
$n_{11} \leftarrow h_{10|2} \cdot n_{00} + h_{11|2} \cdot n_{01}$
$n_{12} \leftarrow h_{00|2} \cdot n_{02} + h_{01|2} \cdot n_{03}$
$n_{13} \leftarrow h_{10|2} \cdot n_{02} + h_{11|2} \cdot n_{03}$
$n_{20} \leftarrow g_{00|1} \cdot n_{00} + g_{01|1} \cdot n_{02}$ (▷) Compute the action of $(P_1', 0)$ for the third row
$n_{21} \leftarrow g_{00|1} \cdot n_{01} + g_{01|1} \cdot n_{03}$
$n_{22} \leftarrow g_{10|1} \cdot n_{00} + g_{11|1} \cdot n_{02}$
$n_{23} \leftarrow g_{10|1} \cdot n_{01} + g_{11|1} \cdot n_{03}$
$n_{30} \leftarrow g_{00|1} \cdot n_{10} + g_{01|1} \cdot n_{12}$ (▷) Compute the action of (P_1', Q_2') for the final row
$n_{31} \leftarrow g_{00|1} \cdot n_{11} + g_{01|1} \cdot n_{13}$
$n_{32} \leftarrow g_{10|1} \cdot n_{10} + g_{11|1} \cdot n_{12}$
$n_{33} \leftarrow g_{10|1} \cdot n_{11} + g_{11|1} \cdot n_{13}$
return N (▷) Total cost: $100\mathbf{M} + 8\mathbf{S} + 4\mathbf{I}$

be computed as

$$\left(\tilde{\theta}_i^B(f(P))\right)_i \star \left(\tilde{\theta}_i^B(0)\right)_i = \mathcal{H}\left(\left(\theta_i^A(P)\right)_i \star \left(\theta_i^A(P)\right)_i\right),$$

and similarly for $\left(\tilde{\theta}_i^B(f(Q))\right)_i$. The quantity $\left(\tilde{\theta}_i^B(0)\right)_i$ is actually not needed, as we only need $\left(\tilde{\theta}_i^B(0)\right)_i \star \left(\tilde{\theta}_i^B(0)\right)_i$ if we use

$$\left(\tilde{\theta}_i^B(f(P))\right)_i \star \left(\tilde{\theta}_i^B(0)\right)_i \star \left(\tilde{\theta}_i^B(f(Q))\right)_i \star \left(\tilde{\theta}_i^B(0)\right)_i,$$

to compute $\left(\tilde{\theta}_i^B(f(P))\right)_i \star \left(\tilde{\theta}_i^B(f(Q))\right)_i$. For the sake of compactness, we introduce the operator $\mathcal{S}$ that, on input $\left(\theta_i^A(P)\right)_i$, returns $\left(\theta_i^A(P)\right)_i \star \left(\theta_i^A(P)\right)_i$. We formalise this procedure in Algorithm 3.

Let $(a : b : c : d)$ be a theta-null point for A and define $(\alpha : \beta : \gamma : \delta)$ to be the dual coordinates $(\tilde{\theta}_i^B(0))_i$ of the theta-null point $(\theta^B(0)_i)_i$. For simplicity, let us assume that $\alpha \cdot \beta \cdot \gamma \cdot \delta \neq 0$; the (rare) case when one of the dual coordinates is zero is briefly treated in Remark 5. Equation 2 proves that $(\alpha^2 : \beta^2 : \gamma^2 : \delta^2) = \mathcal{H}(a^2 : b^2 : c^2 : d^2)$. However,

since we are working projectively, we need to use the quantities $(\alpha^2/\beta^2, \alpha^2/\gamma^2, \alpha^2/\delta^2)$ and $(a/b, a/c, a/d)$, which can we computed via batched inversions.

Algorithm 3 Differential addition

Input: The theta coordinates $(x_P : y_P : z_P : w_P)$ of P, the theta coordinates $(x_Q : y_Q : z_Q : w_Q)$ of Q, the theta coordinates $(x_{P-Q} : y_{P-Q} : z_{P-Q} : w_{P-Q})$ of $P - Q$ and $(\tilde{\lambda}_1, \tilde{\lambda}_2, \tilde{\lambda}_3) = (\alpha^2/\beta^2, \alpha^2/\gamma^2, \alpha^2/\delta^2)$.
Output: The theta coordinates $(x_{P+Q} : y_{P+Q} : z_{P+Q} : w_{P+Q})$ of $P + Q$.

$X_P, Y_P, Z_P, W_P \leftarrow \mathcal{H} \circ \mathcal{S}(x_P, y_P, z_P, w_P)$
$X_Q, Y_Q, Z_Q, W_Q \leftarrow \mathcal{H} \circ \mathcal{S}(x_Q, y_Q, z_Q, w_Q)$
$X_{f(P)f(Q)} \leftarrow X_P \cdot X_Q$
$Y_{f(P)f(Q)} \leftarrow \tilde{\lambda}_1 \cdot Y_P \cdot Y_Q$
$Z_{f(P)f(Q)} \leftarrow \tilde{\lambda}_2 \cdot Z_P \cdot Z_Q$
$W_{f(P)f(Q)} \leftarrow \tilde{\lambda}_3 \cdot W_P \cdot W_Q$
$X_{PQ}, Y_{PQ}, Z_{PQ}, W_{PQ} \leftarrow \mathcal{H}(X_{f(P)f(Q)}, Y_{f(P)f(Q)}, Z_{f(P)f(Q)}, W_{f(P)f(Q)})$
$xy_{P-Q}, zt_{P-Q} \leftarrow x_{P-Q} \cdot y_{P-Q}, z_{P-Q} \cdot t_{P-Q}$
$x_{P+Q} \leftarrow X_{PQ} \cdot zt_{P-Q} \cdot y_{P-Q}$
$y_{P+Q} \leftarrow Y_{PQ} \cdot zt_{P-Q} \cdot x_{P-Q}$
$z_{P+Q} \leftarrow Z_{PQ} \cdot xy_{P-Q} \cdot w_{P-Q}$
$w_{P+Q} \leftarrow W_{PQ} \cdot xy_{P-Q} \cdot z_{P-Q}$
return $x_{P+Q}, y_{P+Q}, z_{P+Q}, w_{P+Q}$ (▷) Total cost: $8\mathbf{S} + 17\mathbf{M}$

To obtain an algorithm to double a point $P \in A$, we proceed as before with the only difference that we only need $(a/b, a/c, a/d)$. We provide a detailed description of the doubling in Algorithm 4.

Algorithm 4 Doubling

Input: The theta coordinates $(x_P : y_P : z_P : w_P)$ of P, $(\tilde{\lambda}_1, \tilde{\lambda}_2, \tilde{\lambda}_3) = (\alpha^2/\beta^2, \alpha^2/\gamma^2, \alpha^2/\delta^2)$ and $(\lambda_1, \lambda_2, \lambda_3) = (a/b, a/c, a/d)$.
Output: The theta coordinates $(x_{2P} : y_{2P} : z_{2P} : w_{2P})$ of $2P$.

$X_P, Y_P, Z_P, W_P \leftarrow \mathcal{H} \circ \mathcal{S}(x_P, y_P, z_P, w_P)$
$X^2_{f(P)} \leftarrow X_P \cdot X_P$
$Y^2_{f(P)} \leftarrow \tilde{\lambda}_1 \cdot Y_P \cdot Y_P$
$Z^2_{f(P)} \leftarrow \tilde{\lambda}_2 \cdot Z_P \cdot Z_P$
$W^2_{f(P)} \leftarrow \tilde{\lambda}_3 \cdot W_P \cdot W_P$
$X'_P, Y'_P, Z'_P, W'_P \leftarrow \mathcal{H}(X^2_{f(P)}, Y^2_{f(P)}, Z^2_{f(P)}, W^2_{f(P)})$
return $X'_P, \lambda_1 \cdot Y'_P, \lambda_2 \cdot Z'_P, \lambda_3 \cdot W'_P$ (▷) Total cost: $8\mathbf{S} + 6\mathbf{M}$

Remark 5. In practice, we will always be in the case that $\alpha \cdot \beta \cdot \gamma \cdot \delta \neq 0$. We may incur in such an exception when we are working on a product of elliptic curves but with a non product theta structure (see [35, § 16.4]): this is due to the fact we may perform a change of basis as in Subsection 2.3 to put the kernel in the "right position" – see Section 4.1.

In this case we do not have to worry since we could perform arithmetic on the elliptic curves and then convert to the theta model afterwards. However, if one wants to deal with this case in the theta model, one may first act with a symplectic automorphism ψ sending the dual theta-null point to a all-non-zero one, use the addition formulae and eventually switch back to the former theta structure acting by ψ^{-1}.

To compute $(2^n, 2^n)$-isogenies, we will only use doublings. Algorithm 4 works only if $a \cdot b \cdot c \cdot d \neq 0$. The case $a \cdot b \cdot c \cdot d = 0$ can happen only if the codomain B is a product of elliptic curves with non product theta structure by [35, § 16.4]. In this case, we can use the same solution as above, by using $\psi = \mathcal{H}$ as our symplectic transformation.

4 The Isogeny Formula

In this section, we explain how to derive an isogeny formula for $(2, 2)$-isogenies from Subsection 2.2. Ultimately, we focus on isogenies between products of two elliptic curves. However, we first show how to compute isogenies when we have an abelian surface already endowed with a theta structure compatible with the kernel of the $(2, 2)$-isogeny we want to compute.

Let A be an abelian surface endowed with a theta structure of type $(2, 2)$, and let $\langle S_1, S_2, T_1, T_2\rangle$ be the canonical symplectic basis associated with the symplectic decomposition $A[2] = K_1 \oplus K_2$; to be more specific, $K_1 = \langle S_1, S_2\rangle$ and $K_2 = \langle T_1, T_2\rangle$. Let us recall that a theta-null point is fixed by a symplectic basis of the 4-torsion [21, Remark 4, p. 319], and let $\langle S'_1, S'_2, T'_1, T'_2\rangle$ be such a basis; in particular $2S'_i = S_i$ and $2T'_i = T_i$. Our goal is to compute a $(2, 2)$-isogeny $f : A \to B$. If the kernel $\ker f$ does not coincide with K_2, we can always apply a change of basis as described in Subsection 2.3. Hence, we can assume that $\ker f = K_2$.

Before outlining the explicit procedure, let us assume that we have T''_1, T''_2 such that $\langle T''_1, T''_2\rangle[4] = \langle T'_1, T'_2\rangle$, $2T''_i = T'_i$ and their Weil pairing $e_8(T''_1, T''_2) = 1$.[7] These conditions are not restrictive since they are naturally verified for chains of $(2, 2)$-isogenies, which are our end goal. In particular, $\langle f(T''_1), f(T''_2)\rangle$ are two of the 4-torsion points inducing the theta-null point on B laying above the two 2-torsion points in K_2. Hence, the points $\langle f(T''_1), f(T''_2)\rangle$ lay above the canonical points in K_1 for the dual coordinates.

Let us remark that $f(T''_i) + 2f(T''_i) = -f(T''_i)$. So, as highlighted in Equation 1 and since we are on the Kummer, we have

$$\left(\tilde{\theta}^B_{00}(f(T''_1)) : \tilde{\theta}^B_{10}(f(T''_1)) : \tilde{\theta}^B_{01}(f(T''_1)) : \tilde{\theta}^B_{11}(f(T''_1))\right) =$$
$$\left(\tilde{\theta}^B_{10}(f(T''_1)) : \tilde{\theta}^B_{00}(f(T''_1)) : \tilde{\theta}^B_{11}(f(T''_1)) : \tilde{\theta}^B_{01}(f(T''_1))\right),$$

and

$$\left(\tilde{\theta}^B_{00}(f(T''_2)) : \tilde{\theta}^B_{10}(f(T''_2)) : \tilde{\theta}^B_{01}(f(T''_2)) : \tilde{\theta}^B_{11}(f(T''_2))\right) =$$
$$\left(\tilde{\theta}^B_{01}(f(T''_2)) : \tilde{\theta}^B_{11}(f(T''_2)) : \tilde{\theta}^B_{00}(f(T''_2)) : \tilde{\theta}^B_{10}(f(T''_2))\right).$$

[7] Such a subgroup $\langle T''_1, T''_2\rangle$ is said to be *isotropic*.

Define $(\alpha : \beta : \gamma : \delta)$ to be the dual theta-null point of B, i.e.

$$\left(\tilde{\theta}^B_{00}(0) : \tilde{\theta}^B_{10}(0) : \tilde{\theta}^B_{01}(0) : \tilde{\theta}^B_{11}(0)\right) = (\alpha : \beta : \gamma : \delta).$$

Then, combining Equation 2 with the above observations, we have

$$\mathcal{H} \circ \mathcal{S}(\theta^A_{00}(T_1''), \theta^A_{10}(T_1''), \theta^A_{01}(T_1''), \theta^A_{11}(T_1'')) = (x\alpha, x\beta, y\gamma, y\delta),$$
$$\mathcal{H} \circ \mathcal{S}(\theta^A_{00}(T_2''), \theta^A_{10}(T_2''), \theta^A_{01}(T_2''), \theta^A_{11}(T_2'')) = (z\alpha, w\beta, z\gamma, w\delta),$$

for some unknown x, y, z, t. Hence, we can recover the dual theta-null point $(\alpha : \beta : \gamma : \delta)$ for B, and in turn its theta-null point $\mathcal{H}(\alpha : \beta : \gamma : \delta)$.

Remark 6 (Technical Remark). It is possible to prove that all x, y, z, t must be different from zero. If it had not been the case, we would have ended up with a theta-null point with at least two zero coordinates. This is a contradiction since it implies we have more than a zero even theta-null coordinate of level $(2, 2)$ – see Subsection "Gluing Isogeny" for the definition of level-$(2, 2)$ theta coordinates.

In general, the dual theta-null point $(\alpha : \beta : \gamma : \delta)$ has all coordinates different from zero. The only exceptions can be found for certain cases of the gluing isogeny – we will discuss how to handle this case in Subsection 4.1. Once we have computed $(\alpha : \beta : \gamma : \delta)$, we can evaluate the isogeny f at any point P using (again) Equation 2. First, we compute $(x', y', z', w') = \mathcal{H} \circ \mathcal{S}\left((\theta^A_i(P))_i\right)$. Then $(\theta^B_i(f(P)))_i = \mathcal{H}(\alpha^{-1}x', \beta^{-1}y', \gamma^{-1}z', \delta^{-1}w')$. We give a detailed description in Algorithms 5 and 6.

Algorithm 5 Codomain

Input: Theta coordinates $(x_{T_1''} : y_{T_1''} : z_{T_1''} : w_{T_1''})$ of T_1'', theta coordinates $(x_{T_2''} : y_{T_2''} : z_{T_2''} : w_{T_2''})$ of T_2'', where T_i'' is a 8-torsion point laying above the K_2 part of the symplectic 4-torsion basis inducing the theta-null point.

Output: Dual theta-null point $(1 : \beta : \gamma : \delta)$, the inverse of the dual theta-null point $(1 : \beta^{-1} : \gamma^{-1} : \delta^{-1})$ and the theta-null point $(a' : b' : c' : d')$ on B. (▷) Case $\beta \cdot \gamma \cdot \delta \neq 0$

1: $(x\alpha, x\beta, y\gamma, y\delta) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_1''}, y_{T_1''}, z_{T_1''}, w_{T_1''})$
2: $(z\alpha, w\beta, z\gamma, w\delta) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_2''}, y_{T_2''}, z_{T_2''}, w_{T_2''})$
3: Compute $(x\alpha)^{-1}, (x\beta)^{-1}, (y\gamma)^{-1}, (y\delta)^{-1}, (z\alpha)^{-1}, (w\beta)^{-1}, (z\gamma)^{-1}, (w\delta)^{-1}$ via batched inversions.
4: $\beta \leftarrow x\beta \cdot (x\alpha)^{-1}$
5: $\gamma \leftarrow z\gamma \cdot (z\alpha)^{-1}$
6: $\delta \leftarrow w\delta \cdot (w\beta)^{-1} \cdot \beta$
7: $\beta^{-1} \leftarrow x\alpha \cdot (x\beta)^{-1}$
8: $\gamma^{-1} \leftarrow z\alpha \cdot (z\gamma)^{-1}$
9: $\delta^{-1} \leftarrow w\beta \cdot (w\delta)^{-1} \cdot \beta^{-1}$
10: $(a', b', c', d') \leftarrow \mathcal{H}(1, \beta, \gamma, \delta)$
11: **return** $(1, \beta, \gamma, \delta), (1, \beta^{-1}, \gamma^{-1}, \delta^{-1}), (a', b', c', d')$ (▷) Total cost: $8\mathbf{S} + 29\mathbf{M} + 1\mathbf{I}$

Algorithm 6 Evaluation

Input: Theta coordinates $(x_P : y_P : z_P : w_P)$ of P and the dual theta-null point $(1 : \beta^{-1} : \gamma^{-1} : \delta^{-1})$ on B.
Output: Theta coordinates $(x_{f(P)} : y_{f(P)} : z_{f(P)} : w_{f(P)})$ of $f(P)$. (▷) Case $\beta \cdot \gamma \cdot \delta \neq 0$

1: $(X_P, Y_P, Z_P, W_P) \leftarrow \mathcal{H} \circ \mathcal{S}(x_P, y_P, z_P, w_P)$
2: $(X'_{f(P)}, Y'_{f(P)}, Z'_{f(P)}, W'_{f(P)}) \leftarrow (X_P, \beta^{-1} \cdot Y_P, \gamma^{-1} \cdot Z_P, \delta^{-1} \cdot W_P)$
3: $(x_{f(P)}, y_{f(P)}, z_{f(P)}, w_{f(P)}) \leftarrow \mathcal{H}(X'_{f(P)}, Y'_{f(P)}, Z'_{f(P)}, W'_{f(P)})$
4: **return** $(x_{f(P)}, y_{f(P)}, z_{f(P)}, w_{f(P)})$ (▷) Total cost: $4\mathbf{S} + 3\mathbf{M}$

Remark 7. As we explained in Section 3, the inverse squared dual theta-null point $(\alpha^2/\beta^2, \alpha^2/\gamma^2, \alpha^2/\delta^2)$ are also needed for the addition and doubling formulae. If such a quantity has already been computed, we can use it to lower down the cost in Algorithm 5. In line 3, we need only to invert three elements, namely $x\alpha, z\alpha, w\beta$. Then, to obtain $(\beta^{-1}, \gamma^{-1}, \delta^{-1})$, we can simply multiply component-wise $(\alpha^2/\beta^2, \alpha^2/\gamma^2, \alpha^2/\delta^2)$ by (β, γ, δ). The total cost in this optimised case is $8\mathbf{S} + 9\mathbf{M} + 1\mathbf{I}$.

4.1 Computing $(2^n, 2^n)$-isogenies between Elliptic Products

Now, we specialise to the case of a $(2^n, 2^n)$-isogeny $f : E_1 \times E_2 \to E'_1 \times E'_2$ between elliptic products, which will be computed as a chain of $(2, 2)$-isogenies. Let K be the kernel of this isogeny and suppose that we have two points of order 2^{n+2} on $E_1 \times E_2$ above K forming an isotropic group. To apply the formulae we described above, we need to be sure that $K[4]$ is in "the right position". Given $K[4]$, we apply Algorithm 2 to obtain a theta-null point induced by the symplectic 4-torsion decomposition $\langle S'_1, S'_2 \rangle \oplus \langle T'_1, T'_2 \rangle$, where $K[4] = \langle T'_1, T'_2 \rangle$

If $n > 2$, we have that $K[8] = \langle T''_1, T''_2 \rangle$ is the isotropic 8-torsion above $K[4]$. This means we could apply Algorithms 5 and 6 to compute the first step of the isogeny f, i.e. the isogeny $f_1 : E_1 \times E_2 \to A_1$ with kernel $K[2]$. However, we should be careful as on the product structure, one of the coordinates of the dual theta-null point on A_1 may be equal to zero. We explain why this happens and how to bypass this obstacle in the Subsection "Gluing Isogeny" below.

After the first step, we also end up with a complete description of the theta structure on A_1; let $A_1[2] = K_1 \oplus K_2$. The points $f_1(T''_1)$ and $f_1(T''_2)$ are two of the 4-torsion elements describing the theta-null point on A_1. If $n > 3$, we can use $f_1(K)[8]$ to describe the 8-torsion above $\langle f_1(T''_1), f_1(T''_2) \rangle$ and iterate the process.

Once we reach the second last step $f_{n-1} : A_{n-2} \to A_{n-1}$, we cannot inherit the 8-torsion above the K_2 part of the A_{n-2} anymore. However, thanks to the assumption that we have to two points of order 2^{n+2} on $E_1 \times E_2$ above K forming an isotropic group, we can use the same exact strategy using the images of such points. We explain how to relax this condition in Subsection 4.2.

In the last step $f_n : A_{n-1} \to E'_1 \times E'_2$, we map onto an elliptic product. Even though we can reuse the same computational strategies when we stay in the theta model, for most of the cryptographic applications we have to explicitly recover the equations of the curves E'_1 and E'_2, and we also have to have map points onto these curves. We describe how to do so in the Subsection "Splitting Isogeny".

Gluing Isogeny In this subsection, we focus on the first step of the isogeny chain, an isogeny originating from an elliptic product; let $f : E_1 \times E_2 \to A$ be such an isogeny. Theta structures on elliptic products $E_1 \times E_2$ verify some additional properties with respect to level-$(2,2)$ theta coordinates. We refer to Dupont's PhD thesis [9] for background material. For the case at hand, we briefly recall some fundamental facts.

Theta coordinates of level $(2,2)$ are indexed by a pair of elements in $K(2,2)$. A level-$(2,2)$ theta coordinate $U_{i,j}$ is said to be *even* if $i \cdot j^T = 0 \pmod 2$; otherwise it is said to be *odd*. Moreover, at most one of the even indices (i,j) verifies $U_{i,j}(0) = 0$, and there is exactly one zero even index if and only if the theta structure is associated with a product of two elliptic curves.

Given level-2 theta coordinates $(\theta_i(P))_i$, we can compute the square of its level-$(2,2)$ theta coordinates as

$$U^2_{i,j}(P) = \sum_t (-1)^{i \cdot t^T} \theta_t(P)\theta_{t+j}(P).$$

In [9, Proposition 6.5], Dupont shows that a theta-null point $(\theta_i(0))_i$ comes from the product theta structure of two elliptic curves if and only if $U_{11,11}(0) = 0$. This means that if we are working on a product structure, all the coordinates of the dual theta-null point $(\alpha : \beta : \gamma : \delta)$ are non-zero since $(U_{00,00}(0) : U_{10,00}(0) : U_{01,00}(0) : U_{11,00}(0)) = (\alpha : \beta : \gamma : \delta)$. However, when we perform a change of basis, we might move the zero even index of the level-$(2,2)$ theta-null point around, and potentially we might have one of the dual theta-null coordinates equal to zero.

In fact, unless A is a product of elliptic curves (which would be the case if the kernel is a product kernel), then we know that one of $\alpha, \beta, \gamma, \delta$ is zero. If it were not the case, we could compute $f(P)$ from P. But since we work with theta coordinates of level 2, on the product $E_1 \times E_2$ we are really working with the product of Kummer lines $E_1/\pm 1 \times E_2/\pm 1$. The automorphism group by which we quotient is thus $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ compared to $\mathbb{Z}/2\mathbb{Z}$ when working on the Kummer surface $A/\pm 1$ of an abelian surface with a non product principal polarisation. Thus, when going from P to $f(P)$, there is an ambiguity coming from an action of $\mathbb{Z}/2\mathbb{Z}$, which can only be resolved by either taking a square root, or as we will explain next, by using extra information coming from the arithmetic of $E_1 \times E_2$ (rather than $E_1/\pm 1 \times E_2/\pm 1$).

Let us handle the case where one of the coordinates of the dual theta-null point $(\alpha : \beta : \gamma : \delta)$ is zero; let us first analyse the case $\alpha = 0$. In Algorithm 5, we normalised everything with respect to δ. This actually simplifies the codomain computation. We explain how to do so in Algorithm 7.

As explained above, mapping points under this isogeny requires extra care. If we simply use Algorithm 6, we cannot retrieve the first coordinate of a point. To be precise, if we want to evaluate f at the point $(\theta_i^{E_1 \times E_2}(P))_i$, we have

$$\mathcal{H} \circ \mathcal{S}((\theta_i^{E_1 \times E_2}(P))_i) = (0, \beta\tilde{\theta}^A_{10}(f(P)), \gamma\tilde{\theta}^A_{01}(f(P)), \delta\tilde{\theta}^A_{11}(f(P))). \tag{4}$$

Multiplying by β^{-1}, γ^{-1} and δ^{-1} the components $\beta\tilde{\theta}^A_{10}(f(P)), \gamma\tilde{\theta}^A_{01}(f(P)), \delta\tilde{\theta}^A_{11}(f(P))$, we retrieve all the dual components but $\tilde{\theta}^A_{00}(f(P))$.

The component $\tilde{\theta}^A_{00}(f(P))$ can be computed using the additional information coming from the theta structure. Let T_1' be the point above $T_1 \in K_2$ as in the previous

Algorithm 7 Special Codomain, $\alpha = 0$

Input: In the same notation as before, theta coordinates $(x_{T_1''} : y_{T_1''} : z_{T_1''} : w_{T_1''})$ of T_1'', theta coordinates $(x_{T_2''} : y_{T_2''} : z_{T_2''} : w_{T_2''})$ of T_2'', where T_i'' is a 8-torsion point laying above the K_2 part of the symplectic 4-torsion basis inducing the theta-null point.

Output: Dual theta-null point $(0 : \beta : \gamma : 1)$, the "inverse" of the dual theta-null point $(0 : \beta^{-1} : \gamma^{-1} : 1)$ and the theta-null point $(a' : b' : c' : d')$ on A. (▷) Case $\alpha = 0$

1: $(0, x\beta, y\gamma, y\delta) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_1''}, y_{T_1''}, z_{T_1''}, w_{T_1''})$
2: $(0, w\beta, z\gamma, w\delta) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_2''}, y_{T_2''}, z_{T_2''}, w_{T_2''})$
3: Compute $(y\gamma)^{-1}, (w\beta)^{-1}, (y\delta)^{-1}, (w\delta)^{-1}$ via batched inversions.
4: $\beta \leftarrow w\beta \cdot (w\delta)^{-1}$
5: $\gamma \leftarrow y\gamma \cdot (y\delta)^{-1}$
6: $\beta^{-1} \leftarrow w\delta \cdot (w\beta)^{-1}$
7: $\gamma^{-1} \leftarrow y\delta \cdot (y\gamma)^{-1}$
8: $(a', b', c', d') \leftarrow \mathcal{H}(0, \beta, \gamma, 1)$
9: **return** $(0, \beta, \gamma, 1), (0, \beta^{-1}, \gamma^{-1}, 1), (a', b', c', d')$ (▷) Total cost: $8\mathbf{S} + 13\mathbf{M} + 1\mathbf{I}$

subsection. Then,

$$\mathcal{H} \circ \mathcal{S}((\theta_i^{E_1 \times E_2}(P + T_1'))_i) = (0, \beta\tilde{\theta}_{00}^A(f(P)), \gamma\tilde{\theta}_{11}^A(f(P)), \delta\tilde{\theta}_{01}^A(f(P))). \tag{5}$$

However, multiplying the component $\beta\tilde{\theta}_{00}^A(f(P))$ by β^{-1} is not enough since we are working up to projective factors.

Once we recover $\tilde{\theta}_{10}^A(f(P)), \tilde{\theta}_{01}^A(f(P)), \tilde{\theta}_{11}^A(f(P))$ from Equation 4, we can compute $\lambda\tilde{\theta}_{01}^A(f(P))$ from Equation 5 for some projective factor λ: we simply multiply the last component of $\mathcal{H} \circ \mathcal{S}((\theta_i^{E_1 \times E_2}(P + T_1'))_i)$ by δ^{-1}. If $\tilde{\theta}_{01}^A(f(P)) \neq 0$, we can actually compute the inverse of the projective factor by $\tilde{\theta}_{01}^A(f(P))/(\lambda\tilde{\theta}_{01}^A(f(P)))$. Otherwise, we repeat the same process with the second last component of $\mathcal{H} \circ \mathcal{S}((\theta_i^{E_1 \times E_2}(P + T_1'))_i)$.

Once we have λ, we extract $\tilde{\theta}_{00}^A(f(P))$ from the second component of $\mathcal{H} \circ \mathcal{S}((\theta_i^{E_1 \times E_2}(P + T_1'))_i)$: we multiply the second component of $\mathcal{H} \circ \mathcal{S}((\theta_i^{E_1 \times E_2}(P + T_1'))_i)$ by $\lambda^{-1} \cdot \beta^{-1}$. Finally, we obtain the image of the point P under f via

$$\mathcal{H}(\tilde{\theta}_{00}^A(f(P)), \tilde{\theta}_{10}^A(f(P)), \tilde{\theta}_{01}^A(f(P)), \tilde{\theta}_{11}^A(f(P))).$$

We summarise everything in Algorithm 8. Note that for both Algorithm 7 and 8, the case for β, γ or $\delta = 0$ follows almost identically, see the implementation for a concrete example of how all cases can be considered concisely. We note that Algorithm 8 requires the knowledge not only of the theta coordinates of P, but also of $P + T_1'$. From the knowledge of the $(\theta_i(P))$ and $(\theta_i(T_1'))$, we may only recover $(\theta_i(P \pm T_1'))$, hence extract $(\theta_i(P + T_1'))$ via a square root, consistent with the fact that in a gluing isogeny we have an ambiguity for images coming from an action by $\mathbb{Z}/2\mathbb{Z}$. Luckily we can compute this addition on each elliptic curve separately, using Weierstrass coordinates, before switching to the level-2 theta coordinates on the surface $E_1 \times E_2$.

Splitting Isogeny In this last step of the isogeny chain, we need to compute an isogeny $f : A \to E_1' \times E_2'$ mapping onto an elliptic product. We can compute the theta-null point of $E_1' \times E_2'$ and mapping points under f using Algorithms 5 and 6. However,

Algorithm 8 Special Evaluation, $\alpha = 0$

Input: Theta coordinates $(x_P : y_P : z_P : w_P)$ of P, theta coordinates $(x_{P+T_1'} : y_{P+T_1'} : z_{P+T_1'} : w_{P+T_1'})$ of $P + T_1'$ and the inverse of the dual theta-null point $(0 : \beta^{-1} : \gamma^{-1} : 1)$ on A.

Output: Theta coordinates $(x_{f(P)} : y_{f(P)} : z_{f(P)} : w_{f(P)})$ of $f(P)$. (▷) Case $\alpha = 0$

$(0, Y_P, Z_P, W_P) \leftarrow \mathcal{H} \circ \mathcal{S}(x_P, y_P, z_P, w_P)$
$(0, Y_{P+T_1}, Z_{P+T_1}, W_{P+T_1}) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{P+T_1} : y_{P+T_1} : z_{P+T_1} : w_{P+T_1})$
$(Y'_{f(P)}, Z'_{f(P)}, W'_{f(P)}) \leftarrow (\beta^{-1} \cdot Y_P, \gamma^{-1} \cdot Z_P, W_P)$
if $Z'_{f(P)} \neq 0$ **then**
    $\lambda^{-1} \leftarrow Z'_{f(P)} / W_{P+T_1}$
else
    $Z'_{f(P+T_1)} \leftarrow \gamma^{-1} \cdot Z_{P+T_1}$
    $\lambda^{-1} \leftarrow W'_{f(P)} / Z'_{P+T_1}$
$X'_{f(P)} \leftarrow \lambda^{-1} \cdot \beta^{-1} \cdot Y_{P+T_1}$
$(x_{f(P)}, y_{f(P)}, z_{f(P)}, w_{f(P)}) \leftarrow \mathcal{H}(X'_{f(P)}, Y'_{f(P)}, Z'_{f(P)}, W'_{f(P)})$
return $(x_{f(P)}, y_{f(P)}, z_{f(P)}, w_{f(P)})$ (▷) Total cost: $8\mathbf{S} + 5\mathbf{M} + 1\mathbf{I}$

we still need to retrieve the explicit equations for the curves E'_1 and E'_2. This can be done using level-$(2,2)$ theta coordinates $U_{i,j}$.

Since the theta structure on the image surface underlies an elliptic product, we know that one of the even indices – say (i, j) – of the level-$(2,2)$ theta-null point is equal to zero. Also, we know that if we compute a symplectic automorphism ψ mapping (i, j) onto $(11, 11)$, the action of ψ on the theta-null point obtained via Algorithm 5 gives back a theta-null point associated with the product theta structure.

From the above, it can be seen that there are ten distinct even indices. For each of these indices, we computed a symplectic automorphism sending this index to $(11, 11)$. For efficiency reasons, we hardcoded the action of each of the symplectic automorphisms onto theta coordinates of level 2 in the reference implementation. These symplectic automorphisms and their actions have been derived from [31, p. 28] using the following sequential steps.

Let (i, j) be the even index such that $U_{i,j}(0) = 0$, and, for ease of notation, let $(a_{00} : a_{10} : a_{01} : a_{11})$ be the underlying theta-null point.

1. If $i = j = 00$, we act by the symplectic automorphism with matrix form

$$\begin{pmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

   We then obtain the theta-null point $(a_{00} : \sqrt{-1} \cdot a_{10} : a_{01} : \sqrt{-1} \cdot a_{11})$, which means that $U_{10,00}(0) = 0$.
2. If $j = 00$ and $i \neq 00$, we act by $\mathcal{H}$, which swaps the roles of i and j. We can now assume that $j \neq 00$.

3. Let A be any invertible matrix such that $A \cdot j^T = 11^T$. Then, the action of the symplectic automorphism with matrix

$$\begin{pmatrix} A & 0 \\ 0 & A^{T^{-1}} \end{pmatrix}$$

maps the theta-null point $(a_{00} : a_{10} : a_{01} : a_{11})$ to $(a_{00} : a_{10\cdot A^T} : a_{01\cdot A^T} : a_{11\cdot A^T})$. This means that $U_{i',j'}(0) = 0$, where $i' = i \cdot A$ and $j' = 11$. We can now assume that $j = 11$.
4. Now, either $i = 00$ or $i = 11$. If $i = 11$, we are done. Otherwise, we act by the symplectic automorphism with matrix form

$$\begin{pmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

We then obtain the theta-null point $(a_{00} : \sqrt{-1}\cdot a_{10} : \sqrt{-1}\cdot a_{01} : a_{11})$, which means that $U_{11,11}(0) = 0$.

Thus, we can assume we are now working on the product theta structure.

If $(a : b : c : d)$ is theta-null point on $E_1 \times E_2$, from Proposition 2, it follows that $(a : b)$ is a theta-null point for E_1 and $(b : d)$ is a theta-null point for E_2. Also, if $f(P) = (P_1, P_2) \in E_1 \times E_2$ is represented in theta coordinates as $(x : y : z : w)$, we have that $(x : y)$ is the representation of P_1 in theta coordinates for E_1 and $(y : w)$ is the representation of P_2 in theta coordinates for E_2. Finally, to convert from theta coordinates to the Montgomery model we can use the formulae in [1].

4.2 Computing Isogenies without Extra Isotropic Information

In this subsection, we relax the condition on the two points of order 2^{n+2} on $E_1 \times E_2$ above K forming an isotropic group. This does not represent a problem when computing a $(2^n, 2^n)$-isogeny, except for the two last steps. We discuss two cases: when we can work with 2^{n+2}-torsion, and when we cannot.[8]

Let us discuss the former case. Let $K = \langle P_1, P_2\rangle \subset E_1 \times E_2$. To apply the previous algorithm, we would like to have an isotropic $\langle P_1'', P_2''\rangle$ above K such that $P_i = [4]P_i''$. However, it suffices to pick up any Q_1'', Q_2'', not necessarily isotropic, as long as $P_i = [4]Q_i''$. Indeed, one can check that applying the algorithm of section 4.1 on these Q_i'' gives a theta-null point that differs from the one given by isotropic P_i'' by an automorphism of the theta group (see Subsection 2.1) induced by a symplectic automorphism,. Hence, it still corresponds to the correct codomain, but with a different theta structure.

In the latter case, we cannot use the 2^{n+2}-torsion at all. A way to circumvent this problem is to use square roots to compute the codomains for the last two steps. Once we have the codomain, the image evaluation is unaffected. There is no way to avoid the

[8] For instance, it is preferable not to work with the 2^{n+2}-torsion when it is defined over a field extension of the base field.

square root computations: the theta-null point requires a theta structure of level 2, so in particular a full basis of the 2-torsion and some extra information on the 4-torsion. If we don't have the 2^{n+2}-torsion at the beginning, we miss the necessary information on the 4-torsion at the penultimate step and on the 2-torsion on the last step. To reconstruct this information requires making choices, hence taking square roots.

At the penultimate step $f : A \to B$, we have T_1' and T_2' of 4-torsion but not the 8-torsion points T_1'' and T_2'' anymore. This means that on the codomain, we only have the 2-torsion determined. We have several choices of possible compatible theta structure, but we still want to use the information at hand.

Let $(\alpha : \beta : \gamma : \delta)$ be the dual theta-null point on B. Applying Equation 2 to the theta-null point $(a : b : c : d)$, we have

$$\mathcal{H} \circ \mathcal{S}(a : b : c : d) = (\alpha^2 : \beta^2 : \gamma^2 : \delta^2). \tag{6}$$

Also, since $f(T_1')$ is in K_1 for the dual theta structure, we have

$$\left(\tilde{\theta}_i^B(f(T_1'))\right)_i = (\beta : \alpha : \delta : \gamma).$$

Therefore, from Equation 2,

$$\mathcal{H} \circ \mathcal{S}(\left(\theta_i^A(T_1')\right)_i) = (\alpha\beta, \alpha\beta, \gamma\delta, \gamma\delta). \tag{7}$$

Fix $\alpha = 1$. From Equation 6, we can compute any square root of β^2 for β and any square root of γ^2 for γ. From Equation 7 and β we can recover the correct lifting of $\gamma\delta$, and in turn, we can recover δ. The four choices we can make on the square roots of γ^2 and δ^2 describe different theta structures underlying the same abelian surface since they differ by the action of a symplectic automorphism [35, Example B.3].

At the last step, we only have T_1 and T_2. As a result, we can only recover the squares $(\alpha^2 : \beta^2 : \gamma^2 : \delta^2)$ of the dual theta-null point $(\alpha : \beta : \gamma : \delta)$. We can fix $\alpha = 1$ and compute β, γ, δ via three square roots. Once again, we can check that these 8 choices all come from a valid theta structure [35, Example B.3].

To sum up, if the 2^{n+2}-torsion is available, we need no square root. If the 2^{n+1}-torsion is available, we need two square roots. If only the 2^n-torsion is available, we need $2 + 3 = 5$ square roots.

5 Implementation

We have implemented the computation of an isogeny between elliptic products in the theta model using both the programming language Rust and the computer algebra system SageMath. The SageMath implementation has been designed to follow the API of isogenies between elliptic curves and is intended to be a tool in both experimentation and in constructing proof-of-concept implementations of isogeny-based cryptographic primitives. For those who have previously relied on the SageMath implementation of [26], the function `EllipticProductIsogeny(kernel, n)` has been designed to be a drop-in replacement for the $(2^n, 2^n)$-isogeny computed using via the Richelot correspondence and the algorithms presented in [36].

The Rust implementation has been designed with constructive cryptographic implementations in mind, and in particular, it has been written to be constant time.[9] The finite field arithmetic and certain elliptic curve functions have been adapted from the `crrl` library [28] maintained by Thomas Pornin as well as other ongoing collaborations. An effort has been made to ensure the code is (reasonably) flexible so that without too much tweaking, this work can be ported to other Rust projects. As an example of this flexibility, we show timings of isogenies of various lengths between elliptic products over three distinct base fields.

Both the SageMath and Rust implementations are made available via the following GitHub repository:

https://github.com/ThetaIsogenies/two-isogenies

5.1 Performance

In this subsection, we include the performance of our algorithm for three distinct isogeny chains between elliptic products over a range of base fields. We include the timings for both the constant-time Rust implementation as well as the proof-of-concept SageMath implementation, together with a comparison to previous work on isogenies between elliptic products in the Mumford model [26] using the optimisations introduced in the implementation of [2].

Table 2. Running times of computing the codomain and evaluating a $(2^n, 2^n)$-isogeny between elliptic products over the base field $\mathbb{F}_{p^2}$. Times were recorded on a Intel Core i7-9750H CPU with a clock-speed of 2.6 GHz with turbo-boost disabled.

		Codomain			Evaluation		
$\log p$	n	Theta Rust	Theta SageMath	Richelot SageMath [26]	Theta Rust	Theta SageMath	Richelot SageMath [26]
254	126	**2.85 ms**	108 ms	1028 ms	**161 μs**	5.43 ms	114 ms
381	208	**11.2 ms**	201 ms	1998 ms	**411 μs**	8.68 ms	208 ms
1293	632	**495 ms**	1225 ms	12840 ms	**17.8 ms**	40.8 ms	1203 ms

This triplet of comparisons has a twofold advantage. Firstly, the Rust implementation we present is the first (to our knowledge) constant time implementation of dimension two isogenies between elliptic products. By including the timings of both our Rust implementation and the SageMath implementation, we hope that researchers can estimate a performance gain if they were to write efficient and cryptographically minded implementations following the proof-of-concept scripts which currently exist in the higher-dimensional isogeny-based cryptography literature.

[9] The implementation assumes the kernel generators are *good* with respect to them generating an isogeny between elliptic products. Designing the algorithm to run in constant time with malformed input extends beyond the goals of this paper but may be necessary for protection against side-channel attacks against schemes which rely on this algorithm.

Secondly, our SageMath implementation allows an honest comparison of the isogenies in the theta model to the Richelot isogenies in the Mumford model. We compare against the implementation of [26] together with the additional optimisations introduced for the proof-of-concept of [2] which offered more than a two times speed up by optimising both the arithmetic on Jacobians as well as the isogenies themselves.

We note here that an alternative and faster implementation of $(2,2)$-isogenies in the Mumford model is available [14]. However, this algorithm requires a precomputation for a symplectic basis derived from the starting elliptic product of the isogeny chain. In the case of Kunzweiler's work – for example in genus two SIDH [11], or for applications to the SIDH attack – this optimisation is natural. However, in the more generic case, the starting product is unknown until run-time, and this basis would been to be computed on the fly. Despite this, the work of [14] offers (approximately) a two times speed up to [26], which is similar (but distinct) to the optimisations included in the implementation of [2]. Kunzweiler also has an unpublished implementation in SageMath of $(2,2)$-isogenies using Kummer surfaces in the Mumford model rather than in the Jacobian model. This is a fairer comparison because we work with level-2 theta coordinates, so also on Kummer surfaces. On the Kummer surface, the codomain computation in the theta model is around five times faster than in the Mumford model, and images around three times faster. Although theta coordinates are faster, working in the Mumford model is interesting if the level-2 theta coordinates are not rational, so using theta coordinates would require to work on a field extension. Since our domain is a product of elliptic curves, the theta coordinates are rational when each elliptic curve is described by rational theta coordinates. By [1], this is the case for an elliptic curve E whenever $E[4] = K_1' \oplus K_2'$ and the K_i' are rational subgroups (equivalently they are generated by a 4-torsion point T_i' which is rational on the Kummer line $E/\pm 1$). In particular, a supersingular elliptic curve $E/\mathbb{F}_{p^2}$ always has rational level-2 theta coordinates (assuming p odd). However, if E is a supersingular elliptic curve defined over $\mathbb{F}_p$, the theta coordinates are rational only if E lies at the top of the 2-isogeny volcano.

The run-times displayed in Table 2 were captured on a Intel Core i7-9750H CPU with a clock-speed of 2.6 GHz with turbo-boost disabled for stable measurements. The Rust code was compiled with the Rust compiler version `1.75.0-nightly` with the flag `-C target-cpu=native` to allow the compiler to use CPU specific opcodes (specifically, `mulx` for the finite field arithmetic). The arithmetic is written using Rust, rather than optimised assembly for each base field; the inclusion of which would allow dramatically faster results, especially for base fields with large characteristic. This form of optimisation is better suited to particular protocols, and we would expect to see this in optimised implementations of isogeny-based cryptographic primitives.

Comparing our SageMath implementation to the isogeny chain in the Mumford model, we find that the codomain computation is consistently faster by a factor of ten, while the image computation is more than twenty times faster. For smaller characteristic, the Rust implementation is approximately forty times faster than the same algorithm written in SageMath, but this gap closes significantly for larger primes. For example, the FESTA sized parameters run only 2.5 times faster than the SageMath

code. Note that the Rust implementation has been written to run in constant time and so the underlying arithmetic between these two implementations is incomparable.[10]

Comparison with Dimension One In Table 3, we provide a timing comparison using SageMath between a 2^n-isogeny in dimension one using the efficient formulae of [29,27] to our formulae in dimension two over the same base field. The dimension two isogeny has degree 2^{2n} so is expected to be slower. Our timings show a consistent factor-two slow down both for the codomain and image computations in dimension two compared to dimension one. This is the best we could hope given the degrees, and actually slightly better than expected. The dominating costs of a 2^n-isogeny are the intermediate doublings and images.

In the following we consider the more costly doublings and images in dimension two which arise from avoiding inversion when computing the codomain. The cost of doubling in dimension one is $4\mathbf{M} + 2\mathbf{S}$ compared to $8\mathbf{M} + 8\mathbf{S}$ in dimension two, and an image is $4\mathbf{M}$ compared to $4\mathbf{M} + 4\mathbf{S}$ in dimension two. Thus, while images are twice slower, doublings are around $2.5\times$ slower, and the intermediate codomain computations are also slower. Furthermore, a lot of doublings are done on the first step of the chain to get the first kernel, so on the elliptic product.

While it might seem at first glance that these doublings would only occur a factor $2\times$ slowdown, in practice, for the gluing images, we need to compute these points in affine (x, y) coordinates rather than x-only coordinates to allow access to addition laws[11] So, all in all, we should expect a slowdown at least greater than $2\times$ for perfect implementations. Our benchmarks show a slowdown slightly less than that, making two-dimensional isogenies perform a little better than expected. This is probably due to SageMath overhead and the fact the dimension one isogenies have been designed to allow arbitrary degree rather than only chains of two isogenies as we have in dimension two. A final caveat is that in dimension one, it is faster to split the 2^n-isogeny using the fast 4-isogenies from [6] rather than using 2-isogenies – we did not do that in our comparison because we do not have efficient 4-isogeny formulae in dimension two yet. Still, taking into account the degrees of the respective isogenies, this shows that our dimension two formulae are somewhat competitive with the best dimension one formulae.

5.2 Implementation details

In this subsection, we explain two optimisations we applied in the implementation. The first one is a direct consequence of Remark 7, where we describe how to lower the complexity of the codomain computation by reusing some constants. The second optimisation consists in the application of *optimal strategies* [10] to our case.

[10] It is not surprising to see this gap close though, as we expect for very large characteristic that the SageMath overhead becomes negligible compared to the cost of the arithmetic. As such, the comparisons of the two run-times boil down to comparing the Rust finite field arithmetic against the SageMath calls to the optimised arithmetic of the C libraries it is built upon.

[11] Rather than differential addition which requires knowledge of $x(P)$, $x(Q)$ and $x(P - Q)$.

Table 3. Comparison of the running times for a 2^n-isogeny in dimension one and dimension two over the same base field. Times were recorded on a Intel Core i7-9750H CPU with a clock-speed of 2.6 GHz with turbo-boost disabled.

		Codomain		Evaluation	
$\log p$	n	Montgomery	Theta	Montgomery	Theta
254	126	63 ms	108 ms	2.24 ms	5.43 ms
381	208	136 ms	201 ms	4.4 ms	8.68 ms
1293	632	727 ms	1225 ms	20 ms	40.8 ms

Reduce, Reuse, Recycle A simple and obvious optimisation is to reuse as many computations as possible throughout the isogeny chain. As mentioned in Remark 7, for each step on the isogeny chain, we can precompute six field elements at a cost of $4\mathbf{S} + 21\mathbf{M} + 1\mathbf{I}$. Knowledge of these values allows the doubling of any theta point on the corresponding theta structure to have a cost of $8\mathbf{S} + 6\mathbf{M}$, but it also allows the following codomain computation cost to be lowered to $8\mathbf{S} + 9\mathbf{M} + 1\mathbf{I}$.

In a similar way, by computing the codomain together with all the images we require at each step, we can precompute two additional field elements which can be reused for each evaluation, bringing the cost of an image to half that of doubling: $4\mathbf{S} + 3\mathbf{M}$.

For the gluing isogeny, the basis change is determined from the kernel and so cannot be precomputed. However, the last step at the end of the isogeny chain requires to find a symplectic transformation that maps the zero even index to the position $(11, 11)$. As there are only ten even indices, we can precompute ten symplectic transforms which map any given zero even index to $(11, 11)$. Computing the basis change is then only a matter of finding the the current zero index and from this, selecting the precomputed matrix and applying the transformation.

On Inversions At each step of the isogeny chain, we compute one inversion for the intermediate codomain. This inversion allows us to reduce the cost of doublings on this codomain from $8\mathbf{M}+8\mathbf{S}$ to $6\mathbf{M}+8\mathbf{S}$ and the cost of images from $4\mathbf{M}+4\mathbf{S}$ to $3\mathbf{M}+4\mathbf{S}$. However, at the end of the isogeny chain, there remain fewer doublings and images to compute, so it would be more efficient to skip this inversion and occur the higher cost. The precise cutoff point would depend on the relative cost of the inversion compared to a multiplication and the length of the chain. This optimisation has not yet been implemented in our code.

On Square Roots As explained in Section 4.2, when we do not have the 2^{n+2}-torsion available, we need to compute some square roots at the end of the chain (five square roots in total). This only changes the computation cost of the last two codomains, and do not affect the images computations. The longer the isogeny chain, the less impactful these square roots will be. Benchmarking our SageMath implementation, we observed that the impact of these five square roots is completely negligible for the chains we consider.

Optimal Strategies As is now standard with computing long isogeny chains, we can reduce the complexity of isogeny chains from a quadratic number of edges in the graph of doublings and evaluations to quasi-linear following the "optimal strategies" introduced in [10]. Essentially, the saving comes from reducing the total number of doublings when computing the kernel for each step in the chain by pushing through intermediate points encountered in the repeated doubling. For isogenies in the theta model, the cost of images is half that of doubling, and so shifting the cost in this way is particularly useful in optimisations.

Although this strategy was first discussed in dimension one for the case of isogenies between elliptic curves, using it in dimension two is a natural generalisation. For the dimension one case, the strategy is computed from balancing the costs of doubling and evaluating the kernel generator through the chain. In dimension two, the $(2, 2)$-isogeny is generated by a *pair* of elements which means twice the number of evaluations, but as the pair of elements must also be doubled to obtain the kernel for each step, essentially nothing changes. The cost weighting for the optimal strategies is a ratio between doublings and evaluations, which means we can naively use an identical method as described in [10] to compute a strategy for our isogeny chain.

Implementing the strategy with the weighting of doublings and images at a cost of $(2 : 1)$, we find an approximate ten times speed up in comparison to an implementation with no strategy. Concretely, for the Rust implementation of the isogeny chain of length $n = 208$, we see a speed up from 107 ms to 11.4 ms.

However, unlike the isogeny chains between elliptic curves, the isogeny chain between elliptic products in our implementation does not have the same costs for every step. For steps in the chain between generic theta structures, the cost weighting is indeed $(2 : 1)$. However, for the first gluing isogeny, doubling an element on the product structure has a cost of $12\mathbf{S} + 12\mathbf{M}$ while the cost for the image is much more expensive.

To compute the image of a point $P \in E_1 \times E_2$ one must first compute the shift $P + T_1'$ for a cost of $10\mathbf{S} + 32\mathbf{M}$ to projectively add a pair of points. Then, for each of these two points on the product, there is a cost of $4\mathbf{M}$ to compute the corresponding theta point from elliptic curve coordinates, and an additional $16\mathbf{M}$ required perform the matrix multiplication for the basis change to ensure a compatible representation. Altogether, this precomputation costs $10\mathbf{S} + 72\mathbf{M}$. Given the theta point corresponding to the pair of points on the product structure, there is still then the final cost of $8\mathbf{S} + 5\mathbf{M} + 1\mathbf{I}$ for the special image itself. Furthermore, for this to be implemented in constant time, both branches depending on whether a coordinate is zero or not must be evaluated, raising the practical cost to $8\mathbf{S} + 10\mathbf{M} + 1\mathbf{I}$.

All things considered, a gluing image costs $18\mathbf{S} + 82\mathbf{M} + 1\mathbf{I}$, making it approximately seven times the cost of the doubling for this first step and fourteen times the cost of a regular image. Visualising the graph of doublings and images as in [10, Figure 2], this means we must weigh the cost of moving down the left most branch with the product doubling and the first step right from the leftmost branch with this high-cost gluing image.

Taking this into account, an optimised strategy for the isogeny between elliptic products for our formula must be tweaked from the original case to find the right balance of doublings and expensive images from this left branch. Applying this modi-

fication, we are able to find the "proper" optimised strategy, which further reduces the run-time of the isogeny chain computation by approximately 2%.[12] For the same chain as above, we see a computation time improve from 11.4ms to 11.2ms. For an explicit description for computing the optimised strategy with a different costs on the left-most branch, see the implementation.

5.3 An Application: FESTA

As an explicit, cryptographic example of the new isogeny formula, we can take our implementation and use it to compute the isogeny between elliptic products which is required within the decryption algorithm of the isogeny-based public key encryption protocol FESTA-128 [2]. Concretely, this requires computing an isogeny of length $n = 632$, where the base field has a characteristic with $\log p = 1293$ bits, and the evaluation of a pair of points on the elliptic product $L_1 = (R_1, R_2)$ and $L_2 = (S_1, S_2)$, $L_i \in E_1 \times E_2$.

A direct swap from the isogeny chain derived from the Richelot correspondence used in the FESTA proof-of-concept would require using Section 4.2 to compute the final two steps without the eight-torsion above the kernel. An implementation of this is available in SageMath, but for the purpose of FESTA, we instead propose to tweak the 128-bit parameter set to instead allow for the additional torsion information to be known, allowing the isogeny chain to be computed as fast as possible while only including an additional 2-bits in the masked torsion data.

Concretely, we propose the following parameters for FESTA-128 using the same notations as in the original paper

$$\begin{aligned}
b &:= 632,\\
d_1 &:= (3^3 \cdot 19 \cdot 29 \cdot 37 \cdot 83 \cdot 139 \cdot 167 \cdot 251 \cdot 419 \cdot 421 \cdot 701 \cdot 839 \cdot 1009 \cdot \\
&\quad 1259 \cdot 3061 \cdot 3779)^2,\\
d_2 &:= 7 \cdot (5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 41 \cdot 43 \cdot 71 \cdot 89 \cdot 127 \cdot 211 \cdot 281 \cdot 503 \cdot 631 \cdot \\
&\quad 2309 \cdot 2521 \cdot 2647 \cdot 2729)^2,\\
d_{A,1} &:= (59 \cdot 6299 \cdot 6719 \cdot 9181)^2,\\
d_{A,2} &:= (3023 \cdot 3359 \cdot 4409 \cdot 5039 \cdot 19531 \cdot 22679 \cdot 41161)^2,\\
m_1 &:= 1492184945093476592520242083925044182103921,\\
m_2 &:= 256173313364299393300166693069,\\
f &:= 4 \cdot 71.
\end{aligned}$$

In practice, we only tweak the cofactor f in the prime $p = 2^b d_1 (d_{A,1} d_{A,2})_{\mathsf{sf}} d_2 f - 1$ to be $f = 0 \pmod 4$.[13]

[12] As an aside, in the original discussion of the optimised costings, it is shown that a 2-3% improvement is gained by moving from a balanced to optimised strategy. Seeing a similar saving from the naive $(2:1)$ weighted optimisation to one carefully handling the cost of the gluing step is then within our expectations.

[13] The notation t_{sf} refers to the square-free part of the integer t.

We find that the our SageMath implementation of the codomain computation has a ten times speed up compared to the proof-of-concept code accompanying [2], and evaluating the pair of points is now thirty times faster. As a hint to what approximate running times may be for FESTA, computing the codomain and both images using our Rust implementation takes only 563ms, a 2.5 times speed up over the SageMath implementation. Note that these computation are precisely that of the final row of Table 2. We again note that further optimisations of the finite field arithmetic could offer substantial speed ups, as seen in the optimised assembly implementations for large characteristic implementations of SIDH [13, Table 2.1] and the efficient algorithms of [16].

Within the context of the proof-of-concept decryption times (in SageMath), the novel algorithms we present here offer a four times speed up in decryption, with run-times for FESTA-128 being reduced from 20.7s to only 5.4s.[14] We see that through computing the dimension two isogeny in the theta model, the time spent computing a $(2^n, 2^n)$-isogeny shrinks from 70% of the run-time to only 25%, with the remaining computation time spent in dimension one, computing various discrete logs and Weil pairings to complete the decryption routine.

6 Conclusions

In this paper, we have described and implemented formulae to compute $(2^n, 2^n)$-isogenies between elliptic products in the theta model. The main goal was to provide a comprehensive and self-contained treatment of the theta model, specialising to the two-dimensional case. The extension of this work to dimension four is therefore a natural consequence, which will greatly benefit isogeny-based cryptography (see [7] for instance), as well optimising the dimension two case for other degrees.

Our algorithm significantly outperforms the previous method in [26,2]: in SageMath, the codomain computation is ten times faster, while the isogeny evaluation is more than twenty times faster. The implementation in Rust has been written to run in constant time, with cryptographic implementations in mind. It runs up to forty times faster than the same algorithm written in SageMath.

We tested our algorithm on the proof-of-concept implementation in [2] and showed a fourfold speed up in decryption, highlighting that the slowest part is now given by the computations in dimension one. Furthermore, our SageMath implementation has been designed to allow protocols whose implementation relies on the previous proof-of-concept in [2] to be easily upgradeable, allowing the theta-model code to be used in many more projects without too much work. Ultimately, the aim is to provide a new tool to facilitate research in higher-dimensional isogeny-based cryptography, allowing us to better understand the practical role of higher-dimensional isogenies in constructive applications.

[14] The speed up here is more than just a faster isogeny chain, but also some other speed-ups which come for free, as the previous implementation required further isomorphisms to ensure the codomain product curves were in the Montgomery model, but for the theta-model isogeny, we get this for free.

References

1. Barbulescu, R., Robert, D., Sarkis, N.: Models of Kummer lines and Galois representation (2023), Private communication
2. Basso, A., Maino, L., Pope, G.: FESTA: Fast Encryption from Supersingular Torsion Attacks. Cryptology ePrint Archive, Paper 2023/660 (2023), https://eprint.iacr.org/2023/660. To appear at ASIACRYPT 2023.
3. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH. pp. 423–447 (2023). https://doi.org/10.1007/978-3-031-30589-4_15
4. Chen, M., Leroux, A.: SCALLOP-HD: group action from 2-dimensional isogenies. Cryptology ePrint Archive, Paper 2023/1488 (2023), https://eprint.iacr.org/2023/1488
5. Cosset, R., Robert, D.: Computing (ℓ, ℓ)-isogenies in polynomial time on Jacobians of genus 2 curves. Mathematics of Computation **84**, 1953–1975 (2015). https://doi.org/10.1090/S0025-5718-2014-02899-8
6. Costello, C., Hisil, H.: A simple and compact algorithm for SIDH with arbitrary degree isogenies. pp. 303–329 (2017). https://doi.org/10.1007/978-3-319-70697-9_11
7. Dartois, P., Leroux, A., Robert, D., Wesolowski, B.: SQISignHD: New Dimensions in Cryptography. Cryptology ePrint Archive, Paper 2023/436 (2023), https://eprint.iacr.org/2023/436
8. Decru, T., Maino, L., Sanso, A.: Towards a Quantum-Resistant Weak Verifiable Delay Function. In: Aly, A., Tibouchi, M. (eds.) Progress in Cryptology - LATINCRYPT 2023. Lecture Notes in Computer Science, vol. 14168, pp. 149–168. Springer (2023). https://doi.org/10.1007/978-3-031-44469-2_8
9. Dupont, R.: Moyenne arithmético-géométrique, suites de Borchardt et applications. Ph.D. thesis, PhD thesis, École polytechnique (2006)
10. Feo, L.D., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. Cryptology ePrint Archive, Paper 2011/506 (2011), https://eprint.iacr.org/2011/506, https://eprint.iacr.org/2011/506
11. Flynn, E.V., Ti, Y.B.: Genus two isogeny cryptography. pp. 286–306 (2019). https://doi.org/10.1007/978-3-030-25510-7_16
12. Gaudry, P.: Fast genus 2 arithmetic based on theta functions. J. Math. Cryptol. **1**(3), 243–265 (2007). https://doi.org/10.1515/JMC.2007.012
13. Jao, D., Azarderakhsh, R., Campagna, M., Costello, C., De Feo, L., Hess, B., Jalali, A., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Renes, J., Soukharev, V., Urbanik, D.: Supersingular isogeny key encapsulation. Submission to https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization (2017), https://sike.org
14. Kunzweiler, S.: Efficient computation of $(2^n, 2^n)$-isogenies. Cryptology ePrint Archive, Paper 2022/990 (2022), https://eprint.iacr.org/2022/990
15. Leroux, A.: Verifiable random function from the Deuring correspondence and higher dimensional isogenies. Cryptology ePrint Archive, Paper 2023/1251 (2023), https://eprint.iacr.org/2023/1251
16. Longa, P.: Efficient Algorithms for Large Prime Characteristic Fields and Their Application to Bilinear Pairings. IACR Transactions on Cryptographic Hardware and Embedded Systems (3), 445–472 (Jun 2023). https://doi.org/10.46586/tches.v2023.i3.445-472
17. Maino, L., Martindale, C., Panny, L., Pope, G., Wesolowski, B.: A direct key recovery attack on SIDH. pp. 448–471 (2023). https://doi.org/10.1007/978-3-031-30589-4_16
18. Milne, J.S.: Abelian Varieties (v2.00) (2008), Available at www.jmilne.org/math/
19. Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. Mathematics of Computation **48**(177), 243–264 (1987). https://doi.org/10.1090/s0025-5718-1987-0866113-7

20. Moriya, T.: IS-CUBE: An isogeny-based compact KEM using a boxed SIDH diagram. Cryptology ePrint Archive, Paper 2023/1506 (2023), https://eprint.iacr.org/2023/1506
21. Mumford, D.: On the Equations Defining Abelian Varieties. I. Inventiones Mathematicae **1** (12 1966). https://doi.org/10.1007/BF01389737
22. Mumford, D.: On the Equations Defining Abelian Varieties. II. Inventiones Mathematicae **3** (01 1967). https://doi.org/10.1007/BF01389741
23. Mumford, D.: On the Equations Defining Abelian Varieties. III. Inventiones Mathematicae **3** (01 1967). https://doi.org/10.1007/BF01425401
24. Mumford, D.: Tata Lectures on Theta I. Birkhäuser, Boston (2007)
25. Nakagawa, K., Onuki, H.: QFESTA: Efficient Algorithms and Parameters for FESTA using Quaternion Algebras. Cryptology ePrint Archive, Paper 2023/1468 (2023), https://eprint.iacr.org/2023/1468
26. Oudompheng, R., Pope, G.: A Note on Reimplementing the Castryck-Decru Attack and Lessons Learned for SageMath. Cryptology ePrint Archive, Paper 2022/1283 (2022), https://eprint.iacr.org/2022/1283
27. Pope, G.: `KummerIsogeny`: Sagemath library for x-only isogenies (2023), https://github.com/GiacomoPope/KummerIsogeny
28. Pornin, T.: `crrl`: Rust library for cryptographic research, version 0.7.0 (2023), https://github.com/pornin/crrl
29. Renes, J.: Computing isogenies between montgomery curves using the action of (0,0). Cryptology ePrint Archive, Paper 2017/1198 (2017), https://eprint.iacr.org/2017/1198
30. Robert, D.: Fonctions thêta et applications à la cryptographie. Ph.D. thesis, Université Henry Poincaré - Nancy 1 (2010), PhD Thesis
31. Robert, D.: Efficient algorithms for abelian varieties and their moduli spaces (2021), Habilitation à Diriger des Recherches
32. Robert, D.: Evaluating isogenies in polylogarithmic time. Cryptology ePrint Archive, Report 2022/1068 (2022), https://eprint.iacr.org/2022/1068
33. Robert, D.: Some applications of higher dimensional isogenies to elliptic curves (overview of results). Cryptology ePrint Archive, Report 2022/1704 (2022), https://eprint.iacr.org/2022/1704
34. Robert, D.: Breaking SIDH in polynomial time. pp. 472–503 (2023). https://doi.org/10.1007/978-3-031-30589-4_17
35. Robert, D.: A note on optimising 2^n-isogenies in higher dimension (6 2023), Private communication.
36. Smith, B.A.: Explicit endomorphisms and correspondences. Ph.D. thesis (2005-12-23), http://hdl.handle.net/2123/1066
37. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 10.0) (2023), https://www.sagemath.org