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

Pierrick Dartois$^{1,2[0009-0008-2808-9867]}$, Luciano Maino$^{3[0009-0005-4495-5102]}$, Giacomo Pope^{3,4}, and Damien Robert$^{1,2[0000-0003-4378-4274]}$

[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 count.

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 [3,25,42] have highlighted the relevance of studying higher-dimensional abelian varieties in isogeny-based cryptography. Following the attacks, it soon became evident that these new tools would have applications beyond cryptanalysis. For instance, Robert leveraged these techniques both to give a representation of isogenies in polylogarithmic time [40] and to compute the endomorphism ring of ordinary elliptic curves in quantum polynomial time [41].

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, three additional protocols employing similar ideas to FESTA have appeared [27,33,28]. One of

Author list in alphabetical order; see https://www.ams.org//profession/leaders/CultureStatement04.pdf. The first and fourth authors have been supported by the Agence Nationale de la Recherche under grant ANR-19-CE48-0008 (CIAO) and the France 2030 program under grant ANR-22-PETQ-0008 (PQ-TLS). The second author has been supported by the UK Engineering and Physical Sciences Research Council (EPSRC) Centre for Doctoral Training (CDT) in Trust, Identity, Privacy and Security in Large-scale Infrastructures (TIPS-at-Scale) at the Universities of Bristol and Bath.

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, SQIsign2D-West [1] uses two-dimensional isogenies between elliptic products to achieve significant speed ups for keygen and signing as well as the fastest SQIsign verification to date. These isogenies are also 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 [15] and verifiable random functions [21]. 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 [45,34]. 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 [2,33], 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 [9], later improved in [24]. The case $\ell = 2$ has been briefly treated in [38, Proposition 6.3.5] and [39, Remarks 2.10.3, 2.10.7, 2.10.14] but never formalised.

The theta model is well known for its efficient arithmetic (in low dimension). For instance, Chudnovsky and Chudnovsky utilised the arithmetic of Kummer surfaces represented in the theta model for factoring integers [6]. Following this work, Gaudry derived fast formulae for the scalar multiplication on the Kummer surface associated to genus-two hyperelliptic curves. Moreover, in [37], Renes, Schwabe, Smith and Batina designed signature schemes for microcontrollers based on the efficient Montgomery ladder scalar multiplication on the Kummer surface.

Despite this efficient arithmetic, curiously, up until now, it had not really been considered for efficient $(2,2)$-isogenies between Kummer surfaces. A notable exception is the work of Costello [10]. Costello employs the theta model to translate the computation of 2^n-isogenies between elliptic curves defined over $\mathbb{F}_{p^2}$ to $(2^n,2^n)$-isogenies between Kummer surfaces over $\mathbb{F}_p$ via the Weil restriction. However, the $(2,2)$-isogenies considered by Costello are of a very special form, i.e. those deriving from very special kernels on the Weil restriction of elliptic curves. Our work can be seen both as a specialisation of the results of [39] to the case most interesting for isogeny-based cryptography (namely $(2^n, 2^n)$-isogenies between product of elliptic curves or Kummer surfaces), and as a generalisation of [10] to general $(2^n, 2^n)$-isogenies.

In this work, we mainly focus on an algorithm to compute $(2^n, 2^n)$-isogenies between products of elliptic curves using the theta model. Since the applications we have in mind fall within the realm of isogeny-based cryptography, we specialise in the case of chains of $(2, 2)$-isogenies whose intermediate abelian surfaces are all Jacobians of genus-two hyperelliptic curves and the kernel generators are all rational. Indeed, in all the current schemes involving two-dimensional isogenies, encountering elliptic products in the middle of these chains occurs with negligible probability. Still, we briefly explain how to extend our algorithms to treat all cases in Appendix A.

Our aim is 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 [17] and also give a precise operation count 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 count. 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. 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.

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.

Table 1. Base field costs of doubling, codomain computation and evaluation for generic (normalised and projective) 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. Computation of a codomain is given along with the precomputation cost which accounts for the one-time cost of computing field elements used when doubling and evaluating theta points along the isogeny chain.

Isogeny Type	Doubling	Codomain		Evaluation
		Precomputations	Codomain	
Normalised	$8\mathbf{S} + 6\mathbf{M}$	$4\mathbf{S} + 24\mathbf{M} + 1\mathbf{I}$	$8\mathbf{S} + 10\mathbf{M} + 1\mathbf{I}$	$4\mathbf{S} + 3\mathbf{M}$
Projective	$8\mathbf{S} + 8\mathbf{M}$	$5\mathbf{S} + 14\mathbf{M}$	$8\mathbf{S} + 7\mathbf{M}$	$4\mathbf{S} + 4\mathbf{M}$
Gluing	$12\mathbf{S} + 12\mathbf{M}$	—	$8\mathbf{S} + 13\mathbf{M} + 1\mathbf{I}$	$8\mathbf{S} + 10\mathbf{M} + 1\mathbf{I}$

Finally, we offer both a constant time implementation of the computation of an isogeny between elliptic products in the programming language Rust, as well as an alternative implementation for the computer algebra system SageMath [46]. 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 [34]. 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 [34]. For characteristic of size 254 bits, 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 (1293 bits). 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 2.13 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 the algorithm to construct theta structures on elliptic products; 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 [26, §10.3.1]. We refer to our implementation for an explicit description of the algorithm.[5]

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 and the anonymous reviewer for pointing out additional applications of the theta model outside isogeny-based cryptography.

2 Preliminaries

We assume the reader has some familiarity with (N,N)-isogenies between principally polarised abelian surfaces; we refer to [25, §2] for a cryptographer-friendly introduction to the subject (see also [7, Ch. V, §8] for a general introduction). Before giving an explicit description of the algorithm used to compute $(2^n,2^n)$-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 the algorithm to construct theta structures on elliptic products 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 [29,30,31]. 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.

[5] Python implementation of batched inversion

2.1 Mumford's Theory

In [44, §2.2], Robert and Sarkis provide a concrete treatment of Mumford's theory in the case of elliptic curves. In this section, we describe Mumford's theory for a principally polarised abelian variety (A, λ) of dimension g in a similar manner, working over an algebraically closed field k of characteristic different from two.

Assume that the principal polarisation λ is represented by a divisor Θ, which we will further assume to be symmetric: $[-1]^*\Theta = \Theta$ (we can always find such a representative). The principal polarisation is then given by $\lambda = \Phi_\Theta : A \to \hat{A}, P \mapsto t_P^*\Theta - \Theta$.[6]

We can then define the polarisation of level n: $\lambda \circ [n] : P \mapsto t_P^* n\Theta - n\Theta$. Its kernel coincides with $A[n]$. This means that, if $P \in A[n]$, $t_P^* n\Theta - n\Theta$ is a principal divisor. We will denote by g_P a function on A with this divisor; the function g_P is well defined up to multiplication by an invertible constant in k. This is a fundamental ingredient to introduce the *theta group*.

The theta group of level n is given by $\mathcal{G}(n\Theta) = \{(P, g_P), P \in A[n]\}$, with group law $(P, g_P) \cdot (Q, g_Q) = (P + Q, g_P(\_)g_Q(P + \_))$. Finally we have an (irreducible) action of $\mathcal{G}(n\Theta)$ on $\Gamma(n\Theta) = \{f \in k(A)^* \mid \mathrm{div}(f) \geq -n\Theta\} \cup \{0\}$, via $(P, g_P) \cdot f = g_P(\_)f(\_ + P)$. We also have an operator δ_{-1} on $\mathcal{G}(n\Theta)$: $\delta_{-1}(P, g_P) = (-P, [-1]^*g_P)$.

Let $\mathcal{G}(n)$ be the Heisenberg group $k^* \times K(n) \times \hat{K}(n)$ where $K(n) = (\mathbb{Z}/n\mathbb{Z})^g$ and $\hat{K}(n) = (\hat{\mathbb{Z}}/n\hat{\mathbb{Z}})^g$, where the multiplication is given by

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

We have an operator δ_{-1} on $\mathcal{G}(n)$ given by $\delta_{-1}(\alpha, x, \chi) = (\alpha, -x, 1/\chi)$, and an irreducible action of $\mathcal{G}(n)$ to $V(n)$, the vector space of functions $(\mathbb{Z}/n\mathbb{Z})^g \longrightarrow k$ generated by the Kronecker delta functions $\delta_i : i \in (\mathbb{Z}/n\mathbb{Z})^g$, via $(\alpha, x, \chi) \cdot \delta_i = \alpha\chi(i)\delta_{x+i}$.

For technical reasons, we will from now on assume that n is even. We will denote by $\mathcal{L}$ the line bundle associated to $n\Theta$. A *symmetric theta structure* $\Theta^{\mathcal{L}}$ of type n is an isomorphism $\mathcal{G}(n) \to \mathcal{G}(n\Theta)$ that commutes with the action of δ_{-1} and which induces the identity on the natural embedding of k^* in both groups. It induces an isomorphism $\overline{\Theta}^{\mathcal{L}} : A[n] \to H(n) = K(n) \times \hat{K}(n)$, which sends the Weil pairing $e_{n\Theta}$ on $A[n]$ to the pairing e_n on $H(n)$ given by $e_n((x_1, \chi_1), (x_2, \chi_2)) = \chi_2(x_1)/\chi_1(x_2)$. In particular, the symmetric theta structure of level n $\Theta^{\mathcal{L}}$ induces a canonical symplectic basis of the n-torsion; and Mumford shows in [29] that conversely $\Theta^{\mathcal{L}}$ is induced by a symplectic basis of the $2n$-torsion. We will say that these bases are *compatible* with $\Theta^{\mathcal{L}}$.

By uniqueness of the irreducible action of the Heisenberg group [29], the theta structure $\Theta^{\mathcal{L}}$ induces an isomorphism $\beta \colon \Gamma(A, n\Theta) \xrightarrow{\sim} V(n)$, uniquely defined up to a scalar. Via β, it is possible to transfer the basis $\delta_i : i \in (\mathbb{Z}/n\mathbb{Z})^g$ of $V(n)$, to a basis $(\theta_i)_{i \in (\mathbb{Z}/n\mathbb{Z})^g}$ on $\Gamma(A, n\Theta)$; the functions θ_i are called *theta coordinates* of *level* n. Using theta coordinates, it is possible to represent abelian varieties via an embedding into the projective space [32, Ch. II, Theorem 1.3]. In particular, if $n > 2$, 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 $\left(\theta_i(0^A)\right)_{i \in (\mathbb{Z}/n\mathbb{Z})^g}$ the *theta-null point*. Given a point $P \in A$ and $T \in A[n]$, we

[6] With $\hat{A}$, we denote the dual abelian variety of A.

can efficiently represent $P+T$ in theta coordinates: if T corresponds to (s,χ) via $\overline{\Theta}^{\mathcal{L}}$,

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

Let $f\colon A \to B$ be an isogeny between abelian varieties, let Θ_A, Θ_B be two divisors inducing principal polarisations on A and B respectively. Suppose there exists an isomorphism $\alpha\colon f^*\Theta_B \xrightarrow{\sim} n\Theta_A$, we say that f is an n-isogeny. Then, one can prove that $\ker(f) \subset A[n]$. On the other hand, given $K \subset A[n]$, it is not generally true that the isogeny $f'\colon A \to B$ of kernel K generates an isomorphism between $n\Theta_A$ and $(f')^*\Theta'_B$ for some divisor Θ'_B on B. By [29], this is exactly true when K is *maximal isotropic* for the Weil pairing e_n on $A[n]$[7]. We note that if we have a theta structure $\Theta^{\mathcal{L}}$ on A, then the image of $K(n) = (\mathbb{Z}/n\mathbb{Z})^g$ and $\hat{K}(n) = (\hat{\mathbb{Z}}/n\hat{\mathbb{Z}})^g$ by $\overline{\Theta}^{\mathcal{L}}$ are maximal isotropic subgroups of $A[n]$. If K is equal to the image of $\hat{K}(n)$ by $\overline{\Theta}^{\mathcal{L}}$, we say that it is *compatible* with the theta structure.

In this work we will consider $(2^n,2^n)$-isogenies $A \to B$ between abelian surfaces, with kernel K maximal isotropic in $A[2^n]$, and A will be endowed with a symmetric theta structure of level two. We will say that K is *compatible with our theta structure* if not only $K[2]$ is compatible in the sense above, but also that $K[4]$ is compatible with the symmetric structure. This is to say that if $T' \in K[4]$ is a point of exact order four, and $T = 2T'$, with T corresponding to $\overline{\Theta}^{\mathcal{L}}(0,\chi)$, then we require the theta coordinates $\theta_i(T')$ to be invariant under the action of $\Theta^{\mathcal{L}}(1,0,\chi)$. Unraveling the definition, this is equivalent to the fact that a basis of $K[4]$ extends to a symplectic basis of $A[4]$ compatible with $\Theta^{\mathcal{L}}$.

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

$$\chi_j(i_k) = \begin{cases} -1 & \text{if } j = k, \\ 1 & \text{if } j \neq k. \end{cases}$$

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 $P+T_2 = (x:y:-z:-t)$.

Remark 2 (Rational theta coordinates). In practice, our base field k is not algebraically closed. However, if we assume that $A[2n]$ is k-rational then the associated level-n theta structure is also k-rational. Especially, the theta-null point is k-rational and level-n theta coordinates of k-rational points are k-rational. In our work focused on cryptographic applications, k will be a finite field (e.g. $\mathbb{F}_{p^2}$) but the formulae we provide are valid on any perfect field.

[7] We say K is isotropic for e_n when $e_n(x,y) = 1$ for all $x,y \in K$ and maximal isotropic if it is maximal as a subgroup of $A[n]$ for this property.

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

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 [39, 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-two 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 [39, 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 $(\theta_{00}(P) : \theta_{10}(P) : \theta_{01}(P) : \theta_{11}(P))$. [8] Specialising [39, Eq. 2.4], we have

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

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

2.2 Duplication Formula

Let $(\theta_i^A)_i$ be theta coordinates of level two 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 $(S_1, \dots, S_g)$ be the canonical basis induced by $\overline{\Theta}^{\mathcal{L}}(K(2))$ and $(T_1, \dots, T_g)$ the canonical basis induced by $\overline{\Theta}^{\mathcal{L}}(\widehat{K}(2))$, which means $K_1 = \langle S_1, \dots, S_g \rangle$ and $K_2 = \langle T_1, \dots, T_g \rangle$. 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 two 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 [29, Theorem 4, p. 302] (see also [38, Theorem 3.6.4]) and duplication formula [29, 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 $(\theta_i^B)_i$.

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

2.3 Theta Structures on Elliptic Products

We now focus on products of two elliptic curves and explain how to endow these products with a theta structure of type two. First, let us recall that every theta structure of type two comes from a symplectic basis of the four-torsion [29, 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 [13, Lemma F.3.1].

Proposition 3. *Let $(a_i : b_i)$ be theta-null points on E_i induced by a symplectic four-torsion basis (e_i, f_i), 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 four-torsion $\langle(e_1, 0), (0, e_2)\rangle \oplus \langle(f_1, 0), (0, f_2)\rangle$.*

However, in the next sections, we might need to work with theta structures associated to a different symplectic four-torsion basis. In what follows, we explain how to construct theta structures associated with a fixed symplectic four-torsion basis. First, we explain how to do it over elliptic curves and then transfer our results to elliptic products.

Let E be an elliptic curve, and (T_1', T_2') a basis of the four-torsion. To compute the theta-null point associated to this basis, we proceed as follows. Given a point $T \in E[2]$, there are two *symmetric elements* $\pm\mathfrak{g}$ (satisfying $\delta_{-1}(\mathfrak{g}) = \mathfrak{g}^{-1}$) above T in the theta group $\mathcal{G}(\mathcal{L}(2(0_E)))$. We can fix a symmetric element via a point T' of four-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. Unraveling the construction by Mumford of a symmetric theta structure of level two induced by a symplectic basis of level four, 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 four-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 two. The element $\mathfrak{g}_{T'}$ in $\mathcal{G}(\mathcal{L}(2(0_E)))$ fixed by T' corresponds to $\pm\mathfrak{g}$. Still by unraveling Mumford's construction, the correct sign choice for the symmetric element $\mathfrak{g}_{T'}$ induced by T' is given by the one that leaves invariant any affine lift of T'. In our case, this is $\mathfrak{g}$.

For a general elliptic curve, if $T' = (x, y, z)$ is a point of four-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

$$M^T U M^{T^{-1}} = \frac{1}{xw - zu} \begin{pmatrix} uz & zw \\ wx^2/z - 2ux & -uz \end{pmatrix},$$

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 4. 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 four-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 $\theta_1 = \mathfrak{g}_1 \cdot \theta_0 = \mathfrak{g}_1 \cdot (X-Z) = z(u-w)/(wx-uz)X + (wx^2/z - 2ux + uz)/(wx-uz)Z = b/aX + b/aZ$. We recover the same conversion formula between Montgomery and theta coordinates as obtained in [43, Ch. 7, Appendix A.1].

We can use the same strategy to compute the theta-null point associated to a symplectic basis of the four-torsion on a product of elliptic curves. If $T' = (T'_1, T'_2) \in E_1 \times E_2$ is a point of four-torsion, the associated element $\mathfrak{g}_{T'}$ is given by $\mathfrak{g}_{T'} = \mathfrak{g}_{T'_1} \otimes \mathfrak{g}_{T'_2}$. Let X_i, Z_i be global sections of $2(0_{E_i})$ defining the x-coordinate on E_i as $x = X_i/Z_i$. 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 four-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 four-torsion of the Kummer line of an elliptic curve
Output: The $2 \times 2$ submatrix $M$ with coefficients $m_{ij}$ describing the action of $\mathfrak{g}_{P'}$ lifting the
    action by translation of $P = 2P'$.
$P \leftarrow [2]P'$                                              (▷) Cost: $2\mathbf{S} + 3\mathbf{M}$
Let $P' = (X : Z)$ and $(U : W) = P$
$WX,\ WZ,\ UX,\ UZ \leftarrow W \cdot X,\ W \cdot Z,\ U \cdot X,\ U \cdot Z$
$\delta \leftarrow WX - UZ$
Compute $\delta^{-1}$, $Z^{-1}$ via batched inversions              (▷) Cost: $3\mathbf{M} + 1\mathbf{I}$
$m_{00} \leftarrow -UZ \cdot \delta^{-1}$
$m_{01} \leftarrow -WZ \cdot \delta^{-1}$
$m_{10} \leftarrow UX \cdot \delta^{-1} - X \cdot Z^{-1}$
$m_{11} \leftarrow -m_{00}$
return $M$                                         (▷) Total cost: $2\mathbf{S} + 14\mathbf{M} + 1\mathbf{I}$
```

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 $S'_2 = (P_1, 0)$.[9] 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 four-torsion basis (S'_1, S'_2, T'_1, T'_2) 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 $N \cdot (X_1 \cdot X_2, X_1 \cdot Z_2, Z_1 \cdot X_2, Z_1 \cdot Z_2)^\intercal$.

[9] Here, we assume that both P_1 and Q_2 have order four. When this is not the case, f is a diagonal isogeny $(P, Q) \mapsto (\phi_1(P), \phi_2(Q))$, which can be computed via two one-dimensional isogenies ϕ_1 and ϕ_2; see also Appendix A.

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 $4 \times 4$ change of basis matrix $N$.
 1: $G_1 \leftarrow$ action_by_translation$(P_1')$          (▷) Algorithm 1: Cost: $2\mathbf{S} + 14\mathbf{M} + 1\mathbf{I}$
 2: $G_2 \leftarrow$ action_by_translation$(P_2')$
 3: $H_1 \leftarrow$ action_by_translation$(Q_1')$
 4: $H_2 \leftarrow$ action_by_translation$(Q_2')$
 5: $t_{00|1} \leftarrow g_{00|1} \cdot h_{00|1} + g_{01|1} \cdot h_{10|1}$          (▷) Compute the first column of $G_1 \times H_1$
 6: $t_{10|1} \leftarrow g_{10|1} \cdot h_{00|1} + g_{11|1} \cdot h_{10|1}$
 7: $t_{00|2} \leftarrow g_{00|2} \cdot h_{00|2} + g_{01|2} \cdot h_{10|2}$          (▷) Compute the first column of $G_2 \times H_2$
 8: $t_{10|2} \leftarrow g_{10|2} \cdot h_{00|2} + g_{11|2} \cdot h_{10|2}$
 9: $n_{00} \leftarrow g_{00|1} \cdot g_{00|2} + h_{00|1} \cdot h_{00|2} + t_{00|1} \cdot t_{00|2} + 1$   (▷) Compute the trace for the first row
10: $n_{01} \leftarrow g_{00|1} \cdot g_{10|2} + h_{00|1} \cdot h_{10|2} + t_{00|1} \cdot t_{10|2}$
11: $n_{02} \leftarrow g_{10|1} \cdot g_{00|2} + h_{10|1} \cdot h_{00|2} + t_{10|1} \cdot t_{00|2}$
12: $n_{03} \leftarrow g_{10|1} \cdot g_{10|2} + h_{10|1} \cdot h_{10|2} + t_{10|1} \cdot t_{10|2}$
13: $n_{10} \leftarrow h_{00|2} \cdot n_{00} + h_{01|2} \cdot n_{01}$          (▷) Compute the action of $(0, Q_2')$ for the second row
14: $n_{11} \leftarrow h_{10|2} \cdot n_{00} + h_{11|2} \cdot n_{01}$
15: $n_{12} \leftarrow h_{00|2} \cdot n_{02} + h_{01|2} \cdot n_{03}$
16: $n_{13} \leftarrow h_{10|2} \cdot n_{02} + h_{11|2} \cdot n_{03}$
17: $n_{20} \leftarrow g_{00|1} \cdot n_{00} + g_{01|1} \cdot n_{02}$          (▷) Compute the action of $(P_1', 0)$ for the third row
18: $n_{21} \leftarrow g_{00|1} \cdot n_{01} + g_{01|1} \cdot n_{03}$
19: $n_{22} \leftarrow g_{10|1} \cdot n_{00} + g_{11|1} \cdot n_{02}$
20: $n_{23} \leftarrow g_{10|1} \cdot n_{01} + g_{11|1} \cdot n_{03}$
21: $n_{30} \leftarrow g_{00|1} \cdot n_{10} + g_{01|1} \cdot n_{12}$          (▷) Compute the action of $(P_1', Q_2')$ for the final row
22: $n_{31} \leftarrow g_{00|1} \cdot n_{11} + g_{01|1} \cdot n_{13}$
23: $n_{32} \leftarrow g_{10|1} \cdot n_{10} + g_{11|1} \cdot n_{12}$
24: $n_{33} \leftarrow g_{10|1} \cdot n_{11} + g_{11|1} \cdot n_{13}$
25: return $N$          (▷) Total cost: $8\mathbf{S} + 100\mathbf{M} + 4\mathbf{I}$
```

Remark 5. 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 Section 2.2. These formulae have already been described in dimension two [17]. However, we prefer to restate them in dimension two to highlight the connection with (2, 2)-isogenies and provide an explicit operation count. In what follows, we use the same notation as in Section 2.2.

Let $P, Q \in A$ and suppose we have $(\theta_i^A(P-Q))_i$. To compute $(\theta_i^A(P+Q))_i$, 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 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 6. 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 be computed via batched inversions.

```
Algorithm 3 Differential addition
Input: The theta coordinates of $P$, $Q$ and $P-Q$, 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 $P+Q$.
$X_P, Y_P, Z_P, W_P \leftarrow \mathcal{H} \circ \mathcal{S}(x_P, y_P, z_P, w_P)$          (▷) Cost: 4S
$X_Q, Y_Q, Z_Q, W_Q \leftarrow \mathcal{H} \circ \mathcal{S}(x_Q, y_Q, z_Q, w_Q)$          (▷) Cost: 4S
$X_{f(P)f(Q)} \leftarrow X_P \cdot X_Q$
$Y_{f(P)f(Q)} \leftarrow \tilde{\lambda}_1 \cdot Y_P \cdot Y_Q$
$Z_{f(P)f(Q)} \leftarrow \tilde{\lambda}_2 \cdot Z_P \cdot Z_Q$
$W_{f(P)f(Q)} \leftarrow \tilde{\lambda}_3 \cdot W_P \cdot W_Q$
$X_{PQ}, Y_{PQ}, Z_{PQ}, W_{PQ} \leftarrow \mathcal{H}(X_{f(P)f(Q)}, Y_{f(P)f(Q)}, Z_{f(P)f(Q)}, W_{f(P)f(Q)})$
$xy_{P-Q} \leftarrow x_{P-Q} \cdot y_{P-Q}$
$zt_{P-Q} \leftarrow z_{P-Q} \cdot t_{P-Q}$
$x_{P+Q} \leftarrow X_{PQ} \cdot zt_{P-Q} \cdot y_{P-Q}$
$y_{P+Q} \leftarrow Y_{PQ} \cdot zt_{P-Q} \cdot x_{P-Q}$
$z_{P+Q} \leftarrow Z_{PQ} \cdot xy_{P-Q} \cdot w_{P-Q}$
$w_{P+Q} \leftarrow W_{PQ} \cdot xy_{P-Q} \cdot z_{P-Q}$
return $x_{P+Q}$, $y_{P+Q}$, $z_{P+Q}$, $w_{P+Q}$          (▷) Total cost: 8S + 17M
```

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.

Remark 6. 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 [43, Ch. 7, § 16.4]): this is due to how we construct the theta structure on the elliptic product in Section 2.3. 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

Algorithm 4 Doubling

```
Input: The theta coordinates of $P$ and $(\tilde{\lambda}_1, \tilde{\lambda}_2, \tilde{\lambda}_3) = (\alpha^2/\beta^2, \alpha^2/\gamma^2, \alpha^2/\delta^2)$ and $(\lambda_1, \lambda_2, \lambda_3) = (a/b, a/c, a/d)$.
Output: The theta coordinates $[2]P$.
$X_P, Y_P, Z_P, W_P \leftarrow \mathcal{H} \circ \mathcal{S}(x_P, y_P, z_P, w_P)$                                  (▷) Cost: 4S
$X'_{f(P)}, Y'_{f(P)}, Z'_{f(P)}, W'_{f(P)} \leftarrow \mathcal{S}(X_P, Y_P, Z_P, W_P)$                      (▷) Cost: 4S
$Y'_{f(P)} \leftarrow \tilde{\lambda}_1 \cdot Y'_{f(P)}$
$Z'_{f(P)} \leftarrow \tilde{\lambda}_2 \cdot Z'_{f(P)}$
$W'_{f(P)} \leftarrow \tilde{\lambda}_3 \cdot W'_{f(P)}$
$X'_P, Y'_P, Z'_P, W'_P \leftarrow \mathcal{H}(X'_{f(P)}, Y'_{f(P)}, Z'_{f(P)}, W'_{f(P)})$
$Y'_P, Z'_P, W'_P \leftarrow \lambda_1 \cdot Y'_P, \lambda_2 \cdot Z'_P, \lambda_3 \cdot W'_P$
return $X'_P, Y'_P, Z'_P, W'_P$                                                    (▷) Total cost: 8S + 6M
```

model, one may first act with a symplectic automorphism ψ sending the dual theta-null point to an 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 [43, Ch. 7, § 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 Section 2.2. Ultimately, we focus on chains of 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 defined over a perfect field k endowed with a k-rational theta structure of level two, and let (S_1, S_2, T_1, T_2) 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 k-rational symplectic basis of the four-torsion [29, Remark 4, p. 319], and let (S'_1, S'_2, T'_1, T'_2) be such a basis; in particular $2S'_i = S_i$ and $2T'_i = T_i$. Our goal is to compute a $(2,2)$-isogeny $f : A \to B$. As explained in Section 2.3, in our case, we will always be working with $\ker(f) = K_2$.

Before outlining the explicit procedure, let us assume that we have k-rational points 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 $e_8(T''_1, T''_2) = 1$.[10] These conditions are not restrictive since they are naturally satisfied for chains of $(2,2)$-isogenies, which are our end goal. In particular, $(f(T''_1), f(T''_2))$ are two of the four-torsion points inducing the theta-null point on B lying above the two two-torsion

[10] Recall that such a subgroup $\langle T''_1, T''_2 \rangle$ is said to be *isotropic*.

points in K_2. Hence, the points $(f(T_1''), f(T_2''))$ 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

$$\left(\tilde{\theta}^B_{00}(f(T_2'')) : \tilde{\theta}^B_{10}(f(T_2'')) : \tilde{\theta}^B_{01}(f(T_2'')) : \tilde{\theta}^B_{11}(f(T_2''))\right) = \\ \left(\tilde{\theta}^B_{01}(f(T_2'')) : \tilde{\theta}^B_{11}(f(T_2'')) : \tilde{\theta}^B_{00}(f(T_2'')) : \tilde{\theta}^B_{10}(f(T_2''))\right).$$

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 7 (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 Section "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 Section 4.1.

Once we have computed $(\alpha : \beta : \gamma : \delta)$, we can evaluate the isogeny f at any point P using (again) Equation 2. To compute the image, we first compute $(x', y', z', w') = \mathcal{H} \circ \mathcal{S}\left((\theta^A_i(P))_i\right)$. Then we find $(\theta^B_i(f(P)))_i = \mathcal{H}(\alpha^{-1}x', \beta^{-1}y', \gamma^{-1}z', \delta^{-1}w')$ using $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$ as input to the evaluation algorithm which can be computed at a one-time cost during the codomain computation.

We give a detailed description of both the codomain and evaluation computations in Algorithms 5 and 6. We note that the cost in parentheses for Algorithm 5 is the cost of the computation of $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$ which is required as input to Algorithm 6.

Remark 8. 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 precomputed at a cost of $4\mathbf{S} + 15\mathbf{M} + 1\mathbf{I}$, we can use

Algorithm 5 Codomain

```
Input: Theta coordinates of $T''_1$ and $T''_2$, where $T''_i$ is a 8-torsion point lying above the $K_2$
    part of the symplectic four-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: $(x\alpha, x\beta, y\gamma, y\delta) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T''_1}, y_{T''_1}, z_{T''_1}, w_{T''_1})$     (▷) Cost: $4\mathbf{S}$
 2: $(z\alpha, w\beta, z\gamma, w\delta) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T''_2}, y_{T''_2}, z_{T''_2}, w_{T''_2})$     (▷) Cost: $4\mathbf{S}$
 3: Invert $(x\alpha, x\beta, z\alpha, w\beta, z\gamma, w\delta)$ using batched inversions.     (▷) Cost: $15\mathbf{M} + 1\mathbf{I}$
 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: $8\mathbf{S} + 10\mathbf{M} + 1\mathbf{I} + (13\mathbf{M})$
```

Algorithm 6 Evaluation

```
Input: Theta coordinates of $P$ and the dual theta-null point $(1 : \beta^{-1} : \gamma^{-1} : \delta^{-1})$ on $B$.
Output: Theta coordinates $f(P)$.     (▷) Case $\beta \cdot \gamma \cdot \delta \neq 0$
$(X_P, Y_P, Z_P, W_P) \leftarrow \mathcal{H} \circ \mathcal{S}(x_P, y_P, z_P, w_P)$     (▷) Cost: $4\mathbf{S}$
$(X'_{f(P)}, Y'_{f(P)}, Z'_{f(P)}, W'_{f(P)}) \leftarrow (X_P, \beta^{-1} \cdot Y_P, \gamma^{-1} \cdot Z_P, \delta^{-1} \cdot W_P)$
$(x_{f(P)}, y_{f(P)}, z_{f(P)}, w_{f(P)}) \leftarrow \mathcal{H}(X'_{f(P)}, Y'_{f(P)}, Z'_{f(P)}, W'_{f(P)})$
return $(x_{f(P)}, y_{f(P)}, z_{f(P)}, w_{f(P)})$     (▷) Total cost: $4\mathbf{S} + 3\mathbf{M}$
```

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 $8\mathbf{S} + 10\mathbf{M} + 1\mathbf{I} + (4\mathbf{M})$.

Projective Algorithms. In Algorithm 5, we use batched inversions to recover $(1 : \beta : \gamma : \delta)$ and $(1 : \beta^{-1} : \gamma^{-1} : \delta^{-1})$. This choice allows us to reduce the number of operations when doubling a point and evaluating an isogeny. However, we can remove the inversion in Algorithm 5 by working projectively at a cost of only a few extra multiplications. We describe a projective version of Algorithm 5 in Algorithm 7, where the cost in parentheses is associated to the computing the input $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$ used for the evaluation of theta points under the action of f.

Evaluating the isogeny and doubling a point when the dual theta-null point is not normalised at $(1 : \beta : \gamma : \delta)$ induces an extra cost of one multiplication for evaluations and two multiplications for doubling. Additionally, the arithmetic precomputation requires $5\mathbf{S} + 14\mathbf{M}$ to precompute eight field elements. Note that if these arithmetic precomputations are available when computing the codomain, we can reduce the stated cost to $8\mathbf{S} + 7\mathbf{M} + (4\mathbf{M})$. Understanding whether to work projectively or using batched inversions with normalised null points boils down to the specifics of the chain length, number of evaluations and the cost of inversion in the base field. In the rest of paper,

Algorithm 7 Projective Codomain

```
Input: Theta coordinates of $T_1''$ and $T_2''$, where $T_i''$ is a 8-torsion point lying above the $K_2$
    part of the symplectic four-torsion basis inducing the theta-null point.
Output: Dual theta-null point $(\alpha : \beta : \gamma : \delta)$, the inverse of the dual theta-null point
    $(\alpha^{-1} : \beta^{-1} : \gamma^{-1} : \delta^{-1})$ and the theta-null point $(a' : b' : c' : d')$ on $B$. (▷) Case $\alpha{\cdot}\beta{\cdot}\gamma{\cdot}\delta \neq 0$
$(x\alpha, x\beta, y\gamma, y\delta) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_1''}, y_{T_1''}, z_{T_1''}, w_{T_1''})$    (▷) Cost: $4\mathbf{S}$
$(z\alpha, w\beta, z\gamma, w\delta) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_2''}, y_{T_2''}, z_{T_2''}, w_{T_2''})$    (▷) Cost: $4\mathbf{S}$
$z\alpha w\beta \leftarrow z\alpha \cdot w\beta$
$\alpha \leftarrow x\alpha \cdot z\alpha w\beta$
$\beta \leftarrow x\beta \cdot z\alpha w\beta$
$\gamma \leftarrow z\gamma \cdot x\alpha \cdot w\beta$
$\delta \leftarrow w\delta \cdot x\beta \cdot z\alpha$
$\alpha\beta, \gamma\delta \leftarrow \alpha \cdot \beta, \gamma \cdot \delta$
$\alpha^{-1} \leftarrow \gamma\delta \cdot \beta$
$\beta^{-1} \leftarrow \gamma\delta \cdot \alpha$
$\gamma^{-1} \leftarrow \alpha\beta \cdot \delta$
$\delta^{-1} \leftarrow \alpha\beta \cdot \gamma$
$(a', b', c', d') \leftarrow \mathcal{H}(\alpha, \beta, \gamma, \delta)$
return $(\alpha, \beta, \gamma, \delta), (\alpha^{-1}, \beta^{-1}, \gamma^{-1}, \delta^{-1}), (a', b', c', d')$    (▷) Total cost: $8\mathbf{S} + 7\mathbf{M} + (6\mathbf{M})$
```

we will work using the normalised implementation for clarity but point out that it is always possible to use projective arithmetic to save inversions at the cost of slightly more expensive doubling and evaluations.

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 defined over a perfect field k, 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 k-rational 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 four-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 Section "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 four-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 Section 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 Section “Splitting Isogeny”.

Gluing Isogeny In this section, 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 $E_1 \times E_2$ satisfy some additional properties with respect to level-two theta coordinates. We refer to Dupont's PhD thesis [16] 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 $K(2)$. A level-two 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) satisfies $U_{i,j}(0) = 0$, 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-two theta coordinates $(\theta_i(P))_i$, we can compute the square of its level-$(2, 2)$ theta coordinates as

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

In [16, 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 two, 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 8.

Algorithm 8 Special Codomain, $\alpha = 0$

```
Input: Theta coordinates of $T_1''$ and $T_2''$, where $T_i''$ is a 8-torsion point lying above the $K_2$
    part of the symplectic four-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 $\alpha = 0$
$(0, x\beta, y\gamma, y\delta) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_1''}, y_{T_1''}, z_{T_1''}, w_{T_1''})$      (▷) Cost: $4\mathbf{S}$
$(0, w\beta, z\gamma, w\delta) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{T_2''}, y_{T_2''}, z_{T_2''}, w_{T_2''})$      (▷) Cost: $4\mathbf{S}$
Compute the inverse of $(y\gamma, w\beta, y\delta, w\delta)$ using batched inversions.      (▷) Cost: $9\mathbf{M} + 1\mathbf{I}$
$\beta \leftarrow w\beta \cdot (w\delta)^{-1}$
$\gamma \leftarrow y\gamma \cdot (y\delta)^{-1}$
$\beta^{-1} \leftarrow w\delta \cdot (w\beta)^{-1}$
$\gamma^{-1} \leftarrow y\delta \cdot (y\gamma)^{-1}$
$(a', b', c', d') \leftarrow \mathcal{H}(0, \beta, \gamma, 1)$
return $(0, \beta, \gamma, 1), (0, \beta^{-1}, \gamma^{-1}, 1), (a', b', c', d')$      (▷) Total cost: $8\mathbf{S} + 13\mathbf{M} + 1\mathbf{I}$
```

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

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

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

The component $\tilde{\theta}_{00}^A(f(P))$ 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 section. 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 9. Note that for both Algorithm 8 and 9, 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 9 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-two theta coordinates on the surface $E_1 \times E_2$.

Algorithm 9 Special Evaluation, $\alpha = 0$

```
Input: Theta coordinates of $P$ and $P + T_1'$ and the “inverse" of the dual theta-null point
    $(0 : \beta^{-1} : \gamma^{-1} : 1)$ on $A$.
Output: Theta coordinates of $f(P)$.                                         (▷) Case $\alpha = 0$
$(0, Y_P, Z_P, W_P) \leftarrow \mathcal{H} \circ \mathcal{S}(x_P, y_P, z_P, w_P)$                          (▷) Cost: 4S
$(0, Y_{P+T_1}, Z_{P+T_1}, W_{P+T_1}) \leftarrow \mathcal{H} \circ \mathcal{S}(x_{P+T_1} : y_{P+T_1} : z_{P+T_1} : w_{P+T_1})$    (▷) Cost: 4S
$(Y'_{f(P)}, Z'_{f(P)}, W'_{f(P)}) \leftarrow (\beta^{-1} \cdot Y_P, \gamma^{-1} \cdot Z_P, W_P)$
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: 8S + 5M + 1I
```

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, 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 hard-coded the action of each of the symplectic automorphisms onto theta coordinates of level two in the reference implementation. These symplectic automorphisms and their actions have been derived from [39, 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 a theta-null point on $E_1 \times E_2$, from Proposition 3, 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 [43, Ch. 7, Appendix A.1], also rederived in Example 4.

4.2 Computing Isogenies without Extra Isotropic Information

In this section, 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.[11]

[11] 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 k.

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 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 Section 2.1) induced by a symplectic automorphism. Hence, it still corresponds to the correct codomain, but with a different theta structure. We refer to [43, Ch. 7, Example B.4]. for more details.

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 square root computations: the theta-null point requires a theta structure of level two, so in particular a full basis of the two-torsion and some extra information on the four-torsion. If we do not have the 2^{n+2}-torsion at the beginning, we miss the necessary information on the four-torsion at the penultimate step and on the two-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 four-torsion but not the 8-torsion points T_1'' and T_2'' anymore. This means that on the codomain, we only have the two-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 [43, Ch. 7, 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 [43, Ch. 7, 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 version 10.2. 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 [34], the function `EllipticProductIsogeny(kernel, n)` has been designed to be a drop-in replacement for the $(2^n, 2^n)$-isogeny computed using the Richelot correspondence and the algorithms presented in [45].

The Rust implementation has been designed with constructive cryptographic implementations in mind, and in particular, it has been written to be constant time.[12] The finite field arithmetic and certain elliptic curve functions have been adapted from the `crrl` library [35] 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 section, 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 [34] using the optimisations introduced in the implementation of [2].

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.

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 [34] 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.

[12] 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.

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 an 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 [34]	Theta Rust	Theta SageMath	Richelot SageMath [34]
254	126	**2.13 ms**	108 ms	1028 ms	**161 μs**	5.43 ms	114 ms
381	208	**9.05 ms**	201 ms	1998 ms	**411 μs**	8.68 ms	208 ms
1293	632	**463 ms**	1225 ms	12840 ms	**17.8 ms**	40.8 ms	1203 ms

The run-times displayed in Table 2 were captured on an 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 `1.80.0-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 (version 10.2) 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 the smaller characteristics studied, 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.[13]

We note here that an alternative and faster implementation of $(2,2)$-isogenies in the Mumford model is available in [19]. In this work, Kunzweiler uses Jacobians of hyperelliptic curves in specific models which allows $(2,2)$-isogeny chains to be computed particularly efficiently. In the initial treatment of this work, isogenies between elliptic products were not considered, leading to FESTA [2] and other projects to rely on [34]. However, Kunzweiler's work can be adapted to the case of isogenies between elliptic products. Additionally, Kunzweiler also has an unpublished SageMath implementation of $(2,2)$-isogenies using Kummer surfaces in the Mumford model rather than in the

[13] 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.

Jacobian model that she kindly provided us.[14] Comparing our results to the computations of Kummer surfaces in the Mumford model is a fairer comparison as we work with level-two theta coordinates, which are also on Kummer surfaces. Comparing against this implementation, the codomain computation in the theta model is around four times faster than in the Mumford model, and evaluations are around four times faster. We give detailed comparison timings in Table 3.

Table 3. Comparison of the SageMath running times for a $(2^n, 2^n)$-isogeny between elliptic products in the theta model against Kunzweiler's implementation in the Mumford model using both Jacobians and Kummer surfaces [20]. Times were recorded on an Intel Core i7-9750H CPU with a clock-speed of 2.6 GHz with turbo-boost disabled.

		Codomain			Evaluation		
$\log p$	n	Theta	Jacobian	Kummer	Theta	Jacobian	Kummer
254	126	108 ms	760 ms	467 ms	5.43 ms	66.7 ms	18.4 ms
381	208	201 ms	1478 ms	858 ms	8.68 ms	119 ms	31.4 ms
1293	632	1225 ms	9196 ms	5150 ms	40.8 ms	593 ms	170 ms

Although theta coordinates are faster, working in the Mumford model is interesting when the level-two theta coordinates are not rational, which would require using theta coordinates 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. Following the discussion in Section 2.3, an elliptic curve E has rational theta coordinates when $E[4]$ is rational.

Comparison with Dimension One In Table 4, we provide a timing comparison using SageMath between a 2^n-isogeny in dimension one using the efficient formulae of [36] 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 essentially the best we could hope given the degrees, and actually 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. First of all, we have around twice as many doublings and images in dimension two than in dimension one because the kernel is of rank two. 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× slower, and the intermediate codomain computations are also slower. Furthermore, a lot of

[14] Kunzweiler's isogenies between elliptic products using both Jacobians and Kummer surfaces are now available via GitHub [20]

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 incur a twofold 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.[15]

So, all in all, we should expect a slowdown around 4× for perfect implementations. Our benchmarks show a slowdown slightly less than 2×, making two-dimensional isogenies perform better than expected (by contrast, the 2× slowdown for images is consistent with the theory). This is probably due to SageMath overhead and the fact that the dimension one implementation has been designed to allow arbitrary degree rather than only chains of two isogenies and is missing some optimisations. A final caveat is that in dimension one, it is faster to split the 2^n-isogeny using the fast four-isogenies from [11] rather than using two-isogenies – we did not do that in our comparison because we do not have efficient four-isogeny formulae in dimension two yet. Still, taking into account the degrees of the respective isogenies, this shows that our dimension two formulae are quite competitive with the best dimension one formulae.

Table 4. 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 an 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

5.2 Implementation details

In this section, we explain two optimisations we applied in the implementation. The first one is a direct consequence of Remark 8, 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* [14] to our case.

Reduce, Reuse, Recycle A simple and obvious optimisation is to reuse as many computations as possible throughout the isogeny chain. As mentioned in Remark 8, for each step on the isogeny chain, we precompute six field elements for doubling with a normalised null point at a cost of $4\mathbf{S} + 21\mathbf{M} + 1\mathbf{I}$ or eight field elements at a cost of $6\mathbf{S} + 16\mathbf{M}$ for the projective null point. Knowledge of these values allows the doubling

[15] We could also use differential additions to compute $[m]P$, $[m]P + T_1'$, but this would be more expensive than just doubling in the affine model.

of any theta point on the corresponding theta structure to have a cost of $8\mathbf{S}+6\mathbf{M}$ and $8\mathbf{S}+8\mathbf{M}$ respectively, but it also allows the following evaluation precomputation cost to be lowered from $13\mathbf{M}$ and $6\mathbf{M}$ to $4\mathbf{M}$ for both cases.

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 number of doublings and evaluations required at each step along the chain. This optimisation has not yet been implemented in our code, where we work with projective null points along the whole chain for simplicity.

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. With 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 [14]. 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 — see for instance [5]. 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 [14] 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 $8\mathbf{S}+10\mathbf{M}+1\mathbf{I}$.

On the whole, 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 [14, 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 modification, we are able to find the "proper" optimised strategy, which further reduces the run-time of the isogeny chain computation by approximately 2%.[16] 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 $L_2 = (S_1, S_2)$, $L_i \in E_1 \times E_2$.

[16] 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.

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 two bits in the masked torsion data.[17]

We find that 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. Optimisations of the finite field arithmetic could offer substantial speed ups, as seen in the optimised assembly implementations for large characteristic SIDH [18, Table 2.1] and the efficient algorithms of [22].

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. When computing the dimension two isogeny in the theta model, the time spent for the $(2^n, 2^n)$-isogeny shrinks from 70% of the run-time to only 25%, with the remaining computation time spent in dimension one, computing various discrete logarithms 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.

Our algorithm significantly outperforms the previous method in [34,2]: 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.

[17] SageMath benchmark of FESTA isogeny

A The General Case

In this section, we briefly explain how to compute a general $(2^n, 2^n)$-isogeny between Kummer surfaces (with decomposable or indecomposable polarisations).

In theory, using the theta model would give an uniform approach to handle both Jacobian of hyperelliptic curves of genus two and product of elliptic curves. However, in order to achieve the best performance, we rely on a theta model of level two rather than higher level $n > 2$ (since in level n we have n^g theta coordinates), which yields the following technical difficulty.

Let A be a principally polarised abelian surface. If A corresponds to a Jacobian, then the level-two theta coordinates give an embedding of the Kummer surface $A/\pm 1$. However, as explained in the main text, if $A = E_1 \times E_2$ is a product of two elliptic curves (with their product polarisation), then the level-two theta coordinates give an embedding of the product of Kummer lines $(E_1/\pm 1) \times (E_2/\pm 1)$. In particular, we do not get an embedding of $(E_1 \times E_2)/\pm 1$ but of a further quotient.

As a consequence, for a gluing image $(E_1/\pm 1) \times (E_2/\pm 1) \to A/\pm 1$, knowing a point $P = (\pm P_1, \pm P_2)$ is not enough to determine its image in A: we need extra data. This is why we had to use a special algorithm for the gluing isogeny in Section 4. In practice, the gluing case can be detected when some of our intermediate theta constants are zero. As mentioned above, if we were working in level $n > 2$, we would always have enough non-zero theta constants to compute images in all cases (by [29]), but we need an alternative strategy for $n = 2$.

We now explain how to deal with all cases for a $(2^n, 2^n)$-isogeny $A \to B$ in level two with kernel K. In the main text, we already dealt with the case where both A, B are product of elliptic curves, but none of the intermediate abelian surfaces are.

1. **When the codomain B is not a product**. In this case we proceed as in the main algorithm, except we do not need to find a product theta structure in the end. If needed, to recover B as a Jacobian and to convert between theta coordinates and Mumford coordinates, we can use Thomae's formula and the conversion formula from [32], see also [47,8,9].
2. **When the domain A is not a product**. If A and the kernel are already described by theta coordinates, we first need to do a symplectic change of theta coordinate to make the kernel compatible with our theta structure. To compute this change of basis, we can proceed as in Section 2.3 by taking suitable traces under the symmetric elements of the theta group induced by $K[4]$. For the case of a product of elliptic curves, we had to give the explicit action of the symmetric theta group element corresponding to a couple of points of four-torsion of elliptic curves on the product of coordinates. In our case, since we already have theta coordinates, this action is already encoded by our theta structure. We refer to the change of coordinates formulae provided in [12, Theorem 12].
   If A, which is a Jacobian $\mathrm{Jac}(C)$ under our hypothesis, is described by the curve C and the kernel K has its generators given in Mumford coordinates, we first need to convert into theta coordinates, using the formulae of [32,47,8,9] as above. In that case, Kunzweiler has formulae[18] for how to take the fourth-roots in Thomae's

[18] Private communication.

formula that directly give the theta constants compatible with the kernel; this allows to bypass the change of basis step once we have the theta coordinates.

3. **When the first step is between elliptic products**. If the chain begins with a $(2,2)$-isogeny between products $\Phi : E_1 \times E_2 \to E_1' \times E_2'$, the isogeny Φ is a diagonal isogeny, i.e. $\Phi = \begin{pmatrix} \phi_1 & 0 \\ 0 & \phi_2 \end{pmatrix}$, where $\phi_i \colon E_i \to E_i'$ is a one-dimensional isogeny. This cases reduces to first computing the one-dimensional isogenies ϕ_i's to encode the first step and then resuming from the resulting elliptic product.

   The only other possibility is that we have an isogeny diamond (i.e., a Kani square) with isogenies of degree one (i.e., isomorphisms). Then Kani's lemma give a $(2,2)$-isogeny Φ. For instance if we take $E_1 = E_2 = E_1' = E_2' = E$ and we consider the automorphisms $\mathrm{Id} : E_i \to E_i$; then we obtain the two-isogeny $\Phi : (P,Q) \mapsto (P+Q, P-Q)$, and whose kernel is $\{(T,T) \mid T \in E[2]\}$. All other $(2,2)$-isogenies $E \times E \to E \times E$ which are not given by diagonal two-isogenies in dimension one are variant of this Φ where we apply some automorphisms to P or Q before. (This only gives a different kernel when $j(E) = 0$ or $j(E) = 1728$ and we have non trivial automorphisms, i.e. different from ±1.)

4. **An intermediate abelian surface is a product**. In that case, the easiest solution would be to restart the computation using level $n = 4$ (which requires 16 coordinates rather than four), or the representation from [23] (which requires eight coordinates), because they give embeddings of the abelian surfaces in both the product and non product case, and allow to treat both cases uniformly.

   Another solution is to switch to the representation from [23] on the fly. Let us treat the case of a gluing directly followed by a splitting: $A \to E_1 \times E_2 \to B$, with $\Phi_1 : A \to E_1 \times E_2$ and $\Phi_2 : E_1 \times E_2 \to B$.

   The splitting step can be handled as in the main text, where we had to compute a splitting as the last step; namely we can compute a product theta structure on $E_1 \times E_2$. The difference is that now, we are not at the last step anymore, so we still need to compute a gluing image afterwards.

   For reasons explained above, knowing $\Phi_1(P)$ in level-two theta coordinates is not enough to compute the gluing $\Phi_2 \circ \Phi_1(P)$. As in Section 4, for the gluing we need $\Phi_1(P)$ and $\Phi_1(P) + T'$ in level-two coordinates, for T' a point of four-torsion in $E_1 \times E_2$.

   One way to obtain these point is to take $T'' \in A$ a point of 8-torsion in A, above $\ker \Phi_2 \circ \Phi_1$. We compute a representation of the set $\{P \pm T''\}$ using the formulae of [23], and we compute $\Phi_1(P)$, $\{\Phi_1(P \pm T'')\}$ in level-two coordinates. From our choice of T'' we have that $T' = \Phi_1(T'')$ is a point of four-torsion in $E_1 \times E_2$. We do not quite have $\Phi_1(P), \Phi_1(P) + T'$, but only $\Phi_1(P) \pm T'$. However, from our choice of T'' we have that $\Phi_2(T')$ is a point of two-torsion, hence the two points $\Phi_2 \circ \Phi_1(P) \pm \Phi_2(T')$ are the same. This means that we can use our gluing algorithm as before.

   The case where we have m several successive product $A \to E_1 \times E_2 \to E_1' \times E_2' \to \cdots \to B$ can be treated in a similar way, by taking a point T'' of 2^{m+2}-torsion above the kernel of $A \to B$, pushing $P, P \pm T''$ through the splitting isogeny and the intermediate isogenies, then taking a final gluing isogeny.

References

1. Basso, A., De Feo, L., Dartois, P., Leroux, A., Maino, L., Pope, G., Robert, D., Wesolowski, B.: SQIsign2D-West: The Fast, the Small, and the Safer. Cryptology ePrint Archive, Paper 2024/760 (2024), https://eprint.iacr.org/2024/760
2. Basso, A., Maino, L., Pope, G.: FESTA: Fast encryption from supersingular torsion attacks. In: Advances in Cryptology – ASIACRYPT 2023. pp. 98–126 (2023). https://doi.org/10.1007/978-981-99-8739-9_4
3. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part V. Lecture Notes in Computer Science, vol. 14008, pp. 423–447. Springer, Heidelberg, Germany, Lyon, France (Apr 23–27, 2023). https://doi.org/10.1007/978-3-031-30589-4_15
4. Chen, M., Leroux, A., Panny, L.: SCALLOP-HD: group action from 2-dimensional isogenies. In: Public-Key Cryptography - PKC 2024. pp. 190–216 (2024). https://doi.org/10.1007/978-3-031-57725-3_7
5. Chi-Domínguez, J.J., Pizarro-Madariaga, A., Riquelme, E.: Computing Quotient Groups of Smooth Order with Applications to Isogenies over Higher-Dimensional Abelian Varieties. Cryptology ePrint Archive, Paper 2023/508 (2023), https://eprint.iacr.org/2023/508
6. Chudnovsky, D., Chudnovsky, G.: Sequences of numbers generated by addition in formal groups and new primality and factorization tests. Advances in Applied Mathematics **7**(4), 385–434 (1986). https://doi.org/10.1016/0196-8858(86)90023-0
7. Cornell, G., Silverman, J.H.: Arithmetic Geometry. Springer (1986), https://doi.org/10.1007/978-1-4613-8655-1
8. Cosset, R.: Application des fonctions thêta à la cryptographie sur courbes hyperelliptiques. Ph.D. thesis (2011)
9. 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
10. Costello, C.: Computing supersingular isogenies on Kummer surfaces. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology – ASIACRYPT 2018, Part III. Lecture Notes in Computer Science, vol. 11274, pp. 428–456. Springer, Heidelberg, Germany, Brisbane, Queensland, Australia (Dec 2–6, 2018). https://doi.org/10.1007/978-3-030-03332-3_16
11. Costello, C., Hisil, H.: A simple and compact algorithm for SIDH with arbitrary degree isogenies. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017, Part II. Lecture Notes in Computer Science, vol. 10625, pp. 303–329. Springer, Heidelberg, Germany, Hong Kong, China (Dec 3–7, 2017). https://doi.org/10.1007/978-3-319-70697-9_11
12. Dartois, P.: Fast computation of 2-isogenies in dimension 4 and cryptographic applications. Cryptology ePrint Archive, Paper 2024/1180 (2024), https://eprint.iacr.org/2024/1180
13. Dartois, P., Leroux, A., Robert, D., Wesolowski, B.: SQIsignHD: New Dimensions in Cryptography. In: Joye, M., Leander, G. (eds.) Advances in Cryptology – EUROCRYPT 2024. pp. 3–32. Springer Nature Switzerland, Cham (2024)
14. De Feo, L., 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
15. 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

16. Dupont, R.: Moyenne arithmético-géométrique, suites de Borchardt et applications. Ph.D. thesis, École polytechnique (2006)
17. 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
18. 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
19. Kunzweiler, S.: Efficient computation of $(2^n, 2^n)$-isogenies. Cryptology ePrint Archive, Paper 2022/990 (2022), https://eprint.iacr.org/2022/990
20. Kunzweiler, S.: Efficient Computation of $(2^n, 2^n)$-isogenies (2023), https://github.com/sabrinakunzweiler/richelot-isogenies
21. 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
22. 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
23. Lubicz, D., Robert, D.: Arithmetic on abelian and kummer varieties. Finite Fields and Their Applications **39**, 130–158 (5 2016). https://doi.org/10.1016/j.ffa.2016.01.009
24. Lubicz, D., Robert, D.: Fast change of level and applications to isogenies. Research in Number Theory (ANTS XV Conference) **9**(1) (12 2022). https://doi.org/10.1007/s40993-022-00407-9
25. Maino, L., Martindale, C., Panny, L., Pope, G., Wesolowski, B.: A direct key recovery attack on SIDH. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part V. Lecture Notes in Computer Science, vol. 14008, pp. 448–471. Springer, Heidelberg, Germany, Lyon, France (Apr 23–27, 2023). https://doi.org/10.1007/978-3-031-30589-4_16
26. 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
27. 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
28. Moriya, T.: LIT-SiGamal: An efficient isogeny-based PKE based on a LIT diagram. Cryptology ePrint Archive, Paper 2024/521 (2024), https://eprint.iacr.org/2024/521
29. Mumford, D.: On the Equations Defining Abelian Varieties. I. Inventiones Mathematicae **1** (12 1966). https://doi.org/10.1007/BF01389737
30. Mumford, D.: On the Equations Defining Abelian Varieties. II. Inventiones Mathematicae **3** (01 1967). https://doi.org/10.1007/BF01389741
31. Mumford, D.: On the Equations Defining Abelian Varieties. III. Inventiones Mathematicae **3** (01 1967). https://doi.org/10.1007/BF01425401
32. Mumford, D.: Tata Lectures on Theta I. Birkhäuser, Boston (2007)
33. 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

34. 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
35. Pornin, T.: `crrl`: Rust library for cryptographic research, version 0.7.0 (2023), https://github.com/pornin/crrl
36. 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
37. Renes, J., Schwabe, P., Smith, B., Batina, L.: μkummer: Efficient hyperelliptic signatures and key exchange on microcontrollers. In: Gierlichs, B., Poschmann, A.Y. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2016. Lecture Notes in Computer Science, vol. 9813, pp. 301–320. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–19, 2016). https://doi.org/10.1007/978-3-662-53140-2_15
38. Robert, D.: Fonctions thêta et applications à la cryptographie. Ph.D. thesis, Université Henry Poincaré - Nancy 1 (2010)
39. Robert, D.: Efficient algorithms for abelian varieties and their moduli spaces (2021), Habilitation à Diriger des Recherches
40. Robert, D.: Evaluating isogenies in polylogarithmic time. Cryptology ePrint Archive, Report 2022/1068 (2022), https://eprint.iacr.org/2022/1068
41. 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
42. Robert, D.: Breaking SIDH in polynomial time. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part V. Lecture Notes in Computer Science, vol. 14008, pp. 472–503. Springer, Heidelberg, Germany, Lyon, France (Apr 23–27, 2023). https://doi.org/10.1007/978-3-031-30589-4_17
43. Robert, D.: Some notes on algorithms for abelian varieties. Cryptology ePrint Archive, Paper 2024/406 (2024), https://eprint.iacr.org/2024/406
44. Robert, D., Sarkis, N.: Computing 2-isogenies between kummer lines. IACR Communications in Cryptology **1**(1) (2024). https://doi.org/10.62056/abvua69p1
45. Smith, B.A.: Explicit endomorphisms and correspondences. Ph.D. thesis (2005-12-23), http://hdl.handle.net/2123/1066
46. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 10.0) (2023), https://www.sagemath.org
47. Van Wamelen, P.: Equations for the jacobian of a hyperelliptic curve. Transactions of the American Mathematical Society **350**(8), 3083–3106 (1998)