Kummer for Genus One over Prime Order Fields *

Sabyasachi Karati[1] and Palash Sarkar[2]

[1] iCIS Lab
Department of Computer Science
University of Calgary
email: sabyasachi.karati@ucalgary.ca***
[2] Applied Statistics Unit
Indian Statistical Institute
203, B.T. Road, Kolkata
India 700108.
e-mail: palash@isical.ac.in

Abstract. This work considers the problem of fast and secure scalar multiplication using curves of genus one defined over a field of prime order. Previous work by Gaudry and Lubicz had suggested the use of the associated Kummer line to speed up scalar multiplication. In this work, we explore this idea in details. The first task is to obtain an elliptic curve in the Legendre form which satisfies necessary security conditions such that the associated Kummer line has small parameters and a base point with small coordinates. In turns out that the Kummer ladder supports parallelism and can be implemented very efficiently in constant time using the single-instruction multiple-data (SIMD) operations available in modern processors. This work presents appropriate Kummer lines over three primes, namely, $2^{251}-9$, $2^{255}-19$ and $2^{266}-3$ all of which are targeted at the 128-bit level. Implementation of scalar multiplication for all three Kummer lines using Intel intrinsics have been done. Timing results indicate that scalar multiplication using all three of these Kummer lines are faster than the best known highly optimised assembly implementation of the well known Curve25519. In fact, the Kummer line over $2^{266}-3$ is both faster and offers about 6 bits of higher security compared to Curve25519. As part of our work, we describe a new multiplication algorithm modulo $2^{255}-19$ which, given the importance of Curve25519, should be of some independent interest.

1 Introduction

Curve-based cryptography provides a platform for secure and efficient implementation of public key schemes whose security rely on the hardness of discrete logarithm problem. Starting from the pioneering work of Koblitz [38] and Miller [41] introducing elliptic curves and the work of Koblitz [39] introducing hyperelliptic curves for cryptographic use, the last three decades have seen an extensive amount of research in the area.

Appropriately chosen elliptic curves and genus two hyperelliptic curves are considered to be suitable for practical implementation. Necessary conditions have been identified for a curve to be considered secure. The currently known necessary conditions can be found at [6]. These necessary conditions, however, are not known to be sufficient. The security of curve-based cryptography is based on the conjecture that the discrete logarithm problem in the relevant group is computationally difficult. This conjecture does not hold in the setting of quantum computing. Since the actual realisation of quantum computers seem to be quite some time away, it is still meaningful to conduct research for identifying new classes of secure and efficient curves.

Table 1 summarises features for some of the concrete curves that have been proposed in the literature. Arguably, the two most well known curves proposed till date for the 128-bit security level are P-256 and

*** Part of the work was done while the author was a post-doctoral fellow at the Turing Laboratory of the Indian Statistical Institute.

* The present version is work in progress. A revised version of the paper with updated timing results will be uploaded soon. This will also be accompanied by a revised version of the code at [26].

Curve25519. Both of these are in the setting of genus one over prime order fields. In particular, we note that Curve25519 has been extensively deployed for various applications. A listing of such applications can be found at [20]. So, from the point of view of deployment, practitioners are very familiar with genus one curves over prime order fields. Influential organisations, such as NIST, Brainpool, Microsoft (the NUMS curve) have concrete proposals in this setting. Bicoins use the `sec256k1` curve. See [6] for a further listing of such primes and curves. We further note that prime order fields are considered important. It has been mentioned in [3] that prime order fields "have the virtue of minimizing the number of security concerns for elliptic-curve cryptography." It is quite likely that any future portfolio of proposals by standardisation bodies will include at least one curve in the setting of genus one over a prime field.

While genus one curves over prime order fields continue to be of enduring interest (as evidenced by [45, 20, 13, 49, 15]), in recent years, the research on efficient implementation [40, 11, 12, 46, 22, 19, 18, 47] has largely focused on either genus 2, or, composite order fields (binary extension, prime-squared) with the goal of improving efficiency. The use of endomorphisms for fields of large characteristic is based on [29, 28] while for binary fields, the base-ϕ expansion of the scalar originates in the work of Koblitz. The setting of genus one over prime order fields received some recent focus in the implementation of the NIST P-256 curve [34] and through [48] which presented complete addition formulas for prime order curves in the short Weierstrass form over finite fields of characteristic not equal to 2 or 3.

Table 1. Features of some curves proposed in the last few years.

Reference	genus	form	field order	endomorphisms
NIST P-256 [45]	1	Weierstrass	prime	no
Curve25519 [3]	1	Montgomery	prime	no
Brainpool [13]	1	Weierstrass	prime	no
NUMS [49]	1	twisted Edwards	prime	no
secp256k1 [15]	1	Weierstrass	prime	yes
Longa-Sica [40]	1	twisted Edwards	p^2	yes
Bos et al. [11]	2	Kummer	prime	yes
Bos et al. [12]	2	Kummer	p^2	yes
Hankerson et al. [35], Oliviera et al. [46]	1	Weierstrass/Koblitz	2^n	yes
Longa-Sica [40], Faz-Hernández et al. [22]	1	twisted Edwards	p^2	yes
Costello et al. [19]	1	Montgomery	p^2	yes
Gaudry-Schost [32], Bernstein et al. [5]	2	Kummer	prime	no
Costello et al. [18]	1	twisted Edwards	p^2	yes
Hankerson et al. [35], Oliviera et al. [47]	1	Weierstrass/Koblitz	2^n	yes
This work	1	Kummer	prime	no

Our Contributions

In this work, we go back to the setting of genus one curves over a prime order field. Two efficient models of curves that have been considered in genus one are the Montgomery [42] and the (twisted) Edwards model [21, 8]. An issue of central importance is to be able to perform scalar multiplications in constant time. The Montgomery form supports a ladder based scalar multiplication which ensures constant time execution. Using unified formula for point addition leads to constant time scalar multiplication for Edwards form curve.

The contribution of this paper is to propose new curves for the setting of genus one and prime order field. Actual computation is done over the Kummer line associated with such a curve. The idea of using Kummer line was proposed by Gaudry and Lubicz [31]. They, however, were not clear about whether competitive speeds can be obtained using this approach. Our main contribution is to show that this can indeed be done using the single-instruction multiple-data (SIMD) instructions available in modern processors. We note that the use of SIMD instructions to speed up computation has been earlier proposed for Kummer surface associated with genus two hyperelliptic curves [31]. The application of this idea, however, to Kummer line has not been considered in the literature. Our work fills this gap and shows that properly using SIMD instructions provide a competitive alternative to known curves in the setting of genus one and prime order fields.

Like in the case of Montgomery curve, scalar multiplication on the Kummer line proceeds via a laddering algorithm. A ladder step corresponds to each bit of the scalar and each such step consists of a doubling and a differential addition irrespective of the value of the bit. This ensures that the resulting code is constant time. We describe and implement a vectorised version of the laddering algorithm which is also constant time. Our target is the 128-bit security level. The work consists of several aspects.

Choice of the underlying field: Our target is the 128-bit security level. To this end, we consider three primes, namely, $2^{251}-9$, $2^{255}-19$ and $2^{266}-3$. These primes are abbreviated as $p2519$, $p25519$ and $p2663$ respectively. The underlying field will be denoted as $\mathbb{F}_p$ where p is one of $p2519$, $p25519$ or $p2663$.

Choice of the Kummer line: Following previous suggestions [11, 4], we work in the square-only setting. In this case, the parameters of the Kummer line are given by two integers a^2 and b^2. We provide appropriate Kummer lines for all three of the primes $p2519$, $p25519$ and $p2663$. In each case, we identify a base point with small coordinates.

SIMD implementation: On Intel processors, it is possible to pack 4 64-bit words into a single 256-bit quantity and then use SIMD instructions to simultaneously work on the 4 64-bit words. We apply this approach to carefully consider various aspects of field arithmetic over $\mathbb{F}_p$. SIMD instructions allow the simultaneous computation of 4 multiplications in $\mathbb{F}_p$ and 4 squarings in $\mathbb{F}_p$. The use of SIMD instructions dovetails very nicely with the scalar multiplication algorithm over the Kummer line as we explain below.

Scalar multiplication over the Kummer line: A constant time, ladder style algorithm is used. In terms of operation count, each ladder step requires 2 field multiplications, 6 field squarings, 6 multiplications by parameters and 2 multiplications by base point coordinates [31]. In contrast, Montgomery ladder requires 4 field multiplications, 4 squarings, 1 multiplication by curve parameter and 1 multiplication by a base point coordinate. This had led to Gaudry and Lubicz [31] commenting that Kummer line can be advantageous provided that the advantage of trading off multiplications for squarings is not offset by the extra multiplications by the parameters and the base point coordinates.

Our choices of the Kummer lines ensure that the parameters and the base point coordinates are indeed very small. This is not to suggest that the Kummer line is only suitable for fixed based point scalar multiplication. The main advantage arises from the structure of the Kummer ladder vis-a-vis the Montgomery ladder.

An example of the Kummer ladder is shown in Figure 1. In the figure, the Hadamard transform $\mathcal{H}(u,v)$ is defined to be $(u+v,u-v)$. Observe that there are 4 layers of 4 simultaneous multiplications. The first layer consists of 2 field multiplications and 2 squarings, while the third layer consists of 4 field squarings. Using 256-bit SIMD instructions, the 2 multiplications and the 2 squarings in the first layer can be computed simultaneously using an implementation of vectorised field multiplication while the third layer can be computed using an implementation of vectorised field squaring. The second layer consists only of multiplications by parameters and is computed using an implementation of vectorised multiplication by constants. The fourth layer consists of two multiplications by parameters and two multiplications by base point coordinates. For fixed base point, this layer can be computed using a single vectorised multiplication by constants while for variable base point, this layer requires a vectorised field multiplication. A major advantage of the Kummer ladder is that the packing and unpacking into 256-bit quantities is done once each. Packing is done at the start of the scalar multiplication and unpacking is done at the end. The entire scalar multiplication can be computed on the packed vectorised quantities.

In contrast, the Montgomery ladder is shown in Figure 2 which has been reproduced from [3]. The structure of this ladder is not as regular as the Kummer ladder. This makes it difficult to optimally group together the multiplications for SIMD implementation. Curve25519 is based on the Montgomery ladder. SIMD implementations of Curve25519 have been reported in [9, 16]. These implementations could group together only two independent multiplications. AVX2 based implementation of Curve25519 has been reported in [23]. This work could also group together only 2 multiplications/squarings. At a forum[3], Tung Chou comments that it would better to find 4 independent multiplications/squarings and vectorise them. It is not clear how this can be done and the previous works [9, 16, 23] on SIMD implementation of Curve25519 do not seem to have been able to identify this. On the other hand, for the Kummer ladder shown in Figure 1, performing vectorisation of 4 independent multiplications/squarings comes quite naturally. This indicates that the Kummer ladder is better suited for SIMD implementation than the Montgomery ladder.

Another choice for implementation is the twisted Edwards form. Using explicit unified formulas for addition leads to constant time scalar multiplication. A non-adjacent form (NAF) representation of the scalar will require 1 doubling and 1/3 additions per bit. Using operation counts from [8] this comes to 6 field mulitplications, 4.33 squarings and 0.33 multiplications by constants per bit; using faster explicit formula from [36] requires 10.33 field multiplications and 1.33 multiplications by constants. These operation counts are higher than either Kummer or Montgomery. Working with a windowed NAF method along with a pre-computed table can provide substantial speed-up in the fixed base scalar multiplication. See [16] for details of this approach for Curve25519 and [34] for NIST P-256. For this work, we focus only on ladder based algorithms which do not require a pre-computed table and works for both fixed base and variable base scalar multiplications.

Implementation: We report implementations of all the three Kummer lines over the three primes $p2519$, $p25519$ and $p2663$. The implementations are in Intel intrinsics and use the AVX2 instructions. All the three Kummer lines are faster than Curve25519 on both the Haswell and the Skylake platforms for both fixed-based and variable-base scalar multiplication.

2 Background

In this section, we provide a sketch of the mathematical background on theta functions over genus one. These functions are defined over the complex field. For cryptographic purposes, our goal is to work over a prime field of large characteristic. All the derivations that are used have a good reduction [31] and so it is possible to use the Lefschetz principle [2, 27] to carry over the identities proved over the complex to those over a large characteristic field.

[3] `https://moderncrypto.org/mail-archive/curves/2015/000637.html`

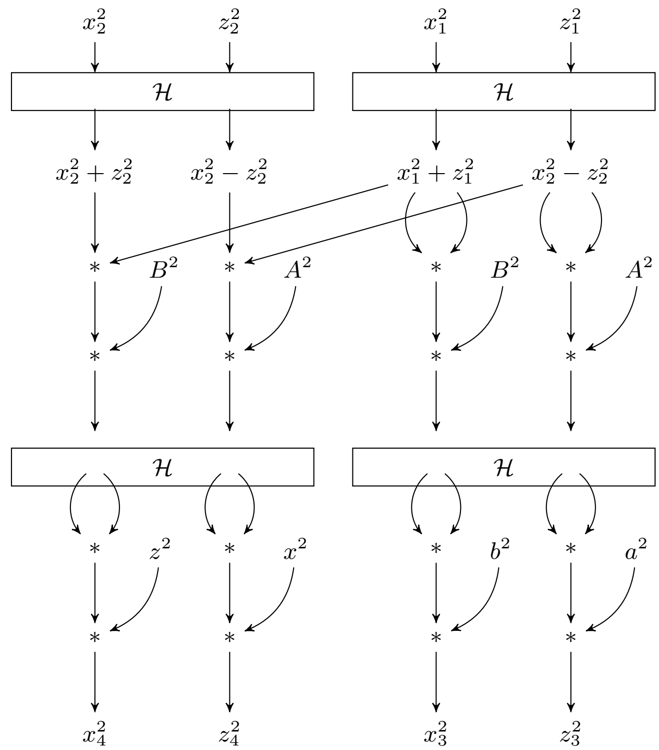

Fig. 1. Kummer ladder.

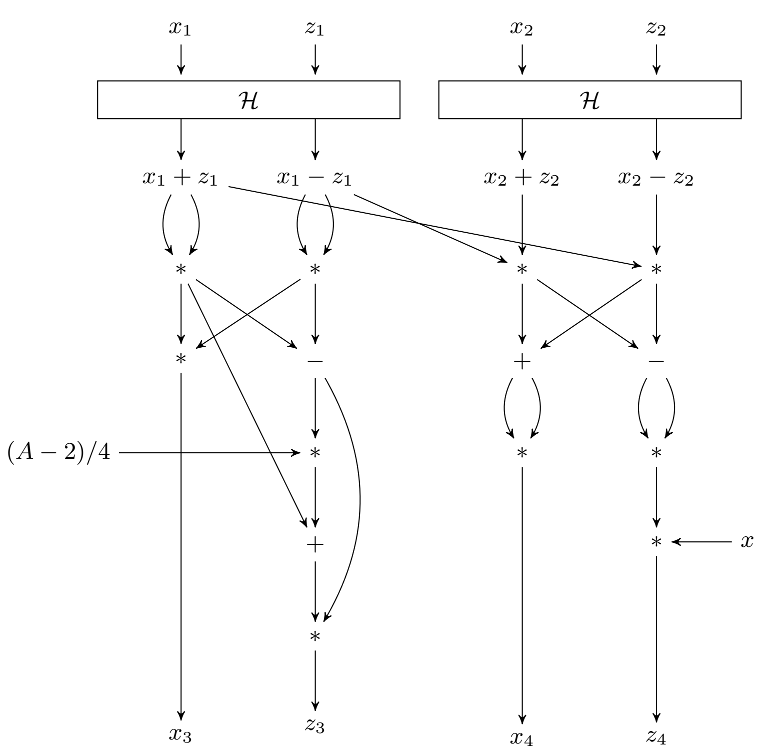

Fig. 2. Montgomery ladder.

2.1 Theta Functions

Theta functions in genus one are called the Jacobi theta functions. For the general theory covering higher genus we refer to [44, 37]. Cryptographic applications of theta functions were pointed out by Gaudry [30] for genus two and Gaudry and Lubicz [31] for genus one (and also for genus two over characteristic two fields). See also [44, 25, 24] for arithmetic on Kummer surface associated to genus two curves.

Let $\tau \in \mathbb{C}$ having a positive imaginary part and $w \in \mathbb{C}$. Let $\xi_1, \xi_2 \in \mathbb{Q}$. Theta functions with characteristics $\vartheta[\xi_1, \xi_2](w, \tau)$ are defined to be the following:

$$\vartheta[\xi_1, \xi_2](w, \tau) = \sum_{n \in \mathbb{Z}} \exp\left[\pi i (n + \xi_1)^2 \tau + 2\pi i (n + \xi_1)(w + \xi_2)\right]. \tag{1}$$

The scalars obtained by evaluating $\vartheta[\xi_1, \xi_2](w, \tau)$ at $w = 0$ are known as theta constants. We only consider the characteristics ξ_1 and ξ_2 which are in $\{0, \frac{1}{2}\}$ giving rise to four possible characteristics. Let $\xi_* = (-1)^{4\xi_1\xi_2}$. The relation between $\vartheta[\xi_1, \xi_2](w, \tau)$ and $\vartheta[\xi_1, \xi_2](-w, \tau)$ is the following. A proof is given in Appendix A.1.

$$\vartheta[\xi_1, \xi_2](-w, \tau) = \xi_* \cdot \vartheta[\xi_1, \xi_2](w, \tau). \tag{2}$$

Using this relation, the four characteristics can be divided into two groups. If $\xi_* = 1$, that is $\vartheta[\xi_1, \xi_2](w, \tau) = \vartheta[\xi_1, \xi_2](-w, \tau)$, then the corresponding characteristic is said to be even and otherwise the characteristic is said to be odd. So, only the characteristics $[\frac{1}{2}, \frac{1}{2}]$ is odd and the other three are even.

For a fixed τ, the following theta constants are defined.

$$\vartheta_1(w) = \vartheta[0,0](w, \tau) \;\text{ and }\; \vartheta_2(w) = \vartheta\left[0, 1/2\right](w, \tau).$$
$$\Theta_1(w) = \vartheta[0,0](w, 2\tau) \;\text{ and }\; \Theta_2(w) = \vartheta\left[1/2, 0\right](w, 2\tau).$$

2.2 Theta Identities

The following identities hold for the theta constants. Proofs are given in Appendices A.2 and A.3.

$$\begin{aligned} 2\Theta_1(w_1 + w_2)\Theta_1(w_1 - w_2) &= \vartheta_1(w_1)\vartheta_1(w_2) + \vartheta_2(w_1)\vartheta_2(w_2); \\ 2\Theta_2(w_1 + w_2)\Theta_2(w_1 - w_2) &= \vartheta_1(w_1)\vartheta_1(w_2) - \vartheta_2(w_1)\vartheta_2(w_2); \end{aligned} \tag{3}$$

$$\begin{aligned} \vartheta_1(w_1 + w_2)\vartheta_1(w_1 - w_2) &= \Theta_1(2w_1)\Theta_1(2w_2) + \Theta_2(2w_1)\Theta_2(2w_2); \\ \vartheta_2(w_1 + w_2)\vartheta_2(w_1 - w_2) &= \Theta_1(2w_1)\Theta_1(2w_2) - \Theta_2(2w_1)\Theta_2(2w_2). \end{aligned} \tag{4}$$

Putting $w_1 = w_2 = w$, we obtain

$$\begin{aligned} 2\Theta_1(2w)\Theta_1(0) &= \vartheta_1(w)^2 + \vartheta_2(w)^2; \\ 2\Theta_2(2w)\Theta_2(0) &= \vartheta_1(w)^2 - \vartheta_2(w)^2; \end{aligned} \tag{5}$$

$$\begin{aligned} \vartheta_1(2w)\vartheta_1(0) &= \Theta_1(2w)^2 + \Theta_2(2w)^2; \\ \vartheta_2(2w)\vartheta_2(0) &= \Theta_1(2w)^2 - \Theta_2(2w)^2. \end{aligned} \tag{6}$$

Putting $w = 0$ in (5), we obtain

$$\begin{aligned} 2\Theta_1(0)^2 &= \vartheta_1(0)^2 + \vartheta_2(0)^2; \\ 2\Theta_2(0)^2 &= \vartheta_1(0)^2 - \vartheta_2(0)^2. \end{aligned} \tag{7}$$

2.3 Kummer Line

Let $\tau \in \mathbb{C}$ having a positive imaginary part and denote by $\mathbb{P}^1(\mathbb{C})$ the projective line over $\mathbb{C}$. The Kummer line $(\mathcal{K})$ associated with τ is the image of the map φ from $\mathbb{C}$ to $\mathbb{P}^1(\mathbb{C})$ defined by

$$\varphi : w \longmapsto (\vartheta_1(w), \vartheta_2(w)). \tag{8}$$

Suppose that $\varphi(w) = [\vartheta_1(w) : \vartheta_2(w)]$ is known for some $w \in \mathbb{F}_q$. Using (5) it is possible to compute $\Theta_1(2w)$ and $\Theta_2(2w)$ and then using (6) it is possible to compute $\vartheta_1(2w)$ and $\vartheta_2(2w)$. So, from $\varphi(w)$ it is possible to compute $\varphi(2w) = [\vartheta_1(2w) : \vartheta_2(2w)]$ without knowing the value of w.

Suppose that $\varphi(w_1) = [\vartheta_1(w_1) : \vartheta_2(w_1)]$ and $\varphi(w_2) = [\vartheta_1(w_2) : \vartheta_2(w_2)]$ are known for some $w_1, w_2 \in \mathbb{F}_q$. Using (5), it is possible to obtain $\Theta_1(2w_1), \Theta_1(2w_2), \Theta_2(2w_1)$ and $\Theta_2(2w_2)$. Then (4) allows the computation of $\vartheta_1(w_1+w_2)\vartheta_1(w_1-w_2)$ and $\vartheta_2(w_1+w_2)\vartheta_2(w_1-w_2)$. Further, if $\varphi(w_1-w_2) = [\vartheta_1(w_1-w_2) : \vartheta_2(w_1-w_2)]$ is known, then it is possible to obtain $\varphi(w_1+w_2) = [\vartheta_1(w_1+w_2) : \vartheta_2(w_1+w_2)]$ without knowing the values of w_1 and w_2.

The task of computing $\varphi(2w)$ from $\varphi(w)$ is called doubling and the task of computing $\varphi(w_1+w_2)$ from $\varphi(w_1)$, $\varphi(w_2)$ and $\varphi(w_1-w_2)$ is called differential (or pseudo) addition.

2.4 Square Only Setting

Let $P = \varphi(w) = [x : z]$ be a point on the Kummer line. As described above, doubling computes the point $2P$ and suppose that $2P = [x_3 : z_3]$. Further, suppose that instead of $[x : z]$, we have the values x^2 and z^2 and after the doubling we are interested in the values x_3^2 and z_3^2. Then the doubling operation only involves the squared quantities $\vartheta_1(0)^2, \vartheta_2(0)^2, \Theta_1(0)^2, \Theta_2(0)^2$ and x^2, z^2. As a consequence, the double of $[x : z]$ and $[x : -z]$ are same.

Similarly, consider that from $P_1 = \varphi(w_1) = [x_1 : z_1]$, $P_2 = \varphi(w_2) = [x_2 : z_2]$ and $P = P_1 - P2 = \varphi(w_1 - w_2) = [x : z]$ the requirement is to compute $P_1 + P_2 = \varphi(w_1 + w_2) = [x_3 : z_3]$. If we have the values $x_1^2, z_1^2, x_2^2, z_2^2$ and x^2, z^2 along with $\vartheta_1(0)^2, \vartheta_2(0)^2, \Theta_1(0)^2, \Theta_2(0)^2$ then we can compute the values x_3^2 and z_3^2.

This approach requires only squared values, i.e., it starts with squared values and also returns squared values. Hence, this is called the square only setting. Note that in the square only setting, $[x^2 : z^2]$ represents two points $[x : \pm z]$ on the Kummer line. For the case of genus two, the square only setting was advocated in [11, 4].

Let

$$a^2 = \vartheta_1(0)^2,\ b^2 = \vartheta_2(0)^2,\ A^2 = a^2 + b^2 \text{ and } B^2 = a^2 - b^2.$$

Then from (7) we obtain $\Theta_1(0)^2 = A^2/2$ and $\Theta_2(0)^2 = B^2/2$. *By $\mathcal{K}_{a^2,b^2}$ we denote the Kummer line having the parameters a^2 and b^2.*

We provide the details of doubling in the square only setting. Using (5), we obtain

$$\Theta_1(2w)^2 = \frac{(x^2+z^2)^2}{2A^2};\ \Theta_2(2w)^2 = \frac{(x^2-z^2)^2}{2B^2}.$$

Then from (6)

$$x_3'^2 = \vartheta_1(2w)^2 = \frac{(\Theta_1(2w)^2 + \Theta_2(2w)^2)^2}{a^2};\ z_3'^2 = \vartheta_2(2w)^2 = \frac{(\Theta_1(2w)^2 - \Theta_2(2w)^2)^2}{b^2}.$$

For $[x : z] \in \mathbb{P}^1(\mathbb{C})$, $[x : z] = [\lambda x : \lambda z]$ for any non-zero λ. Suppose that we compute λx^2 and λz^2 for some non-zero λ. Since $\mathbb{C}$ is algebraically closed, there is a $\zeta \in \mathbb{C}$ such that $\zeta^2 = \lambda$ and so we in effect have $[\zeta^2 x^2 : \zeta^2 z^2]$. By taking square roots, we obtain the point $[\zeta x : \pm\zeta z] = [x : \pm z]$. So, in the square only setting, it is sufficient to compute $[\lambda x^2 : \lambda z^2]$ for some non-zero λ. Using this, we have

$$\begin{aligned}[x_3'^2 : z_3'^2] &= \left[\frac{(\Theta_1(2w)^2 + \Theta_2(2w)^2)^2}{a^2} : \frac{(\Theta_1(2w)^2 - \Theta_2(2w)^2)^2}{b^2}\right] \\ &= \left[b^2(\Theta_1(2w)^2 + \Theta_2(2w)^2)^2 : a^2(\Theta_1(2w)^2 - \Theta_2(2w)^2)^2\right] \\ &= \left[b^2\left(\frac{(x^2+z^2)^2}{2A^2} + \frac{(x^2-z^2)^2}{2B^2}\right)^2 : a^2\left(\frac{(x^2+z^2)^2}{2A^2} - \frac{(x^2-z^2)^2}{2B^2}\right)^2\right] \\ &= \left[b^2\left(B^2(x^2+z^2)^2 + A^2(x^2-z^2)^2\right)^2 : a^2\left(B^2(x^2+z^2)^2 - A^2(x^2-z^2)^2\right)^2\right] \\ &= [x_3^2 : z_3^2].\end{aligned}$$

So, it is sufficient to compute $[x_3^2 : z_3^2]$. The computation of $[x_3^2 : z_3^2]$ is shown as Algorithm dbl in Table 2. Note that $[x_3^2 : z_3^2]$ are the squared values of the double of $[x : \pm z]$ and is not the double of $[x^2 : z^2]$. Informally, however, we will say that $[x_3^2 : z_3^2]$ is the double of $[x^2 : z^2]$. Given $x_1^2, z_1^2, x_2^2, z_2^2, x^2$ and z^2,

$\mathsf{dbl}(x^2, z^2)$	$\mathsf{diffAdd}(x_1^2, z_1^2, x_2^2, z_2^2, x^2, z^2)$
$s_0 = B^2(x^2+z^2)^2;$	$s_0 = B^2(x_1^2+z_1^2)(x_2^2+z_2^2);$
$t_0 = A^2(x^2-z^2)^2;$	$t_0 = A^2(x_1^2-z_1^2)(x_2^2-z_2^2);$
$x_3^2 = b^2(s_0+t_0)^2;$	$x_3^2 = z^2(s_0+t_0)^2;$
$z_3^2 = a^2(s_0-t_0)^2;$	$z_3^2 = x^2(s_0-t_0)^2;$
return (x_3^2, z_3^2).	return (x_3^2, z_3^2).

Table 2. Double and differential addition in the square-only setting.

the computation of x_3^2 and z_3^2 is shown as Algorithm $\mathsf{diffAdd}$ in Table 2.

In $\mathcal{K}_{a^2,b^2}$, the point $[a^2 : b^2]$ acts as the identity element for the differential addtion. Also, the double of $[b^2 : a^2]$ is $[a^2 : b^2]$ so that $[b^2 : a^2]$ is a point of order two. These facts can be proved directly from the formulas provided for doubling and differential addition. The calculations, however, become very messy. Instead, in Appendix B we provide a SAGE script which provides the symbolic verifications of these calculations.

In the rest of the paper, we will work in the square only setting over a Kummer line $\mathcal{K}_{a^2,b^2}$ for some values of the parameters a^2 and b^2.

2.5 Scalar Multiplication

Suppose $P = [x_1^2 : z_1^2]$ and n be a positive integer. We wish to compute $nP = [x_n^2 : z_n^2]$. The method for doing this is given by Algorithm $\mathsf{scalarMult}$ in Table 3. Let the ℓ-bit binary expansion of n be $n = (1, n_{\ell-2}, \ldots, n_0)$. Algorithm $\mathsf{scalarMult}$ goes through $\ell - 1$ ladder steps. Each ladder step takes the squared coordinates of two points as input and provides as output the squared coordinates of two other points.

A conceptual description of a ladder step is given in Figure 1. Suppose the squared coordinates of the two input points to a ladder step are $[x_1^2 : z_1^2]$ and $[x_2^2 : z_2^2]$. Also assume that the double of the point $[x_1^2 : z_1^2]$, and addition of the points $[x_1^2 : z_1^2]$ and $[x_2^2 : z_2^2]$ are required to be performed. Then the ladder produces $[x_3^2 : z_3^2]$ and $[x_4^2 : z_4^2]$, where $[x_3^2 : z_3^2] = \mathsf{dbl}(x_1^2 : z_1^2)$ and $[x_4^2 : z_4^2] = \mathsf{diffAdd}(x_1^2, z_1^2, x_2^2, z_2^2)$.

The input to the first ladder step are the (squared) coordinates of $(P, 2P)$. Suppose, at the i-th iteration, the input to the ladder step corresponds to $(kP, (k+1)P)$. If $n_i = 0$, then the output consists of the (squared) coordinates of the points $(2kP, (2k+1)P)$ and if $n_i = 1$, then the output consists of the (squared) coordinates of $((2k+1)P, (2k+2)P)$.

scalarMult(P, n)	ladder(R, S, b)
input: $P \in \mathcal{K}_{a,b}$; ℓ-bit scalar $n = (1, n_{\ell-2}, \ldots, n_0)$; output: nP; set $R = P$ and $S = \mathsf{dbl}(P)$; for $i = \ell-2, \ell-3, \ldots, 0$ do $(R, S) = \mathsf{ladder}(R, S, n_i)$; return R.	if $(b = 0)$ $S = \mathsf{diffAdd}(R, S, P)$; $R = \mathsf{dbl}(R)$; else $R = \mathsf{diffAdd}(R, S, P)$; $S = \mathsf{dbl}(S)$; return (R, S).

Table 3. Scalar multiplication using a ladder.

2.6 Legendre Form Elliptic Curve

Let E be an elliptic curve and $\sigma : E \to E$ be the automorphism which maps a point of E to its inverse, i.e., for $(a, b) \in E$, $\sigma(a, b) = (a, -b)$.

For $\mu \in \mathbb{F}_q$, let

$$E_\mu : Y^2 = X(X-1)(X-\mu) \tag{9}$$

be an elliptic curve in the Legendre form. Let $\mathcal{K}_{a^2,b^2}$ be a Kummer line such that

$$\mu = \frac{a^4}{a^4 - b^4}. \tag{10}$$

An explicit map $\psi : \mathcal{K}_{a^2,b^2} \to E_\mu/\sigma$ has been given in [31]. In the square only setting, let $[x^2 : z^2]$ represent the points $[x : \pm z]$ of the Kummer line $\mathcal{K}_{a^2,b^2}$ such that $[x^2 : z^2] \neq [b^2 : a^2]$. Recall that $[b^2 : a^2]$ has order two and $[a^2 : b^2]$ acts as the identity in $\mathcal{K}_{a^2,b^2}$. Then from [31],

$$\psi([x^2 : z^2]) = \begin{cases} \infty & \text{if } [x^2 : z^2] = [a^2 : b^2]; \\ \left(\frac{a^2x^2}{a^2x^2 - b^2z^2}, \ldots\right) & \text{otherwise }. \end{cases} \tag{11}$$

Given $X = a^2x^2/(a^2x^2 - b^2z^2)$, it is possible to find $\pm Y$ from the equation of E, though it is not possible to uniquely determine the sign of Y. The inverse ψ^{-1} maps an element of E_μ/σ to the squared coordinates of points in $\mathcal{K}_{a^2,b^2}$. Let $\mathbf{P} = (X, \ldots) \in E_\mu/\sigma$ be a point which is not of order two so that $X \neq 0, 1, \mu$. Then

$$\psi^{-1}(\mathbf{P}) = \begin{cases} [a^2 : b^2] & \text{if } \mathbf{P} = \infty; \\ \left[\frac{b^2X}{a^2(X-1)} : 1\right] & \text{if } \mathbf{P} = (X, \ldots). \end{cases} \tag{12}$$

Notation: We will use upper-case bold face letters to denote points of E_μ and upper case normal letters to denote points of $\mathcal{K}_{a^2,b^2}$.

2.7 Consistency

Let $\mathcal{K}_{a^2,b^2}$ and E_μ be such that (10) holds. Consider the point $\mathbf{T} = (\mu, 0)$ on E_μ. Note that $\mathbf{T}$ is a point of order two. Given any point $\mathbf{P} = (X, \ldots)$ of E_μ, let $\mathbf{Q} = \mathbf{P} + \mathbf{T}$. Then it is easy to verify that

$$\mathbf{Q} = \left(\frac{\mu(X-1)}{X-\mu}, \ldots\right).$$

Consider the map $\widehat{\psi} : \mathcal{K}_{a^2,b^2} \to E_\mu$ such that for a point $[x : \pm z]$ represented by $[x^2 : z^2]$ in the square only setting

$$\widehat{\psi}([x^2 : z^2]) = \psi([x^2 : z^2]) + \mathbf{T}. \tag{13}$$

The inverse map $\widehat{\psi}^{-1}$ takes a point $\mathbf{P}$ of E_μ to squared coordinates in $\mathcal{K}_{a^2,b^2}$ and is given by

$$\widehat{\psi}^{-1}(\mathbf{P}) = \psi^{-1}(\mathbf{P} + \mathbf{T}). \tag{14}$$

(Since $\mathbf{T}$ is a point of order two, $\mathbf{T} = -\mathbf{T}$.)

For any points $\mathbf{P}_1, \mathbf{P}_2$ on E_μ which are not of order two and $\mathbf{P} = \mathbf{P}_1 - \mathbf{P}_2$ the following properties hold.

$$\left.\begin{aligned} \mathsf{dbl}\left(\widehat{\psi}^{-1}(\mathbf{P}_1)\right) &= \widehat{\psi}^{-1}\left(2\mathbf{P}_1\right); \\ \mathsf{diffAdd}\left(\widehat{\psi}^{-1}(\mathbf{P}_1), \widehat{\psi}^{-1}(\mathbf{P}_2), \widehat{\psi}^{-1}(\mathbf{P})\right) &= \widehat{\psi}^{-1}\left(\mathbf{P}_1 + \mathbf{P}_2\right). \end{aligned}\right\} \tag{15}$$

The proofs of these statements can be derived from the formulas for addition and doubling on E_μ and the formulas arising from the Algorithms dbl and diffAdd. The SAGE script provided in Appendix B does the symbolic verification of the required calculations.

The relations given by (15) have an important consequence to scalar multiplication. Suppose P is in $\mathcal{K}_{a^2,b^2}$ and $\mathbf{P} = \widehat{\psi}(P)$. Then $\widehat{\psi}(nP) = n\mathbf{P}$. Figure 3 depicts this in pictorial form.

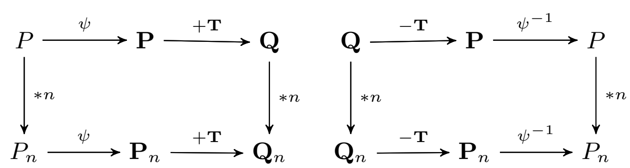

Fig. 3. Consistency of scalar multiplications on E_μ and $\mathcal{K}_{a^2,b^2}$.

2.8 Relation Between the Discrete Logarithm Problems

Suppose the Kummer line $\mathcal{K}_{a^2,b^2}$ is chosen such that the corresponding curve E_μ has a cyclic subgroup $\mathfrak{G}$ which can be used for cryptographic purposes. So, in particular, the order of $\mathfrak{G}$ is a prime. Let $\mathfrak{G} = \langle \mathbf{P} \rangle$.

Given $\mathbf{Q} \in \mathfrak{G}$, the discrete logarithm problem in $\mathfrak{G}$ is to obtain an n such that $\mathbf{Q} = n\mathbf{P}$. This problem can be reduced to computing discrete logarithm problem in $\mathcal{K}_{a^2,b^2}$. Map the point $\mathbf{P}$ (resp. $\mathbf{Q}$) to $P \in \mathcal{K}_{a,b}$ (resp. $Q \in \mathcal{K}_{a,b}$) using $\widehat{\psi}^{-1}$ Find n such that $Q = nP$ and return n. Similarly, the discrete logarithm problem in $\mathcal{K}_{a,b}$ can be reduced to the discrete logarithm problem in E_μ.

The above shows the equivalence of the hardness of solving the discrete logarithm problem in either E_μ or in $\mathcal{K}_{a^2,b^2}$. So, if E_μ is a well chosen curve such that the discrete logarithm problem in E_μ is conjectured to be hard, then the discrete logarithm problem in the associated $\mathcal{K}_{a^2,b^2}$ will be equally hard. This fact forms the basis for using Kummer line for cryptographic applications.

2.9 Recovering y-Coordinate

Suppose $\mathbf{Q} = (X_Q, Y_Q), \mathbf{R} = (X_R, Y_R), \mathbf{S} = (X_S, Y_S)$ are points in E_μ such that $\infty \neq \mathbf{Q} = \mathbf{R} - \mathbf{S}$, $\mathbf{Q} \neq \mathbf{R}$ and $\mathbf{Q}$ is not a point of order 2. The last two conditions imply that $X_Q \neq X_R$ and $Y_Q \neq 0$. So, it is allowed to divide by both $(X_R - X_Q)$ and Y_Q.

Suppose that X_Q, Y_Q, X_R and X_S are known. We show that Y_R is uniquely determined and can be computed from these four quantities. This is based on a similar calculation in [42, 14]. For the genus two case, this problem has been addressed in [17].

Consider the chord-and-tangent rule for addition on E_μ. The points $\mathbf{R}$ and $-\mathbf{S}$ determine a line $Y = mX + c$. This line intersects the curve E_μ at the point $-\mathbf{Q} = (X_Q, -Y_Q)$. So, m can be determined as $m = (Y_R + Y_Q)/(X_R - X_Q)$. Substituting $Y = mX + c$ into the equation of the curve we obtain:

$$X^3 - (\mu + 1 + m^2)X^2 + (\mu - 2mc)X - c^2 = 0.$$

Since X_Q, X_R, X_S are roots of this equation, we have

$$X_Q + X_R + X_S = \mu + 1 + m^2.$$

Using the value for $m = (Y_R + Y_Q)/(X_R - X_Q)$, we have

$$(Y_R + Y_Q)^2 = (X_R - X_Q)^2(X_Q + X_R + X_S - \mu - 1).$$

Write $f(X) = X(X - 1)(X - \mu)$. Then $Y_R^2 = f(X_R)$ and we obtain

$$Y_R = \frac{1}{2Y_Q}\left((X_R - X_Q)^2(X_Q + X_R + X_S - \mu - 1) - f(X_R) - Y_Q^2\right).$$

2.10 Scalar Multiplication in E_μ

Let E_μ be a Legendre form curve and $\mathcal{K}_{a^2,b^2}$ be a Kummer line in square only setting. Suppose $\mathfrak{G} = \langle \mathbf{P} = (X_P, Y_P)\rangle$ is a cryptographically relevant subgroup of E_μ. Further, suppose a point $P = [x^2 : z^2]$ in $\mathcal{K}_{a^2,b^2}$ is known such that $(X_P, \ldots) = \widehat{\psi}(P) = \psi(P) + \mathbf{T}$ where as before $\mathbf{T} = (\mu, 0)$. The point P is the base point on $\mathcal{K}_{a^2,b^2}$ which corresponds to the point $\mathbf{P}$ on E_μ.

Let n be a non-negative integer which is less than the order of $\mathfrak{G}$. We show how to compute the scalar multiplication $n\mathbf{P}$ via the laddering algorithm on the Kummer line $\mathcal{K}_{a^2,b^2}$. First, the ladder algorithm is applied to the input P and n. This results in a pair of points Q and R, where $Q = nP$ and $R = (n+1)P$ so that $Q - R = -P$. The square only ladder algorithm will return $[x_Q^2 : z_Q^2]$ to represent Q and $[x_R^2 : z_R^2]$ to represent R. Let $\mathbf{Q} = \widehat{\psi}(Q) = \psi(Q) + \mathbf{T}$ and $\mathbf{R} = \widehat{\psi}(R) = \psi(R) + \mathbf{T}$. By the consistency of scalar multiplication, we have $\mathbf{Q} = n\mathbf{P}$.

Consider $\mathbf{Q} = \psi(Q) + \mathbf{T}$. Let $\alpha_Q = a^2x_Q^2$ and $\beta_Q = a^2x_Q^2 - b^2z_Q^2$ so that $\psi(Q) = (\alpha_Q/\beta_Q, \ldots)$. Writing $\mathbf{Q} = (X_Q, Y_Q)$ and applying the addition rule on E_μ we obtain $X_Q = \gamma_Q/\delta_Q$ where $\gamma_Q = \mu(\alpha_Q - \beta_Q)$ and $\delta_Q = \alpha_Q - \mu\beta_Q$. Similarly, we obtain $\mathbf{R} = (X_R, \ldots)$ where $X_R = \gamma_R/\delta_R$ and γ_R, δ_R are given by the previous expression with Q replaced by R.

At this point, we have $\mathbf{P} = (X_P, Y_P)$, $\mathbf{Q} = (X_Q, Y_Q)$ and $\mathbf{R} = (X_R, \ldots)$ where $\mathbf{Q} - \mathbf{R} = -\mathbf{P}$. The y-coordinate Y_Q of $\mathbf{Q}$ can be recovered as discussed in Section 2.9. This gives

$$\begin{aligned} Y_Q &= \frac{-1}{2Y_P}\left((X_Q - X_P)^2(X_P + X_Q + X_R - \mu - 1) - f(X_Q) - Y_P^2\right) \\ &= -\frac{1}{2Y_P}\left(\left(\frac{\gamma_Q}{\delta_Q} - X_P\right)^2\left(X_P + \frac{\gamma_Q}{\delta_Q} + \frac{\gamma_R}{\delta_R} - \mu - 1\right) - f\left(\frac{\gamma_Q}{\delta_Q}\right) - Y_P^2\right). \end{aligned}$$

This shows that given n, it is possible to compute $\mathbf{Q} = (X_Q, Y_Q)$ such that $\mathbf{Q} = n\mathbf{P}$.

It is required to compute both Y_Q and X_Q. Using Montgomery's trick, the two inversions required for computing Y_Q and X_Q can be done using one inversion and 3 multiplications. So, the entire computation of X_Q and Y_Q from Q, R and P can be done using one inversion and a few multiplications in $\mathbb{F}_p$. The main time consuming step will be that of the inversion. If projective coordinates are used to represent the point of E_μ, then the field inversion can be avoided.

3 Kummer Line Over Prime Order Fields

Let p be a prime and $\mathbb{F}_p$ be the field of p elements. As mentioned earlier, using on the Lefschetz principle, the theta identities also hold over $\mathbb{F}_p$. Consequently, it is possible to work over a Kummer line $\mathcal{K}_{a^2,b^2}$ and associated elliptic curve E_μ defined over the algebraic closure of $\mathbb{F}_p$. The only condition for this to be meaningful is that $a^4 - b^4 \neq 0 \bmod p$ so that $\mu = a^4/(a^4 - b^4)$ is defined over $\mathbb{F}_p$. We choose a^2 and b^2 to be small values while p is a large prime and so the condition $a^4 - b^4 \neq 0 \bmod p$ easily holds. Note that we will choose a^2 and b^2 to be in $\mathbb{F}_p$ without necessarily requiring a and b themselves to be in $\mathbb{F}_p$. Similarly, in the square only setting when we work with squared representation $[x^2 : z^2]$ of points $[x : \pm z]$, the values x^2, z^2 will be in $\mathbb{F}_p$ and it is not necessary for x and z themselves to be in $\mathbb{F}_p$.

Our target is the 128-bit security level. To this end, we consider the three primes $p2519$, $p25519$ and $p2663$. The choice of these three primes is motivated by the consideration that these are of the form $2^m - \delta$, where m is around 256 and δ is a small positive integer. For m in the range 250 to 270 and $\delta < 20$, the only three primes of the form $2^m - \delta$ are $p2519$, $p25519$ and $p2663$. We later discuss the comparative advantages and disadvantages of using Kummer lines based on these three primes.

3.1 Finding a Secure Kummer Line

For each prime p, the procedure for finding a suitable Kummer line is the following. The value of a^2 is increased from 2 onwards and for each value of a^2, the value of b^2 is varied from 1 to $a^2 - 1$; for each pair (a^2, b^2), the value of $\mu = a^4/(a^4 - b^4)$ is computed and the order of $E_\mu(\mathbb{F}_p)$ is computed. Let $t = p + 1 - \#E_\mu(\mathbb{F}_p)$. Let ℓ and ℓ_T be the largest prime factors of $p + 1 - t$ and $p + 1 + t$ respectively and let $h = (p + 1 - t)/\ell$ and $h_T = (p + 1 + t)/\ell_T$. Here h and h_T are the co-factors of the curve and its quadratic twists respectively. If both h and h_T are small, then (a^2, b^2) is considered. Among the possible (a^2, b^2) that were obtained, we have used the one with the minimum value of a^2. After fixing (a^2, b^2) the following parameters for E_μ have been computed.

1. Embedding degrees k and k_T of the curve and its twist. Here k (resp. k_T) is the smallest positive integer such that $\ell | p^k - 1$ (resp. $\ell_T | p^{k_T} - 1$). This is given by the order of p in $\mathbb{F}_\ell$ (resp. $\mathbb{F}_{\ell_T}$) and is found by checking the factors of $\ell - 1$ (resp. $\ell_T - 1$).
2. The complex multiplication field discriminant D. This is defined in the following manner (`https://safecurves.cr.yp.to/disc.html`): By Hasse's theorem, $|t| \leq 2\sqrt{p}$ and in the cases that we considered $|t| < 2\sqrt{p}$ so that $t^2 - 4p$ is a negative integer; let s^2 be the largest square dividing $t^2 - 4p$; define $D = (t^2 - 4p)/s^2$ if $t^2 - 4p \bmod 4 = 1$ and $D = 4(t^2 - 4p)/s^2$ otherwise. (Note that D is different from the discriminant of E_μ which is equal to $\mu^4 - 2\mu^3 + \mu^2$.)

Table 4 provides the three Kummer lines and (estimates of) the sizes of the the various parameters. As part of [26], we provide Magma code for computing these parameters and also their exact values. The Kummer line $\mathcal{K}_{a^2,b^2}$ over $p2519$ is compactly denoted as $\mathsf{KL2519}(a^2, b^2)$ and similarly for Kummer lines over $p25519$ and $p2663$. For each Kummer line reported in Table 4, the base point $[x^2 : z^2]$ is such that its order is ℓ. Table 4 also provides the corresponding details for Curve25519 and P-256. This will help in comparing the new proposals with the two most important proposals over prime fields that are present in the literature.

We note that for $\mathsf{KL2519}(81, 20)$, $[15 : 1]$ is another choice of base point. Also, for $p2519$, $\mathsf{KL2519}(101, 61)$ is another good choice for which both h and h_T are 8, the other security parameters have large values and $[4 : 1]$ is a base point. For $p25519$ and $p2663$, for a^2 up to 512, we were unable to find any Kummer line for which both h and h_T are at most 8.

Based on the values listed in Table 4 it is possible to conclude that the new proposals provide security at approximately the 128-bit security level.

Table 4. New Kummer lines and their parameters in comparison to Curve25519 and P-256.

	KL2519(81, 20)	KL25519(82, 77)	KL2663(260, 139)	Curve25519 [3]	P-256 [45]
$(\lg \ell, \lg \ell_T)$	$(248, 248)$	$(251.4, 252)$	$(262.4, 263)$	$(252, 253)$	$(256, 240)$
(h, h_T)	$(8, 8)$	$(12, 8)$	$(12, 8)$	$(8, 4)$	$(1, 3 \cdot 5 \cdot 13 \cdot 179)$
(k, k_T)	$(\ell - 1, (\ell_T - 1)/7)$	$(\ell - 1, \ell_T - 1)$	$((\ell - 1)/2, \ell_T - 1)$	$((\ell - 1)/6, \ell_T - 1)$	$((\ell - 1)/3, (\ell_T - 1)/2)$
$\lg(-D)$	246.3	255	266	254.7	258
base point	$[64 : 1]$	$[31 : 1]$	$[2 : 1]$	$(9, \ldots)$	large

4 Field Arithmetic

We consider three primes $p2519 = 2^{251} - 9$, $p25519 = 2^{255} - 19$ and $p2663 = 2^{266} - 3$. The general form of these primes is $p = 2^m - \delta$. Let η and ν be such that $m = \eta(\kappa - 1) + \nu$ with $0 \leq \nu < \eta$. The values of m, δ, κ, η and ν for $p2519$, $p25519$ and $p2663$ are given in Table 5. The value of κ indicates the number of limbs used to represent elements of $\mathbb{F}_p$; the value of η represents the number of bits in the first $\kappa - 1$ limbs; and the value of ν is the number of bits in the last limb. For each prime two sets of values of κ, η and ν are provided. This indicates that two different representations of each prime will be used.

1. For a single field multiplication, the representation with smaller κ will be used. In this case, each limb will fit into a 64-bit word. A field multiplication is computed using several $64 \times 64 \to 128$ multiplications.
2. A Kummer line allows the execution of four simultaneous multiplications (and four simultaneous squarings). This can be computed using SIMD instructions (specifically, the AVX2 instructions on modern Intel processors). To avail such instructions, the limbs of four field elements are stored in one 256-bit word. A single SIMD multiplication performs four simultaneous $32 \times 32 \to 64$ multiplications. In this case, the representation of field elements with the larger value of κ will be used.

The scalar multiplication on the Kummer line will be computed entirely using SIMD instructions. At the end, the result $[x^2 : z^2]$ is obtained and it is required to return $w = x^2/z^2$. This involves an inversion in $\mathbb{F}_p$. The inversion is done using a fixed addition chain to compute $w^{p-2} \bmod p$. This requires multiplications and squarings over $\mathbf{F}_p$ which do not involve SIMD instructions. So, the representation with longer κ is changed to the representation with smaller κ and then the inverse is computed.

Table 5. The different values of κ, η and ν corresponding to the primes $p2519$, $p25519$ and $p2663$.

prime	m	δ	κ	η	ν
$p2519$	251	9	9	28	27
			5	51	47
$p25519$	255	19	10	26	21
			5	51	51
$p2663$	266	3	10	27	23
			5	54	50

In the following sections, we describe methods to perform arithmetic over $\mathbb{F}_p$. Most of the description is in general terms of κ, η and ν. The specific values of κ, η and ν are required only when we argue that no overfull occurs.

4.1 Representation of Field Elements

Let $\theta = 2^\eta$ and consider the polynomial $A(\theta)$ defined in the following manner:

$$A(\theta) = a_0 + a_1\theta + \cdots + a_{\kappa-1}\theta^{\kappa-1} \tag{16}$$

where $0 \le a_0, \ldots, a_{\kappa-1} < 2^\eta$ and $0 \le a_{\kappa-1} < 2^\nu$. Such a polynomial will be called a *proper* polynomial.

Note that proper polynomials are in 1-1 correspondence with the integers $0, \ldots, 2^m - 1$. This leads to non-unique representation of some elements of $\mathbb{F}_p$: specifically, the elements $0, \ldots, \delta - 1$ are also represented as $2^m - \delta, \ldots, 2^m - 1$. This, however, does not cause any of the computations to become incorrect. Conversion to unique representation is done once at the end of the computation. The issue of non-unique representation was already mentioned in [3] where the following was noted: 'Note that integers are not converted to a unique "smallest" representation until the end of the Curve25519 computation. Producing reduced representations is generally much faster than producing "smallest" representations.'

4.2 Representation of the Prime p

The representation of the prime p will be denoted by $\mathfrak{P}(\theta)$ where

$$\begin{aligned}
\mathfrak{P}(\theta) &= \textstyle\sum_{i=0}^{\kappa-1} \mathfrak{p}_i\theta^i \text{ with} \\
\mathfrak{p}_0 &= 2^\eta - \delta; \\
\mathfrak{p}_i &= 2^\eta - 1; \qquad i = 1, \ldots, \kappa - 2; \text{ and} \\
\mathfrak{p}_{\kappa-1} &= 2^\nu - 1.
\end{aligned} \tag{17}$$

This representation will only be required for the longer value of κ.

4.3 Reduction

This operation will be required for both values of κ.

Using $p = 2^m - \delta$, for $i \ge 0$, we have $2^{m+i} = 2^i \times 2^m = 2^i(2^m - \delta) + 2^i\delta \equiv 2^i\delta \bmod p$. So, multiplying by 2^{m+i} modulo p is the same as multiplying by $2^i\delta$ modulo p. Recall that we have set $\theta = 2^\eta$ and so $\theta^\kappa = 2^{\eta\kappa} = 2^{m+\eta-\nu}$ which implies that

$$\theta^\kappa \bmod p = 2^{\eta-\nu}\delta. \tag{18}$$

Suppose $C(\theta) = \sum_{i=0}^{\kappa-1} c_i\theta^i$ is a polynomial such that for some $\mathfrak{m} \le 64$, $c_i < 2^{\mathfrak{m}}$ for all $i = 0, \ldots, 7$. If for some $i \in \{0, \ldots, \kappa - 2\}$, $c_i \ge 2^\eta$, or $c_{\kappa-1} \ge 2^\nu$, then $C(\theta)$ is not a proper polynomial. Following the idea in [3, 9, 16], we describe a method to obtain a polynomial $D(\theta) = \sum_{i=0}^{\kappa-1} d_i\theta^i$ such that $D(\theta) \equiv C(\theta) \bmod p$.

$\mathsf{reduce}(C(\theta))$
input: $C(\theta) = c_0 + c_1\theta + \cdots + c_{\kappa-1}\theta^{\kappa-1}$, $c_i < 2^{\mathfrak{m}}$, $i = 0, \ldots, \kappa - 1$;
output: polynomial $D(\theta)$ such that $D(\theta) \equiv C(\theta) \bmod p$;
$s_0 \leftarrow 0$;
for $i = 0, \ldots, \kappa - 2$ do
    $d_i \leftarrow \mathsf{lsb}_\eta(c_i + s_i)$; $s_{i+1} \leftarrow (c_i + s_i)/2^\eta$;
end for;
$d_{\kappa-1} \leftarrow \mathsf{lsb}_\nu(c_{\kappa-1} + s_{\kappa-1})$; $t_0 = (c_{\kappa-1} + s_{\kappa-1})/2^\nu$;
$e_0 \leftarrow \mathsf{lsb}_\eta(d_0 + 2^{\eta-\nu}\delta t_0)$; $t_1 \leftarrow (d_0 + 2^{\eta-\nu}\delta t_0)/2^\eta$; $[t_2 \leftarrow \lfloor (d_1 + t_1)/2^\eta \rfloor]$
$d_0 \leftarrow e_0$; $d_1 \leftarrow d_1 + t_1$;
return $D(\theta)$.

For $i = 0, \ldots, \kappa - 2$, Step 3 ensures $c_i + s_i = d_i + 2^\eta s_{i+1}$ and $d_i < 2^\eta$; Step 5 ensures $c_{\kappa-1} + s_{\kappa-1} = d_{\kappa-1} + 2^\nu t_0$ and $d_{\kappa-1} < 2^\nu$. In Step 6, t_2 is actually not computed, it is provided for the ease of analysis.

Using $\theta = 2^\eta$, the computation can be written out more explicitly in the following manner.

$$\begin{aligned}
C(\theta) &= c_0 + c_1\theta + \cdots + c_{\kappa-1}\theta^{\kappa-1} \\
&= (d_0 + s_1\theta) + c_1\theta + \cdots + c_{\kappa-1}\theta^{\kappa-1} \\
&= d_0 + (s_1 + c_1)\theta + \cdots + c_{\kappa-1}\theta^{\kappa-1} \\
&= d_0 + ((d_1 + s_2\theta))\theta + \cdots + c_{\kappa-1}\theta^{\kappa-1} \\
&= d_0 + d_1\theta + (s_2 + c_2)\theta^2 + \cdots + c_{\kappa-1}\theta^{\kappa-1} \\
&\cdot\ \cdots \\
&= d_0 + d_1\theta + d_2\theta^2 + \cdots + (s_{\kappa-1} + c_{\kappa-1})\theta^{\kappa-1} \\
&= d_0 + d_1\theta + d_2\theta^2 + \cdots + (d_{\kappa-1} + t_0\theta)\theta^{\kappa-1} \\
&= d_0 + d_1\theta + d_2\theta^2 + \cdots + d_{\kappa-1}\theta^{\kappa-1} + t_0\theta^\kappa \\
&\equiv (d_0 + 2^{\eta-\nu}\delta t_0) + d_1\theta + d_2\theta^2 + \cdots + d_{\kappa-1}\theta^{\kappa-1} \pmod p \\
&= (e_0 + t_1\theta) + d_1\theta + d_2\theta^2 + \cdots + d_{\kappa-1}\theta^{\kappa-1} \\
&= e_0 + (d_1 + t_1)\theta + d_2\theta^2 + \cdots + d_{\kappa-1}\theta^{\kappa-1} \\
&= D(\theta).
\end{aligned}$$

We have used the fact that $\theta^\kappa \equiv 2^{\eta-\nu}\delta \bmod p$. This computation shows that $D(\theta) \equiv C(\theta) \bmod p$.

For $i = 1, \ldots, \kappa - 1$, let $\mathfrak{b}_i$ be the maximum value that s_i can take; let $\mathfrak{d}_0$ and $\mathfrak{d}_1$ respectively be the maximum values that t_0 and t_1 can take. Then

- $\mathfrak{b}_1 = 2^{\mathfrak{m}-\eta}$;
- $\mathfrak{b}_i = \lfloor (2^{\mathfrak{m}} + \mathfrak{b}_{i-1})/2^\eta \rfloor$, for $i = 2, \ldots, \kappa - 1$;
- $\mathfrak{d}_0 = \lfloor (2^{\mathfrak{m}} + \mathfrak{b}_{\kappa-1})/2^\nu \rfloor$;
- $\mathfrak{d}_1 = \lfloor (2^\eta - 1 + 2^{\eta-\nu}\delta\mathfrak{d}_0)/2^\eta \rfloor$.

The values of $\mathfrak{b}_i$, $\mathfrak{d}_0$ and $\mathfrak{d}_1$ are determined entirely by $\mathfrak{m}$, η, ν and δ. The maximum possible value of $\mathfrak{m}$ is 64 and the values of η and ν are determined by the choice of the prime p. For each of the primes $p2519$, $p25519$ and $p2663$, it turns out that $\mathfrak{d}_1 < 2^\eta - 1$. Since $d_1 \leq 2^\eta - 1$, $t_2 \leq 1$ and so the updated value of d_1 at Step 7 is less than $2^{\eta+1} - 2$. So, if $t_2 = 1$, then $\mathsf{lsb}_\eta(d_1)$ is less than $2^\eta - 1$. So, $D(\theta)$ is not necessarily a proper polynomial as the bound on d_1 can possibly be violated, though the bounds on all the other d_i's hold.

We first argue that $\mathsf{reduce}(D(\theta))$ is indeed a proper polynomial. Suppose $\mathsf{reduce}(D(\theta))$ returns $D'(\theta) = \sum_{i=0}^{\kappa-1} d'_i\theta^i$. The values of $d'_0, \ldots, d'_{\kappa-1}$ are computed by the reduce algorithm with d'_0 and d'_1 being first computed in Step 3 and then updated in Step 7. Let $s'_1, \ldots, s'_{\kappa-1}, t'_0, t'_1$ be the values of the s and t variables when reduce is applied to $D(\theta)$. Since $d_0 < 2^\eta$, $s'_1 = 0$ and we have $s'_2 = t_2$. It is now easy to argue that the corresponding $s'_3, \ldots, s'_{\kappa-1}, t'_0, t'_1$ are all at most 1. We have $d'_0 = d_0$ and $d'_1 = \mathsf{lsb}_\eta(d_1)$ at Step 3. If $t_2 = 1$, then $d'_1 < 2^\eta - 1$ and so $d'_1 + t'_1 \leq 2^\eta - 1$. So, for $D'(\theta)$ that is returned, we have $d'_0, \ldots, d'_{\kappa-2} < 2^\eta$ and $d_{\kappa-1} < 2^\nu$. This shows that $D'(\theta)$ is indeed a proper polynomial.

So, two successive invocations of reduce on $C(\theta)$ reduces it to a proper polynomial. In practice, however, this is not done at each step. Only one invocation is made. As observed above, $\mathsf{reduce}(C(\theta))$ returns $D(\theta)$ for which all coefficients $d_0, d_2, \ldots, d_{\kappa-1}$ satisfy the appropriate bounds and only d_1 theta can possibly require $\eta + 1$ bits to represent instead of the required η-bit representation. This does not cause any overfull in the intermediate computation and so we do not reduce $D(\theta)$ further. It is only at the end, that an additional invocation of reduce is made to ensure that a proper polynomial is obtained on which we apply the $\mathsf{makeUnique}$ procdure to ensure unique representation of elements of $\mathbb{F}_p$.

4.4 Field Addition

This operation will only be required for the representation using the longer value of κ.

Let $A(\theta) = \sum_{i=0}^{\kappa-1} a_i\theta^i$ and $B(\theta) = \sum_{i=0}^{\kappa-1} b_i\theta^i$ be two polynomials. Let $C(\theta) = \sum_{i=0}^{\kappa-1} c_i\theta^i$ where $c_i = a_i + b_i$ for $i = 0, \ldots, \kappa - 1$. From the bounds on a_i and b_i, we have $c_i < 2^{\eta+1} - 1$ for $i = 0, \ldots, \kappa - 2$ and $c_{\kappa-1} < 2^{\nu+1} - 1$. The operation $\mathsf{sum}(A(\theta), B(\theta))$ is defined to be $D(\theta)$ which is obtained as $D(\theta) = \mathsf{reduce}(C(\theta))$.

4.5 Field Negation

This operation will only be required for the representation using the longer value of κ.

Let $A(\theta) = \sum_{i=0}^{\kappa-1} a_i\theta^i$ be a polynomial. We wish to compute $-A(\theta) \bmod p$. Let $\mathfrak{n}$ be the least integer such that all the coefficients of $2^{\mathfrak{n}}\mathfrak{P}(\theta) - A(\theta)$ are non-negative. By $\mathsf{negate}(A(\theta))$ we denote $T(\theta) = 2^{\mathfrak{n}}\mathfrak{P}(\theta) - A(\theta)$. Reducing $T(\theta)$ modulo p gives the desired answer. Let $T(\theta) = \sum_{i=0}^{\kappa-1} t_i\theta^i$ so that $t_i = 2^{\mathfrak{n}}\mathfrak{p}_i - a_i \geq 0$.

The condition of non-negativity on the coefficients of $T(\theta)$ eliminates the situation in two's complement subtraction, where the result can be negative. Considering all values to be 64-bit quantities, the computation of t_i is done in the following manner: $t_i = ((2^{64} - 1) - a_i) + (1 + 2^{\mathfrak{n}}\mathfrak{p}_i) \bmod 2^{64}$. The operation $(2^{64} - 1) - a_i$ is equivalent to taking the bitwise complement of a_i which is equivalent to $1^{64} \oplus a_i$.

From (17), $\mathfrak{p}_0 = 2^\eta - \delta$, $\mathfrak{p}_i = 2^\eta - 1$ for $i = 1, \ldots, \kappa - 2$ and $\mathfrak{p}_{\kappa-1} = 2^\nu - 1$.

1. If $A(\theta)$ is a proper polynomial, then $\mathfrak{n} = 1$ is sufficient to ensure the non-negativity constraint on the coefficients of $T(\theta)$. Using $\mathfrak{n} = 1$, ensures that $t_0, \ldots, t_{\kappa-2} \leq 2\mathfrak{p}_i = 2^{\eta+1} - 2$ and $t_{\kappa-1} \leq 2\mathfrak{p}_{\kappa-1} = 2^{\nu+2} - 2$. So, $t_0, \ldots, t_{\kappa-2}$ can be represented using $\eta + 1$ bits and $t_{\kappa-1}$ can be represented using $\nu + 1$ bits.
2. More generally, suppose that $A(\theta)$ is equal to $2^{\mathfrak{r}}$ times a proper polynomial. Then choosing $\mathfrak{n} = \mathfrak{r} - \nu + 1$ is sufficient to ensure the non-negativity condition on the coefficients of $T(\theta)$.

Later we explain how the above two situations arise.

Given the operation of negation, subtraction can be done by first negating the subtrahend and then adding to the minuend followed by a reduction.

4.6 Multiplication by a Small Constant

This operation will only be required for the representation using the longer value of κ.

Let $A(\theta) = \sum_{i=0}^{\kappa-1} a_i\theta^i$ be a polynomial and c be a small positive integer considered to be an element of $\mathbb{F}_p$. In our applications, c will be at most 9 bits. The operation $\mathsf{constMult}(A(\theta), c)$ will denote the polynomial $C(\theta) = \sum_{i=0}^{\kappa-1}(ca_i)\theta^i$. We do not apply the algorithm reduce to $C(\theta)$. This is because in our application, multiplication by a constant will be followed by a Hadamard operation and the reduce algorithm is applied after the Hadamard operation. This improves efficiency.

4.7 Field Multiplication

This operation is required for both values of κ.

Suppose that $A(\theta) = \sum_{i=0}^{\kappa-1} a_i\theta^i$ and $B(\theta) = \sum_{i=0}^{\kappa-1} b_i\theta^i$ are to be multiplied. The operation mult is the following:

input: $\mathsf{mult}(A(\theta), B(\theta))$
output: $C(\theta)$
1. $C(\theta) \leftarrow \mathsf{polyMult}(A(\theta), B(\theta))$;
2. $[C(\theta) \leftarrow \mathsf{expand}(C(\theta))]$; (required only for $p25519$)
3. $C(\theta) \leftarrow \mathsf{fold}(C(\theta))$;
4. return $\mathsf{reduce}(C(\theta))$.

Let $C(\theta)$ be the result of $\mathsf{polyMult}(A(\theta), B(\theta))$. Then $C(\theta)$ can be written as

$$C(\theta) = c_0 + c_1\theta + \cdots + c_{2\kappa-2}\theta^{2\kappa-2} \tag{19}$$

where $c_t = \sum_{s=0}^{t} a_s b_{t-s}$ with the convention that a_i, b_j is zero for $i, j > \kappa - 1$. For $s = 0, \ldots, \kappa - 1$, the coefficient $c_{\kappa-1\pm s}$ is the sum of $(\kappa - s)$ products of the form $a_i b_j$. Since $a_i, b_j < 2^\eta$, it follows that for $s = 0, \ldots, \kappa - 1$,

$$c_{\kappa-1\pm s} \leq (\kappa - s)(2^\eta - 1)^2. \tag{20}$$

In particular, for all three of $p2519$, $p25519$ and $p2663$, each c_t fits in a 64-bit word.

Step 2 of $\mathsf{mult}(A(\theta), B(\theta))$, i.e., $\mathsf{expand}(C(\theta))$ is required only for $p25519$ and is not required for either $p2519$ or $p2663$. The step $\mathsf{polyMult}$ multiplies $A(\theta)$ and $B(\theta)$ as polynomials in θ and returns the result polynomial of degree $2\kappa - 2$. In case of $p25519$, the step expand is applied to this polynomial and returns a polynomial of degree $2\kappa - 1$; for $p2519$ and $p2663$, this step is not required. For uniformity we assume that the input to fold is a polynomial of degree $2\kappa - 1$ where for $p2519$ and $p2663$, the highest degree coefficient is 0.

The computation of $\mathsf{fold}(C(\theta))$ is described as follows.

$$\begin{aligned}
C(\theta) &= c_0 + c_1\theta + \cdots + c_{\kappa-1}\theta^{\kappa-1} + \theta^\kappa \left(c_\kappa + c_{\kappa+1}\theta + \cdots + c_{2\kappa-1}\theta^{\kappa-1}\right) \\
&\equiv c_0 + c_1\theta + \cdots + c_{\kappa-1}\theta^{\kappa-1} + 2^{\eta-\nu}\delta \left(c_\kappa + c_{\kappa+1}\theta + \cdots + c_{2\kappa-1}\theta^{\kappa-1}\right) \bmod p \\
&= (c_0 + \mathfrak{h}c_\kappa) + (c_1 + \mathfrak{h}c_{\kappa+1})\theta + \cdots + (c_{\kappa-1} + \mathfrak{h}c_{2\kappa-1})\theta^{\kappa-1}
\end{aligned}$$

where $\mathfrak{h} = 2^{\eta-\nu}\delta$. The polynomial in the last line is the output of $\mathsf{fold}(C(\theta))$. For $p2519$ and $p2663$, if $A(\theta)$ and $B(\theta)$ are proper polynomials, $\mathsf{fold}(\mathsf{polyMult}(A(\theta), B(\theta)))$ returns a polynomial in θ of degree $\kappa - 1$, where all the coefficients fit into 64-bit words.

For $p25519$, however, some of the coefficients of $\mathsf{fold}(\mathsf{polyMult}(A(\theta), B(\theta)))$ do not fit into 64-bit words. This was already mentioned in [3] and it is for this reason that the "base 2^{26} representation" was discarded. We show that the problem can be handled quite easily using the expand routine.

$\mathsf{expand}(C(\theta))$
input: $C(\theta) = c_0 + c_1\theta + \cdots + c_{2\kappa-2}\theta^{2\kappa-2}$
output: $D(\theta) = d_0 + d_1\theta + \cdots + d_{2\kappa-1}\theta^{2\kappa-1}$
1. for $i = 0, \ldots, \kappa - 1$, $d_i \leftarrow c_i$;
2. $s_0 \leftarrow 0$;
3. for $i = 0, \ldots, \kappa - 2$, $d_{\kappa+i} \leftarrow \mathsf{lsb}_\eta(c_{\kappa+i} + s_i)$; $s_{i+1} \leftarrow (c_{\kappa+i} + s_i)/2^\eta$;
4. $d_{2\kappa-1} \leftarrow s_{\kappa-1}$;
5. return $D(\theta)$.

Note that for $D(\theta)$ that is returned we have $d_\kappa, \ldots, d_{2\kappa-1} < 2^\eta$.

Computation of polyMult We discuss strategies for polynomial multiplication using the representation for the larger value of κ. Similar considerations hold for the representation using the smaller value of κ.

There are several strategies for multiplying two polynomials. For $p2519$, $\kappa = 9$, while for $p25519$ and $p2663$, $\kappa = 10$. Let $C(\theta) = \mathsf{polyMult}(A(\theta), B(\theta))$ where $A(\theta)$ and $B(\theta)$ are proper polynomials. Computing the coefficients of $C(\theta)$ involve 32-bit multiplications and 64-bit additions. In the present context, the usual measure one would apply for assessing the efficacy of a polynomial multiplication algorithm is the number of 32-bit multiplications that would be required. Algorithms from [43] provide the smallest counts of 32-bit multiplication. This measure, however, does not necessarily provide the fastest implementation. Additions and dependencies do play a part and it turns out that an algorithm using a higher number of 32-bit multiplications turn out to be faster in practice. We discuss the cases of $\kappa = 9$ and $\kappa = 10$ separately. In the following, we abbreviate a 32-bit multiplication as [M].

Case $\kappa = 9$: Using 3-3 Karatsuba requires 36[M]. An algorithm given in [43] requires 34[M], but, this algorithm also requires multiplication by small constants which slows down the implementation. We have experimented with several variants and have found the following variant to provide the fastest speed (on the platform for implementation that we used). Consider the 9-limb multiplication to be 8-1 Karatsuba, i.e., the degree 8 polynomial is considered to be a degree 7 polynomial plus the term of degree 8. The two degree 7 (i.e., 8-limb) polynomials are multiplied by 3-level recursive Karatsuba requiring a total of 27[M]. The multiplication by the coefficients of the two degree 8 terms requires an additional 17[M]. So, the whole multiplication requires 44[M].

Case $\kappa = 10$: Using binary Karatsuba, this can be broken down into 3 5-limb multiplications. Two strategies for 5-limb multiplications in [43] require 13[M] and 14[M]. The strategy requiring 13[M] also requires multiplications by small constants and turns out to have a slower implementation than the strategy requiring 14[M]. So, our implementation of 10-limb multiplication requires 42[M].

Comparison to previous multiplication algorithm for $p25519$: In the original paper [3] which introduced Curve25519, it was mentioned that for $p25519$, a 10-limb representation using base 2^{26} cannot be used as this leads to an overfull. Instead an approach called "base $2^{25.5}$" was advocated. This approach has been followed in later implementations [9, 16]. of Curve25519. In this representation a 255-bit integer A is written as

$$A = a_0 + 2^{26}a_1 + 2^{51}a_2 + 2^{77}a_3 + 2^{102}a_4 + 2^{128}a_5 + 2^{153}a_6 + 2^{179}a_7 + 2^{204}a_8 + 2^{230}a_9$$

where $a_0, a_2, a_4, a_6, a_8 < 2^{26}$ and $a_1, a_3, a_5, a_7, a_9 < 2^{25}$. Note that this representation cannot be considered a polynomial in some quantity and so the multiplication of two such representations cannot benefit from the various polynomial multiplication algorithms. Instead, multiplication of two integers A and B in this representation requires all the 100 pairwise multiplications of a_i and b_j along with a few other multiplications by small constants. A total of 109[M] are required to compute the product.

For $p25519$, we have described a 10-limb representation using base as $\theta = 2^{26}$ and have described a multiplication algorithm using this representation. Given the importance of Curve25519, this itself is of some interest. The advantage of the new algorithm is that it can benefit from the various polynomial multiplication strategies. In particular, a field multiplication can be computed using 42[M]. On the other hand, the drawback is that the reduction requires a little more time, since the expand step has to be applied.

Following previous work [3, 9], the Sandy2x implementation used SIMD instructions to simultaneously compute two field multiplications. The `vpmuludq` instruction is used to simultaneously carry out two 32-bit multiplications. As a result, the 109 multiplications can be implemented using 54.5 `vpmuludq` instructions per field multiplication. The issue of computing four simultaneous field multiplications was not considered in [9, 16], since for the Montgomery ladder there is no opportunity for using such a strategy.

The new multiplication algorithm for $p25519$ can also be vectorised using `vpmulqdq` to compute two simultaneous field multiplications. We have, however, not implemented this. Since our target is Kummer line computation, we used AVX2 instructions to simultaneously compute four field multiplications. It would be of independent interest to explore the 2-way vectorisation of the new multiplication algorithm for use in the Montgomery ladder.

A previous implementation [7] of Curve25519 performs field arithmetic for $p25519$ using the representation with $\kappa = 5$ and $\eta = \nu = 51$. The Sandy2x code provides an assembly implementation of this multiplication algorithm and an implementation of the inversion algorithm for $p25519$. We have used this implementation to perform the inversion required after the Kummer line computation over $\mathsf{KL25519}(82, 77)$.

4.8 Field Squaring

This operation is required for both values of κ.

Let $A(\theta)$ be a proper polynomial. We define $\mathsf{sqr}(A(\theta))$ to be the proper polynomial $C(\theta)$ such that $C(\theta) \equiv A^2(\theta) \bmod p$. The computation of $\mathsf{sqr}(A(\theta))$ is almost the same as that of $\mathsf{mult}(A(\theta))$, except that $\mathsf{polyMult}(A(\theta), B(\theta))$ is replaced by $\mathsf{polySqr}(A(\theta))$ where $\mathsf{polySqr}(A(\theta))$ returns $A^2(\theta)$ as the square of the polynomial $A(\theta)$.

For $\kappa = 10$, $\mathsf{sqr}(A(\theta))$ requires 42[M] as before. On the other hand, for $\kappa = 9$, $\mathsf{sqr}(A(\theta))$ requires 36[M]. This is because the 17[M] required by the coefficients of the two degree 8 terms in the case of multiplication reduces to 9[M] in the case of squaring.

4.9 Hadamard Transform

This operation is required only for the representation using the larger value of κ.

Let $A_0(\theta)$ and $A_1(\theta)$ be two polynomials. By $\mathcal{H}(A_0(\theta), A_1(\theta))$ we denote the pair $(B_0(\theta), B_1(\theta))$ where

$$\begin{aligned} B_0(\theta) &= \mathsf{reduce}(A_0(\theta) + A_1(\theta)); \\ B_1(\theta) &= \mathsf{reduce}(A_0(\theta) - A_1(\theta)) = \mathsf{reduce}(A_0(\theta) + \mathsf{negate}(A_1(\theta))). \end{aligned}$$

In our context, there is an application of the Hadamard transform to the output of multiplication by constant. Since the output of multiplication by constant is not reduced, the coefficients of the input polynomials to the Hadamard transform do not necessarily respect the bounds required for proper polynomials. As explained earlier, the procedure negate works correctly with looser bounds on the coefficients of the input polynomial.

4.10 Field Inversion

This operation is required only for the representation using the smaller value of κ.

Suppose the inversion of $A(\theta)$ is required. Inversion is computed in constant time using a fixed addition chain to compute $A(\theta)^{p-2} \bmod p$. This computation boils down to computing a fixed number of squarings and multiplications.

In our context, field inversion is required only for conversion from projective to affine coordinates. The output of the scalar multiplication is in projective coordinates and if for some application the output is required in affine coordinates, then only a field inversion is required. For a proper polynomial

$A(\theta)$, by $\mathcal{I}(A(\theta))$ we denote the inverse of $A(\theta)$. The timing measurements that we report later includes the time required for inversion.

As mentioned earlier, the entire Kummer line computation is done using the larger value of κ. Before performing the inversion, the operands are converted to the representation using the smaller value of κ. This leads to a faster speed for the inversion. Since the inversion requires a substantial fraction of the total time, improving the speed of inversion is important for improving the overall speed.

5 Vector Operations

While considering vector operations, we consider the representation of field elements using the larger value of κ.

SIMD instructions in modern processors allow parallelism where the same instruction can be applied to multiple data. To take advantage of SIMD instructions it is convenient to organise the data as vectors. The Intel instructions that we target apply to 256-bit registers which are considered to be 4 64-bit words (or, as 8 32-bit words). So, we consider vectors of length 4.

Let $\mathbf{A}(\theta) = (A_0(\theta), A_1(\theta), A_2(\theta), A_3(\theta))$ where $A_k(\theta) = \sum_{i=0}^{\kappa-1} a_{k,i}\theta^i$ are proper polynomials. We will say that such an $\mathbf{A}(\theta)$ is a proper vector. So, $\mathbf{A}(\theta)$ is a vector of 4 elements of $\mathbb{F}_p$. Recall that each $a_{k,i}$ is stored in a 64-bit word. Conceptually one may think of $\mathbf{A}(\theta)$ to be given by a $\kappa \times 4$ matrix of 64-bit words.

We describe a different way to consider $\mathbf{A}(\theta)$. Let $\mathbf{a}_i = (a_{0,i}, a_{1,i}, a_{2,i}, a_{3,i})$ and define $\mathbf{a}_i\theta^i = (a_{0,i}\theta^i, a_{1,i}\theta^i, a_{2,i}\theta^i, a_{3,i}\theta^i)$. Then we can write $\mathbf{A}(\theta)$ as $\mathbf{A}(\theta) = \sum_{i=0}^{\kappa-1} \mathbf{a}_i\theta^i$. Each $\mathbf{a}_i$ is stored as a 256-bit value. We define the following operations.

- $\mathsf{pack}(a_0, a_1, a_2, a_3)$: returns a 256-bit quantity $\mathbf{a}$. Here each a_i is a 64-bit quantity and $\mathbf{a}$ is obtained by concatenating $a_0, \ldots, a_3$.
- $\mathsf{unpack}(\mathbf{a})$: returns (a_0, a_1, a_2, a_3). Here $\mathbf{a}$ is a 256-bit quantity and the a_i's are 64-bit quantities such that $\mathbf{a}$ is the concatenation of $a_0, \ldots, a_3$.
- $\mathsf{pack}(A_0(\theta), A_1(\theta), A_2(\theta), A_3(\theta))$: returns $\mathbf{A}(\theta)$ represented as $\mathbf{A}(\theta) = \sum_{i=0}^{\kappa-1} \mathbf{a}_i\theta^i$, where $\mathbf{a}_i = \mathsf{pack}(a_{i,0}, a_{i,1}, a_{i,2}, a_{i,3})$.
- $\mathsf{unpack}(\mathbf{A}(\theta))$: returns $(A_0(\theta), A_1(\theta), A_2(\theta), A_3(\theta))$, where $\mathbf{A}(\theta) = \sum_{i=0}^{\kappa-1} \mathbf{a}_i\theta^i$, for $j = 0, 1, 2, 3$, $A_j(\theta) = \sum_{i=0}^{\kappa-1} a_{j,i}\theta^i$ and $(a_{0,i}, a_{1,i}, a_{2,i}, a_{3,i}) = \mathsf{unpack}(\mathbf{a}_i)$.

In the above, we use pack to denote both the packing of 4 64-bit words into a 256-bit quantity and also the limb-wise packing of four field elements into a vector. Similar overloading of notation is used for unpack.

We define the following vector operations. The operand $\mathbf{A}(\theta)$ represents $(A_0(\theta), A_1(\theta), A_2(\theta), A_3(\theta))$ and similarly, $\mathbf{B}(\theta)$ represents $(B_0(\theta), B_1(\theta), B_2(\theta), B_3(\theta),)$.

- $\mathsf{reduce}(\mathbf{A}(\theta)$: returns $(\mathsf{reduce}(A_0(\theta)), \mathsf{reduce}(A_1(\theta)), \mathsf{reduce}(A_2(\theta)), \mathsf{reduce}(A_3(\theta)))$.
- $\mathcal{M}^4(\mathbf{A}(\theta), \mathbf{B}(\theta))$: returns $\mathbf{C}(\theta) = \sum_{i=0}^{\kappa-1} \mathbf{c}_i\theta^i$ representing $(C_0(\theta), C_1(\theta), C_2(\theta), C_3(\theta))$ where $C_k(\theta) = \mathsf{mult}(A_k(\theta), B_k(\theta))$ for $k = 0, \ldots, 3$.
- $\mathcal{S}^4(\mathbf{A}(\theta))$: returns $\mathbf{C}(\theta) = \sum_{i=0}^{\kappa-1} \mathbf{c}_i\theta^i$ representing $(C_0(\theta), C_1(\theta), C_2(\theta), C_3(\theta))$ where $C_k(\theta) = \mathsf{sqr}(A_k(\theta))$ for $k = 0, \ldots, 3$.
- $\mathcal{C}^4(\mathbf{A}(\theta), \mathbf{d})$: returns $\mathbf{C}(\theta) = \sum_{i=0}^{\kappa-1} \mathbf{c}_i\theta^i$ representing $(C_0(\theta), C_1(\theta), C_2(\theta), C_3(\theta))$ where $\mathbf{d} = (d_0, d_1, d_2, d_3)$; $C_k(\theta) = \mathsf{constMult}(A_k(\theta), d_k)$ for $k = 0, \ldots, 3$. Recall that the output of constMult is not reduced and so neither is the output of $\mathcal{C}^4$.

The key Intel intrinsics operations that are required to implement the above vector operations are the following.

- `_mm256_add_epi64`: On inputs $\mathbf{a} = (a_0, a_1, a_2, a_3)$ and $\mathbf{b} = (b_0, b_1, b_2, b_3)$ returns $(a_0 + b_0, a_1 + b_1, a_2 + b_2, a_3 + b_3)$ with each component reduced modulo 2^{64}.
- `_mm256_sub_epi64`: On inputs $\mathbf{a} = (a_0, a_1, a_2, a_3)$ and $\mathbf{b} = (b_0, b_1, b_2, b_3)$ returns $(a_0 - b_0, a_1 - b_1, a_2 - b_2, a_3 - b_3)$ with each component reduced modulo 2^{64}. We have used this operation only in context of Karatsuba multiplication, i.e., for a subtraction of the type $(a + b)(c + d) - (ac + bd) = ad + bc$ for non-negative integers a, b, c and d. The result is guaranteed to be non-negative and so there is no need to handle the sign.
- `_mm256_mul_epu32`: On inputs $\mathbf{a} = (a_0, a_1, a_2, a_3)$ and $\mathbf{b} = (b_0, b_1, b_2, b_3)$ returns $(a_0b_0, a_1b_1, a_2b_2, a_3b_3)$ with each component reduced modulo 2^{64}.

Using the above SIMD operations to replace the 32-bit operations in the algorithm for multiplying a pair of field elements directly provides an algorithm for multiplying four pairs of field elements. Similar vectorisation is achieved for squaring.

5.1 Vector Hadamard Operation

The Hadamard operation $\mathcal{H}(A(\theta), B(\theta))$ is required to output $(C(\theta), D(\theta))$ where $C(\theta) \equiv A(\theta) + B(\theta) \bmod p$ and $D(\theta) \equiv A(\theta) - B(\theta) \bmod p$. We define the vector extension of the Hadamard operation, which computes two simultaneous Hadamard operations using SIMD vector instructions. For a 256-bit quantity $\mathbf{a} = (a_0, a_1, a_2, a_3)$ we define $\mathsf{dup}_1(\mathbf{a}) = (a_0, a_0, a_2, a_2)$ and $\mathsf{dup}_2(\mathbf{a}) = (a_1, a_1, a_3, a_3)$.

The Hadamard operation involves a subtraction. As explained in Section 4.5 this is handled by first computing a negation followed by an addition. Negation of a polynomial is computed as subtracting the given polynomial from $2^{\mathfrak{n}}\mathfrak{P}(\theta)$ where $\mathfrak{n}$ is chosen to ensure that all the coefficients of the result are positive.

$\mathcal{H}^2(\mathbf{A}(\theta))$
input: $\mathbf{A}(\theta) = \sum_{i=0}^{\kappa-1} \mathbf{a}_i\theta^i$ representing $(A_0(\theta), A_1(\theta), A_2(\theta), A_3(\theta))$;
output: $\mathbf{C}(\theta) = \sum_{i=0}^{\kappa-1} \mathbf{c}_i\theta^i$ representing $(A_0(\theta) + A_1(\theta), A_0(\theta) - A_1(\theta), A_2(\theta) + A_3(\theta), A_2(\theta) - A_3(\theta))$
with each component reduced modulo p;
1. for $i = 0, \ldots, \kappa - 1$ do
2. $\quad\mathbf{s} = \mathsf{dup}_1(\mathbf{a}_i)$;
3. $\quad\mathbf{t} = \mathsf{dup}_2(\mathbf{a}_i)$;
4. $\quad\mathbf{t} = \mathbf{t} \oplus (0^{64}, 1^{64}, 0^{64}, 1^{64})$;
5. $\quad\mathbf{t} = \mathbf{t} + (0^{64}, 2^{\mathfrak{n}}\mathfrak{p}_i + 1, 0^{64}, 2^{\mathfrak{n}}\mathfrak{p}_i + 1)$;
6. $\quad\mathbf{c}_i = \mathbf{t} + \mathbf{s}$;
7. end for;

return $\mathsf{reduce}(\mathbf{C}(\theta))$.

The $\oplus$ operation is implemented using `_mm256_xor_si256`; the additions in Steps 5 and 6 can be implemented using `_mm256_add_epi32`; the operations dup_1 and dup_2 are implemented using `_mm256_permute4x64_epi64`.

1. The operation $\mathcal{C}^4$ (which is the vector version of constMult) multiplies the input proper polynomials with constant and the result is not reduced (since the output of constMult is not reduced). The constant is one of the parameters A^2 and B^2 of the Kummer line. The output of $\mathcal{C}^4$ forms the input to $\mathcal{H}^2$. Choosing $\mathfrak{n} = \lceil \log_2 \max(A^2, B^2) \rceil$ ensures the non-negativity condition for the subtraction operation.
2. We define a unreduced version of $\mathcal{H}^2$ to be unreduced-$\mathcal{H}^2$. This procedure is the same as $\mathcal{H}^2$ except that at the end instead of returning $\mathsf{reduce}(\mathbf{C}(\theta))$, $\mathbf{C}(\theta)$ is returned. Following the discussion in Section 4.5, to apply the procedure unreduced-$\mathcal{H}^2$ to a proper polynomial it is sufficient to choose $\mathfrak{n} = 1$. The first $\kappa - 2$ coefficients of the output can be represented using $\eta + 1$ bits and the last coefficient can be represented using $\nu + 1$ bits.

5.2 Vector Duplication

Let $\mathbf{a} = (a_0, a_1, a_2, a_3)$ and b be a bit. We define an operation $\mathsf{copy}(\mathbf{a}, b)$ as follows: if $b = 0$, return (a_0, a_1, a_0, a_1); and if $b = 1$, return (a_2, a_3, a_2, a_3). The operation copy is implemented using `_mm256_permutevar8x32_epi32`.

Let $\mathbf{A}(\theta) = \sum_{i=0}^{\kappa-1} \mathbf{a}_i \theta^i$ be a proper vector and b be a bit. We define the operation $\mathcal{P}^4(\mathbf{A}, b)$ to return $\sum_{i=0}^{\kappa-1} \mathsf{copy}(\mathbf{a}_i, b)\theta^i$. If $\mathbf{A}(\theta)$ represents $(A_0(\theta), A_1(\theta), A_2(\theta), A_3(\theta))$, then

$$\mathcal{P}^4(\mathbf{A}, b) = \begin{cases} (A_0(\theta), A_1(\theta), A_0(\theta), A_1(\theta)) \text{ if } b = 0; \\ (A_2(\theta), A_3(\theta), A_2(\theta), A_3(\theta)) \text{ if } b = 1. \end{cases}$$

6 Vectorised Scalar Multiplication

Scalar multiplication on the Kummer line is computed from a base point $[x : z]$ represented as (x^2, z^2) in the square only setting and an ℓ-bit non-negative integer n. The quantities x^2 and z^2 are elements of $\mathbb{F}_p$. If x^2 and z^2 are small as in the fixed base point of $\mathcal{K}_{101,61}$, then these two values are represented as 32-bit words (or even as bytes). In general, the values x^2 and z^2 will be arbitrary elements of $\mathbb{F}_p$ and will have a 9-limb representation as has been described above.

The scalar multiplication on $\mathcal{K}_{a^2,b^2}$ over the field $\mathbb{F}_p$ with p to be one of $p2519$, $p25519$ or $p2663$ is given below. Recall that $A^2 = a^2 + b^2$ and $B^2 = a^2 - b^2$.

$\mathsf{scalarMult}(P, n)$
Input: base point $P = [X(\theta) : Z(\theta)]$; and ℓ-bit scalar n given as $(1, n_{\ell-2}, \ldots, n_0)$;
Output: $U(\theta)/V(\theta)$ where $nP = [U(\theta) : V(\theta)]$;

1. $\mathfrak{a} = \mathsf{pack}(B^2, A^2, B^2, A^2)$;
2. $\mathfrak{c}_0 = \mathsf{pack}(b^2, a^2, z^2, x^2)$; $\mathfrak{c}_1 = \mathsf{pack}(b^2, a^2, z^2, x^2)$;
3. compute $2P = (X_2(\theta), Z_2(\theta))$;
4. $\mathbf{T}(\theta) = \mathsf{pack}(X(\theta), Z(\theta), X_2(\theta), Z_2(\theta))$;
5. for $i = \ell - 2$ down to 0
6. $\quad\mathbf{T}(\theta) = \mathcal{H}^2(\mathbf{T}(\theta))$;
7. $\quad\mathbf{S}(\theta) = \mathcal{P}^4(\mathbf{T}(\theta), n_i)$;
8. $\quad\mathbf{T}(\theta) = \mathcal{M}^4(\mathbf{T}(\theta), \mathbf{S}(\theta))$;
9. $\quad\mathbf{T}(\theta) = \mathcal{C}^4(\mathbf{T}(\theta), \mathfrak{a})$;
10. $\quad\mathbf{T}(\theta) = \mathcal{H}^2(\mathbf{T}(\theta))$;
11. $\quad\mathbf{T}(\theta) = \mathcal{S}^4(\mathbf{T}(\theta))$;
12. $\quad\mathbf{T}(\theta) = \mathcal{C}^4(\mathbf{T}(\theta), \mathfrak{c}_{n_i})$;
13. end for;
14. $(U(\theta), V(\theta), \cdot, \cdot) = \mathsf{unpack}(\mathsf{reduce}(\mathbf{T}(\theta)))$;
15. return $U(\theta)/V(\theta)$.

Note that we are working in the square only setting. So, $X(\theta), Z(\theta), U(\theta)$ and $V(\theta)$ should be considered as squared quantities, but, whose square roots are not necessarily in $\mathbb{F}_p$. This though, is not important for the computation of the scalar multiplication.

An inversion is required at Step 15. The representations of $U(\theta)$ and $V(\theta)$ are first converted to the one using the smaller value of κ. Let these be denoted as u and v: $w = v^{-1}$ and then $x = w \cdot u$ are computed. As mentioned in Section 4.10, the inversion is computed in constant time. The multiplications and squarings in this computation are performed using the representation with smaller κ so that both w and x are also represented using the smaller value of κ. A final reduce call is made on x followed by a makeUnique call whose output is returned.

Modification for variable base scalar multiplication: In the above description, we have assumed that the base point $[x^2 : z^2]$ is represented by small integers. This is true if the scalar multiplication is for a fixed base point. On the other hand, for variable base point, this is no longer true. The following modifications are made.

1. In Step 12, the operation $\mathcal{M}^4$ is used instead of the operation $\mathcal{C}^4$.
2. For $p25519$ a further modification is made. In Step 6, $\mathcal{H}^2$ is replaced by unreduced-$\mathcal{H}^2$. This eliminates the reduction step of the Hadamard operation. Consequently, the first 9 limbs of the output consists of 27-bit values, while the last limb consists of 22-bit values. Applying the multiplication algorithm of $p25519$ to such values do not lead to an overfull. This is due to the expand operation which is part of the mult operation for $p25519$. Since $p2519$ and $p2663$ do not use the expand operation, the replacement of $\mathcal{H}^2$ by unreduced-$\mathcal{H}^2$ is not possible for these two primes.

Correctness: Steps 6 to 12 constitute a single vectorised ladder step. At the i-th iteration, suppose $(kP, (k+1)P)$, for some k, is a pair of points which forms the input to the ladder step. If $n_i = 0$, then the output of the ladder step is the pair of points $(2kP, (2k+1)P)$; and if $n_i = 1$, then the output of the ladder step is the pair of points $((2k+1)P, (2k+2)P)$. Suppose $kP = [X_1(\theta) : Z_1(\theta)]$ and $(k+1)P = [X_2(\theta) : Z_2(\theta)]$. These two points are represented in packed form as $(X_1(\theta), Z_1(\theta), X_2(\theta), Z_2(\theta))$ which is the vector input to the ladder step. Denoting the output of the ladder step as $(X_3(\theta), Z_3(\theta), X_4(\theta), Z_4(\theta))$, the operation of the ladder step is shown in Figure 4. The correctness of the ladder step is easy to argue from which the correctness of the vectorised scalar multiplication follows.

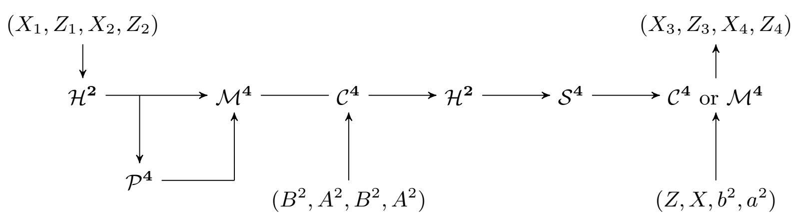

Fig. 4. One vectorised ladder step

7 Implementation and Timings

We have implemented the vectorised scalar multiplication algorithm in 64-bit AVX2 intrinsics instructions. The code implements the vectorised ladder algorithm which takes the same amount of time for all scalars. Consequently, our code also runs in constant time. The code is publicly available at [26].

Timing experiments were carried out on a single core of the following two platforms.

Haswell: Intel®Core™i7-4790 4-core CPU @ 3.60GHz. The OS is 64-bit Ubuntu-14.04 LTS operating system and the C code was complied using GCC version 4.8.4.

Skylake: Intel®Core™i7-6700 4-core CPU @ 3.40GHz. The OS is 64-bit Ubuntu 16.04 LTS and the C code was compiled using GCC version 5.4.0.

Timing measurements were performed using the methodology described at [33]. During measurement, turbo boost and hyperthreading were turned off. We used average from 100,000 iterations. Cache training was done using 25000 iterations. The Time Stamp Counter (TSC) was read from the CPU to RAX and

RDX registers by RDTSC instruction. For the actual measurements, the header file "measurement.h", given in [33] was used.

Table 6 compares the number of cycles required by our implementation with that of a few other concrete curve proposals[4]. All the timings are for constant time code on the Haswell processor using variable base scalar multiplication. For Four-$\mathbb{Q}$, $\mathcal{K}_{11,-22,-19,-3}$ and the results from [47] and [34], the timings are obtained from the respective papers. For Curve25519, we downloaded the `Sandy2x`[5] library and measured the performance using the methodology from [33]. The cycle count of 158169 on Haswell that we obtain for Curve25519 is close to 156076 cycles reported by Tung Chou at `https://moderncrypto.org/mail-archive/curves/2015/000637.html` and the count of about 156500 cycles reported in [23]. Further, EBACS (`https://bench.cr.yp.to/results-dh.html`) also mentions about 156000 cycles on the machine `titan0`.

curve	genus	security	field	endomorphism	cycles	pre-comp tab
Curve25519 [16]	1	126	$p25519$	no	**158169**	no
NIST P-256 [34]	1	128	$\mathbb{F}_{2^{256}-2^{224}+2^{192}+2^{96}-1}$	no	291000	no
Four-$\mathbb{Q}$ [18][6]	1	123	$\mathbb{F}_{(2^{127}-1)^2}$	yes	59000	2048 bits
				no	109000	no
$\mathcal{K}_{11,-22,-19,-3}$ [5][7]	2	125	$\mathbb{F}_{2^{127}-1}$	no	60468	no
Koblitz [47] (3-ρNAF)	1	128	$\mathbb{F}_{4^{149}}$	yes	69,656	4768 bits
KL2519(81, 20)	1	124	$p2519$	no	**118550**	no
KL25519(82, 77)	1	125.7	$p25519$	no	**144542**	no
KL2663(260, 139)	1	131.2	$p2663$	no		no

Table 6. Timing comparison for variable base scalar multiplication on Haswell. The references point to the best known implementations. Curve25519 was proposed in [3]; NIST P-256 was proposed in [45]; the curve used in [47] was proposed in [40]; and $\mathcal{K}_{11,-22,-19,-3}$ was proposed in [32]; the other curves were proposed in the referenced papers themselves.

Timing results on Haswell and Skylake platforms for Curve25519 and the Kummer lines for both fixed base and variable base scalar multiplications are shown in Table 7. We note that for fixed base scalar multiplication it is possible to use a pre-computed table to improve the speed. Using this approach, [34] reports much faster timing for NIST P-256 and [16] reports much faster timing for Curve25519. We have not investigated the use of pre-computed tables for Kummer lines.

Based on entries in Table 7, we conclude the following[8].

1. All three of KL2519(81,20), KL25519(82,77) and KL2663(260,139) are faster than Curve25519 on both the Haswell and the Skylake platforms for both fixed base and variable base scalar multiplications. This points to the fact that the Kummer ladder is more SIMD friendly than the Montgomery ladder. In particular, we note that even though Curve25519 and KL25519(82,77) use the same underlying prime $p25519$, KL25519(82,77) has speed improvements over Curve25519.

[4] The reported timings for the Kummer lines are tentative results. Updated timing results accompanied by an updated version of the code will soon be made available.

[5] Downloaded from `https://bench.cr.yp.to/supercop/supercop-20160910.tar.xz`. We used `crypto_scalarmult(q,n,p)` to measure variable base scalar multiplication and `crypto_scalarmult_base(q,n)` to measure fixed base scalar multiplication.

[6] Improved timing results of 54000 and 104000 respectively for implementation with and without automorphism for Four-$\mathbb{Q}$ have been reported in the extended version `http://eprint.iacr.org/2015/565.pdf`.

[7] The original speed reported in [5] was 54389. The figure 60468 is reported to be the median cycles per byte at `https://bench.cr.yp.to/results-dh.html` for the machine `titan0`. See `http://eprint.iacr.org/2015/565.pdf` for a possible explanation of the discrepancy.

[8] The comments regarding KL2663(260,139) are based on tentative timing results which we do not report here. The code for KL2663(260,139) is being updated and once this is ready, we will report the timing results.

2. KL2519(81,20) is the fastest, followed by KL2663(260,139), KL25519(82,77) and Curve25519.
3. In terms of security, KL2663(260,139) offers the highest security followed by Curve25519, KL25519(82,77) and KL2519(81,20). The security gap between KL2663(260,139) and Curve25519 is 6 bits; between KL25519(82,77) and Curve25519 is 0.3 bits; and between KL2519(81,20) and Curve25519 is 2 bits.

It is noticeable that compared to Curve25519, KL2663(260,139) provides both higher security and faster scalar multiplication. This is due to the fact that a field multiplication in $p2663$ is faster than that in $p25519$. If one is interested in obtaining the maximum security, then KL2663(260,139) should be used. On the other hand, if one consider 124 bits of security to be adequate, then KL2519(81,20) should be used. The only reason for consider the prime $p25519$ in comparison to either $p2519$ or $p2663$ is that 255 is closer to a multiple of 32 than either of 251 or 266. So, if public keys are transmitted as 32-bit words, then the wastage of bits would be minimum for $p25519$ compared to $p2519$ or $p2663$. Whether this is an overriding reason for discarding the higher security and higher speed offered by $p2663$ or the much higher speed and small loss in security offered by $p2519$ would probably depend on the application at hand. If for some reason, $p25519$ is preferred to be used, then KL25519(82,77) offers higher speed than Curve25519 at a loss of only 0.3 bits of security.

curve	security	Haswell		Skylake	
		fixed base	var base	fixed base	var base
Curve25519 [16]	126	144350	158169	126522	136209
KL2519(81,20)	124	94265	118550	78599	98445
KL25519(82,77)	125.7	114726	144542	92153	119568
KL2663(260,139)	131.2				

Table 7. Timing comparison of Kummer lines with Curve25519 on Haswell and Skylake platforms.

8 Conclusion

This work has shown that compared to Curve25519, Kummer line based scalar multiplication for genus one over prime order fields offers competitive performance using SIMD operations. Previous works on implementation of Kummer arithmetic had focused completely on genus two. By showing competitive implementation also in genus one, our work fills a gap in the existing literature. Our implementation of the Kummer line computations use Intel intrinsics. It is likely that assembly implementation will further improve the speed. On the other hand, we would like to note that we have comprehensively considered the different possibilities for algorithmic improvements to the basic idea.

Acknowledgement

We would like to thank Pierrick Gaudry for helpful comments and clarifying certain confusion regarding conversion from Kummer line to elliptic curve. We would also like to thank Peter Schwabe for clarifying certain implementation issues regarding Curve25519 and Kummer surface computation on genus 2. Thanks to Alfred Menezes, René Struik and Patrick Longa for comments.

References

1. Diego F. Aranha, Paulo S. L. M. Barreto, Geovandro C. C. F. Pereira, and Jefferson E. Ricardini. A note on high-security general-purpose elliptic curves. Cryptology ePrint Archive, Report 2013/647, 2013. `http://eprint.iacr.org/2013/647`.

2. J. Barwise and P. Eklof. Lefschetz's principle. *Journal of Algebra*, 13(4):554–570, 1969.
3. D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In *Public Key Cryptography - PKC*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.
4. D. J. Bernstein. Elliptic vs. hyperelliptic, part I. *Talk at ECC*, 2006.
5. D. J. Bernstein, C. Chuengsatiansup, T. Lange, and P. Schwabe. Kummer strikes back: New DH speed records. In *Advances in Cryptology - ASIACRYPT*, volume 8873 of *Lecture Notes in Computer Science*, pages 317–337. Springer, 2014.
6. D. J. Bernstein and T. Lange. Safecurves: choosing safe curves for elliptic-curve cryptography. `http://safecurves.cr.yp.to/index.html`, accessed 15th September, 2016.
7. Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 124–142. Springer, 2011.
8. Daniel J. Bernstein and Tanja Lange. Faster addition and doubling on elliptic curves. In Kaoru Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 29–50. Springer, 2007.
9. Daniel J. Bernstein and Peter Schwabe. NEON crypto. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 320–339. Springer, 2012.
10. Guido Bertoni and Jean-Sébastien Coron, editors. *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*. Springer, 2013.
11. Joppe W. Bos, Craig Costello, Hüseyin Hisil, and Kristin E. Lauter. Fast cryptography in genus 2. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 194–210. Springer, 2013.
12. Joppe W. Bos, Craig Costello, Hüseyin Hisil, and Kristin E. Lauter. High-performance scalar multiplication using 8-dimensional GLV/GLS decomposition. In Bertoni and Coron [10], pages 331–348.
13. Brainpool. ECC standard. `http://www.ecc-brainpool.org/ecc-standard.htm`.
14. Eric Brier and Marc Joye. Weierstraß elliptic curves and side-channel attacks. In David Naccache and Pascal Paillier, editors, *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*, volume 2274 of *Lecture Notes in Computer Science*, pages 335–345. Springer, 2002.
15. Daniel R. L. Brown. SEC 2: Recommended elliptic curve domain parameters. `http://www.secg.org/sec2-v2.pdf`, 2010.
16. Tung Chou. Sandy2x: New Curve25519 speed records. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, volume 9566 of *Lecture Notes in Computer Science*, pages 145–160. Springer, 2015.
17. Ping Ngai Chung, Craig Costello, and Benjamin Smith. Fast, uniform scalar multiplication for genus 2 Jacobians with fast Kummers. In *Selected Areas in Cryptography*, 2016. to appear.
18. C. Costello and P. Longa. Four($\mathbb{Q}$): Four-dimensional decompositions on a $\mathbb{Q}$-curve over the Mersenne prime. In *Advances in Cryptology - ASIACRYPT Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 214–235. Springer, 2015.
19. Craig Costello, Hüseyin Hisil, and Benjamin Smith. Faster compact Diffie-Hellman: Endomorphisms on the x-line. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2014.
20. Curve25519. Wikipedia page on Curve25519. `https://en.wikipedia.org/wiki/Curve25519`, accessed 15th September, 2016.
21. Harold M. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44:393–422, 2007.
22. Armando Faz-Hernández, Patrick Longa, and Ana H. Sánchez. Efficient and secure algorithms for GLV-based scalar multiplication and their implementation on GLV-GLS curves. In Josh Benaloh, editor, *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, volume 8366 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 2014.
23. Armando Faz-Hernández and Julio López. Fast implementation of Curve25519 using AVX2. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, volume 9230 of *Lecture Notes in Computer Science*, pages 329–345. Springer, 2015.

24. E.V. Flynn. Formulas for Kummer on genus 2. http://people.maths.ox.ac.uk/flynn/genus2/kummer/, accessed on 15th September, 2016.
25. E.V. Flynn. The group law on the Jacobian of a curve of genus 2. *J. reine angew. Math.*, 439:45–69, 1993.
26. Code for Kummer Line Computations. https://github.com/skarati/KummerLineV02.
27. G. Frey and H.-G. Rück. The strong lefschetz principle in algebraic geometry. *Manuscripta Mathematica*, 55(3):385–401, 1986.
28. Steven D. Galbraith, Xibin Lin, and Michael Scott. Endomorphisms for faster elliptic curve cryptography on a large class of curves. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 518–535. Springer, 2009.
29. Robert P. Gallant, Robert J. Lambert, and Scott A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer, 2001.
30. P. Gaudry. Fast genus 2 arithmetic based on theta functions. *J. Mathematical Cryptology*, 1(3):243–265, 2007.
31. P. Gaudry and D. Lubicz. The arithmetic of characteristic 2 Kummer surfaces and of elliptic Kummer lines. *Finite Fields and Their Applications*, 15(2):246–260, 2009.
32. P. Gaudry and É. Schost. Genus 2 point counting over prime fields. *J. Symb. Comput.*, 47(4):368–400, 2012.
33. S. Gueron. Software optimizations for cryptographic primitives on general purpose x86_64 platforms. *Tutorial at IndoCrypt*, 2011.
34. Shay Gueron and Vlad Krasnov. Fast prime field elliptic-curve cryptography with 256-bit primes. *J. Cryptographic Engineering*, 5(2):141–151, 2015.
35. Darrel Hankerson, Koray Karabina, and Alfred Menezes. Analyzing the galbraith-lin-scott point multiplication method for elliptic curves over binary fields. *IEEE Trans. Computers*, 58(10):1411–1420, 2009.
36. Hüseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Twisted Edwards curves revisited. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings*, volume 5350 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2008.
37. Jun ichi Igusa. *Theta functions*. Springer, 1972.
38. Neal Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, 48(177):203–209, 1987.
39. Neal Koblitz. Hyperelliptic cryptosystems. *J. Cryptology*, 1(3):139–150, 1989.
40. Patrick Longa and Francesco Sica. Four-dimensional Gallant-Lambert-Vanstone scalar multiplication. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 718–739. Springer, 2012.
41. Victor S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology - CRYPTO'85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, pages 417–426. Springer Berlin Heidelberg, 1985.
42. Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.
43. Peter L. Montgomery. Five, six, and seven-term karatsuba-like formulae. *IEEE Trans. Computers*, 54(3):362–369, 2005.
44. D. Mumford. *Tata lectures on theta I.* Progress in Mathematics 28. Birkh äuser, 1983.
45. U.S. Department of Commerce/National Institute of Standards and Technology. Digital Signature Standard (DSS). FIPS-186-3. http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf, 2009.
46. Thomaz Oliveira, Julio López, Diego F. Aranha, and Francisco Rodríguez-Henríquez. Lambda coordinates for binary elliptic curves. In Bertoni and Coron [10], pages 311–330.
47. Thomaz Oliveira, Julio López, and Francisco Rodríguez-Henríquez. Software implementation of Koblitz curves over quadratic fields. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 259–279. Springer, 2016.
48. Joost Renes, Craig Costello, and Lejla Batina. Complete addition formulas for prime order elliptic curves. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 403–428. Springer, 2016.
49. NUMS: Nothing up my sleeve. https://tools.ietf.org/html/draft-black-tls-numscurves-00.

A Proofs of Theta Identities

For the sake of completeness, we provide the proofs of (2), (3) and (4). These are derived from the definition of theta functions with characteristics given by (1).

A.1 Proof of (2)

We can write $\vartheta[\xi_1, \xi_2](-w, \tau)$ as

$$\vartheta[\xi_1, \xi_2](-w, \tau) = \sum_{n \in \mathbb{Z}} \exp\left[\pi i(n + \xi_1)^2 \tau + 2\pi i(n + \xi_1)(-w + \xi_2)\right] \tag{21}$$

Let $m = -n - 2\xi_1$. As n runs through all the integers and $\xi_1 \in \{0, \frac{1}{2}\}$, m will also run over all the integers. Using $m + \xi_1 = -n - \xi_1$ in (21) we have

$$\begin{aligned}
\vartheta[\xi_1, \xi_2](-w, \tau) &= \sum_{m \in \mathbb{Z}} \exp\left[\pi i(m + \xi_1)^2 \tau + 2\pi i(-m - \xi_1)(-w + \xi_2)\right] \\
&= \sum_{m \in \mathbb{Z}} \exp\left[\pi i(m + \xi_1)^2 \tau + 2\pi i(m + \xi_1)(w - \xi_2)\right] \\
&= \sum_{m \in \mathbb{Z}} \exp\left[\pi i(m + \xi_1)^2 \tau + 2\pi i(m + \xi_1)(w + \xi_2) - 4\pi i(m + \xi_1)\xi_2\right] \\
&= \sum_{m \in \mathbb{Z}} \exp\left[\pi i(m + \xi_1)^2 \tau + 2\pi i(m + \xi_1)(w + \xi_2)\right] \exp(-4\pi i m \xi_2) \exp(-4\pi i \xi_1 \xi_2)
\end{aligned}$$

Now $4\xi_2$ is either zero or an even integer and m is an integer and so $\exp(-4\pi i m\xi_2) = 1$. Similarly, $4\xi_1\xi_2$ is also zero or one. Therefore, $\exp(-4\pi i\xi_1\xi_2) = \exp(-\pi i)^{4\xi_1\xi_2} = (-1)^{4\xi_1\xi_2}$. This shows

$$\vartheta[\xi_1, \xi_2](-w, \tau) = (-1)^{4\xi_1\xi_2} \sum_{m \in \mathbb{Z}} \exp\left[\pi i(m + \xi_1)^2 \tau + 2\pi i(m + \xi_1)(w + \xi_2)\right]$$

which proves (2).

A.2 Proof of (3)

We can write the $\vartheta_1(w)$, $\vartheta_2(w)$, $\Theta_1(w)$ and $\Theta_2(w)$ as given below:

$$\vartheta_1(w) = \vartheta[0, 0](w, \tau) = \sum_{n \in \mathbb{Z}} \exp\left[\pi i n^2 \tau + 2\pi i n w\right] \tag{22}$$

$$\vartheta_2(w) = \vartheta\left[0, 1/2\right](w, \tau) = \sum_{n \in \mathbb{Z}} \exp\left[\pi i n^2 \tau + \pi i n + 2\pi i n w\right] \tag{23}$$

$$\Theta_1(w) = \vartheta[0, 0](w, 2\tau) = \sum_{n \in \mathbb{Z}} \exp\left[2\pi i n^2 \tau + 2\pi i n w\right] \tag{24}$$

$$\Theta_2(w) = \vartheta\left[1/2, 0\right](w, 2\tau) = \sum_{n \in \mathbb{Z}} \exp\left[2\pi i(n + 1/2)^2 \tau + 2\pi i(n + 1/2)w\right]. \tag{25}$$

Let w_1 and w_2 be in $\mathbb{C}$. From (24), we can write

$$\begin{aligned}
&\Theta_1(w_1+w_2)\Theta_1(w_1-w_2)\\
&=\left(\sum_{n_1\in\mathbb{Z}}\exp\left[2\pi i n_1^2\tau+2\pi i n_1(w_1+w_2)\right]\right)\left(\sum_{n_2\in\mathbb{Z}}\exp\left[2\pi i n_2^2\tau+2\pi i n_2(w_1-w_2)\right]\right)\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[2\pi i n_1^2\tau+2\pi i n_1(w_1+w_2)+2\pi i n_2^2\tau+2\pi i n_2(w_1-w_2)\right]\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[2\pi i n_1^2\tau+2\pi i n_2^2\tau+2\pi i(n_1+n_2)w_1+2\pi i(n_1-n_2)w_2\right]\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\begin{bmatrix}\pi i n_1^2\tau+\pi i n_2^2\tau+2\pi i n_1 n_2\tau++2\pi i(n_1+n_2)w_1+\\ \pi i n_1^2\tau+\pi i n_2^2\tau-2\pi i n_1 n_2\tau+2\pi i(n_1-n_2)w_2\end{bmatrix}\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i(n_1+n_2)^2\tau+\pi i(n_1-n_2)^2\tau+2\pi i(n_1+n_2)w_1+2\pi i(n_1-n_2)w_2)\right].
\end{aligned}$$

Let $m_1=n_1+n_2$ and $m_2=n_1-n_2$ so that $m_1+m_2=2n_1$ is even. We obtain

$$\Theta_1(w_1+w_2)\Theta_1(w_1-w_2)=\sum_{\substack{m_1\in\mathbb{Z}\\ m_1+m_2=even}}\sum_{m_2\in\mathbb{Z}}\exp\left[\pi i m_1^2\tau+\pi i m_2^2\tau+2\pi i m_1 w_1+2\pi i m_2 w_2)\right]. \tag{26}$$

From (22), we can write

$$\begin{aligned}
&\vartheta_1(w_1)\vartheta_1(w_2)\\
&=\left(\sum_{n_1\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+2\pi i n_1 w_1\right]\right)\left(\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_2^2\tau+2\pi i n_2 w_2\right]\right)\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+2\pi i n_1 w_1+\pi i n_2^2\tau+2\pi i n_2 w_2\right].
\end{aligned}\tag{27}$$

Similarly, from (23), we have

$$\begin{aligned}
&\vartheta_2(w_1)\vartheta_2(w_2)\\
&=\left(\sum_{n_1\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+\pi i n_1+2\pi i n_1 w_1\right]\right)\left(\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_2^2\tau+\pi i n_2+2\pi i n_2 w_2\right]\right)\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+\pi i n_1+2\pi i n_1 w_1+\pi i n_2^2\tau+\pi i n_2+2\pi i n_2 w_2\right].
\end{aligned}\tag{28}$$

Adding (27) and (28), we get

$$\begin{aligned}
&\vartheta_1(w_1)\vartheta_1(w_2)+\vartheta_2(w_1)\vartheta_2(w_2)\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+2\pi i n_1 w_1+\pi i n_2^2\tau+2\pi i n_2 w_2\right]\\
&+\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+\pi i n_1+2\pi i n_1 w_1+\pi i n_2^2\tau+\pi i n_2+2\pi i n_2 w_2\right]\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+2\pi i n_1 w_1+\pi i n_2^2\tau+2\pi i n_2 w_2\right](1+\exp(\pi i(n_1+n_2)))\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+2\pi i n_1 w_1+\pi i n_2^2\tau+2\pi i n_2 w_2\right]\left(1+(-1)^{(n_1+n_2)}\right)
\end{aligned}$$

If n_1+n_2 is odd, the corresponding terms in the series will vanish, whereas if n_1+n_2 is even, there will be a factor of 2. Therefore

$$\begin{aligned}\vartheta_1(w_1)\vartheta_1(w_2)+\vartheta_2(w_1)\vartheta_2(w_2) &= 2\sum_{\substack{n_1\in\mathbb{Z}\\ n_1+n_2=even}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+2\pi i n_1 w_1+\pi i n_2^2\tau+2\pi i n_2 w_2\right]\\ &= 2\Theta_1(w_1+w_2)\Theta_1(w_1-w_2) \text{ (using (26))}. \end{aligned} \tag{29}$$

From (25), we can write

$$\begin{aligned}
&\Theta_2(w_1+w_2)\Theta_2(w_1-w_2)\\
&=\left(\sum_{n_1\in\mathbb{Z}}\exp\left[2\pi i\left(n_1+\frac{1}{2}\right)^2\tau+2\pi i\left(n_1+\frac{1}{2}\right)(w_1+w_2)\right]\right)\\
&\quad\times\left(\sum_{n_2\in\mathbb{Z}}\exp\left[2\pi i\left(n_2+\frac{1}{2}\right)^2\tau+2\pi i\left(n_2+\frac{1}{2}\right)(w_1-w_2)\right]\right)\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\begin{bmatrix}2\pi i\left(n_1+\frac{1}{2}\right)^2\tau+2\pi i\left(n_1+\frac{1}{2}\right)(w_1+w_2)\\ +2\pi i\left(n_2+\frac{1}{2}\right)^2\tau+2\pi i\left(n_2+\frac{1}{2}\right)(w_1-w_2)\end{bmatrix}\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[2\pi i\left(n_1+\frac{1}{2}\right)^2\tau+2\pi i\left(n_2+\frac{1}{2}\right)^2\tau+2\pi i(n_1+n_2+1)w_1+2\pi i(n_1-n_2)w_2\right]\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\begin{bmatrix}\pi i\left(n_1+\frac{1}{2}\right)^2\tau+2\pi i\left(n_1+\frac{1}{2}\right)\left(n_2+\frac{1}{2}\right)\tau+\pi i\left(n_2+\frac{1}{2}\right)^2\tau+\\ \pi i\left(n_1+\frac{1}{2}\right)^2\tau-2\pi i\left(n_1+\frac{1}{2}\right)\left(n_2+\frac{1}{2}\right)\tau+\pi i\left(n_2+\frac{1}{2}\right)^2\tau+\\ 2\pi i(n_1+n_2+1)w_1+2\pi i(n_1-n_2)w_2\end{bmatrix}\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i(n_1+n_2+1)^2\tau+2\pi i(n_1+n_2+1)w_1+\pi i(n_1-n_2)^2\tau+2\pi i(n_1-n_2)w_2\right]
\end{aligned}$$

Let $m_1=n_1+n_2+1$ and $m_2=n_1-n_2$ so that $m_1+m_2=2n_1+1$. So, we have

$$\Theta_2(w_1+w_2)\Theta_2(w_1-w_2)=\sum_{\substack{m_1\in\mathbb{Z}\\ m_1+m_2=odd}}\sum_{m_2\in\mathbb{Z}}\exp\left[\pi i m_1^2\tau+\pi i m_2^2\tau+2\pi i m_1 w_1+2\pi i m_2 w_2)\right]. \tag{30}$$

Subtracting (28) from (27) and simplifying gives

$$\vartheta_1(w_1)\vartheta_1(w_2)-\vartheta_2(w_1)\vartheta_2(w_2) = 2\sum_{\substack{n_1\in\mathbb{Z}\\ n_1+n_2=odd}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+2\pi i n_1 w_1+\pi i n_2^2\tau+2\pi i n_2 w_2\right] \tag{31}$$

$$= 2\Theta_2(w_1+w_2)\Theta_2(w_1-w_2) \text{ (using (30))}. \tag{32}$$

This completes the proof of (3).

A.3 Proof of (4)

From (22), it can be written that

$$\begin{aligned}
&\vartheta_1(w_1+w_2)\vartheta_1(w_1-w_2)\\
&=\left(\sum_{n_1\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+2\pi i n_1(w_1+w_2)\right]\right)\left(\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_2^2\tau+2\pi i n_2(w_1-w_2)\right]\right)\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+2\pi i n_1(w_1+w_2)+\pi i n_2^2\tau+2\pi i n_2(w_1-w_2)\right]\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+2\pi i(n_1+n_2)w_1+\pi i n_2^2\tau+2\pi i(n_1-n_2)w_2\right].
\end{aligned} \tag{33}$$

In (33), note that n_1+n_2 and n_1-n_2 are both even or both odd. From (23), it can be written that

$$\begin{aligned}
&\vartheta_2(w_1+w_2)\vartheta_2(w_2-w_2)\\
&=\left(\sum_{n_1\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+\pi i n_1+2\pi i n_1(w_1+w_2)\right]\right)\left(\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_2^2\tau+\pi i n_2+2\pi i n_2(w_1-w_2)\right]\right)\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+\pi i n_1+2\pi i n_1(w_1+w_2)+\pi i n_2^2\tau+\pi i n_2+2\pi i n_2(w_1-w_2)\right]\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+\pi i n_1+2\pi i(n_1+n_2)w_1+\pi i n_2^2\tau+\pi i n_2+2\pi i(n_1-n_2)w_2\right]\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+2\pi i(n_1+n_2)w_1+\pi i n_2^2\tau+2\pi i(n_1-n_2)w_2\right]\exp(\pi i(n_1+n_2))\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+2\pi i(n_1+n_2)w_1+\pi i n_2^2\tau+2\pi i(n_1-n_2)w_2\right](-1)^{(n_1+n_2)}.
\end{aligned}$$

Again n_1+n_2 and n_1-n_2 are both even or both odd. If n_1+n_2 is even, then $\exp(\pi i(n_1+n_2))=1$ and if n_1+n_2 is odd, then $\exp(\pi i(n_1+n_2))=-1$.

From (24), we can write

$$\begin{aligned}
&\Theta_1(2w_1)\Theta_1(2w_2)\\
&=\left(\sum_{n_1\in\mathbb{Z}}\exp\left[2\pi i n_1^2\tau+2\pi i n_1(2w_1)\right]\right)\left(\sum_{n_2\in\mathbb{Z}}\exp\left[2\pi i n_2^2\tau+2\pi i n_2(2w_2)\right]\right)\\
&=\left(\sum_{n_1\in\mathbb{Z}}\exp\left[2\pi i n_1^2\tau+4\pi i n_1 w_1\right]\right)\left(\sum_{n_2\in\mathbb{Z}}\exp\left[2\pi i n_2^2\tau+4\pi i n_2 w_2\right]\right)\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[2\pi i n_1^2\tau+4\pi i n_1 w_1+2\pi i n_2^2\tau+4\pi i n_2 w_2\right]\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i n_1^2\tau+\pi i n_2^2\tau+2\pi i n_1 n_2\tau+4\pi i n_1 w_1+\pi i n_1^2\tau+\pi i n_2^2\tau-2\pi i n_1 n_2\tau+4\pi i n_2 w_2\right]\\
&=\sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}}\exp\left[\pi i(n_1+n_2)^2\tau+4\pi i n_1 w_1+\pi i(n_1-n_2)^2\tau+4\pi i n_2 w_2\right].
\end{aligned}$$

Let $m'_1 = n_1 + n_2$ and $m'_2 = n_1 - n_2$, where $n_1, n_2 \in \mathbb{Z}$. Therefore $m'_1 + m'_2 = 2n_1$ and $m'_1 - m'_2 = 2n_2$ are both even. So,

$$\Theta_1(2w_1)\Theta_1(2w_2) = \sum_{\substack{m'_1\in\mathbb{Z}\\ m'_1+m'_2=even}}\sum_{m'_2\in\mathbb{Z}} \exp\left[\pi i m'^2_1\tau + 2\pi i(m'_1+m'_2)w_1 + \pi i m'^2_2\tau + 2\pi i(m'_1-m'_2)w_2\right] \tag{34}$$

From (25), we can write

$$\begin{aligned}
&\Theta_2(2w_1)\Theta_2(2w_2)\\
&= \left(\sum_{n_1\in\mathbb{Z}} \exp\left[2\pi i\left(n_1+\frac{1}{2}\right)^2\tau + 2\pi i\left(n_1+\frac{1}{2}\right)(2w_1)\right]\right)\\
&\quad\times\left(\sum_{n_2\in\mathbb{Z}} \exp\left[2\pi i\left(n_2+\frac{1}{2}\right)^2\tau + 2\pi i\left(n_2+\frac{1}{2}\right)(2w_2)\right]\right)\\
&= \left(\sum_{n_1\in\mathbb{Z}} \exp\left[2\pi i\left(n_1+\frac{1}{2}\right)^2\tau + 4\pi i\left(n_1+\frac{1}{2}\right)w_1\right]\right)\\
&\quad\times\left(\sum_{n_2\in\mathbb{Z}} exp\left[2\pi i\left(n_2+\frac{1}{2}\right)^2\tau + 4\pi i\left(n_2+\frac{1}{2}\right)w_2\right]\right)\\
&= \sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}} \exp\left[2\pi i\left(n_1+\frac{1}{2}\right)^2\tau + 4\pi i\left(n_1+\frac{1}{2}\right)w_1 + 2\pi i\left(n_2+\frac{1}{2}\right)^2\tau + 4\pi i\left(n_2+\frac{1}{2}\right)w_2\right]\\
&= \sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}} \exp\begin{bmatrix}\pi i\left(n_1+\frac{1}{2}\right)^2\tau + \pi i\left(n_2+\frac{1}{2}\right)^2\tau + 2\pi i\left(n_1+\frac{1}{2}\right)\left(n_2+\frac{1}{2}\right)\tau + 4\pi i\left(n_1+\frac{1}{2}\right)w_1\\ \pi i\left(n_1+\frac{1}{2}\right)^2\tau + \pi i\left(n_2+\frac{1}{2}\right)^2\tau - 2\pi i\left(n_1+\frac{1}{2}\right)\left(n_2+\frac{1}{2}\right)\tau + 4\pi i\left(n_2+\frac{1}{2}\right)w_2\end{bmatrix}\\
&= \sum_{n_1\in\mathbb{Z}}\sum_{n_2\in\mathbb{Z}} \exp\left[\pi i(n_1+n_2+1)^2\tau + 4\pi i\left(n_1+\frac{1}{2}\right)w_1 + \pi i(n_1-n_2)^2\tau + 4\pi i\left(n_2+\frac{1}{2}\right)w_2\right].
\end{aligned}$$

Let $m'_1 = n_1 + n_2 + 1$ and $m'_2 = n_1 - n_2$ and so $m'_1 + m'_2$ and $m'_1 - m'_2$ are both odd. We can write

$$\Theta_2(2w_1)\Theta_2(2w_2) = \sum_{\substack{m'_1\in\mathbb{Z}\\ m'_1+m'_2=odd}}\sum_{m'_2\in\mathbb{Z}} \exp\left[\pi i m'^2_1\tau + 2\pi i(m'_1+m'_2)w_1 + \pi i m'^2_2\tau + 2\pi i(m'_1-m'_2)w_2\right] \tag{35}$$

Addition of (34) and (35) creates a series where m'_1 and m'_2 varies over $\mathbb{Z}$ and $m'_1 + m'_2$ and $m'_1 - m'_2$ are either both odd or both even. Therefore the series is exactly the same as the series defined by (33) and we get the identity

$$\vartheta_1(w_1+w_2)\vartheta_1(w_1-w_2) = \Theta_1(2w_1)\Theta_1(2w_2) + \Theta_2(2w_1)\Theta_2(2w_2)$$

Similarly, if we subtract (35) from (34), we get the exact series defined by (34). This gives the identity

$$\vartheta_2(w_1+w_2)\vartheta_2(w_1-w_2) = \Theta_1(2w_1)\Theta_1(2w_2) - \Theta_2(2w_1)\Theta_2(2w_2)$$

which completes the proof of (4).

B SAGE Verification Script

```
reset()
var('asq,bsq,Asq,Bsq,mu')

def SetKL(a2,b2):
    global asq,bsq,Asq,Bsq,mu
    asq = a2
    bsq = b2
    Asq = asq+bsq
    Bsq = asq-bsq
    mu = asq^2/(asq^2-bsq^2)

def KummerDbl(xsq,zsq):
    global asq,bsq,Asq,Bsq,mu
    s0 = Bsq*(xsq+zsq)^2
    t0 = Asq*(xsq-zsq)^2
    x3sq = bsq*(s0+t0)^2
    z3sq = asq*(s0-t0)^2
    return (x3sq,z3sq)

def KummerAdd(x1sq,z1sq,x2sq,z2sq,xsq,zsq):
    global asq,bsq,Asq,Bsq,mu
    s0 = Bsq*(x1sq+z1sq)*(x2sq+z2sq)
    t0 = Asq*(x1sq-z1sq)*(x2sq-z2sq)
    x3sq = zsq*(s0+t0)^2
    z3sq = xsq*(s0-t0)^2
    return (x3sq,z3sq)

def psi(xsq,zsq):
    global asq,bsq,Asq,Bsq,mu
    s0 = asq*xsq
    t0 = asq*xsq - bsq*zsq
    return s0/t0

def psiInv(X):
    global asq,bsq,Asq,Bsq,mu
    xsq = X*bsq
    zsq = (X-1)*asq
    return (xsq,zsq)

def addT2(alpha,X):
    global asq,bsq,Asq,Bsq,mu
    Ysq = X*(X-1)*(X-mu)
    tmp = mu + 1 + Ysq/(X-alpha)^2 - X - alpha
    return tmp

def ECdblXcoordinate(X1):
```

```
    global asq,bsq,Asq,Bsq,mu
    Y1sq = X1*(X1-1)*(X1-mu)
    msq = (3*X1^2-2*(mu+1)*X1+mu)^2/(4*Y1sq)
    X3 = mu+1+msq-2*X1
    return X3

def ECadd(X1,Y1,X2,Y2):
    global asq,bsq,Asq,Bsq,mu
    m = (Y1-Y2)/(X1-X2)
    X3 = mu + 1 + m^2 - X1 - X2
    return X3

def ECsubtract(X1,Y1,X2,Y2):
    global asq,bsq,Asq,Bsq,mu
    m = (Y1+Y2)/(X1-X2)
    X = mu + 1 + m^2 - X1 - X2
    return X

def ECaddXcoordinate(X1,X2,X):
    global asq,bsq,Asq,Bsq,mu
    Y1sq = X1*(X1-1)*(X1-mu)
    Y2sq = X2*(X2-1)*(X2-mu)
    Y1plusY2sq = (X1+X2+X-mu-1)*(X2-X1)^2
    Y2minusY1sq = 2*Y1sq + 2*Y2sq - Y1plusY2sq
    msq = Y2minusY1sq/(X2-X1)^2
    X3 = mu + 1 + msq - X1 - X2
    return X3

def verifyDblKummerToEC(xsq,zsq):
    global asq,bsq,Asq,Bsq,mu
    X = psi(xsq,zsq).simplify_full()
    X1 = X
    X3 = ECdblXcoordinate(X1).simplify_full()
    val = KummerDbl(xsq,zsq)
    x3sq = val[0]
    z3sq = val[1]
    X3prime = psi(x3sq,z3sq)
    X3prime = addT2(mu,X3prime).simplify_full()
    tmp = X3 - X3prime
    return tmp.simplify_full()

def verifyDblECToKummer(X1):
    global asq,bsq,Asq,Bsq,mu
    val = psiInv(X1)
    x1sq = val[0]
    z1sq = val[1]
    val = KummerDbl(x1sq,z1sq)
    x3sq = val[0]
```

```
    z3sq = val[1]
    X3 = ECdblXcoordinate(X1)
    X3 = addT2(mu,X3)
    val = psiInv(X3)
    x3primesq = val[0]
    z3primesq = val[1]
    tmp = x3sq*z3primesq - x3primesq*z3sq
    return tmp.simplify_full()

def verifyAddECToKummer(X1,Y1,X2,Y2):
    global asq,bsq,Asq,Bsq,mu

    X3 = ECadd(X1,Y1,X2,Y2)
    X = ECsubtract(X1,Y1,X2,Y2)

    X1prime = addT2(mu,X1)
    val = psiInv(X1prime)
    x1sq = val[0]
    z1sq = val[1]

    X2prime = addT2(mu,X2)
    val = psiInv(X2prime)
    x2sq = val[0]
    z2sq = val[1]

    X3prime = addT2(mu,X3)
    val = psiInv(X3prime)
    x3sq = val[0]
    z3sq = val[1]

    Xprime = addT2(mu,X)
    val = psiInv(Xprime)
    xsq = val[0]
    zsq = val[1]

    val = KummerAdd(x1sq,z1sq,x2sq,z2sq,xsq,zsq)
    x3primesq = val[0]
    z3primesq = val[1]

    tmp = x3sq*z3primesq - x3primesq*z3sq
    tmp1 = tmp.numerator()
    g1 = Y1^2 - X1*(X1-1)*(X1-mu)
    g2 = Y2^2 - X2*(X2-1)*(X2-mu)
    tmp2 = tmp1.maxima_methods().divide(g1)
    tmp1 = tmp2[1]
    tmp2 = tmp1.maxima_methods().divide(g2)
    tmp1 = tmp2[1]
    return tmp1.simplify_full()
```

```
var('asq,bsq')
SetKL(asq,bsq)

val = KummerDbl(asq,bsq) # double of (a^2,b^2) is (a^2,b^2)
tmp = val[0]/val[1]
print tmp.full_simplify()

val = KummerDbl(bsq,asq) # double of (b^2,a^2) is (a^2,b^2)
tmp = val[0]/val[1]
print tmp.full_simplify()

var('xsq,zsq')
val = KummerAdd(xsq,zsq,asq,bsq,xsq,zsq) # addition of (x^2,z^2) and (a^2,b^2) is (x^2,z^2)
tmp = val[0]/val[1]
print tmp.full_simplify()

val = KummerAdd(xsq,zsq,xsq,zsq,asq,bsq) # addition of (x^2,z^2) and (x^2,z^2) is 2(x^2,z^2)
tmp1 = val[0]/val[1]
val = KummerDbl(xsq,zsq)
tmp2 = val[0]/val[1]
print tmp1 - tmp2

var('xsq,zsq')
print verifyDblKummerToEC(xsq,zsq) # consistency of doubling on Kummer Line and EC

var('X1')
print verifyDblECToKummer(X1) # consistency of doubling on EC and Kummer Line

var('X1,Y1,X2,Y2')
print verifyAddECToKummer(X1,Y1,X2,Y2) # consistency of pseudo-addition on EC and Kummer Line
```