Binary Kummer Line

Sabyasachi Karati

R. C. Bose Centre for Cryptology and Security
Indian Statistical Institute Kolkata, India
skarati@isical.ac.in

Abstract. Gaudry and Lubicz introduced the idea of Kummer line in 2009, and Karati and Sarkar proposed three Kummer lines over prime fields in 2017. In this work, we explore the problem of secure and efficient scalar multiplications on binary field using Kummer line and investigate the possibilities of speedups using Kummer line compared to Koblitz curves, binary Edwards curve and Weierstrass curves. We propose a binary Kummer line BKL251 over binary field $\mathbb{F}_{2^{251}}$ where the associated elliptic curve satisfies the required security conditions and offers 124.5-bit security which is the same as that of Binary Edwards curve BEd251 and Weierstrass curve CURVE2251. BKL251 has small curve parameter and small base point. We implement our software of BKL251 using the instruction PCLMULQDQ of modern Intel processors and batch software BBK251 using bitslicing technique. For fair comparison, we also implement the software BEd251 for binary Edwards curve. In both the implementations, scalar multiplications take constant time which use Montgomery ladders. In case of left-to-right Montgomery ladder, both the Kummer line and Edwards curve have almost the same number of field operations. For right-to-left Montgomery ladder scalar multiplication, each ladder step of binary Kummer line needs less number of field operations compared to Edwards curve. Our experimental results show that left-to-right Montgomery scalar multiplications of BKL251 are 9.63% and 0.52% faster than those of BEd251 for fixed-base and variable-base, respectively. Left-to-right Montgomery scalar multiplication for variable-base of BKL251 is 39.74%, 23.25% and 32.92% faster than those of the curves CURVE2251, K-283 and B-283 respectively. Using right-to-left Montgomery ladder with precomputation, BKL251 achieves 17.84% speedup over BEd251 for fixed-base scalar multiplication. For batch computation, BBK251 has comparatively the same (slightly faster) performance as BBE251 and sect283r1. Also it is clear from our experiments that scalar multiplications on BKL251 and BEd251 are (approximately) 65% faster than one scalar multiplication (after scaling down) of batch software BBK251 and BBE251.

Keywords: Binary Finite Field Arithmetic, Elliptic Curve Cryptography, Kummer line, Edwards Curve, Montgomery Ladder, Scalar Multiplication.

1 Introduction

Diffie-Hellman (DH) key agreement [18] protocol is one of the most important protocols of modern cryptography. It allows two users to communicate over a public channel and create a shared key to establish a secure communication session. The protocol has two phases: i) exchange of public keys, and ii) computation of a shared key using their own secret key and the received public key.

Elliptic curve cryptography was introduced separately by Miller [35] and Koblitz [31] and hyperelliptic curve cryptography by Koblitz [32]. Elliptic Curve DH (ECDH) is an instantiation of DH protocol designed on the cyclic group of Elliptic curves where the corresponding discrete logarithm problem is computationally hard. ECDH is the fastest and has smallest key size among all the other variants of DH protocol.

All SSH/TLS communications start with DH key agreement protocol between client and server. TLS 1.3 uses Ephemeral Diffie-Hellman key-exchange protocol [47, 50] and includes new elliptic curves which target at 128-bit and the 224-bit security. Famous Montgomery curve Curve25519 has been included in TLS 1.3 which provides 128-bit security. Kummer lines are proposed in [23, 21] and have been considered as an alternative to elliptic curves. [30, 28] propose three Kummer lines over prime fields with 128-bit target security level. Concrete implementations of these Kummer lines using SIMD vector instructions of modern processors show that these Kummer lines are significantly faster than SIMD-based implementation of Curve25519 [40].

TLS 1.3 and cryptographic libraries like OpenSSL support elliptic curves over both the prime and binary fields [49, 41, 13]. K-233, B-233, K-283, B-283 [42] of TLS 1.3 and OpenSSL are elliptic curves over binary fields which target 128-bit security. Binary Edwards curve BEd251 [6, 9] and CURVE2251 [51] over binary field $\mathbb{F}_{2^{251}}$ are other prominent curves which also target 128-bit security.

Performance of elliptic curve cryptography depends on the efficiency of scalar multiplications. computation of side-channel attack resistant scalar multiplication is a prerequisite for cryptographic applications. If we consider important primitives of public key cryptography like DH protocol or digital signatures [42], we see that fixed-base scalar multiplications take the most important role to determine the performance. In DH protocol, computation of public key is done by fixed-base scalar multiplication and one variable-base scalar multiplication is required to compute the shared secret. The performance of DH is measured by the sum of those two scalar multiplications. If we consider digital signature algorithms as ECDSA [24] or qDSA [30, 48], key generation and signing use only fixed-base scalar multiplications. Verification of ECDSA uses a double-base scalar multiplication. On the other hand, qDSA is designed using the x-coordinate based constant-time scalar multiplications over Kummer line. As a consequence, only the x-coordinates of the generator of the group and the public key of the signer are available. The signature verification modules of the software [30, 48], therefore, use one fixed-base and one variable-base scalar multiplication. It is shown in [5, 30] that small base point can improve the performance of fixed-base scalar multiplication significantly compared to fixed-base scalar multiplication with large base point and variable-base scalar multiplication. For an example, Curve25519 and Kummer lines KL2519, KL25519 and KL2663 have small base points and as a consequence they achieve $18\% - 24\%$ faster fixed-base scalar multiplication compared to variable-base scalar multiplication.

But the NIST Koblitz curves K-233 and K-283 and NIST random curves B-233, and B-283 do not possess any small base-point. [13] reports the fastest implementation of these four curves for variable-base scalar multiplications which use LD-Montgomery method. Also the base-point of CURVE2251 is large and similarly the available fastest implementation does not get any advantage of small base point [51]. For efficient arithmetic, BEd251 uses WZ-coordinate system and the fact $W = X + Y$ prevents to have a small base point. Therefore, to have efficient implementation, software BBE251 computes 128 variable-base scalar multiplications of binary Edwards curve BEd251 in a batch [6]. Batch implementation of 128 variable-base scalar multiplications of binary Weierstrass curve CURVE2251 is available at [15]. These software of batch computation use the bitslicing technique to achieve the best possible result by avoiding the shifting operations. They compute $1 \leqslant n \leqslant 128$ scalar multiplications in batches and take the same amount of time irrespective of the value of n. In other words, without any requirement of a large number of scalar multiplications, we can not get the speedups mentioned for these software. For a busy server, these software are effective ones. But for simple clients machines, these software are not suitable.

1.1 Our Contributions

This work introduces concrete proposition of binary Kummer line BKL251 over field $\mathbb{F}_{2^{251}}$ which has small base point and targets at 128-bit security. We show the efficient behavior of BKL251 through field operation count comparison and by developing software using `PCLMULQDQ` instructions along with the efficient implementation of binary Edwards curve BEd251 [6]. We also provide software which computes 128 scalar multiplications for both the fixed-base and variable-base in batch for BKL251. The experimental results show that BKL251 offers the fastest DH protocol [18] on binary elliptic curves among all the constant-time implementations which do not use endomorphism. In this work, we use the conservative set-up over binary field "with as little algebraic structure as possible" [12]. Therefore, we do not use any special field with beneficial properties (like $\mathbb{F}_{q^2}$) or any special algebraic properties (like endomorphism).

1. **Binary Kummer line.** Following the introduction of Kummer line over binary fields by Gaudry and Lubicz in [21], this work shows that it is possible to achieve competitive speed using binary Kummer line compared to Weierstrass curves, Koblitz curves, binary Edwards curve or Kummer lines over prime fields. In Section 2, we provide the new theoretical details of binary Kummer line along with the basic details available at [21]:

- We introduce the identity and the point of order 2 on binary Kummer line. Along with the theoretical interests, the identity also has an important role in the right-to-left Montgomery ladder scalar multiplication.
- Mapping π from Kummer line to associated elliptic curve was given in [21]. We give the explicit formulas for the mapping π and that of π^{-1} (Section 2.2). But the mapping π alone does not preserve the consistency between scalar multiplications on Kummer lines and associated elliptic curves. We extend the mapping π to $\hat{\pi}$ by adding a particular point of order two on the associated elliptic curve and proves the consistency of scalar multiplications (Section 2.3). We also prove the equivalence between the hardness of discrete logarithm problem (DLP) defined on Kummer line and associated elliptic curve.
- Let P be a point on the elliptic curve and n be a scalar, and the scalar multiplication is done via binary Kummer line. Here, we also supply the explicit formulas required to recover the y-coordinate of the point nP (Section 2.4).

In this work, we propose a binary Kummer line BKL251 (Section 5) whose details are given below:

- **Choice of Finite Field.** Our target is 128-bit security. For fair comparison with BEd251 and CURVE2251, we choose the finite field $\mathbb{F}_{2^{251}}$.
- **Choice of Kummer line.** The binary Kummer line BKL251 is one with the smallest curve parameter $b = t^{13} + t^9 + t^8 + t^7 + t^2 + t + 1 \in \mathbb{F}_{2^{251}}$ such that it satisfies certain security conditions. For the line, we also identify a base point $(t^3 + t^2, 1)$ with small coordinates. Later we provide the details of the above mentioned security conditions and show that BKL251 offers 124.5-bit security similar to BEd251 and CURVE2251.

2. **Implementation of Scalar Multiplication over binary Kummer line.** Section 6.1 deals with one of the major concern: implementation of scalar multiplication resistant against side-channel attacks like timing attacks [9]. The solution is constant time scalar multiplication and one of the popular choices is Montgomery ladder [38] with a differential addition and a doubling operation. In binary Kummer line, one left-to-right Montgomery ladder step requires 4[M], 5[S] along with 1[M] by Kummer line parameter and 1[M] by base point[1]. By carefully choosing small Kummer line parameter and small base point, we achieve one ladder step at the cost of 4[M]+5[S]+2[C] for fixed-base scalar multiplications and 5[M]+5[S]+1[C] for variable-base scalar multiplications. With precomputed multiplies of the base point, each step of the right-to-left Montgomery ladder of fixed-base scalar multiplication of binary Kummer line needs 4[M]+2[S] operations. The implemented software for BKL251 are publicly available at:

   BKL251: `https://github.com/skarati/BKL251`

3. **Implementation of Scalar Multiplication over binary Edwards Curve.** To make a fair comparison, the possible candidates are mpfq [22], CURVE2251 and BEd251 along with K-283, B-283, K-233 and B-233. But there exist certain problems associated with the available software of each of the mpfq [22], CURVE2251 and BEd251.
   (a) mpfq uses similar curve arithmetic as binary Kummer line but the software is approximately 12 years old and it does not take advantage of the present processors. The software was developed for the curve
   $$E_m : Y^2 + XY = X^3 + (t^{13} + t^9 + t^8 + t^7 + t^2 + t + 1),$$
   and $(t^3 + 1, 1)$ has been used as base point for fixed-based scalar multiplications. But $(t^3 + 1, 1)$ is not a point of E_m rather it is a point on the twist of E_m [20].
   (b) Latest implementation of CURVE2251 is available as a part of the RELIC [1, 2] but each ladder step takes 6[M]+5[S] [52] which is higher than the operation count of ladder step of the binary Edwards curve.
   (c) There does not exist a single software (best to our knowledge) which provides efficient implementation of single fixed-base and variable-base scalar multiplications of BEd251.

[1] [M], [S], [C] and [B] stand for field multiplication, field squaring, multiplication by small constant and multiplication by base-point respectively.

One ladder step of binary Edwards Curve [9] needs 4[M], 4[S] along with 2[M] by curve parameters and 1[M] by base point in WZ-coordinate system. Because of WZ-coordinate system, base point becomes a full length element of the field. As a consequence, the operation count for each ladder step of left-to-right Montgomery scalar multiplication becomes 5[M]+4[S]+2[C] for both the cases of fixed-base and variable-base. On the other hand, each ladder step of fixed-base scalar multiplication of binary Edwards curve needs 5[M]+2[S]+2[C] for right-to-left Montgomery ladder with precomputation. We also implement the binary Edwards curve BEd251 using left-to-right and right-to-left Montgomery ladder which take the above mentioned field operations and the implemented software are available at:

BEd251: `https://github.com/skarati/BEd251`

4. **Batch Binary Kummer** BBK251. The software BBE251 of BEd251 is available at [7] and uses bitslicing technique to compute 128 variable-base scalar multiplications in batches. In this work, we also implement software which computes 128 scalar multiplications for fixed-base and variable-base in batch for BKL251 using bitslicing technique and we name the software as BBK251. The software is publicly available at:

BBK251: `https://github.com/skarati/Batch-Binary-Kummer-BBK251`

2 Binary Kummer line

Let k be a finite field of characteristic two. Let $b \in k$ and $b \neq 0$. Let E_b be an elliptic curve defined over k by Equation (1):

$$E_b : Y^2 + XY = X^3 + b^4. \tag{1}$$

In this work, we explore the problem of efficient and timing-attack resistant computation of scalar multiplication via the Kummer line associated with the elliptic curve E_b. Gaudry and Lubicz define the Kummer line associated with E_b using algebraic theta functions [21] and we denote this Kummer line by $\mathsf{BKL}_{(1,b)}$. We refer [21] to interested reader for further details.

The arithmetic of Kummer line is given in projective coordinate system. Let $\mathbf{P} = (x_1, z_1)$ and $\mathbf{Q} = (x_2, z_2)$ be two points on the Kummer line such that $\mathbf{P} \neq (0,0)$ and $\mathbf{Q} \neq (0,0)$. We say that $\mathbf{P}$ and $\mathbf{Q}$ are equivalent, denoted by $\mathbf{P} \sim \mathbf{Q}$, if there exists a $\xi \in k^*$ such that $x_1 = \xi x_2$ and $z_1 = \xi z_2$.

Suppose that $\mathbf{P} = (x_1, z_1)$ is a projective point on the Kummer line $\mathsf{BKL}_{(1,b)}$. Given the point $\mathbf{P}$, doubling Algorithm `dbl` of Table 1 computes $2\mathbf{P} = (x_3, z_3)$. Let $\mathbf{Q} = (x_2, z_2)$ be another point on the $\mathsf{BKL}_{(1,b)}$ and we want to compute $\mathbf{P} + \mathbf{Q} = (x_4, z_4)$. Given the point $\mathbf{P} - \mathbf{Q} = (x, z)$, computation of (x_4, z_4) is shown by the differential addition Algorithm `diffAdd` of Table 1.

$(x_3, z_3) = \mathtt{dbl}(x_1, z_1)$:	$(x_4, z_4) = \mathtt{diffAdd}(x_1, z_1, x_2, z_2, x, z)$:
$x_3 = b(x_1^2 + z_1^2)^2$;	$x_4 = z(x_1 x_2 + z_1 z_2)^2$;
$z_3 = (x_1 z_1)^2$;	$z_4 = x(x_1 z_2 + x_2 z_1)^2$;

Table 1. Doubling and Differential Addition on Binary Kummer line

In Kummer line $\mathsf{BKL}_{(1,b)}$, the point $\mathbf{I} = (1, 0)$ acts as an identity. This can be proved by showing

$$\left.\begin{array}{r} \mathtt{diffAdd}(x, z, 1, 0, x, z) = (x^2 z, x z^2) \sim (x, z) \\ \mathtt{diffAdd}(1, 0, x, z, x, z) = (x^2 z, x z^2) \sim (x, z) \\ \mathtt{dbl}(1, 0) = (b, 0) \sim (1, 0) \end{array}\right\} \tag{2}$$

It also can be shown that the point $(0, 1)$ is a point of order 2 as given in Equation (3)

$$\mathtt{dbl}(0, 1) = (b, 0) \sim (1, 0) \tag{3}$$

In the rest of the paper, we will consider Kummer line $\mathsf{BKL}_{(1,b)}$ for some non-zero element $b \in k$.

2.1 Scalar Multiplication

Let $\mathbf{P} = (x, z)$ be a point on the Kummer line $\mathsf{BKL}_{(1,b)}$ and n be a l-bit scalar as $n = \{1, n_{l-2}, \ldots, n_0\}$. Our objective is to compute $n\mathbf{P} = (x_n, z_n)$. We apply Montgomery ladder to perform this operation [36]. The ladder step iterates for $l - 1$ times and each ladder step performs a `dbl` and a `diffAdd` operation.

Assume that at a ladder step, the inputs are the points (x_1, z_1) and (x_2, z_2). At the end of the ladder step, the outputs are two points (x_3, z_3) and (x_4, z_4). Suppose that we need to compute double of the point (x_1, z_1) and addition of the points (x_1, z_1) and (x_2, z_2), then during the ladder step we compute $(x_3, z_3) = \mathtt{dbl}(x_1, z_1)$ and $(x_4, z_4) = \mathtt{diffAdd}(x_1, z_1, x_2, z_2, x, z)$. The details of the Algorithms `scalarMult` and `ladderStep` are given in the Table 2. Notice that, Algorithm `ladderStep` uses "If" condition, but in our implemented code of the ladder step we do not use any branching instruction.

We start Algorithm `scalarMult` with two points $\mathbf{S} = \mathbf{P}$ and $\mathbf{R} = 2\mathbf{P} = \mathtt{dbl}(\mathbf{P})$. Let at i-th iteration, the inputs be the points $\mathbf{S} = m\mathbf{P}$ and $\mathbf{R} = (m+1)\mathbf{P}$. Then if $n_i = 0$, the `ladderStep` outputs the points $\mathbf{S} = 2m\mathbf{P}$ and $\mathbf{R} = (2m+1)\mathbf{P}$. On the other hand, if $n_i = 1$, the `ladderStep` computes the points $\mathbf{S} = (2m+1)\mathbf{P}$ and $\mathbf{R} = 2(m+1)\mathbf{P}$.

$n\mathbf{P} = \mathtt{scalarMult}(\mathbf{P}, n)$:	$\mathtt{ladderStep}(\mathbf{S}, \mathbf{R}, n_i)$:
1. Let $n = \{1, n_{l-2}, \ldots, n_0\}$;	1. If $n_i = 0$ then
2. Set $\mathbf{S} = \mathbf{P}$ and $\mathbf{R} = \mathtt{dbl}(\mathbf{P})$;	2. $\mathbf{R} = \mathtt{diffAdd}(\mathbf{S}, \mathbf{R}, \mathbf{P}); \mathbf{S} = \mathtt{dbl}(\mathbf{S})$;
3. For $i = l - 2$ to 0 do	3. Else If $n_i = 1$ then
4. $\mathtt{ladderStep}(\mathbf{S}, \mathbf{R}, n_i)$;	4. $\mathbf{S} = \mathtt{diffAdd}(\mathbf{S}, \mathbf{R}, \mathbf{P}); \mathbf{R} = \mathtt{dbl}(\mathbf{R})$;
5. End For;	5. End If;
6. Return $\mathbf{S}$;	

Table 2. (Left-to-Right) Scalar Multiplication and Ladder Step

2.2 Binary Kummer line and the associated Elliptic Curve

We can map a point of Kummer line $\mathbf{P}$ to elliptic curve by the mapping $\pi : \mathsf{BKL}_{(1,b)} \to E_b/\{\pm 1\}$ [21] which is defined as

$$\pi(\mathbf{P} = (x, z)) = \begin{cases} (bz, \cdot, x), & \text{if } x \neq 0 \\ \infty, & \text{if } x = 0. \end{cases} \tag{4}$$

Putting $X = \frac{bz}{x}$ in Equation (1), we can compute the Y-coordinate up to elliptic involution. We can also move back to Kummer line $\mathsf{BKL}_{(1,b)}$ from $E_b/\{\pm 1\}$ using the inverse mapping π^{-1} as defined by Equation (5). Let $P = (X, \cdot, Z)$ be a point on E_b then

$$\pi^{-1}(P) = \begin{cases} (bZ, \cdot, X), & \text{if } X \neq 0 \\ (0, 1), & \text{if } X = 0. \end{cases} \tag{5}$$

But the mapping π alone does not conserve the consistency of the scalar multiplications between Kummer line $\mathsf{BKL}_{(1,b)}$ and the elliptic curve E_b. We also need a point of order two of the elliptic curve E_b as given in the next section.

2.3 Equivalence between $\mathsf{BKL}_{(1,b)}$ and E_b

Let $\mathsf{BKL}_{(1,b)}$ be a Kummer line on the binary field k and E_b be the associated elliptic curve as defined by Equation (1). Let $\mathbf{P}$ be a point on Kummer line $\mathsf{BKL}_{(1,b)}$. Also consider the point $T_2 = (0, b^2)$ which is a point of order two on E_b. As $\pi(\mathbf{P})$ is a point on E_b, $\pi(\mathbf{P}) + T_2$ is also a point on E_b. Now we extend the mapping π to $\hat{\pi}$ by Equation (6):

$$\hat{\pi}(\mathbf{P}) = \pi(\mathbf{P}) + T_2. \tag{6}$$

The inverse mapping of $\hat{\pi}$ is defined as

$$\hat{\pi}^{-1}(P) = \pi^{-1}(P + T_2), \tag{7}$$

where P is a point on E_b. Let $\mathbf{P} = (x, z)$ be a point on the Kummer line such that it is not a point of order 2 or identity, then Equation (8) holds.

$$2\hat{\pi}(\mathbf{P}) = \hat{\pi}(\texttt{dbl}(\mathbf{P})) \tag{8}$$

Let P_1 and P_2 be any two points on E_b such that $P_1 \neq \pm P_2$ and neither of them is point at infinity nor of order 2, then Equation (9) holds.

$$\left.\begin{aligned} \hat{\pi}\left(\texttt{dbl}(\hat{\pi}^{-1}(P_i))\right) &= 2P_i, i = 1, 2 \\ \hat{\pi}\left(\texttt{diffAdd}(\hat{\pi}^{-1}(P_1), \hat{\pi}^{-1}(P_2), \hat{\pi}^{-1}(P_1 - P_2))\right) &= P_1 + P_2 \end{aligned}\right\} \tag{9}$$

Notice that $2\hat{\pi}(\mathbf{P}) = 2(\pi(\mathbf{P}) + T_2) = 2\pi(\mathbf{P})$. The proofs of Equations (8) and (9) are trivial, but the expressions grow very fast and hard to compute manually. Therefore, we used a GP/PARI [53] script to verify them symbolically and made available along with the software. Equations (8) and (9) are important as they form the consistency between the scalar multiplications on binary Kummer line $\mathsf{BKL}_{(1,b)}$ and elliptic curve E_b. Let $\mathbf{P} \in \mathsf{BKL}_{(1,b)}$. Then we have $\hat{\pi}(n\mathbf{P}) = n\hat{\pi}(\mathbf{P})$. Again, we have that $\hat{\pi}(n\mathbf{P}) = \pi(n\mathbf{P}) + T_2$ and $n\hat{\pi}(\mathbf{P}) = n(\pi(\mathbf{P}) + T_2) = n\pi(\mathbf{P}) + n \pmod 2\, T_2$. Therefore, $\hat{\pi}(n\mathbf{P}) = n\hat{\pi}(\mathbf{P})$ can be rewritten as:

$$\pi(n\mathbf{P}) = n\pi(\mathbf{P}) + (1 + n \pmod 2)\, T_2$$

which is pictorially shown in Figure 1. The equivalence of scalar multiplication on Kummer line and the

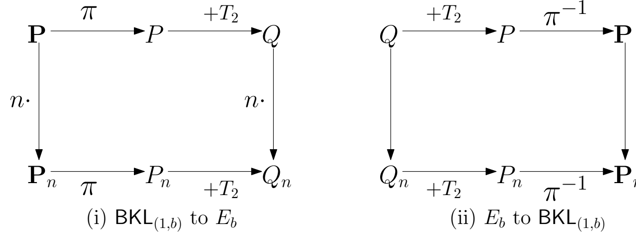

Fig. 1. Consistency of scalar multiplications between $\mathsf{BKL}_{(1,b)}$ and E_b

associated elliptic curve is exactly the same as Kummer line on prime field [30]. From Figure 1, we conclude that discrete logarithm problem is equally hard on the Kummer line $\mathsf{BKL}_{(1,b)}$ and the elliptic curve E_b.

2.4 Retrieving y-coordinate

One of the main purposes of the Kummer line is to perform faster scalar multiplications. Let $P = (X_P, Y_P, Z_P)$ be a point on elliptic curve E_b and n be a scalar. The objective is to compute $S = nP$ via Kummer line $\mathsf{BKL}_{(1,b)}$ and this can be done in the following manner.

Set $\mathbf{P} = \hat{\pi}^{-1}(P)$ and compute $(\mathbf{S}, \mathbf{R}) = \texttt{scalarMult}(\mathbf{P}, n)$ where $\mathbf{S} = n\mathbf{P}$ and $\mathbf{R} = \mathbf{S} + \mathbf{P}$. By the consistency of scalar multiplications $S = \hat{\pi}(\mathbf{S}) = (X_S, Y_S, Z_S)$ and $R = S + P = \hat{\pi}(\mathbf{R}) = (X_R, Y_R, Z_R)$. But the problem is that scalar multiplication by Kummer line does not provide the Y_S explicitly. In this section, we provide the method to recover Y_S from the known values $S = (X_S, \cdot, Z_S)$ and $R = (X_R, \cdot, Z_R)$ and $P = (X_P, Y_P, Z_P)$ following the approaches given in [29, 43]. Let the affine coordinates of the points be $S = (x_s, y_s) = \left(\frac{X_S}{Z_S}, \frac{Y_S}{Z_S}\right)$, $R = (x_r, y_r) = \left(\frac{X_R}{Z_R}, \frac{Y_R}{Z_R}\right)$ and $P = (x_p, y_p) = \left(\frac{X_P}{Z_P}, \frac{Y_P}{Z_P}\right)$ where Y_S and Y_R are unknown to us.

By chord-and-tangent rule for addition on E_b, the points S, P and $-R$ lie on the straight line $Y = mX + c$ where $m = \frac{y_s+y_p}{x_s+x_p}$. Substituting $Y = mX + c$ into the equation of the curve we obtain:

$$X^3 + (m + m^2)X^2 + cX + (c^2 + b^4) = 0. \tag{10}$$

Now x_s, x_p and x_r are roots of Equation (10) and we have:

$$x_s + x_p + x_r = m + m^2. \tag{11}$$

Putting $m = \frac{y_s+y_p}{x_s+x_p}$ in Equation (11), we recover y_s as given in Equation (12).

$$y_s = \frac{1}{x_p}\left[(x_s x_p + x_p x_r + x_r x_s)(x_s + x_p) + y_p x_s\right]. \tag{12}$$

Substituting $x_s = \frac{X_S}{Z_S}$, $y_s = \frac{Y_S}{Z_S}$, $x_r = \frac{X_R}{Z_R}$, $x_p = \frac{X_P}{Z_P}$ and $y_p = \frac{Y_P}{Z_P}$ in Equation (12), we get projective coordinate Y_S as given in Equation (13):

$$Y_S = \frac{Z_S Z_P}{X_P}\left[\left(\frac{X_S X_P}{Z_S Z_P} + \frac{X_P X_R}{Z_P Z_R} + \frac{X_R X_S}{Z_R Z_S}\right)\left(\frac{X_S}{Z_S} + \frac{X_P}{Z_P}\right) + \frac{Y_P X_S}{Z_P Z_S}\right]. \tag{13}$$

3 Binary Edwards Curve

In this section, we give a brief description of binary Edwards curve to make the paper self-contained. Let k be a field of characteristic 2 and $d \in k \setminus \{0\}$. We define binary Edwards curve [6, 9] by Equation (14).

$$\mathsf{BEd} : d(x + x^2 + y + y^2) = (x + x^2)(y + y^2) \tag{14}$$

The neutral element is the point $(0, 0)$. The point $(1, 1)$ has order 2. The Edwards curve BEd is birationally equivalent to the ordinary curve E_d of Equation (15).

$$E_d : X^2 + XY = X^3 + (d^2 + d)X + d^8. \tag{15}$$

The mapping from BEd to E_d is given by Equation (16).

$$\begin{aligned}\phi : (x, y) &\mapsto (X, Y)\\ X &= \frac{d^3(x + y)}{xy + d(x + y)}\\ Y &= d^3\left(\frac{x}{xy + d(x + y)} + d + 1\right)\end{aligned} \tag{16}$$

[6, 9] suggest the use of WZ-coordinate system, where $W = X + Y$, which provides the minimum operation count for each of the ladder step of Montgomery scalar multiplication. Let $P, Q \in \mathsf{BEd}$ such that $P = (w_2, z_2)$, $Q = (w_3, z_3)$ and $P - Q = (w_1, 1)$ are given. We compute $2P = (w_4, z_4)$ and $P + Q = (w_5, z_5)$ using mixed differential and doubling operation as given in Table 3. We refer [6, 9] for further details of binary Edwards curve.

$c = w_2(w_2 + z_2)$;	$w_4 = c^2$;	$z_4 = d(z_2^2)^2 + w_4$;
$v = cw_3(w_3 + z_3)$;	$z_5 = v + d(z_2 z_3)^2$;	$w_5 = v + z_5 w_1$;

Table 3. Mixed differential and doubling of binary Edwards Curve

4 Right-to-Left Montgomery ladder

The x-coordinate-based scalar multiplication of Section 2.1 is the traditional one and sometimes is referred as left-to-right Montgomery ladder. We also have right-to-left Montgomery ladder method of scalar multiplication which was introduced in [27]. This section provides the details of the right-to-left Montgomery ladder double-and-add algorithm of [45] in the context of $\mathsf{BKL}_{(1,b)}$ and BEd.

Let $\mathbf{P}$ be a point on the Kummer line $\mathsf{BKL}_{(1,b)}$ (or BEd) and n be a scalar. We compute scalar multiplication $n\mathbf{P}$ using the Algorithm `scalarMultR2L` as given in Table 4. We start with the points $\mathbf{R_0} = \mathbf{P}$, $\mathbf{R_1} = \mathbf{I}$ and $\mathbf{R_2} = \mathbf{P}$. If the bit in the scalar is 1, then we compute $\mathbf{R_1} = \mathtt{diffAdd}(\mathbf{R_0}, \mathbf{R_1}, \mathbf{R_2})$ else we compute $\mathbf{R_2} = \mathtt{diffAdd}(\mathbf{R_0}, \mathbf{R_2}, \mathbf{R_1})$. At the end of each iteration, we compute $\mathbf{R_0} = \mathtt{dbl}(\mathbf{R_0})$. Notice that, the invariant $\mathbf{R_0} = \mathbf{R_1} + \mathbf{R_2}$ always holds at the beginning of and end of each iteration.

$n\mathbf{P} = \mathtt{scalarMultR2L}(\mathbf{P}, n)$:	$\mathtt{ladderStepR2L}(\mathbf{R_0}, \mathbf{R_1}, \mathbf{R_2}, n_i)$:
1. Let $n = \{n_{l-1}, n_{l-2}, \ldots, n_0\}$;	1. If $n_i = 1$ then
2. Set $\mathbf{R_0} = \mathbf{P}$, $\mathbf{R_1} = \mathbf{I}$ and $\mathbf{R_2} = \mathbf{P}$;	2. $\mathbf{R_1} = \mathtt{diffAdd}(\mathbf{R_0}, \mathbf{R_1}, \mathbf{R_2})$;
3. For $i = 0$ to $l - 1$ do	3. Else If $n_i = 0$ then
4. $\mathtt{ladderStepR2L}(\mathbf{R_0}, \mathbf{R_1}, \mathbf{R_2}, n_i)$;	4. $\mathbf{R_2} = \mathtt{diffAdd}(\mathbf{R_0}, \mathbf{R_2}, \mathbf{R_1})$;
5. End For;	5. End If;
6. Return $\mathbf{R_1}$;	6. $\mathbf{R_0} = \mathtt{dbl}(\mathbf{R_0})$;

Table 4. Right-to-Left Scalar Multiplication and Ladder Step

$n\mathbf{P} = \mathtt{scalarMultR2LwPrecomp}(\mathbf{P}, n)$:	$\mathtt{ladderStepR2L}(\mathbf{R_0}, \mathbf{R_1}, \mathbf{R_2}, n_i)$:
1. Let $n = \{n_{l-1}, n_{l-2}, \ldots, n_0\}$;	1. If $n_i = 1$ then
2. Set $\mathbf{R_0} = \mathbf{P}$, $\mathbf{R_1} = \mathbf{I}$ and $\mathbf{R_2} = \mathbf{P}$;	2. $\mathbf{R_1} = \mathtt{diffAdd}(\mathbf{R_0}, \mathbf{R_1}, \mathbf{R_2})$;
3. For $i = 0$ to $l - 1$ do	3. Else If $n_i = 0$ then
4. $\mathtt{ladderStepR2L}(\mathbf{R_0}, \mathbf{R_1}, \mathbf{R_2}, n_i)$;	4. $\mathbf{R_2} = \mathtt{diffAdd}(\mathbf{R_0}, \mathbf{R_2}, \mathbf{R_1})$;
5. $\mathbf{R_0} = 2^{i+1}\mathbf{P}$;	5. End If;
6. End For;	
7. Return $\mathbf{R_1}$;	

Table 5. Scalar Multiplication with Precomputation and Right-to-Left Ladder Step

In Algorithm `scalarMultR2L`, all the three points $\mathbf{R_0}$, $\mathbf{R_1}$ and $\mathbf{R_2}$ keep changing at each ladder step. Therefore, we can not take the advantage of small base point to have efficient implementation of fixed-base scalar multiplication. However, it is possible to achieve significant speedups by precomputing the multiples of the base point, that is, the points $2^i\mathbf{P}$ for $0 \leqslant i \leqslant l$. The details of the right-to-left fixed-base scalar multiplication algorithm with precomputation `scalarMultR2LwPrecomp` is given in Table 5. As the access of the lookup table does not depend on the input scalar and is sequential, Algorithm `scalarMultR2LwPrecomp` is of constant time and side-channel attack resistant.

4.1 Differential addition formulas in `scalarMultR2LwPrecomp`

The point $\mathbf{R_0} = (x_0, z_0)$ of $\mathsf{BKL}_{(1,b)}$ (or $\mathbf{R_0} = (w_0, z_0)$ of BEd) is precomputed in ladder step of Table 5. We can save one field multiplication for each differential addition of line 2 and 4 of the `ladderStepR2L` if we store the values of $\mathbf{R_0}$ as $(\frac{x_0}{z_0}, 1)$ (or $(\frac{x_0}{z_0}, 1)$) for all $2^i\mathbf{P}$. Then following [46], we can modify the differential addition formula for both the $\mathsf{BKL}_{(1,b)}$ (given in Table 1) and BEd (available at [9, 10]) as given below: where $\mathbf{R_2} = (x_2, z_2)$ (or $\mathbf{R_2} = (w_2, z_2)$) and $\mathbf{R_1} = (x_1, z_1)$ (or $\mathbf{R_1} = (w_1, z_1)$).

Curve	$\texttt{diffAdd}(\mathbf{R_0}, \mathbf{R_2}, \mathbf{R_1})$
$\mathsf{BKL}_{(1,b)}$	$x_4 = z_2(x_1 \frac{x_0}{z_0} + z_1)^2$; $z_4 = x_1(x_1 + z_1 \frac{x_0}{z_0})^2$;
BEd	$A = (\frac{w_0}{z_0})w_2$; $B = (\frac{w_0}{z_0} + 1)(w_2 + z_2)$ $w_5 = z_1(d(A + B + z_2)^2)$ $z_5 = w_1(AB + dz_2)$

Table 6. Modified Differential Addition Formula for Fixed-base Right-to-Left Montgomery Ladder

5 Binary Kummer line over Field $\mathbb{F}_{2^{251}}$

Let q be an integer and $k = \mathbb{F}_{2^q}$ be a finite field of characteristic 2 with 2^q elements. We choose Kummer line $\mathsf{BKL}_{(1,b)}$ such that $b \in \mathbb{F}_{2^q} \setminus \{0\}$ and then we check whether the associated elliptic curve E_b is the one with all the required security criteria like curve and the twist of it have near prime orders, have large prime subgroups, resistance against pairing attacks and others which we discuss in details for the proposed Kummer line later. In this work, we target 128-bit security and we choose field $\mathbb{F}_{2^{251}} = \mathbb{F}_2[t]/(t^{251} + t^7 + t^4 + t^2 + 1)$ as the binary Edwards curve BEd251 [6, 9] and the binary Weierstrass curve CURVE2251 [52]. For BEd251, curve parameter is $d = t^{57} + t^{54} + t^{44} + 1$.

To find the appropriate Kummer line, the value of b was increased from 1 onwards, and then we computed the associated elliptic curve and checked the security details. In our experiment, we found that $b = t^{13} + t^9 + t^8 + t^7 + t^2 + t + 1$ is the smallest value for which the associated elliptic curve E_b of the Kummer line $\mathsf{BKL}_{(1,b)}$ satisfies the following security details:

1. Order of the curve is $4p_1$ where $p_1 = 2^{249} - \delta_1$ and $\delta_1 = 16097863035244644589\backslash\ 8362306660609333279$. Therefore, the curve order is near prime [6] with cofactor $h = 4$.
2. Order of the twist curve is $2p_2$ where $p_2 = 2^{250} + \delta_2$ and $\delta_2 = 32195726070492\backslash\ 891796724613321218666559$. Similarly, the twist curve order is near prime [6] with cofactor $h_T = 2$.
3. The largest prime subgroup has order p_1 and of size 249-bit. Therefore the curve provides approximately 124.5-bit security against discrete logarithms problem.
4. Avoiding subfields: The j-invariant $1/b^4$ is a primitive element of the field $\mathbb{F}_{2^{251}}$.
5. The discriminant of the curve is $\Delta = \big((2^{251} + 1 - 4p_1) - 4 \times 2^{251}\big)$ which is 1 (mod 4) and a square-free term. The discriminant is also divisible by the large prime $\Delta/(-7 \times 31 \times 599 \times 2207)$.
6. The multiplicative order of $2^{251} \pmod{p_1}$ and $2^{251} \pmod{p_2}$ are very large and they are respectively $\lambda = (p_1 - 1)/2$ and $\lambda_T = (p_2 - 1)/6$. Therefore, the curve is resistant against pairing attacks.
7. Similar to the BEd251, it is also resistant against GHS attack as the degree of the extension field is 251 which is a prime [6, 34].

From hereon, BKL251 denotes the $\mathsf{BKL}_{(1,b)}$ with $b = t^{13} + t^9 + t^8 + t^7 + t^2 + t + 1$ over the finite field $\mathbb{F}_{2^{251}}$. The Kummer line BKL251 also has a small base point $(t^3 + t^2, 1)$, where other two curves have large base points. Table 7 lists the comparative study of (estimates of) the sizes of the various parameters of the elliptic curve associated with the proposed Kummer line BKL251 with respect to the BEd251 and the CURVE2251. From Table 7, it can be concluded that BKL251 is as secure as BEd251 and CURVE2251.

5.1 Set of Scalars

In this work, the allowed scalars are of length 251 bits. In case of left-to-right Montgomery ladder, scalars have the form

$$2^{250} + 4 \times \{0, 1, 2, \ldots, 2^{248} - 1\}.$$

	$(\lg p_1, \lg p_2)$	(h, h_T)	(λ, λ_T)	$\lg(-\Delta)$	Base point
BEd251 [9, 6]	(249,250)	(4,2)	$(\frac{p_1-1}{2}, \frac{p_2-1}{2})$	252	-
CURVE2251 [52]	(249,250)	(4,2)	$(\frac{p_1-1}{2}, \frac{p_2-1}{6})$	253	(α_1, γ_1)
BKL251 (**this work**)	(249,250)	(4,2)	$(\frac{p_1-1}{2}, \frac{p_2-1}{6})$	253	$(\alpha_2, 1)$

$\alpha_1 =$ 0x6AD0278D8686F4BA4250B2DE565F0A373AA54D9A154ABEFACB90\
DC03501D57C,
$\gamma_1 =$ 0x50B1D29DAD5616363249F477B05A1592BA16045BE1A9F218180C5150\
ABE8573,
$\alpha_2 =$ 0xC.

Table 7. Comparison of BKL251 against BEd251 and CURVE2251

On the other hand, scalars of the right-to-left Montgomery ladder have the form

$$2^{250} + 4 \times \{1, 3, 5, \ldots, 2^{248} - 1\}.$$

We call these scalars as **clamped scalar** following the terminology of [30]. Use of clamped scalars ensures two things:

1. **Resistance to Small Subgroup Attacks.** Small subgroup attacks are effective when curves have small cofactors [33]. These attacks become infeasible if the scalars are the multiples of the cofactor. The clamped scalars, here, are all multiples of 4 where 4 is the cofactor of the curves.
2. **Constant time scalar multiplication.** The most traditional way to achieve constant time scalar multiplication is the use of Montgomery ladder. In Montgomery ladder, the ladder step iterates $(l-1)$ times where l is the bit-length of the scalar. This implies that the constant time is relative to the length of scalar. By clamping, we ensure the use of constant number of iterations of the ladder step irrespective of the choice of the scalar.
   In case of left-to-right Montgomery ladder, we always need 250 `diffAdd`s and `dbl`s. In case of right-to-left Montgomery ladder, $\mathbf{R_2} = \texttt{diffAdd}(\mathbf{R_0}, \mathbf{R_2}, \mathbf{R_1})$ (line 4 of `ladderStepR2L` in Table 5) becomes $\mathbf{R_2} = \texttt{dbl}(\mathbf{R_2})$ as $\mathbf{R_2} = \mathbf{R_0}$ and $\mathbf{R_1} = \mathbf{I}$ for all the consecutive 0's from the least significant bit. After the first 1 from the least significant bit, the ladder starts performing the operation `diffAdd`. The above mentioned scalars keep the number of `diffAdd`s and `dbl`s constant while avoiding the small subgroup attacks. Therefore during right-to-left Montgomery ladder, we need 248 `diffAdd`s and 250 `dbl`s.

Generation of Clamped Scalars. One can create a clamped scalar from a 32-byte random binary string. For left-to-right ladder, first we clear the least significant two bits (that is, set zero to bit number 0 and 1 of 0-th byte). Second, we clear the most significant 5 bits (that is, set 0 to bit number 7, 6, 5, 4, and 3 of 31-st byte). Lastly, we set the 3-rd least significant bit of 31-st byte to 1 (that is, set 1 to bit number 2 of 31-st byte).

For right-to-left ladder, we first clamp the scalar as a scalar of left-to-right ladder. Then we set the third least significant bit of the scalar as 1 (that is, bit number 2 of 0-th byte).

6 Scalar Multiplication

In this section, we explain the details of the algorithms implemented to compute scalar multiplication of BKL251 and BEd251 where each algorithm contains the explicit formulas of each ladder step.

$\texttt{BKLscalarMult}(\mathbf{P}, n)$: Input: Base Point = $(x, 1)$ $n = \{1, n_{l-2}, n_{l-3}, \ldots, n_0\}$ Output: x_n	$\texttt{BKLscalarMultR2L}(P, n)$: Input: Base Point = $(x, 1)$ $n = \{1, n_{l-2}, \ldots, n_3, 1, 0, 0\}$ Output: x_n
1. $\mathsf{sx} = x$; $\mathsf{sz} = 1$;	1. $\mathsf{r0x} = x$; $\mathsf{r0z} = 1$;
2. $\mathsf{rx} = b(\mathsf{sx} + \mathsf{sz})^4$;	2. $\mathsf{r1x} = 1$; $\mathsf{r1z} = 0$;
3. $\mathsf{rz} = \mathsf{sx}^2$;	3. For $i = 0$ to 1 do
4. $\mathfrak{pb} = 0$;	4. $t_1 = b * (\mathsf{r0x} + \mathsf{r0z})^4$;
5. For $i = (l-2)$ to 0 do	5. $\mathsf{r0z} = (\mathsf{r0x} * \mathsf{r0z})^2$;
6. $\mathfrak{b} = (\mathfrak{pb} \oplus n_i)$;	6. $\mathsf{r0x} = t_1$;
7. $\texttt{condSwapConst}(\mathsf{sx}, \mathsf{rx}, \mathfrak{b})$;	7. End For;
8. $\texttt{condSwapConst}(\mathsf{sz}, \mathsf{rz}, \mathfrak{b})$;	8. $\mathsf{r2x} = \mathsf{r0x}$; $\mathsf{r2z} = \mathsf{r0z}$;
9. $t_1 = \mathsf{sx} + \mathsf{sz}$;	9. $\mathfrak{pb} = 1$;
10. $t_2 = (t_1 * (\mathsf{rx} + \mathsf{rz}))^2$;	10. For $i = 2$ to $l-1$ do
11. $\mathsf{rz} = (\mathsf{sx} * \mathsf{rz} + \mathsf{sz} * \mathsf{rx})^2$;	11. $\mathfrak{b} = (\mathfrak{pb} \oplus n_i)$;
12. $\mathsf{rx} = t_2 + \mathsf{rz}$;	12. $\texttt{condSwapConst}(\mathsf{r1x}, \mathsf{r2x}, \mathfrak{b})$;
13. $\mathsf{rz} = x * \mathsf{rz}$;	13. $\texttt{condSwapConst}(\mathsf{r1z}, \mathsf{r2z}, \mathfrak{b})$;
14. $\mathsf{sz} = (\mathsf{sx} * \mathsf{sz})^2$;	14. $t_1 = \mathsf{r0x} + \mathsf{r0z}$;
15. $\mathsf{sx} = b * t_1^4$;	15. $t_2 = (t_1 * (\mathsf{r1x} + \mathsf{r1z}))^2$;
16. $\mathfrak{pb} = n_i$;	16. $\mathsf{r1z} = (\mathsf{r0x} * \mathsf{r1z} + \mathsf{r1x} * \mathsf{r0z})^2$;
17. End For;	17. $\mathsf{r1x} = \mathsf{r2z} * (t_2 + \mathsf{r1z})$;
18. $\texttt{condSwapConst}(\mathsf{sx}, \mathsf{rx}, n_0)$;	18. $\mathsf{r1z} = \mathsf{r1z} * \mathsf{r2x}$;
19. $\texttt{condSwapConst}(\mathsf{sz}, \mathsf{rz}, n_0)$;	19. $\mathsf{r0z} = (\mathsf{r0x} * \mathsf{r0z})^2$;
20. Return $(\mathsf{sx}/\mathsf{sz})$;	20. $\mathsf{r0x} = b * t_1^4$;
21.	21. $\mathfrak{pb} = n_i$;
22.	22. End For;
23.	23. Return $(\mathsf{r1x}/\mathsf{r1z})$;

Table 8. Algorithms `BKLscalarMul` and `BKLscalarMultR2L`

6.1 Scalar Multiplication of BKL251

The algorithm for traditional left-to-right Montgomery scalar multiplication for variable-base `BKLscalarMult` is given in Table 8. For fixed-base scalar multiplications, we precompute the point $\texttt{dbl}(\mathbf{P})$ and keep it in memory. We always consider that the z-coordinate of the input base point is 1 and the implementation is designed to take advantage of that. The total operation count of a ladder step of `BKLscalarMult` is $5[\mathsf{M}] + 5[\mathsf{S}] + 1[\mathsf{C}]$. $1[\mathsf{C}]$ refers the multiplication by Kummer line parameter b (line 15 of `BKLscalarMult`). In our implementation, the base point is small $(t^3 + t^2, 1)$ and consequently the field multiplication in line 13 of `BKLscalarMult` becomes a multiplication by constant. Therefore, the total operation count of a ladder step becomes $4[\mathsf{M}] + 5[\mathsf{S}] + 2[\mathsf{C}]$ for fixed-base scalar multiplication.

Details of the right-to-left Montgomery ladder based scalar multiplication algorithm `BKLscalarMultR2L` for variable base is also given in Table 8. The ladder step for variable-base and fixed-base are the same. The total operation count of a ladder step of `BKLscalarMultR2L` is $6[\mathsf{M}] + 5[\mathsf{S}] + 1[\mathsf{C}]$. In case of fixed-base scalar multiplication without precomputation table, we precompute $4\mathbf{P}$ and assign to $(\mathsf{r0x}, \mathsf{r0z})$ at line 1 of `BKLscalarMultR2L` and remove the computation described in lines 3–7.

On the other hand, fixed-base right-to-left Montgomery ladder based scalar multiplication with precomputation `BKLscalarMultR2LPrecompFB` is given in Table 10. We always consider that the z-coordinates of the precomputed points are one. Each ladder step requires only $4[\mathsf{M}] + 2[\mathsf{S}]$ operations.

6.2 Scalar Multiplication of BEd251

For scalar multiplication on binary Edwards curve BEd251, WZ-coordinate system is used which has minimum operation count per ladder step as suggested in [6, 9, 8]. The algorithm for scalar multiplication

BEdscalarMult(P, n) : Input: Base Point $= (w, 1)$ $n = \{1, n_{l-2}, n_{l-3}, \ldots, n_0\}$ Output: x_n	BEdscalarMultR2L(P, n) : Input: Base Point $= (w, 1)$ $n = \{1, n_{l-2}, \ldots, n_3, 1, 0, 0\}$ Output: x_n
1. $\mathsf{sw} = w$; $\mathsf{sz} = 1$;	1. $\mathsf{r0w} = w$; $\mathsf{r0z} = 1$;
2. $\mathsf{rw} = (\mathsf{sw} * (\mathsf{sw} + \mathsf{sz}))^2$;	2. $\mathsf{r1x} = 0$; $\mathsf{r1z} = 1$;
3. $\mathsf{rz} = \mathsf{rw} + d$;	3. For $i = 0$ to 1 do
4. $\mathfrak{pb} = 0$;	4. $t_1 = (\mathsf{r0w} * (\mathsf{r0w} + \mathsf{r0z}))^2$;
5. For $i = (l-2)$ to 0 do	5. $\mathsf{r0z} = \mathsf{r0w} + d * \mathsf{r0z}^4$;
6. $\mathfrak{b} = (\mathfrak{pb} \oplus n_i)$;	6. $\mathsf{r0w} = t_1$;
7. condSwapConst$(\mathsf{sw}, \mathsf{rw}, \mathfrak{b})$;	7. End For;
8. condSwapConst$(\mathsf{sz}, \mathsf{rz}, \mathfrak{b})$;	8. $\mathsf{r2w} = \mathsf{r0w}$; $\mathsf{r2z} = \mathsf{r0z}$;
9. $t_1 = \mathsf{sw} * (\mathsf{sw} + \mathsf{sz})$;	9. $\mathfrak{pb} = 1$;
10. $t_2 = t_1 * (\mathsf{rw} * (\mathsf{rw} + \mathsf{rz}))$;	10. For $i = 2$ to $l - 1$ do
11. $t_3 = t_2 + d * (\mathsf{sz} * \mathsf{rz})^2$;	11. $\mathfrak{b} = (\mathfrak{pb} \oplus n_i)$;
12. $\mathsf{sw} = t_1^2$;	12. condSwapConst$(\mathsf{r1w}, \mathsf{r2w}, \mathfrak{b})$;
13. $\mathsf{sz} = \mathsf{sw} + d * \mathsf{sz}^4$;	13. condSwapConst$(\mathsf{r1z}, \mathsf{r2z}, \mathfrak{b})$;
14. $\mathsf{rw} = w * t_3 + t_2$;	14. $t_1 = \mathsf{r0w} * (\mathsf{r0w} + \mathsf{r0z})$
15. $\mathsf{rz} = t_3$;	15. $t_2 = t_1 * (\mathsf{r1w} * (\mathsf{r1w} + \mathsf{r1z}))$;
16. $\mathfrak{pb} = n_i$;	16. $t_3 = t_2 + d * (\mathsf{r0z} * \mathsf{r1z})^2$;
17. End For;	17. $\mathsf{r0w} = t_1^2$;
18. condSwapConst$(\mathsf{sw}, \mathsf{rw}, n_0)$;	18. $\mathsf{r0z} = \mathsf{r0w} + d * \mathsf{r0z}^4$;
19. condSwapConst$(\mathsf{sz}, \mathsf{rz}, n_0)$;	19. $\mathsf{r1w} = \mathsf{r2w} * t_3 + \mathsf{r2z} * t_2$;
20. Return $(\mathsf{sw}/\mathsf{sz})$;	20. $\mathsf{r1z} = t_3 * \mathsf{r2z}$;
21.	21. $\mathfrak{pb} = n_i$;
22.	22. End For;
23.	23. Return $(\mathsf{r1x}/\mathsf{r1z})$;

Table 9. Algorithms BEdscalarMult and BEdscalarMultR2L

BKLscalarMultR2LPrecompFB$(\mathbf{P}, n)$: Input: Base Points $= (x_i, 1)$, where $x_i =$ x-coordinate of $2^i\mathbf{P}$, $\forall 0 \leqslant i \leqslant l-1$ $n = \{1, n_{l-2}, n_{l-3}, \ldots, n_3, 1, 0, 0\}$ Output: x_n	BEdscalarMultR2LPrecompFB$(\mathbf{P}, n)$: Input: Base Points $= (w_i, 1)$, where $w_i =$ w-coordinate of $2^i\mathbf{P}$, $\forall 0 \leqslant i \leqslant l-1$ $n = \{1, n_{l-2}, n_{l-3}, \ldots, n_3, 1, 0, 0\}$ Output: x_n
1. $\mathsf{r0x} = x_2$;	1. $\mathsf{r0x} = w_2$;
2. $\mathsf{r1x} = 1$; $\mathsf{r1z} = 0$;	2. $\mathsf{r1x} = 1$; $\mathsf{r1z} = 0$;
3. $\mathsf{r2x} = x_2$; $\mathsf{r2z} = 1$;	3. $\mathsf{r2x} = w_2$; $\mathsf{r2z} = 1$;
4. $\mathfrak{pb} = 1$;	4. $\mathfrak{pb} = 1$;
5. For $i = 2$ to $l - 1$ do	5. For $i = 2$ to $l - 1$ do
6. $\mathfrak{b} = (\mathfrak{pb} \oplus n_i)$;	6. $\mathfrak{b} = (\mathfrak{pb} \oplus n_i)$;
7. condSwapConst$(\mathsf{r1x}, \mathsf{r2x}, \mathfrak{b})$;	7. condSwapConst$(\mathsf{r1w}, \mathsf{r2w}, \mathfrak{b})$;
8. condSwapConst$(\mathsf{r1z}, \mathsf{r2z}, \mathfrak{b})$;	8. condSwapConst$(\mathsf{r1z}, \mathsf{r2z}, \mathfrak{b})$;
9. $t_1 = \mathsf{r2z} * (\mathsf{r0x} * \mathsf{r1x} + \mathsf{r1z})^2$;	9. $t_1 = \mathsf{r0w} * \mathsf{r1w}$;
10. $\mathsf{r1z} = \mathsf{r2x} * (\mathsf{r0x} * \mathsf{r1z} + \mathsf{r1x})^2$;	10. $t_2 = (\mathsf{r1w} + 1) * (\mathsf{r1w} + \mathsf{r1w})$;
11. $\mathsf{r1x} = t_1$;	11. $\mathsf{r1w} = \mathsf{r2z} * (d * (t_1 + t_2 + \mathsf{r1z})^2)$;
12. $\mathsf{r0x} = x_{i+1}$;	12. $\mathsf{r1z} = \mathsf{r2w} * (t_1 * t_2 + d * \mathsf{r1z}^2)$;
13. $\mathfrak{pb} = n_i$;	13. $\mathsf{r0w} = w_{i+1}$;
14. End For;	14. $\mathfrak{pb} = n_i$;
15. Return $(\mathsf{r1x}/\mathsf{r1z})$;	15. End For;
16.	16. Return $(\mathsf{r1w}/\mathsf{r1z})$;

Table 10. Algorithms BKLscalarMultR2LPrecompFB and BEdscalarMultR2LPrecompFB

BEdscalarMult is given in the Table 9 which is a left-to-right Montgomery ladder based variable-base scalar multiplication. The operation count of a ladder step of BEdscalarMult is $5[\mathsf{M}] + 4[\mathsf{S}] + 2[\mathsf{C}]$ where $[\mathsf{C}]$ is the multiplication by Edwards curve parameter d (lines 11 and 13 of BEdscalarMult). In WZ-coordinate system, $W = X + Y$. It is very hard to find a base point (x, y) on Edwards curve such that (x, y) is the generator of the largest prime subgroup and $w = x + y$ becomes small enough to be considered as a small constant. Even if we try to make x small, y becomes a random element of the field which satisfies the Equation (14), that is y becomes the roots of the quadratic Equation (17)

$$\left(1 + \frac{d}{x + x^2}\right) y^2 + \left(1 + \frac{d}{x + x^2}\right) y + d = 0. \tag{17}$$

Similar thing happens if we try to control the size of y. In our experiment we could not find such a point and it seems that the only way is to check all the points of BEd251 by brute-forced method[2]. As a result, multiplication by the base point w becomes a full field multiplication and the operation count of each ladder step of left-to-right Montgomery ladder for fixed-base remains the same as that of BEdscalarMult, that is, $5[\mathsf{M}] + 4[\mathsf{S}] + 2[\mathsf{C}]$.

Right-to-left Montgomery scalar multiplication algorithm BEdscalarMultR2L for variable base is also given in Table 9. Similar to Kummer line, the ladder steps for variable-base and fixed-base take the same amount of field operations. From [9, 11], the minimum number of required field operations of a ladder step of BKLscalarMultR2L is $7[\mathsf{M}] + 4[\mathsf{S}] + 2[\mathsf{C}]$. In case of fixed-base scalar multiplication without precomputation table, we set $(\mathsf{r0w}, \mathsf{r0z})$ to $4\mathbf{P}$ at line 1 of BKLscalarMultR2L and remove the computation described in lines 3–7.

If we use precomputation table for fixed-base right-to-left Montgomery ladder based scalar multiplication, minimum number of operations required for each ladder step is $5[\mathsf{M}] + 2[\mathsf{S}] + 2[\mathsf{C}]$ (from Table 6). Details of the algorithm is given in Table 10 and we call it BEdscalarMultR2LPrecompFB.

6.3 Operation Count Comparison

In Table 11, we provide the comparison between $\mathsf{BKL}_{(1,b)}$ and BEd with respect to minimum number of field operations required per ladder step. $\mathsf{BKL}_{(1,b)}$ takes one more $[\mathsf{S}]$ operation and one less $[\mathsf{C}]$ operation compared to BEd during left-to-right scalar multiplication for variable base. During fixed-base case, $\mathsf{BKL}_{(1,b)}$ gains advantage as it takes one less field multiplication at the cost of one multiplication by small constant. For the rest of the cases, $\mathsf{BKL}_{(1,b)}$ always takes less number of field operations than BEd and as a result $\mathsf{BKL}_{(1,b)}$ performs better. Our experimental results also support the fact and we provide the details of the implemented software in the upcoming sections.

Montgomery ladder Method		$\mathsf{BKL}_{(1,b)}$	BEd
Left-to-Right	Fixed-Base	$4[\mathsf{M}] + 5[\mathsf{S}] + 2[\mathsf{C}]$	$5[\mathsf{M}] + 4[\mathsf{S}] + 2[\mathsf{C}]$
	Var Base	$5[\mathsf{M}] + 5[\mathsf{S}] + 1[\mathsf{C}]$	$5[\mathsf{M}] + 4[\mathsf{S}] + 2[\mathsf{C}]$
Right-to-Left	Fixed-Base	$6[\mathsf{M}] + 5[\mathsf{S}] + 1[\mathsf{C}]$	$7[\mathsf{M}] + 4[\mathsf{S}] + 2[\mathsf{C}]$
	Var Base	$6[\mathsf{M}] + 5[\mathsf{S}] + 1[\mathsf{C}]$	$7[\mathsf{M}] + 4[\mathsf{S}] + 2[\mathsf{C}]$
Right-to-Left with Precomputation	Fixed-Base	$4[\mathsf{M}] + 2[\mathsf{S}]$	$5[\mathsf{M}] + 2[\mathsf{S}] + 2[\mathsf{C}]$

Table 11. Comparison of field operation counts per ladder step of $\mathsf{BKL}_{(1,b)}$ and BEd

[2] [9, 6] also do not mention anything about small base-point

7 Field Arithmetic of the Implemented Software

Efficient field arithmetic is extremely necessary to have efficient scalar multiplication. There are many efficient algorithms which are available for binary field arithmetic, but we focus only on the finite field $\mathbb{F}_{2^{251}} = \mathbb{F}_2[t]/f(t)$ where $f(t) = t^{251} + r(t)$ is a irreducible polynomial with $r(t) = (t^7 + t^4 + t^2 + 1)$. Each element $u \in \mathbb{F}_{2^{251}}$ can be represented as a polynomial of the form

$$u = u_{250}t^{250} + u_{249}t^{249} + \cdots + u_1 t + u_0, \text{ where each } u_i \in \mathbb{F}_2, \forall 0 \leqslant i \leqslant 250.$$

Element u can also be represented as binary vector of the form $(u_{250}, u_{249}, \ldots, u_1, u_0)$. This vector can be divided into ν small vectors and we call these small vectors as limbs. Assume that the least significant $\nu - 1$ limbs have length κ and then the length of the most significant limb is $\eta = 251 - \kappa \times (\nu - 1)$.

Our software explore the instruction `PCLMULQDQ` of Intel Intrinsic [26] to achieve efficient implementation. Let x and y be two 128-bit registers as `__m128i`. We represent x as a vector (x_0, x_1) where x_0 is the least significant 64 bits and x_1 is the most significant 64 bits. Similarly, we also represent y as (y_0, y_1). Instruction `PCLMULQDQ` takes two `__m128i` variables and an 8-bit integer `0x`ij (`0x` stands for hexadecimal representation), where $i, j \in \{0, 1\}$, as inputs. Let z be another `__m128i` register. The `PCLMULQDQ` outputs

$$z = (z_0, z_1) = \texttt{PCLMULQDQ}(x, y, \texttt{0x}ij) = x_i \odot_2 y_j,$$

where $(z_1 \| z_0)$ is the result of the binary multiplication of x_i and y_j, $\|$ denotes string concatenation and $\odot_2$ denotes multiplication on $\mathbb{F}_2$. Notice that `PCLMULQDQ` can only multiply two binary elements of length 64-bit. Because of this, we choose $\kappa = 64$ and consequently we have $\nu = 4$. The length of the ν_3 is $\eta = 59$ bits.

7.1 Field Element Representation

Let $\theta = t^{64} \in \mathbb{F}_2[t]$. Each element $u \in \mathbb{F}_{2^{251}}$ is represented as $u(\theta)$ as given below:

$$u(\theta) = u_0 + u_1\theta + u_2\theta^2 + u_3\theta^3.$$

We call $u(\theta)$ has proper representation if each $u_i < \theta$ for $i = 0, 1, 2$ and $u_3 < t^{59}$. In other words, $\text{len}(u_i) \leqslant 64$ for $i = 0, 1, 2$ and $\text{len}(u_3) \leqslant 59$ as $\text{len}(u_i) = \deg(u_i) + 1$ where u_i is a binary polynomial[3].

7.2 Reduction

Reduction is one of the most important and time-consuming algorithm of field arithmetic. Let $u(t) \in \mathbb{F}_2[t]$ such that $\deg(u) = 251 + i$. Then we can write $u = h(t)t^{251} + g(t)$ where $h(t), g(t) \in \mathbb{F}_2[t]/f(t)$ such that $\deg(h(t)) = i$ and $\deg(g(t)) \leqslant 250$. Then we have

$$u(t) = h(t)t^{251} + g(t) = r(t)h(t) + g(t) \pmod{f(t)}.$$

Let $u(\theta) = \sum_{i=0}^{3} u_i\theta^i$ where $\deg(u_i) \leqslant 126 (= 63 + 63)$ for $i = 0, 1, 2$ and $\deg(u_3) \leqslant 121 (= 63 + 58)$. If for any $i = 0, 1, 2$, $\deg(u_i) > 63$ and/or $\deg(u_3) > 58$ then $u(\theta)$ does not have proper representation. Following the ideas of [4, 5, 24, 30], the reduction algorithm `reduce` is given in Table 12. Notice that the returned $v(\theta)$ is of proper representation. After the For loop at line 4, $\text{len}(v_i) \leqslant 64$ for $i = 0, 1, 2$ and $\text{len}(v_3) \leqslant \max\{\text{len}(u_3), \text{len}(w_2)\} \leqslant \max\{122, 127 - 64\} = 122$. After line 5, $\text{len}(v_3) \leqslant 59$ and $\text{len}(w_3) \leqslant 63$. This implies that w_3 is a binary polynomial of maximum degree 62. As r is a polynomial of degree 7, $\deg(w_3)$ can be at most 69 after line 6 that is $\text{len}(w_3) \leqslant 70$. Then v_0 can be of length 70 bits after line 7 which is 6-bit greater than the allowed 64-bit. Line 8 takes care of this overflow from v_0. As XOR do not increase the length of the input binary strings and $\text{len}(v_1) \leqslant 64$ at the beginning of line 8, then $\text{len}(v_1)$ is still at most 64 bits after line 8. This concludes that the output $v(\theta)$ is the proper representation of $u(\theta)$.

[3] Let u be a polynomial with coefficients from $\{0, 1\}$. Then u can be represented as a string of the coefficients which is basically a binary string. $\text{len}(u)$ denotes the length of the binary string of coefficients of u and $\deg(u)$ provides the degree of the polynomial u. Let $u = t^5 + t^2 + 1$, then string of coefficients is 100101. Therefore, $\text{len}(u) = 6$ and $\deg(u) = 5$.

$v(\theta) = \texttt{reduce}(u(\theta))$ where $u(\theta) = \sum_{i=0}^{3} u_i\theta^i$:
1. $v_0 = u_0$;
2. For $i = 0$ to 2 do
3. $w_i = v_i \gg_{64}$; $v_i = \text{lsb}_{64}(v_i)$; $v_{i+1} = u_{i+1} \oplus w_i$;
4. End For;
5. $w_3 = v_3 \gg_{59}$; $v_3 = \text{lsb}_{59}(v_3)$;
6. $w_3 = w_3 \odot_2 r$, where r is the binary vector representation of $r(t)$;
7. $v_0 = v_0 \oplus w_3$;
8. $w_0 = v_0 \gg_{64}$; $v_0 = \text{lsb}_{64}(v_0)$; $v_1 = v_1 \oplus w_0$;
9. Return $v(\theta) = \sum_{i=0}^{3} v_i\theta^i$;

Table 12. Algorithm `reduce`

Remark: For the field reduction polynomial $f(t) = (t^{251} + t^7 + t^4 + t^2 + 1)$, we did not find any trinomial $g(t) = (t^k + t^j + t^i)$ for $252 \leqslant k \leqslant 512$, $1 \leqslant j \leqslant 192$ and $0 \leqslant i \leqslant j - 1$ such that $f(t)$ divides $g(t)$. Therefore, we could not use the redundant trinomials strategy [14, 19, 44] for reduction of field elements.

7.3 Addition and Subtraction

Let $u(\theta) = \sum_{i=0}^{3} u_i\theta^i$ and $v(\theta) = \sum_{i=0}^{3} v_i\theta^i$ be two elements of $\mathbb{F}_{2^{251}}$ with proper representations. Let $w(\theta) = u(\theta) + v(\theta) \in \mathbb{F}_{2^{251}}$. Addition over binary field only needs XOR operations. We compute addition algorithm `add` as $w_i = u_i \oplus v_i$ for all $i = 0, 1, 2, 3$ where $w(\theta) = \sum_{i=0}^{3} w_i\theta^i$ is also in proper representation. As binary addition operation is component-wise XOR, it is not followed by the `reduce` algorithm.

On binary field, subtraction is the same as the addition as $-1 = 2 - 1 = 1$. Therefore, we do not define subtraction separately.

7.4 Multiplication by Small Constant

Let $u(\theta) = \sum_{i=0}^{3} u_i\theta^i$ be an element of the field $\mathbb{F}_{2^{251}}$. Let $c(\theta)$ be a small constant such that $c \in \mathbb{F}_2[t]$ and $\deg(c) \leqslant 63$ and thus can be stored using one limb. We compute the multiplication of $u(\theta)$ by $c(\theta)$ as $u'(\theta) = \sum_{i=0}^{3}(c \odot_2 u_i)\theta^i$ and then apply `reduce` on $u'(\theta)$ to achieve proper representation.

7.5 Field Multiplication

Let $u(\theta) = \sum_{i=0}^{3} u_i\theta^i$ and $v(\theta) = \sum_{i=0}^{3} v_i\theta^i$ be two elements with proper representations to be multiplied. The multiplication algorithm is given in the Table 13. The function `polyMult` of $u(\theta)$ and $v(\theta)$ computes a polynomial of degree 6 in θ. Let the polynomial be $w(\theta) = \sum_{i=0}^{6} w_i\theta^i$. We apply `expandM` function on $w(\theta)$ to achieve a polynomial of 8 limbs where each limb is of at most 64-bit. The steps of the Algorithm `expandM` are also given in the Table 13. Let the expanded polynomial be $w(\theta) = \sum_{i=0}^{7} w_i\theta^i$ with $\text{len}(w_i) \leqslant 64$. We can derive Equation (19) from the output of the function `expandM` (that is Equation (18)) using the function `fold`$(w(\theta))$ as given below.

$$
\begin{aligned}
w(\theta) &= w_7\theta^7 + w_6\theta^6 + w_5\theta^5 + w_4\theta^4 + w_3\theta^3 + w_2\theta^2 + w_1\theta + w_0 && (18)\\
&= (w_3 + w_7rt^5)\theta^3 + (w_2 + w_6rt^5)\theta^2 + (w_1 + w_5rt^5)\theta + (w_0 + w_4rt^5) && (19)
\end{aligned}
$$

Notice that the `expandM` is absolutely necessary. After `polyMult` at line 1 of `mult`$(u(\theta), v(\theta))$, we have $w(\theta) = \sum_{i=0}^{6} w_i\theta^i$. In the absence of `expandM` function, consider the terms w_irt^5 for $i = 4, 5, 6$. After `polyMult`$(u(\theta), v(\theta))$, w_i is a polynomial of degree at most 126. As $\deg(r) = 7$, then w_irt^5 can be a polynomial of degree $126 + 7 + 5 = 138$ which requires 139 bits to be stored. In our implementation, we use `__m128i` registers whose capacities are 128-bit. Therefore, without `expandM`, there will be overflow.

$w(\theta) = \texttt{mult}(u(\theta), v(\theta))$:	$w(\theta) = \texttt{expandM}(w(\theta))$:
1. $w(\theta) = \texttt{polyMult}(u(\theta), v(\theta))$	1. $w_7 = 0$;
2. $w(\theta) = \texttt{expandM}(w(\theta))$;	2. For $i = 0$ to 6 do
3. $w(\theta) = \texttt{fold}(w(\theta))$;	3. $w_{i+1} = w_{i+1} \oplus (w_i \gg_{64})$; $w_i = \mathrm{lsb}_{64}(w_i)$;
4. $w(\theta) = \texttt{reduce}(w(\theta))$	4. End For;
5. Return $w(\theta) = \sum_{i=0}^{3} w_i\theta^i$;	5. Return $w(\theta) = \sum_{i=0}^{7} w_i\theta^i$;

Table 13. Algorithms `mult` and `expandM`

Computation of `polyMult`$(u(\theta), v(\theta))$. Let $u(\theta)$ and $v(\theta)$ be in proper representation with 4 limbs and let $w(\theta) = \texttt{polyMult}(u(\theta), v(\theta)) = \sum_{i=0}^{6} w_i\theta^i$. The main objective of `polyMult` is the computation of the coefficients of $w(\theta)$, that is w_i for $i = 0, 1, \ldots, 6$. We use 2-2 Karatsuba [37] method with `PCLMULQDQ` and XOR instructions to compute the coefficients of $w(\theta)$. The details are given below:

$$
\begin{aligned}
w(\theta) &= \texttt{polyMult}(u(\theta), v(\theta)) \\
&= \texttt{polyMult2}(u_1\theta + u_0, v_1\theta + v_0) + \texttt{polyMult2}(u_3\theta + u_2, v_3\theta + v_2)\theta^4 + \\
&\quad (\texttt{polyMult2}((u_1 \oplus u_3)\theta + (u_0 \oplus u_2), (v_1 \oplus v_3)\theta + (v_0 \oplus v_2)) + \\
&\quad (\texttt{polyMult2}(u_1\theta + u_0, v_1\theta + v_0) + \texttt{polyMult2}(u_3\theta + u_2, v_3\theta + v_2)))\,\theta^2
\end{aligned}
$$

We also compute `polyMult2` using Karatsuba method as

$$
\begin{aligned}
&\texttt{polyMult2}(u_1\theta + u_0, v_1\theta + v_0) \\
&= (u_1 \odot_2 v_1)\theta^2 + (((u_0 \oplus u_1) \odot_2 (v_0 \oplus v_1)) \oplus (u_0 \odot_2 v_0) \oplus (u_1 \odot_2 v_1))\,\theta \\
&+ (u_0 \odot_2 v_0)
\end{aligned}
$$

Each `polyMult2` requires 3 `PCLMULQDQ` operations and 4 XORs. Consequently, `polyMult` requires 9 `PCLMULQDQ` operations and 22 XORs.

Remark: We also implemented the hybrid method [30] and the schoolbook method. Our experiments show that Karatsuba method produces the best result (See Table 17).

Unreduced Field Multiplication (`multUnreduced`). Let $u(\theta)$ and $v(\theta)$ be two elements of $\mathbb{F}_{2^{251}}$ with proper representation, that is $u(\theta) = \sum_{i=0}^{3} u_i\theta^i$ and $v(\theta) = \sum_{i=0}^{3} v_i\theta^i$. Let $w(\theta)$ is a polynomial of the form $w(\theta) = \sum_{i=0}^{6} w_i\theta^i$. We define `multUnreduced` as

$$
w(\theta) = \texttt{multUnreduced}\,(u(\theta), v(\theta)) = \texttt{polyMult}(u(\theta), v(\theta)).
$$

`multUnreduced`$\,(u(\theta), v(\theta))$ is exactly the same as the `mult` without `expandM`, `fold` and `reduce`.

Field Addition of unreduced field elements (`addReduce`). Let $u(\theta) = \sum_{i=0}^{6} u_i\theta^i$ and $v(\theta) = \sum_{i=0}^{6} v_i\theta^i$ be results of two `multUnreduced`s, because `addReduce` is used on the outputs of two `multUnreduced` in our implementation. As addition over binary field is simply the bit-wise XOR of the inputs, it does not increase the length and thus there is no issue of overflow. The details of the algorithm of `addReduce` is given in Table 14. On the XORed value, we apply `expandM`, `fold` and `reduce` to achieve a proper representation.

7.6 Field Squaring

Field squaring is much less expensive in binary fields compared to prime fields as here squaring means relabeling the exponent of the input binary element. Let $u(\theta) = \sum_{i=0}^{3} u_i\theta^i$ be the element to be squared.

$w(\theta) = \texttt{addReduce}(u(\theta), v(\theta))$:
1. For $i = 0$ to 6 do
2. $w_i = u_i \oplus v_i$;
3. End For;
4. $w(\theta) = \texttt{expandM}(w(\theta))$;
5. $w(\theta) = \texttt{fold}(w(\theta))$;
6. $w(\theta) = \texttt{reduce}(w(\theta))$;
7. Return $w(\theta) = \sum_{i=0}^{3} w_i\theta^i$;

Table 14. Algorithms `addReduce`

The squaring algorithm is given in Table 15. The `polySq` function creates a polynomial $w(\theta) = \sum_{i=0}^{6} w_i\theta^i$ from $u(\theta)$ as given in Equation 20.

$$w_i = \begin{cases} u_{i/2}^2 = u_{i/2} \odot_2 u_{i/2}, & i = 0 \pmod 2 \\ 0, & i = 1 \pmod 2 \end{cases} \tag{20}$$

$w(\theta) = \texttt{sq}(u(\theta))$:	$w(\theta) = \texttt{expandS}(w(\theta))$:
1. $w(\theta) = \texttt{polySq}(u(\theta), v(\theta))$	1. For $i = 0, 2, 4, 6$ do
2. $w(\theta) = \texttt{expandS}(w)$;	2. $w_{i+1} = (w_i \gg_{64})$; $w_i = \text{lsb}_{64}(w_i)$;
3. $w(\theta) = \texttt{fold}(w)$;	3. End For;
4. $w(\theta) = \texttt{reduce}(w(\theta))$	4. Return $w(\theta) = \sum_{i=0}^{7} w_i\theta^i$;
5. Return $w(\theta) = \sum_{i=0}^{3} w_i\theta^i$;	

Table 15. Algorithms `sq` and `expandS`

The `expandS` is also slightly different than `expandM`. In function `expandS`, if $i = 0 \pmod 2$, then we divide the w_i in to two parts and assign the least significant 64 bits to w_i and the remaining most significant bits to w_{i+1}. If $i = 1 \pmod 2$, we do nothing. The details of the function `expandS` are also given in Table 15. On the output of `expandS`, we apply `fold` and `reduce` to compute the proper representation of the squared value.

Notice that, `polySq` only needs 4 `PCLMULQDQ` operations and no XORs which is less than half of the operation counts of `polyMult`. As a result, `sq` is significantly faster than `mult`.

Remark: Squaring also can be done by inserting zero between every consecutive bits using the help of a lookup table as proposed in [3]. But to process each 64-bit limb, it requires 2 AND, 1 Right Shift by a Byte, 2 table lookup operations, 1 INTERLO and 1 INTERHI operations. Each table lookup also depends on the input element and therefore it is not of constant time. To achieve constant time implementation, we avoid table lookup and implement field squaring operation `polySq` using only 4 PCLMULQDQ operations.

7.7 Field Inverse

We compute Kummer line scalar multiplication in projective coordinate system and receive an projective point (x_n, z_n) at the end of the iterations of ladder steps. Therefore, we compute the affine output as x_n/z_n which requires one field inversion and one field multiplication. We compute field inversion as $z_n^{-1} = z_n^{2^{251}-2}$ in constant time using 250 field squaring and 10 field multiplications following the sequence given in [6]. The multiplications produce the terms z_n^3, z_n^7, $z_n^{2^6-1}$, $z_n^{2^{12}-1}$, $z_n^{2^{24}-1}$, $z_n^{2^{25}-1}$, $z_n^{2^{50}-1}$, $z_n^{2^{100}-1}$, $z_n^{2^{125}-1}$ and $z_n^{2^{250}-1}$.

7.8 Conditional Swap

The `laddersteps` of Tables 2, 4 and 5 use conditional swap based on the input bit from the scalar. But to achieve constant time scalar multiplication, no use of branching instructions is prerequisite. Therefore, we perform the conditional swap without any branching instruction as given in Table 16. In Table 16, the algorithm `condSwap` uses branching instruction where `condSwapConst` performs the same job as the `condSwap` without any branching instruction. In computer, 0 is represented as a binary string of all zeros and -1 is represented as a binary string of all ones in 2's complement representation. Therefore, if $\mathfrak{b}$ is 0 then $w = 0$ at the end of the line 2 of Algorithm `condSwapConst` else it is $w = u_i \oplus v_i$. As a consequence, if $\mathfrak{b} = 0$, there is no change of values in u_i and v_i as $u_i = u_i \oplus 0 = u_i$ and $v_i = v_i \oplus 0 = v_i$. On the other hand, if $\mathfrak{b} = 1$, then u_i and v_i get swapped as $u_i = u_i \oplus w = u_i \oplus u_i \oplus v_i = v_i$ and $v_i = v_i \oplus w = v_i \oplus u_i \oplus v_i = u_i$.

`condSwap`$(u(\theta), v(\theta), \mathfrak{b})$	`condSwapConst`$(u(\theta), v(\theta), \mathfrak{b})$
1. If $(\mathfrak{b} = 1)$ then	1. For $i = 0$ to 4 do
2. For $i = 0$ to 4 do	2. $w = u_i \oplus v_i$; $w = w\&(-\mathfrak{b})$;
3. $w = u_i$; $u_i = v_i$; $v_i = w$;	3. $u_i = u_i \oplus w$; $v_i = v_i \oplus w$;
4. End For;	4. End For;
5. End If;	

Table 16. Algorithm Conditional Swap

8 Implementation of Batch Binary Kummer lines

In this work, we also provide an efficient software which computes variable-base and fixed-base scalar multiplications in a batch of 128. In this implementation, we use the software BBE251 available at [6] as the base implementation. As the underlying fields are the same for both the cases, we only had to modify the `fieldelement.h`, `core2.cpp` and `gates.cpp` files of the available software to make it work for the Kummer line BKL251. We also make the modified code publicly available.

9 Implementations and Timings

We have implemented the software BKL251 and BEd251 using the Intel intrinsic instructions applicable to __m128i. All the modules of the field arithmetic and the ladder step are written in assembly language to achieve the most optimized implementation. The 64×64 bit binary field multiplications are done using `pclmulqdq` instruction. We compute 128-bit bit-wise XOR and AND operations using instructions `pxor` and `pand` respectively. For byte-wise and bit-wise right-shift, we use `psrldq` and `psrlq`. We implement the scalar multiplication function with clamped scalars and Montgomery ladder algorithm with `condSwapConst` and constant-time field inverse operation. Consequently, our codes achieve constant run-time.

We use `reduce` algorithm with `mult`, `sq`, `multConst` and `addReduce`. In case of function `mult` and `sq`, the size of the limbs are at most 76 bits after `fold` operations. Therefore, the w_3 of line 5 of Algorithm `reduce` (Table 12) will be 17 bits at most and in turn w_3 becomes 24-bit after line 6. Therefore, there will be no overflow from v_0 of line 7 of Table 12. Similar thing happens for the `addReduce`.

In case of `multConst` operation in scalar multiplications `BKLscalarMult` and `BKLscalarMultR2L`, the maximum length of the constant is the length of the Kummer line parameter b which is 14-bit (where the base point is of 4 bits). Therefore, the maximum possible length of u'_3 after line 3 of Algorithm `multConst` is 72-bit. During `reduce` of `multConst`, w_3 of line 7 of Table 12 becomes 20-bit long and in turn there will be no overflow from v_0. Similarly, in scalar multiplications `BEdscalarMult` and `BEdscalarMultFB`, the maximum length of the constant is curve parameter d which is of degree 57. Therefore, the maximum possible length

of u'_3 after line 3 of Algorithm multConst is 116-bit. During reduce of multConst, w_3 of line 7 of Table 12 can be at most 64-bit long and thus there will also be no overflow from v_0.

As there will be no overflow from v_0 after line 7 of reduce in all possible cases of mult, sq, multConst and addReduce in context of BKL251 and BEd251, we further optimize the field arithmetic by removing the line 8 of reduce in Table 12 during implementation.

In the modules of field multiplications and squaring, a significant amount of time is taken by the attempt of achieving the proper representation. In other words, the operations expandM/expandS, fold and reduce, in total, take a considerable amount of time compared to polyMult/polySq. Using multUnreduced and addReduce in BKLscalarMult (line 11), BKLscalarMultR2L (line 16) and BEdscalarMultR2L (line 19), we avoid one set of expandM, fold and reduce operations at each ladder step and it produces a significant speedup. The rest of the details of the implementation can be found from the available software.

Timing experiments were carried out on a single core on the platform:

Skylake: Intel®Core™i3-6100U 2-core CPU @2.30GHz running

OS of the computer is 64-bit Ubuntu 18.04.5 LTS and the code was compiled using GCC version 8.4.0.

During timing measurements, turbo boost and hyperthreading were turned off. An initial cache warming was done with 25,000 iterations and then the median of 100,000 iterations was recorded. The Time Stamp Counter (TSC) was read from the CPU to RAX and RDX registers by RDTSC instruction.

In Table 17, we provide the timing results of left-to-right Montgomery scalar multiplications of BKL251 and BEd251 which are implemented using three different field multiplication methods: Schoolbook, Hybrid and Karatsuba. [5, 25] use schoolbook method to obtain best result over prime field for Curve25519. On the other hand, [30] uses hybrid method to produce best result for Kummer lines over prime fields. But in our work, we found that Karatsuba method implemented using PCLMULQDQ provides the best running time for binary Kummer line. Our experiments show that fixed-base scalar multiplication of BKL251 is 9.63% faster than binary Edwards curve BEd251. On the other hand, variable-base scalar multiplications on binary Kummer line and Binary Edwards curve have almost same performance, and BKL251 is 0.52% faster than BEd251.

In Table 18, we compare the performances of right-to-left Montgomery ladder scalar multiplications. It shows that the BKL251 provides the fastest result. BKL251 is 17.84% faster than BEd251 and 45.99% faster than Koblitz over $\mathbb{F}_{4^{163}}$ [44] for fixed-base scalar multiplication with precomputation. Using right-to-left Montgomery ladder with precomputation, we can achieve 15.84% faster fixed-base scalar multiplications compared to left-to-right Montgomery ladder fixed-base scalar multiplication, but we get only 7.13% speedup in case of BEd251. On the other hand, left-to-right Montgomery ladder is 7.04% faster than right-to-left Montgomery ladder for BKL251 in case of variable base scalar multiplication.

Table 19 provides a comparative study of variable-base scalar multiplications of a few curves which target at 128-bit security and have timing attack resistant implementation. Even though this work *does not* consider special fields or special algebraic properties, we include Four-$\mathbb{Q}$ curve or Koblitz curves [44] over quadratic field to the comparison for the sake of completeness[4]. BKL251 is 39.74% faster than CURVE2251 and, 23.25%, 32.92% and 0.48% faster than the NIST propsed binary curves K-283, B-283 and B-233 respectively. On the other hand, BKL251 is 11.8% slower than the K-233.

Kummer lines over prime field KL2519 and KL25519 are 11.82% and 2.1% faster than BKL251. Notice that both the implementations of KL2519 and KL25519 use SIMD parallelization where BKL251 does not use any parallelization. BKL251 is 2.08% faster than Curve25519 even though it uses SIMD parallelization.

Table 20 lists the timing and bit-operation comparisons among the batch software: BBE251, Curve2251, sect283r1 and BBK251[5]. Provided timings are scaled as (total batch computation time)/128. Performances of batch binary Kummer and Edwards are almost the same. It also shows that each variable-base scalar

[4] All the Skylake performances are measured in our experimental system. But due to publicly unavailabile code, all the Haswell performances are obtained from the referred articles.

[5] Reported details of BBE251 and BBK251 are obtained from out experimental setup. But due to unavailability of the public code, data of Curve2251 and sect283r1 are obtained from the referred article.

Field Multiplication Algorithm	BKL251		BEd251	
	Fixed-Base	Var-Base	Fixed-Base	Var-Base
School-book	83,598	91,570	90,894	91,062
Hybrid	83,398	91,484	91,422	91,586
Karatsuba	**82,062**	**90,560**	**90,812**	**91,036**

Table 17. Timings of Left-to-Right Montgomery Scalar Multiplications of BKL251 and BEd251 in clock cycles (cc)

Curve	Fixed-Base		Var-Base	
	with Precomp.	without Precomp.	with Precomp.	without Precomp.
Koblitz over $\mathbb{F}_{4^{163}}$ [44]	128,284	145,188	-	-
BKL251 (**this work**)	**69,292**	**97,170**	-	**97,416**
BEd251(**this work**)	84,334	101,922	-	102,010

Table 18. Timings of Right-to-Left Montgomery Scalar Multiplications of BKL251 and BEd251 in clock cycles (cc)

Curve	Field	Endomorphism	SIMD Parallelization	Timing (in cc)	Architecture
NIST K-233 [13]	$\mathbb{F}_{2^{233}}$	no	no	81,000	Haswell
NIST B-233 [13]	$\mathbb{F}_{2^{233}}$	no	no	91,000	Haswell
NIST K-283 [13]	$\mathbb{F}_{2^{283}}$	no	no	118,000	Haswell
NIST B-283 [13]	$\mathbb{F}_{2^{283}}$	no	no	135,000	Haswell
CURVE2251 [51]	$\mathbb{F}_{2^{251}}$	no	no	150293	Skylake
KL2519 [30, 40]	$\mathbb{F}_{2^{251}-9}$	no	yes	80,987	Skylake
KL25519 [30, 40]	$\mathbb{F}_{2^{255}-19}$	no	yes	88,678	Skylake
Curve25519 [39]	$\mathbb{F}_{2^{255}-19}$	no	yes	92,485	Skylake
Koblitz [44]	$\mathbb{F}_{4^{143}}$	yes	yes	82,872	Haswell
Koblitz [44]	$\mathbb{F}_{4^{163}}$	no	yes	105,952	Haswell
Koblitz [44]	$\mathbb{F}_{4^{163}}$	no	yes	145,188	Haswell
Four-$\mathbb{Q}$ [16, 17]	$\mathbb{F}_{(2^{127}-1)^2}$	yes	yes	50621	Skylake
Four-$\mathbb{Q}$ [16, 17]	$\mathbb{F}_{(2^{127}-1)^2}$	no	yes	89,917	Skylake
BEd251 (**this work**)	$\mathbb{F}_{2^{251}}$	no	no	**91,036**	Skylake
BKL251 (**this work**)	$\mathbb{F}_{2^{251}}$	no	no	**90,560**	Skylake

Table 19. Comparison between variable-base scalar multiplications on different curves for 128-bit security level

Curve	Fixed-Base		Var-Base	
	Timing(cc)	Bit operation	Timing(cc)	Bit operation
BBE251 [6]	-	-	260,843	44,679,665
Curve2251 [15]	106,391	-	-	-
sect283r1 [15]	218,130	-	-	-
BBK251(**this work**)	213,928	36,172,773	**260,220**	**44,634,234**

Table 20. Timings of Batch Binary Edwards, Kummer and Short Weierstrass

multiplication of BKL251 or BEd251 (from Table 19) is approximately 65% faster than one variable-base scalar multiplication (after scaling down) of batch software BBK251 or BBE251 (from Table 20).

Diffie-Hellman Key Exchange. In two-party Diffie-Hellman key exchange [18] protocol, each party has to compute two scalar multiplication: one fixed-base and one variable-base. Ignoring the communication time, the total computation time required by each party is the sum of the computation time of both the scalar multiplication. BKL251 takes $69,292 + 90,560 = 159,852$ cc to compute a shared key. On the other hand, BEd251 needs $84,334 + 91,036 = 175,370$ cc. Therefore, BKL251 is 8.85% faster than BEd251 to perform Diffie-Hellman key exchange.

10 Conclusion

This work proposes the first ever binary Kummer line, namely BKL251. It also fills a gap in the existing literature by exhibiting that Binary Kummer line based scalar multiplication offers competitive performance compared to existing proposals like K-283, B-283, K-233, B-233, BEd251 and CURVE2251 over finite field of characteristic 2 using PCLMULQDQ. Previous implementations of BEd251 and CURVE2251 focus on batch implementation using bitslicing technique. This work presents the first ever implementation of the proposed BKL251 and BEd251 using the instruction PCLMULQDQ (best to our knowledge) along with the batch binary implementation of BKL251. From the experimental results, we conclude that BKL251 is faster than all the binary curves which target at 128-bit security and have timing-attack resistant implementation without any morphism.

Acknowledgement. We thank Palash Sarkar and Kaushik Nath for their valuable comments.

References

1. D. F. Aranha, 2019. Personal Communication.
2. D. F. Aranha. Relic-toolkit, 2019. `https://github.com/relic-toolkit`.
3. D. F. Aranha, J. López, and D. Hankerson. Efficient software implementation of binary field arithmetic using vector instruction sets. In *Progress in Cryptology - LATINCRYPT*, volume 6212 of *LNCS*, pages 144–161. Springer, 2010.
4. R. M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Chapman & HallCRC, 1st edition, 2006.
5. D. J. Bernstein. Curve25519: New diffie-hellman speed records. In *Public Key Cryptography – PKC*, volume 3958 of *LNCS*, pages 207–228. Springer, 2006.
6. D. J. Bernstein. Batch binary edwards. In *Advances in Cryptology - CRYPTO*, volume 5677 of *LNCS*, pages 317–336. Springer, 2009.
7. D. J. Bernstein. Batch binary edwards, 2017. `https://binary.cr.yp.to/edwards.html`.
8. D. J. Bernstein and T. Lange. Explicit-formulas database, 2019. `https://www.hyperelliptic.org/EFD/`.
9. D. J. Bernstein, T. Lange, and R. R. Farashahi. Binary edwards curves. In *Cryptographic Hardware and Embedded Systems - CHES*, volume 5154 of *LNCS*, pages 244–265, 2008.
10. D. J. Bernstein, T. Lange, and R. R. Farashahi. Explicit-formulas database, 2008. `https://www.hyperelliptic.org/EFD/g12o/auto-edwards-wz-1.html#diffadd-dadd-2008-blr-3`.
11. D. J. Bernstein, T. Lange, and R. R. Farashahi. Explicit-formulas database, 2008. `https://www.hyperelliptic.org/EFD/g12o/auto-edwards-wz-1.html#ladder-ladd-2008-blr-1`.
12. S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls), 2006. `https://datatracker.ietf.org/doc/html/rfc4492`.
13. M. Bluhm and S. Gueron. Fast software implementation of binary elliptic curve cryptography. *J. Cryptographic Engineering*, 5(3):215–226, 2015.
14. R. P. Brent and P. Zimmermann. Algorithms for finding almost irreducible and almost primitive trinomials. In *in Primes and Misdemeanours: Lectures in Honour of the Sixtieth Birthday of Hugh Cowie Williams, Fields Institute*, page 212, 2003.

15. B. B. Brumley, S. ul Hassan, A. Shaindlin, N. Tuveri, and Kide Vuojärvi. Batch binary weierstrass. In *Advances in Cryptology - LATINCRYPT*, volume 11774 of *LNCS*, pages 364–384. Springer, 2019.
16. 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 *LNCS*, pages 214–235. Springer, 2015.
17. C. Costello and P. Longa. Fourqlib v2.0, 2021. `https://www.microsoft.com/en-us/download/details.aspx?id=52310`.
18. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions of Information Theory*, 22(6):644–654, 1976.
19. C. Doche. Redundant trinomials for finite fields of characteristic 2. In *ACISP*, volume 3574 of *LNCS*, pages 122–133, 2004.
20. P. Gaudry, 2019. Personal Communication.
21. 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.
22. P. Gaudry and E. Thomé. The mpfq library and implementing curve-based key exchanges. *SPEED: Software Performance Enhancement for Encryption and Decryption*, pages 49–64, 06 2007.
23. Pierrick Gaudry. Fast genus 2 arithmetic based on theta functions. *Journal of Mathematical Cryptology*, 1(3):243–265, 2007.
24. D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer Publishing Company, Incorporated, 1st edition, 2010.
25. H. Hisil, B. Egrice, and M. Yassi. Fast 4 way vectorized ladder for the complete set of montgomery curves, 2020. `https://eprint.iacr.org/2020/388`.
26. Intel. Intel intrinsics guide, 2019. `https://software.intel.com/sites/landingpage/IntrinsicsGuide/\#`.
27. M. Joye. Highly regular right-to-left algorithms for scalar multiplication. In *Cryptographic Hardware and Embedded Systems - CHES*, volume 4727 of *LNCS*, pages 135–147, 2007.
28. S. Karati and P. Sarkar. Kummer for genus one over prime-order fields. In *Advances in Cryptology - ASIACRYPT*, volume 10625 of *LNCS*, pages 3–32. Springer, 2017.
29. S. Karati and P. Sarkar. Connecting legendre with kummer and edwards. *Advances in Mathematics of Communications*, 13(1):41–66, 2019.
30. S. Karati and P. Sarkar. Kummer for genus one over prime-order fields. *Journal of Cryptology*, pages 1–38, 2019. `https://doi.org/10.1007/s00145-019-09320-4`.
31. N. Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, 48(177):203–209, 1987.
32. Neal Koblitz. Hyperelliptic cryptosystems. *Journal of Cryptology*, 1(3):139–150, 1989.
33. C. H. Lim and P. J. Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In *Advances in Cryptology - CRYPTO*, volume 1294 of *LNCS*, pages 249–263. Springer, 1997.
34. A. Menezes and M. Qu. Analysis of the weil descent attack of gaudry, hess and smart. In *Topics in Cryptology – CT-RSA*, volume 2020 of *LNCS*, pages 308–318. Springer, 2001.
35. V. S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology - CRYPTO*, LNCS, pages 417–426. Springer, 1985.
36. P. L. Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243â243, 1987.
37. P. L. Montgomery. Five, six, and seven-term karatsuba-like formulae. *IEEE Trans. Computers*, 54(3):362–369, 2005.
38. Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.
39. K. Nath and P. Sarkar. Efficient 4-way vectorizations of themontgomery ladder, 2020. `https://eprint.iacr.org/2020/378`.
40. K. Nath and P. Sarkar. Kummer versus montgomery face-off over prime order fields, 2021. `https://eprint.iacr.org/2021/019`.
41. Y. Nir, S. Josefsson, and M. Pegourie-Gonnard. Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls) versions 1.2 and earlier, 2018. `https://datatracker.ietf.org/doc/html/rfc8422`.
42. NIST. Fips pub 186-4: Digital signature standard (dss), 2013. `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`.
43. K. Okeya and K. Sakurai. Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the y-coordinate on a montgomery-form elliptic curve. In *Cryptographic Hardware and Embedded Systems - CHES*, volume 2162 of *LNCS*, pages 126–141, 2001.

44. T. Oliveira, J. L. Hernandez, Daniel Cervantes-Vázquez, and F. Rodríguez-Henríquez. Koblitz curves over quadratic fields. *Journal of Cryptology*, 32(3):867–894, 2019.
45. T. Oliveira, J. L. Hernandez, and F. Rodríguez-Henríquez. The montgomery ladder on binary elliptic curves. *J. Cryptographic Engineering*, 8(3):241–258, 2018.
46. T. Oliveira, J. López, H. Hisil, A. Faz-Hernández, and F. Rodríguez-Henríquez. A note on how to (pre-)compute a ladder, 2017. `https://eprint.iacr.org/2017/264`.
47. Thomaz Oliveira, D. F. Aranha, J. Lopez, and F. Rodríguez-Henríquez. Fast point multiplication algorithms for binary elliptic curves with and without precomputation. In *Selected Areas in Cryptography – (SAC)*, volume 8781 of *LNCS*, pages 324–344, 2014.
48. J. Renes and B. Smith. qDSA: Small and secure digital signatures with curve-based Diffie-Hellman. In *Advances in Cryptology - ASIACRYPT*, volume 10625 of *LNCS*, pages 273–302. Springer, 2017.
49. E. Rescorla. The transport layer security (tls) protocol version 1.3 – rfc8446, 2018. `https://tools.ietf.org/html/rfc8446`.
50. J. Salowey. Confirming consensus on removing rsa key transport from tls 1.3, 2014. `https://mailarchive.ietf.org/arch/msg/tls/f7WVUwsTe5ACGhIPxXe3BSlvI3M/`.
51. J. Taverne, A. Faz-Hernández, D. F. Aranha, F. Rodríguez-Henríquez, D. Hankerson, and J. López Hernandez. Software implementation of binary elliptic curves: Impact of the carry-less multiplier on scalar multiplication. In *Cryptographic Hardware and Embedded Systems – CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 108–123. Springer, 2011.
52. J. Taverne, A. Faz-Hernández, D. F. Aranha, F. Rodríguez-Henríquez, D. Hankerson, and J. López Hernandez. Speeding scalar multiplication over binary elliptic curves using the new carry-less multiplication instruction. *J. Cryptographic Engineering*, 1(3):187–199, 2011.
53. The PARI Group, University of Bordeaux. *PARI/GP version* `2.7.5`, 2018. `http://pari.math.u-bordeaux.fr/`.