Security and Efficiency Trade-offs for Elliptic Curve Diffie-Hellman at the 128-bit and 224-bit Security Levels

Kaushik Nath and Palash Sarkar

Applied Statistics Unit
Indian Statistical Institute
203, B. T. Road
Kolkata - 700108
India
{kaushikn_r,palash}@isical.ac.in

Abstract

Within the Transport Layer Security (TLS) Protocol Version 1.3, RFC 7748 specifies elliptic curves targeted at the 128-bit and the 224-bit security levels. For the 128-bit security level, the Montgomery curve Curve25519 and its birationally equivalent twisted Edwards curve Ed25519 are specified; for the 224-bit security level, the Montgomery curve Curve448, the Edwards curve Edwards448 (which is isogenous to Curve448) and another Edwards curve which is birationally equivalent to Curve448 are specified. Our first contribution is to provide the presently best known 64-bit assembly implementations of Diffie-Hellman shared secret computation using Curve25519. The main contribution of this work is to propose new pairs of Montgomery-Edwards curves at the 128-bit and the 224-bit security levels. The new curves are *nice* in the sense that they have very small curve coefficients and base points. Compared to the curves in RFC 7748, the new curves lose two bits of security. The gain is improved efficiency. For Intel processors, we have made different types of implementations of the Diffie-Hellman shared secret computation using the new curves. The new curve at the 128-bit level is faster than Curve25519 for all types of implementations, while the new curve at the 224-bit level is faster than Curve448 using 64-bit sequential implementation using schoolbook multiplication, but is slower than Curve448 for vectorized implementation using Karatsuba multiplication. Overall, the new curves provide good alternatives to Curve25519 and Curve448.

Keywords: Elliptic curve cryptography, Montgomery form, Edwards form, Transport layer security, Diffie-Hellman Protocol, Curve25519, Curve448

1 Introduction

Elliptic curves were independently introduced in cryptography by Koblitz [24] and Miller [27]. Since their introduction, a large literature has developed around the theory and application of elliptic curves in cryptography. Presently, elliptic curve cryptography is widely used in practical systems. Several standards and proposals have been put forward by a number of influential organizations [10, 14, 36, 38].

The Transport Layer Security (TLS) Protocol, Version 1.3 [37] has been proposed by the Internet Engineering Task Force. This includes RFC 7748 [25] which specifies certain elliptic curves. The document specifies Montgomery form curves and their birationally equivalent Edwards form curves.

Given a prime p, a parameter $A \in \mathbb{F}_p \setminus \{-2, 2\}$ defines the Montgomery curve $E_{M,A,1} : y^2 = x^3 + Ax^2 + x$. Similarly, a parameter $d \in \mathbb{F}_p \setminus \{0, 1, -1\}$ defines the Edwards curve $E_{E,1,d} : u^2 + v^2 = 1 + du^2v^2$ or the twisted Edwards curve $E_{E,-1,d} : -u^2 + v^2 = 1 + du^2v^2$.

We introduce notation for denoting primes and curves. The prime $2^{251} - 9$ will be denoted as $p251\text{-}9$, $2^{255} - 19$ will be denoted as $p255\text{-}19$, $2^{444} - 17$ will be denoted as $p444\text{-}17$, and $2^{448} - 2^{224} - 1$ will be denoted as $p448\text{-}224\text{-}1$. A Montgomery curve $E_{M,A,1}$ will be denoted as $\mathrm{M}[\![A]\!]$, an Edwards curve $E_{E,1,d}$ will be denoted as $\mathrm{E}[\![d]\!]$, and a twisted Edwards curve $\mathrm{E}_{E,-1,d}$ will be denoted as $\widetilde{\mathrm{E}}[\![d]\!]$. If we wish to emphasize the underlying field $\mathbb{F}_p$, we will write $\mathrm{M}[\![p, A]\!]$, $\mathrm{E}[\![p, d]\!]$ and $\widetilde{\mathrm{E}}[\![p, d]\!]$ instead of $\mathrm{M}[\![A]\!]$, $\mathrm{E}[\![d]\!]$ and $\widetilde{\mathrm{E}}[\![d]\!]$ respectively. In terms of this naming convention, the parameters of the new curves and those in RFC 7748 are shown in Table 1.

Curves proposed in RFC 7748:

Over p255-19: The birationally equivalent pair $(M[\![486662]\!], \widetilde{\mathrm{E}}[\![121665/121666]\!])$ has been proposed. The curve M$[\![486662]\!]$ is the famous Curve25519 and was introduced in [3]. The curve $\widetilde{\mathrm{E}}[\![121665/121666]\!]$ is the famous Ed25519 curve and was introduced in [6].

Over p448-224-1: The curves M$[\![156326]\!]$ E$[\![39082/39081]\!]$ and E$[\![39081]\!]$ have been proposed. The curve M$[\![156326]\!]$ has been named Curve448 in [25]. The curve E$[\![39081]\!]$ was proposed in [18] where it was named Ed448-Goldilocks, and in [25] it has been called Edwards448. The isogenies between M$[\![156326]\!]$ and E$[\![39081]\!]$ and the birational equivalence between M$[\![156326]\!]$ and E$[\![39082/39081]\!]$ have been identified in [25].

Curve25519 and Ed25519 are targeted at the 128-bit security level while Curve448 and Edwards448 are targeted at the 224-bit security level.

Our Contributions

Our first contribution is to provide new and the presently most efficient 64-bit implementations of Diffie-Hellman shared secret computations using Curve25519. These implementations are targeted at Intel processors. Compared to the previous best implementations [33], on the Skylake processor, the new implementations provide about 17% speed-up compared to [33]; for the previous generation Haswell processor, the improvement is nominal. The major speed improvement in Skylake makes the new implementation attractive for practical applications.

The main contribution of the paper is to propose new curves at the 128-bit and 224-bit security levels. Similar to Curve25519 and Curve448, we consider prime order fields and pairs of birationally equivalent curves. In particular the following pairs of curves are introduced.

Over p251-9*:* $(M[\![4698]\!], E[\![1175/1174]\!])$.

Over p444-17*:* $(M[\![4058]\!], E[\![1015/1014]\!])$.

The prime p251-9 was considered in [7] where the curve $u^2 + v^2 = 1 - 1174u^2v^2$ was introduced and named Curve1174. The Montgomery curve $(4/1175)y^2 = x^3 + (4/1175 - 2)x^2 + x$ with base point $(4, \cdot)$ was considered as birationally equivalent to Curve1174; the corresponding base point on Curve1174 is $(\cdot, 3/5)$. Using the isogenies given in [12], it can be shown that M$[\![p251\text{-}9, 4698]\!]$ is 4-isogenous to Curve1174 which was introduced in [7].

To the best of our knowledge, neither M$[\![p251\text{-}9, 4698]\!]$ nor E$[\![p251\text{-}9, 1175/1174]\!]$ has been earlier considered in the literature. The prime p444-17 has been mentioned in a CFRG mailing list [26], but, neither the curve M$[\![p444\text{-}17, 4058]\!]$ nor E$[\![p444\text{-}17, 1015/1014]\!]$ have been considered in the literature.

Sec Level	Prime	Mont	(h, h_T)	(k, k_T)	Security	Mont Base Pt	Ed	Ed Base Pt
≈ 128	p251-9	M$[\![4698]\!]$	$(4, 4)$	$(\ell - 1, \frac{\ell_T - 1}{2})$	124.5	$(3, \cdot)$	E$[\![\frac{1175}{1174}]\!]$	$(\cdot, 2)$
	p255-19	M$[\![486662]\!]$	$(8, 4)$	$(\frac{\ell-1}{6}, \frac{\ell_T - 1}{2})$	126	$(9, \cdot)$	$\widetilde{\mathrm{E}}[\![\frac{121665}{121666}]\!]$	$(\cdot, 4/5)$
≈ 224	p444-17	M$[\![4058]\!]$	$(4, 4)$	$(\frac{\ell-1}{3}, \ell_T - 1)$	221	$(3, \cdot)$	E$[\![\frac{1015}{1014}]\!]$	$(\cdot, 2)$
	p448-224-1	M$[\![156326]\!]$	$(4, 4)$	$(\frac{\ell-1}{2}, \frac{\ell_T - 1}{4})$	223	$(5, \cdot)$	E$[\![\frac{39082}{39081}]\!]$	$(\cdot, -3/2)$

Table 1: Parameters of curves. See Section 2.3 for the definition of the parameters.

Table 1 compares the parameters of the newly proposed curves with those in RFC 7748. Note that the curve coefficients of the new curves are quite small. Also, the fixed base points for the new Montgomery and Edwards curve are also very small. In fact, the fixed base point over both the new Edwards curves is $(\cdot, 2)$. As we explain later, this has an effect on the speed of fixed base point scalar multiplication over such curves.

For 64-bit implementations of the Montgomery ladder, scalar multiplication over the new curves is faster than that over Curve25519 and Curve448. The improvement arises due to working with a slightly smaller prime. Suppose $m = \lceil \log_2 p \rceil$ and elements of $\mathbb{F}_p$ are represented using κ 64-bit words. We show that if $64\kappa - m \geq 3$, then it is possible to omit performing the reduction step on the outputs of all the addition/subtraction operations in the ladder step. This is the major reason for obtaining faster ladder computation modulo p251-9 compared to p255-19 and for obtaining faster ladder computation modulo p444-17 compared to p448-224-1. Using 64-bit implementations, for both the 128-bit and 224-bit security levels, compared to Curve25519 and Curve448, the new curves provide about 10% speed-up on the Haswell and the Skylake processors.

Recently, vectorization strategies for the Montgomery ladder have been proposed [20, 32]. We have made vectorized implementations for the new curves. For such implementations, the above mentioned advantage no longer applies. At the 128-bit level, M⟦4698⟧ provides about 4% speed-up over Curve25519. The situation for the 224-bit level is different. For 4-way vectorized implementation using 256-bit registers, the representation of field elements has 16 limbs. This makes Karatsuba multiplication more efficient than schoolbook multiplication. Since p448-224-1 is particularly efficient for Karatsuba multiplication, the vectorized implementation of Curve448 turns out to be faster than that of M⟦4058⟧. We note that future availability of wider vector operations would reduce the number of limbs and would possibly lead to schoolbook becoming faster than Karatsuba and consequently, M⟦4058⟧ being faster than Curve448.

To summarize, at the 128-bit security level, M⟦4698⟧ provides speed-up over Curve25519 for all the types of implementations that we have considered. At the 224-bit security level, M⟦4058⟧ is faster than Curve448 when schoolbook is faster than Karatsuba. So overall, the new curves provide efficient alternatives to Curve25519 and Curve448.
The assembly codes of all our implementations are available at the following links.

https://github.com/kn-cs/nice-curves
https://github.com/kn-cs/x25519
https://github.com/kn-cs/x448/tree/master/8limb

Related Works

In this work, we consider elliptic curves over large prime order fields. We note that elliptic curves over composite order fields have been proposed in the literature [11, 19]. Cryptography over hyper-elliptic curves was proposed by Koblitz [19] and there have been concrete proposals for cryptography in genus 2 [1, 9, 17]. For the same security level, computations over these proposals are faster than over genus one prime order field curves. On the other hand, the security perception for composite order fields and genus two curves is different from that of elliptic curves over prime order fields. It is perhaps due to this perception issue that elliptic curves over prime order fields remain to be of primary interest.

Variable base scalar multiplication over Kummer lines associated with Legendre form elliptic curves have been proposed in the literature [16]. These have very efficient vectorized implementations [22]. So, if applications are targeted primarily for vector implementations, then the curves proposed in [22] will be the primary choice. On the other hand, for non-vectorized implementations, Montgomery curves will be faster.

2 Montgomery and (Twisted) Edwards Form Elliptic Curves

We consider elliptic curves over a field $\mathbb{F}_p$ where p is a prime.

In general, the Montgomery form elliptic curve $E_{M,A,B}$ is given by the equation $E_{M,A,B} : By^2 = x^3 + Ax^2 + x$ with $A \in \mathbb{F}_p \setminus \{-2, 2\}$ and $B \in \mathbb{F}_p \setminus \{0\}$. In general, the twisted Edwards form elliptic curve $E_{E,a,d}$ is given by the equation $E_{E,a,d} : au^2 + v^2 = 1 + du^2v^2$ with $a, d \in \mathbb{F}_p \setminus \{0\}$ and $a \neq d$. If $a = 1$, then the corresponding curve is simply called an Edwards form curve (instead of twisted Edwards form curve). If a is a square and d is not a square in $\mathbb{F}_p$, then the addition formula in $E_{E,a,d}$ is complete [4]. In this case, $E_{E,a,d}$ is called a complete twisted Edwards curve. For further details about Montgomery curves, we refer to [8, 13, 28] and for (twisted) Edwards curves, we refer to [2, 4, 15].

In the following discussion, a full field multiplication (resp. squaring) in $\mathbb{F}_p$ will be denoted as [M] (resp. [S]); if one of the multiplicands is a constant, the resulting multiplication will be denoted as [C].

2.1 Addition on Complete (Twisted) Edwards Curves

Following [21], the extended affine coordinate system is (u, v, t) with $t = uv$. The projective version of this coordinate system is (U, V, T, W) where $u = U/W$, $v = V/W$ and $t = T/W$. Suppose, it is required to add $(U_1 : V_1 : T_1 : W_1)$ and $(U_2 : V_2 : T_2 : W_2)$ to obtain $(U_3 : V_3 : T_3 : W_3)$. The formulas for U_3, V_3, T_3 and W_3 are as follows [21].

$$\left.\begin{array}{rcl} U_3 &=& (U_1V_2 + V_1U_2)(W_1W_2 - dT_1T_2) \\ V_3 &=& (V_1V_2 - aU_1U_2)(W_1W_2 + dT_1T_2) \\ T_3 &=& (U_1V_2 + V_1U_2)(V_1V_2 - aU_1U_2) \\ W_3 &=& (W_1W_2 + dT_1T_2)(W_1W_2 - dT_1T_2). \end{array}\right\} \quad (1)$$

1. Computing V_1V_2 and U_1U_2 and then computing $U_1V_2 + V_1U_2$ as $(U_1 + V_1)(U_2 + V_2) - (U_1U_2 + V_1V_2)$ leads to an algorithm for computing U_3, V_3, T_3 and W_3 using 9[M]+2[C] operations, where the

multiplications by the two constants are by a and d. If $a = 1$, then the number of operations is 9[M]+1[C].

2. If $a = -1$, then by first computing $\alpha = (V_1 + U_1)(V_2 + U_2)$, $\beta = (V_1 - U_1)(V_2 - U_2)$ and then computing $2(V_1V_2 + U_1U_2) = \alpha + \beta$ and $2(V_1U_2 + U_1V_2) = \alpha - \beta$, the number of operations can be brought down to 8[M]+1[C] [21], where 1[C] corresponds to a multiplication by d. The relevant formula becomes the following.

$$\left.\begin{array}{rcl} 4U_3 & = & 2(U_1V_2 + V_1U_2)(2W_1W_2 - 2dT_1T_2) = (\alpha - \beta)(2W_1W_2 - 2dT_1T_2) \\ 4V_3 & = & 2(V_1V_2 + U_1U_2)(2W_1W_2 + 2dT_1T_2) = (\alpha + \beta)(2W_1W_2 + 2dT_1T_2) \\ 4T_3 & = & 2(U_1V_2 + V_1U_2)2(V_1V_2 + U_1U_2) = (\alpha - \beta)(\alpha + \beta) \\ 4W_3 & = & (2W_1W_2 + 2dT_1T_2)(2W_1W_2 - 2dT_1T_2). \end{array}\right\} \quad (2)$$

If $W_1 = 1$, the number of operations required is 7[M]+1[C] [21].

3. For $a = -1$, suppose $(U_1 : V_1 : T_1 : W_1)$ is a fixed base point with $W_1 = 1$. By pre-computing and storing $(V_1 - U_1, V_1 + U_1, 2dT_1)$ the number of operations can be brought down to 7[M] [6]. The multiplication by d becomes part of the pre-computed quantity $2dT_1$. In this formula, since the multiplication by d is part of the pre-computed quantity $2dT_1$, the efficiency of the computation is not affected by whether d is small or large. Also, the efficiency of the computation is not affected by whether V_1 (or U_1) is small or large.

Consider (1) for $a = 1$ and suppose $(U_1 : V_1 : T_1 : W_1)$ is a fixed base point where $W_1 = 1$. Further suppose that V_1 is small and $U_1 + V_1$ and dT_1 are pre-computed and stored as part of $(U_1, V_1, U_1 + V_1, dT_1)$. In (1), by directly computing U_1V_2, V_1U_2, U_1U_2 and V_1V_2, $(dT_1)T_2$ along with the other four multiplications, the formulas in (1) can be computed using 7[M]+2[C], where 2[C] counts the multiplications V_1U_2 and V_1V_2. The efficiency of the computation following this strategy is not affected by whether d is small or large. For the curves that we introduce, V_1 is equal to 2 as can be seen from Table 1. So, for fixed base multiplication, the difference in the cost between $a = -1$ and $a = 1$ is essentially two multiplications by very small constants.

For dedicated (not unified) addition in $\widetilde{\mathrm{E}}[\![p, d]\!]$, it has been shown in [21] that 8[M] operations are sufficient without the assumption that $(U_1 : V_1 : T_1 : W_1)$ is a fixed base point. The corresponding formulas do not involve d. Further, Section 4.3 of [21] shows how to perform efficient scalar multiplication using fast formulas for dedicated addition and dedicated doubling that do not involve d. The resulting scalar multiplication is not necessarily constant time and can be used only when the scalars are not secret.

Summary:

Role of d: For the fastest formulas, the size of d does not play a role.

- For fixed base point scalar multiplication, the fastest complete addition formulas over both $\mathrm{E}[\![d]\!]$ and $\widetilde{\mathrm{E}}[\![d]\!]$ do not depend on the size of d.
- For scalar multiplication with non-secret scalars, the fastest formulas do not involve d.

Size of fixed base point:

- For $\widetilde{\mathrm{E}}[\![p, d]\!]$, the fastest formula for complete and unified addition does not depend on the size of any of the components of the fixed base point. The number of operations required is 7[M].
- For $\mathrm{E}[\![p, d]\!]$, the fastest formula for complete and unified addition is achieved when V_1 is small. The number of operations required is 7[M]+2[C], where 2[C] counts two multiplications by very small constants. In particular, for both $\mathrm{E}[\![p251\text{-}9, \frac{1175}{1174}]\!]$ and $\mathrm{E}[\![p444\text{-}17, \frac{1015}{1014}]\!]$, $(\cdot, 2)$ is a base point. So, the multiplication by constant is the operation of multiplying an element of $\mathbb{F}_p$ by 2.

2.2 Birational Equivalences between Montgomery and Edwards Curves

Consider the curves $\mathrm{M}[\![A]\!]$ and $\mathrm{E}[\![d]\!]$ over a field $\mathbb{F}_p$ with $p \equiv 3 \bmod 4$. If $A - 2$ is a square in $\mathbb{F}_p$, then the map

$$(x, y) \quad \mapsto (u, v) \quad = \left(\frac{\delta x}{y}, \frac{x + 1}{x - 1}\right), \quad (3)$$

where $\delta^2 = (A-2)$, is a birational equivalence from $\mathrm{M}[\![A]\!]$ to $\mathrm{E}[\![d]\!]$ with exceptional points $y = 0$ and $x = 1$. Conversely, the map

$$(u, v) \mapsto (x, y) = \left(\frac{v+1}{v-1}, \frac{\delta(v+1)}{u(v-1)}\right), \tag{4}$$

is a birational equivalence from $\mathrm{E}[\![d]\!]$ to $\mathrm{M}[\![A]\!]$ with exceptional points $u = 0$ and $v = 1$. The relation between A and d is $(A-2)/4 = 1/(d-1)$. The above birational equivalences can be obtained using the elementary birational equivalences in [2, 4]. On the other hand, verification of these birational equivalences can be done by direct substitution.

2.3 Security Properties

Let E be an elliptic curve over $\mathbb{F}_p$, where p is a prime.

Let $n = \#E(\mathbb{F}_p)$ and $n_T = 2(p+1) - \#E(\mathbb{F}_p)$, i.e., n and n_T are the orders of $E(\mathbb{F}_p)$ and its twist. Let ℓ (resp. ℓ_T) be a prime such that $n = h \cdot \ell$ (resp. $n_T = h_T \cdot \ell_T$). Cryptography is done over a subgroup of $E(\mathbb{F}_p)$ of size ℓ. The parameters h and h_T are the co-factors of $E(\mathbb{F}_p)$ and its twist respectively.

For a Montgomery curve, the curve order n is a multiple of 4. Using this fact along with $n + n_T = 2(p+1)$, it is easy to argue that if $p \equiv 3 \bmod 4$, then the minimum value of (h, h_T) is $(4, 4)$, while if $p \equiv 1 \bmod 4$, then the minimum value of (h, h_T) is either $(8, 4)$ or $(4, 8)$.

Let k (resp. k_T) be the smallest positive integer such that $\ell | p^k - 1$ (resp. $\ell_T | p^{k_T} - 1$). The parameters k and k_T are the embedding degrees of the curve and its twist respectively.

The complex multiplication field discriminant D of E is defined in the following manner. Let $t = p+1-n$. 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.

SafeCurves[1], recommend all of ℓ, ℓ_T, k, k_T and D to be large. In particular, we are interested in curves for which (h, h_T) has the optimal value.

By security of a curve in terms of bits we will mean the value of the expression $\frac{1}{2}\min(\log_2 \ell, \log_2 \ell_T)$.

3 Concrete Curves

The parameters of the new curves are given below.

Curves over $\mathbb{F}_{2^{251}-9}$: Let $p = 2^{251} - 9 \equiv 3 \bmod 4$. The minimum positive value of A for which the curve $\mathrm{M}[\![p251\text{-}9, A]\!]$ attains the optimal value of (h, h_T) is $A = 4698$. We have that $A - 2$ is a square in $\mathbb{F}_p$. Using the birational equivalences given by (3) and (4), we obtain the pair $(M[\![4698]\!], E[\![1175/1174]\!])$ of birationally equivalent curves. The quantity $1175/1174$ is a non-square modulo $p251\text{-}9$ and so the addition formula over $E_{E,1,1175/1174}$ is complete. The parameters for $\mathrm{M}[\![p251\text{-}9, 4698]\!]$ are as follows.

$$\begin{aligned}
n &= 3618502788666131106986593281521497120369356141117981896093957047094571902404,\\
\ell &= 904625697166532776746648320380374280092339035279495474023489261773642975601,\\
\log_2 \ell &= 249,\\
h &= 4,\\
k &= \ell - 1,\\
n_T &= 3618502788666131106986593281521497120460017900484553356372141953399998700076,\\
\ell_T &= 904625697166532776746648320380374280115004475121138339093035488349999675019,\\
\log_2 \ell_T &= 249,\\
h_T &= 4,\\
k_T &= (\ell_T - 1)/2,\\
D &= -124191225018039974503432777870156724737999714622904784214776464009459350500 60,\\
\lceil \log_2(-D) \rceil &= 253.
\end{aligned}$$

The point $(\cdot, 2)$ is a point of order ℓ on $E_{E,1,1175/1174}$; the corresponding point on $E_{M,4698,1}$ is $(3, \cdot)$.

The set of scalars for $E_{M,4698,1}$ is set to be $4(2^{248} + \{0, 1, \ldots, 2^{248} - 1\})$. Given a 32-byte scalar a, the clamping function $\mathsf{clamp}(a)$ is defined as follows (assuming that the first byte is the least significant byte of a): clear bits 0 and 1 of the first byte; set bit number 2 of the last byte and clear bits numbered 3 to 7 of the last byte.

[1] https://safecurves.cr.yp.to/disc.html, accessed on September 8, 2019.

Curves over $\mathbb{F}_{2^{444}-17}$: Let $p = 2^{444} - 17 \equiv 3 \bmod 4$. The minimum positive value of A for which the curve M$[\![p444\text{-}17, A]\!]$ attains the optimal value of (h, h_T) is $A = 4058$. We have that $A - 2$ is a square in $\mathbb{F}_p$. Using the birational equivalences given by (3) and (4), we obtain the pair $(M[\![4058]\!], E[\![1015/1014]\!])$ of birationally equivalent curves. The quantity $1015/1014$ is a non-square modulo $p444\text{-}17$ and so the addition formula over $E_{E,1,1015/1014}$ is complete. The parameters for M$[\![p444\text{-}17, 4058]\!]$ are as follows.

$$
\begin{aligned}
n &= 4542742026847543065933273799300028339710258504295737876759313744878847882210 99\backslash \\
&\quad 9488778472332545777485712520414512636105020181018664945 2, \\
\ell &= 1135685506711885766483318449825007084927564626073934469189828436219711970552 74\backslash \\
&\quad 9872194618083136444371428130103628159026255045254666236 3, \\
\log_2 \ell &= 442, \\
h &= 4, \\
k &= (\ell - 1)/3, \\
n_T &= 4542742026847543065933273799300028339710258504295737876759313744879143219206 47\backslash \\
&\quad 4552798915801376267083797011105565691119149001501692734 8, \\
\ell_T &= 1135685506711885766483318449825007084927564626073934469189828436219785804801 61\backslash \\
&\quad 8638199728950344066770949252776391422779787250375423183 7, \\
\log_2 \ell_T &= 442, \\
h_T &= 4, \\
k_T &= (\ell_T - 1), \\
D &= -1795290825514957729951691737953552054640775333171791787428768650737402994958 9\backslash \\
&\quad 6775961634341986909858530888358546664503937673912260606892, \\
\lceil \log_2(-D) \rceil &= 446.
\end{aligned}
$$

The point $(\cdot, 2)$ is a point of order ℓ for $E_{E,1,1015/1014}$; the corresponding point on $E_{M,4058,1}$ is $(3, \cdot)$.

The set of scalars is set to be $4(2^{441} + \{0, 1, \ldots, 2^{441} - 1\})$. Given a 56-byte scalar a, the clamping function $\mathsf{clamp}(a)$ is defined as follows (assuming that the first byte is the least significant byte of a): clear bits 0 and 1 of the first byte; set bit number 3 of the last byte and clear bits numbered 4 to 7 of the last byte.

Remark. Using the isogenies given in [12], it can be shown that M$[\![p444\text{-}17, 4058]\!]$ is 4-isogenous to E$[\![p444\text{-}17, -1014]\!]$. Also, it has been mentioned earlier that M$[\![p251\text{-}9, 4698]\!]$ is 4-isogenous to Curve1174. Connecting Montgomery and Edwards using these isogenies can be a problem, since a small base point on one of these curves does not translate to a small base point on the other.

4 Implementation Details

We consider the Montgomery form curve. Let P be a generator of the prime order cyclic subgroup of the elliptic curve over which cryptography is to be done. For all the curves considered in this work, the point P can be chosen such that its x-coordinate is small. Such a fixed point is called the base point of the corresponding curve. The base points of the curves considered in this paper are given in Table 1. For a point $Q \in \langle P \rangle$ and a non-negative integer a which is less than the order of P, the task of computing the a-fold product aQ is called scalar multiplication. In the case, $Q = P$, we will call the operation aP to be fixed base scalar multiplication, while when Q is an arbitrary element of $\langle P \rangle$, we will call the operation aQ to be variable base scalar multiplication. In the Diffie-Hellman protocol, variable base scalar multiplication is required for the shared secret phase, while fixed base scalar multiplication is required for the key generation phase. Our primary focus will be variable base scalar multiplication for the shared secret phase.

The Montgomery ladder [28] is an x-coordinate only algorithm which can be used to compute the x-coordinate of the result of a scalar multiplication. The ladder computation is performed using projective coordinates and at the end, the result is converted to affine coordinates. When the scalar a in the scalar multiplication aQ is a secret, for secure computation, it is important that the computation be implemented in constant time. There are known ways to implement the Montgomery ladder in constant time. A detailed treatment of the Montgomery ladder has been addressed in [8, 13]. For an overview on the constant time sequential algorithm of the ladder we refer to [31], and for 4-way vectorized algorithms we refer to [20, 32].

Intel processors provide two kinds of 64-bit integer multiplication operations, namely `mul` and `mulx`, where `mul` modifies both the carry and overflow flags, but `mulx` does not modify either of these flags. The

`add` and `adc` instructions perform addition and addition-with-carry using the carry flag respectively and modifies both the carry and the overflow flags; the instruction `adcx` performs addition-with-carry using the carry flag, but does not modify the overflow flag, while the instruction `adox` performs addition-with-carry using the overflow flag, but does not modify the carry flag. By `maa` we will denote implementations which use only the `mul`, `add` and `adc` instructions and not any of `mulx`, `adcx` or `adox`; `mxaa` will denote implementations which use `mulx`, `add` and `adc` instructions; while `maax` will denote implementations which use `mulx`, `adcx` and `adox`. The `maa` type implementations are supported across a wide range of Intel processors, the `mxaa` type implementations are supported from the Haswell processor onwards, while the `maax` type instructions are supported on modern generation Intel processors such as Skylake.

The previously best known `maa`, `mxaa` and `maax` type implementations of Curve25519 are available in [33]. For all three types, we provide new implementations of Curve25519 which are faster than the implementations in [33]. For Curve448, `mxaa` and `maax` type implementations are available from [30, 33]. We provide a new `maa` type implementation for Curve448.

Intel processors from Haswell onwards provide `AVX2` instructions which support 4-way SIMD on 256-bit registers. This allows vectorized implementations. For the Montgomery ladder, 4-way vectorized algorithms have been described in [20, 32]. Vectorized implementations of the Montgomery ladder for Curve25519 and Curve448 are known.

We provide `maa`, `mxaa` and `maax` type implementations for the new curves. We also provide vectorized implementations of the new curves which follow the strategy in [32].

4.1 64-Bit Implementations

All 64-bit implementations of the Montgomery curves are based on Algorithms 1 and 7 of [31]. The details of the implementations are discussed below.

Let $m = \lceil \log_2 p \rceil$. Elements of $\mathbb{F}_p$ can be represented as m-bit strings which will be represented as κ 64-bit words. Conventionally, each such word is called a limb. We will consider packed or saturated limb representation. In this representation, m is written as $m = 64(\kappa - 1) + \nu$ with $1 \leq \nu \leq \eta \leq 64$. In other words, the first $(\kappa - 1)$ limbs are 64 bits long, while the size of the last limb is ν which lies between 1 and η. In the appendix, we mention unsaturated limb representation in the context of `maa` type implementations.

Representation of field elements. The representations of the four primes of interest to this work are given in Table 2. Note that for p251-9 and p444-17, $\eta - \nu \geq 3$ (equivalently, the last limb has three or more "free" bits), for p255-19, $\eta - \nu = 1$ (equivalently, the last limb has one "free" bits) and for p448-224-1, $\eta = \nu$ (equivalently, the last limb has no "free" bits). These have significant effect on the ladder computation as we will see below.

Prime	m	κ	η	ν	$\eta - \nu$
p251-9	251	4	64	59	5
p255-19	255	4	64	63	1
p444-17	444	7	64	60	4
p448-224-1	448	7	64	64	0

Table 2: Saturated limb representations of field elements.

Integer multiplication/squaring. The `maax` operations can be used to perform fast integer multiplication using two independent carry chains. For multiplication/squaring of 256-bit numbers, this technique has been explained in the Intel white papers [34, 35]. A general algorithmic description for multiplication/squaring of 64κ-bit numbers, $\kappa \geq 4$ is given in [29]. Saturated limb multiplication/squaring algorithms can also be implemented using a single carry chain with the help of the instructions mulx/add/adc. Such sequential implementations are applicable for the Haswell processor where the instruction mulx is available but the instructions adcx/adox are not.

Reduction. Integer multiplication/squaring of κ-limb quantities produces a 2κ-limb output. The reduction step reduces this output modulo the prime p. A full reduction will reduce the output to a value less than p. For the purposes of efficiency a full reduction is not carried out in the intermediate steps of the computation. Instead a size reduction is done. The size reduction can be of two types, namely,

reduction to an $(m+1)$-bit integer and reduction to an m-bit integer (note than an m-bit integer is not necessarily fully reduced since it is not necessarily less than p). The former is more efficient than the later. Further, the reduced quantity should again be a κ-limb quantity. If $\nu < 64$, i.e., the last limb has at least one free bit, then reduction to an $(m+1)$-bit integer is a κ-limb quantity. On the other hand, if $\nu = 64$, i.e., the last limb has no free bits, then it is a necessity to reduce to an m-bit integer to obtain a κ-limb quantity. Among the primes in Table 2, the prime $2^{448} - 2^{224} - 1$ has no extra bits in the last limb and the reduction for this prime has to be to an m-bit integer. For the other primes, it is possible to reduce to an $(m+1)$-bit integer without any overfull. The size reductions to $(m+1)$ bits modulo $2^{251} - 9$ and $2^{444} - 17$ have been done following the algorithm reduceSLPMP in [29].

Addition and subtraction. Other than multiplication/squarings, the ladder algorithm also uses field addition and subtraction. In the ladder algorithm, the inputs to an addition/subtraction operation are outputs of multiplication/squaring operations and the outputs of addition/subtraction operations are inputs to multiplication/squaring operations. In particular, the outputs of addition/subtraction are never inputs to another addition/subtraction.

We have mentioned that the outputs of multiplication/squaring are size reduced to either m bits or to $(m+1)$ bits. So, the inputs to addition/subtraction operations are either m bits or $(m+1)$ bits. We require the outputs of the addition/subtraction operations to be κ-limb quantities so that the integer multiplication/squaring algorithm can be applied to these outputs. So, it is not always required to size reduce the outputs of addition/subtraction operations to m or $(m+1)$ bits. Depending upon the sizes of the inputs to the addition/subtraction operation and the relative values of η and ν, various cases may arise. We discuss the cases of addition and subtraction separately.

Addition. A field addition is typically an integer addition followed by a possible reduction operation. The integer addition operation increases the size of the output by one bit compared to the sizes of the inputs.

Case p448-224-1*:* In this case, there is no leeway in the last limb and the output of integer addition must necessarily be reduced to obtain a κ-limb quantity.

Case p255-19*:* If the inputs to the addition are m-bit quantities, then it is possible to omit applying the reduction step to the output of the integer addition operation. On the other hand, if the inputs to the addition are $(m+1)$-bit quantities, then the reduction step has to be applied to the output of the integer addition operation. The inputs to the addition operation are the outputs of previous multiplication/squaring operations. So, whether the output of the integer addition needs to be reduced depends on whether the outputs of the multiplication/squaring operation have been reduced to m bits or to $(m+1)$ bits.

Cases p251-9 *and* p444-17*:* In these cases, it is possible to reduce the outputs of multiplication/squaring to $(m+1)$ bits *and* omit the reduction step after the integer addition operation.

Subtraction. A field subtraction is of the type $a - b \bmod p$. To avoid handling negative numbers, a suitable multiple of p is added to a so that the result is guaranteed to be positive. Since the result will be reduced modulo p, the correctness of the result is not affected by adding a multiple of p.

Cases p255-19 *and* p448-224-1*:* The reduction operation must be performed on the output of each subtraction operation to ensure that the result fits in κ limbs.

Cases p251-9 *and* p444-17*:* The operation $a - b \bmod p$ is performed as follows. Note that both a and b are $(m+1)$-bit quantities. The operation $4p + a - b$ is guaranteed to be an $(m+3)$-bit non-negative integer. So, instead of performing $a - b \bmod p$, the operation $(4p + a) - b$ is computed. Since the result is at most an $(m+3)$-bit quantity, it fits within κ limbs. Consequently, no reduction operation is performed on this result.

Remark. We have discussed the issue of avoiding reduction with respect to 64-bit arithmetic. The general idea, on the other hand, holds for saturated limb representations using 32-bit (or, lower) arithmetic. The implementation benefits of of p251-9 and p444-17 over p255-19 and p448-224-1 also holds for 32-bit arithmetic.

Optimizations of the ladder step. Based on the description in Section 4, the following strategy may be adopted for implementing the ladder step for the various primes.

Case $p448$-224-1*:* The outputs of all multiplication/squaring operations are to be size reduced to m bits. Outputs of all addition/subtraction operations are to be size reduced to m bits.

Case $p255$-19*:* The outputs of all multiplication/squaring operations are to be size reduced to $(m+1)$ bits. Outputs of all addition/subtraction operations are to be size reduced to m or $(m+1)$ bits.

Cases $p251$-9 *and* $p444$-17*:* The outputs of all multiplication/squaring operations are to be size reduced to $(m+1)$ bits. Outputs of all addition/subtraction operations are left unreduced.

The above strategy has direct consequences to the efficiencies of the ladder step for the various primes. We summarize these below.

4-*limb representations:* For both $\mathbb{F}_{2^{251}-9}$ and $\mathbb{F}_{2^{255}-19}$, field elements have 4-limb representations. So, the integer multiplication/squaring operations take the same time in both cases. Due to the ability to avoid reductions, the ladder step is significantly faster modulo $2^{251}-9$ compared to $2^{255}-19$.

7-*limb representations:* For the fields $\mathbb{F}_{2^{444}-17}$ and $\mathbb{F}_{2^{448}-2^{224}-1}$, field elements have 7-limb representations. So, the integer multiplication/squaring operations take the same time in both cases. Due to the ability to avoid reductions, the ladder step is significantly faster modulo $2^{444}-17$ compared to $2^{448}-2^{224}-1$.

4.2 Vectorized Implementation

Vectorized implementations of the new Montgomery curves are based on Algorithms 7 and 8 of [32]. We briefly mention the relevant details.

Representation of field elements. The elements of $\mathbb{F}_{2^{251}-9}$ are represented using 9 words, i.e., we consider values of κ as 9. The elements of $\mathbb{F}_{2^{444}-17}$ are represented using 16 words, and here the value of κ is 16. The details of the representations are provided in Table 3.

Prime	m	κ	η	ν	$\eta-\nu$
$p251$-9	251	9	28	27	1
$p444$-17	444	16	28	24	4

Table 3: Representations of field elements for vectorized implementation of the Montgomery ladder.

Multiplication and squaring in $\mathbb{F}_p$. For $p=2^{251}-9$, the schoolbook method is used for multiplication and squaring. For $p=2^{444}-17$ we use a (8+8)-Karatsuba strategy for performing integer multiplication and squaring; for the sub-problems of size 8 we apply directly the schoolbook method. After integer multiplication/squaring we have a 31-limb quantity. Directly trying to reduce this 31-limb quantity to a 16-limb quantity results in overfull. Instead, we first expand the 31-limb quantity to an 32-limb quantity so that the sizes of the limbs get reduced. Then the 32-limb quantity is reduced to a 16-limb quantity. This is essentially the mulle algorithm in [23].

The multiplication and squaring operations in $\mathbb{F}_p$ include a reduction operation. Instead of using the direct carry chain, we use the interleaved carry chain for reduction.

- For $p=2^{251}-9$, we interleave the chains $c_0 \to c_1 \to \cdots \to c_4 \to c_5$ and $c_4 \to c_5 \to \cdots \to c_8 \to c_0 \to c_1$.
- For $p=2^{444}-17$, we interleave the chains $c_0 \to c_1 \to \cdots \to c_7 \to c_8 \to c_9$ and $c_8 \to c_9 \to \cdots \to c_{15} \to c_0 \to c_1$.

Multiplication by a small constant in $\mathbb{F}_p$. Let $A \in \mathbb{F}_p$ have a κ-limb representation $(a_0, a_1, \ldots, a_{\kappa-1})$. Let $\mathfrak{c}$ be a small element in $\mathbb{F}_p$, which can be represented using a single limb. Then multiplication of A by $\mathfrak{c}$ provides κ limbs of the form $(c_0, c_1, \ldots, c_{\kappa-1}) = (a_0 \cdot \mathfrak{c}, a_1 \cdot \mathfrak{c}, \ldots, a_{\kappa-1} \cdot \mathfrak{c})$. This needs to be reduced.

- For $p=2^{251}-9$, we interleave the chains $c_0 \to c_1 \to \cdots \to c_3 \to c_4$ and $c_4 \to c_5 \to \cdots \to c_8 \to c_0$.
- For $p=2^{444}-17$, we interleave the chains $c_0 \to c_1 \to \cdots \to c_7 \to c_8$ and $c_8 \to c_9 \to \cdots \to c_{15} \to c_0$.

Hadamard transforms. The vectorized Montgomery ladder in [32] uses Hadamard transforms. Efficient implementations of these operations can be done using a dense packing of the field elements. We refer to [32] for details. The outputs of the Hadamard transforms are reduced for both the primes.

4.3 Inversion in $\mathbb{F}_p$

The final output of the ladder algorithm needs to compute a modular inverse. The computation of the inverse is done through exponentiation, which needs squaring and multiplication in $\mathbb{F}_p$. For $p = 2^{251} - 9$, the computation of inverse is done using the saturated 4-limb representation, and for $p = 2^{444} - 17$, the saturated 7-limb representation has been used. We provide both `mxaa` and `maax` implementations of the inversion algorithm. The relevant algorithms for field arithmetic from [29] have been used.

5 Timings and Performance Analysis

The timing experiments were carried out on a single core of Haswell and Skylake processors. During measurement of the CPU-cycles, Turbo-Boost and Hyper-Threading features were turned off.

Curve	Haswell	Skylake	κ	Strategy	Implementation	Implementation Type
Curve25519	143956	126940	4	64-bit seq	[33]	`mxaa`, inline assembly
	143369	113481	4	64-bit seq	this work	`mxaa`, assembly
	-	118231	4	64-bit seq	[33]	`maax`, inline assembly
	-	98694	4	64-bit seq	this work	`maax`, assembly
	140996	104519	9	4-way, [20]	[20]	`AVX2`, intrinsics
	121539	99898	9	4-way SIMD [20]	[32]	`AVX2`, assembly
	126521	97590	10	4-way SIMD [20]	[32]	`AVX2`, assembly
	120108	99194	9	4-way SIMD [32]	[32]	`AVX2`, assembly
	123899	95437	10	4-way SIMD [32]	[32]	`AVX2`, assembly
M⟦4698⟧	129732	102570	4	64-bit seq	this work	`mxaa`, assembly
	-	87807	4	64-bit seq	this work	`maax`, assembly
	114937	91203	9	4-way SIMD [32]	this work	`AVX2`, assembly

Table 4: CPU-cycle counts for variable base scalar multiplication at the 128-bit security level.

Curve	Haswell	Skylake	κ	Strategy	Implementation	Implementation Type
Curve448	732013	587389	7	64-bit seq	[33]	`mxaa`, inline assembly
	719217	461379	7	64-bit seq	[30]	`mxaa`, assembly
	-	530984	7	64-bit seq	[33]	`maax`, inline assembly
	-	434831	7	64-bit seq	[30]	`maax`, assembly
	462277	373006	16	4-way SIMD [20]	[32]	`AVX2`, assembly
	441715	357095	16	4-way SIMD [32]	[32]	`AVX2`, assembly
M⟦4058⟧	644791	423042	7	64-bit seq	this work	`mxaa`, assembly
	-	384905	7	64-bit seq	this work	`maax`, assembly
	476866	401809	16	4-way SIMD [32]	this work	`AVX2`, assembly

Table 5: CPU-cycle counts for variable base scalar multiplication at the 224-bit security level.

Platform specifications. The details of the hardware and software tools used in our software implementations are as follows.

Haswell: Intel® Core™ i7-4790 4-core CPU 3.60 Ghz. The OS was 64-bit Ubuntu 14.04 LTS and the source code was compiled using GCC version 7.3.0.

Skylake: Intel® Core™ i7-6500U 2-core CPU @ 2.50GHz. The OS was 64-bit Ubuntu 14.04 LTS and the source code was compiled using GCC version 7.3.0.

Along with the timings of our implementations, we also report timings of previous implementations. For a fair comparison, we have downloaded the relevant codes and have measured the timings of these codes on the same platforms where we measured the timings of our implementations. The timings thus obtained have been reported in the tables. In some cases, these values are different from the timings reported in the original papers. The source of such differences is the possible differences in the micro-architectures of the same family of processors.

Timings in form of CPU-cycles are shown in Tables 4 and 5 for Haswell and Skylake processors. As mentioned earlier, we have carried out `maa`, `mxaa` and `maax` type implementations. The timing results show that the `maa`-type implementations are uniformly slower than the `mxaa`-type and the `maax` type implementations. Due to this, we do not report the timings of the `maa` type implementations in Tables 4 and 5. For reference, we report the timings of the `maa`-type implementations in Table 7 of Appendix A.

For comparison, we report the timings of the previously most efficient (to the best of our knowledge) and publicly available sequential and vectorized implementations. For 64-bit implementations of Curve25519 we refer to [33] and for Curve448 we refer to [30, 33]. For vectorized implementations of Curve25519 and Curve448 we refer to [20, 32]. In the discussion below, we mention speed-up percentages which is defined to be $100(t_1 - t_2)/t_1$, where t_1 and t_2 are the old and new timings respectively.

New implementations of Curve25519. Table 4 shows the timing results for the new implementations of Curve25519 that have done in the context of the present work. The new `maax`-type implementation is about 17% faster over the previous best implementation [33] on Skylake. On Haswell, the performance improvement of the new `mxaa`-type implementation over the `mxaa`-type implementation of [33] is small. We would like to mention two issues.

1. For reduction we have used the algorithm reduceSLPMP from [29], while the algorithm used by [33] is the same as algorithm reduceSLPMPa of [29]. To assess the effect of the reduction algorithm, we made an assembly implementation (note that the code of [33] uses inline assembly) using reduceSLPMPa. It turns out that using reduceSLPMP leads to a faster code.

2. The field operations in the implementations of [33] have been developed using inline assembly and then integrated through a high level function in the Montgomery ladder-step. In contrast, we have developed the entire Montgomery ladder-step as a single hand-optimized assembly code, in which we have judiciously used the available 64-bit registers to minimize the overall load/store operations. The timings indicate that such a strategy to develop the assembly code provides a substantial gain in efficiency on the Skylake processor, while on Haswell the gain is nominal.

From Table 4, we note that the performance of the new `maax` implementation of Curve25519 lies between the performances of the 9-limb and 10-limb 4-way SIMD implementations of [32].

Comparison between M⟦4698⟧ and Curve25519. From Table 4, we see that M⟦4698⟧ is faster than Curve25519 for all the types of implementations, though the speed-up percentages vary. For `mxaa` type implementations, the speed-ups on both Haswell and Skylake are about 9.5%; for `maax` type implementations, the speed-up (on Skylake) is about 11%; for `AVX2` type implementations, the speed-ups on both Haswell and Skylake are about 4.3%.

Comparison between M⟦4058⟧ and Curve448. From Table 5, we observe that M⟦4058⟧ is faster than Curve448 for `mxaa` and `maax` implementations; for `mxaa` implementations, the speed-ups are 10.3% and 8.3% respectively on Haswell and Skylake, while for `maax`, the speed-up is 11.5 % (on Skylake). For `AVX2` implementations, compared to Curve448, M⟦4058⟧ has slowdowns of 7.4% and 11% on Haswell and Skylake respectively.

The explanation for the above observations is as follows. The number of limbs for both Curve448 and M⟦4058⟧ are the same. For `AVX2` implementations (using 256-bit registers) the number of limbs is 16, whereas for `mxaa` and `maax` type implementations the number of limbs is 7. As a result, for `AVX2` type implementations the underlying integer multiplication is faster using Karatsuba rather than schoolbook, while for the other two implementations schoolbook is faster than Karatsuba. The prime $p448$-224-1 on which Curve448 is based has been chosen such that Karatsuba is particularly efficient. So, whenever the underlying integer multiplication is faster using Karatsuba than schoolbook, Curve448 will be faster than M⟦4058⟧. In a similar vein, due to the reasons explained in Section 4, whenever the underlying multiplication is faster using schoolbook than Karatsuba, M⟦4058⟧ will be faster than Curve448. Future availability of wider vector operations would lead to a reduction in the number of limbs. This may result in schoolbook becoming faster than Karatsuba and consequently, M⟦4058⟧ being faster than Curve448.

Remark: It is interesting to note that for both M⟦4698⟧ and M⟦4058⟧, the best timings are obtained for the `maax` implementation rather than the `AVX2` implementation.

Key generation. We have also developed assembly code for fixed base scalar multiplication using Montgomery ladder. This corresponds to the key generation phase of the Diffie-Hellman protocol. The timings are reported in Table 6. For comparison we also report the corresponding timings of Curve25519 and Curve448 from [30, 32]. The timings of the `maa`-type implementations are reported in Table 8 of Appendix A.

Curve	Haswell	Skylake	κ	Strategy	Implementation	Implementation Type
Curve25519	132162	106725	4	64-bit seq	this work	`mxaa`, assembly
	-	93247	4	64-bit seq	this work	`maax`, assembly
	100127	86885	9	4-way SIMD [32]	[32]	`AVX2`, assembly
	106190	84047	10	4-way SIMD [32]	[32]	`AVX2`, assembly
Curve448	653035	427058	7	64-bit seq	[30]	`mxaa`, assembly
	-	396583	7	64-bit seq	[30]	`maax`, assembly
	381417	317778	16	4-way SIMD [32]	[32]	`AVX2`, assembly
M⟦4698⟧	120599	96683	4	64-bit seq	this work	`mxaa`, assembly
	-	82116	4	64-bit seq	this work	`maax`, assembly
	96419	79770	16	4-way SIMD [32]	this work	`AVX2`, assembly
M⟦4058⟧	573782	389371	7	64-bit seq	this work	`mxaa`, assembly
	-	359597	7	64-bit seq	this work	`maax`, assembly
	403697	335314	16	4-way SIMD [32]	this work	`AVX2`, assembly

Table 6: CPU-cycle counts for fixed base scalar multiplication required by the curves Curve25519, Curve448, M⟦4698⟧ and M⟦4058⟧ at 128-bit and 224-bit security levels on Haswell and Skylake processors.

6 Conclusion

In this paper, we have introduced two pairs of Montgomery-Edwards curves for performing cryptography at the 128-bit and the 224-bit security levels. Compared to the curves proposed in IETF RFC 7748, the new curves provide 1.5 to 2 bits less security. We have performed various kinds of implementations of scalar multiplication using the Montgomery ladder for the new curves. At the 128-bit level, the new curve is faster than Curve25519 for all the types of implementations that we have considered. At the 224-bit level, the new curve is faster when the underlying integer multiplication is faster using schoolbook than Karatsuba. Our work provides a wider picture of the efficiency/security trade-off at the 128-bit and the 224-bit security levels.

Acknowledgements: Thanks to Rene Struik for comments on an earlier version of the paper and to Armando Faz Hernández for comments on an earlier version of our implementation code.

References

[1] 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.

[2] D. J. Bernstein and Lange T. Faster addition and doubling on elliptic curves. In *Advances in Cryptology - ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 29–50. Springer, 2007.

[3] Daniel J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.

[4] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted edwards curves. In Serge Vaudenay, editor, *Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings*, volume 5023 of *Lecture Notes in Computer Science*, pages 389–405. Springer, 2008.

[5] 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.

[6] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *J. Cryptographic Engineering*, 2(2):77–89, 2012.

[7] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 967–980. ACM, 2013.

[8] Daniel J. Bernstein and Tanja Lange. Montgomery curves and the Montgomery ladder. In Joppe W. Bos and Arjen K. Lenstra, editors, *Topics in Computational Number Theory inspired by Peter L. Montgomery*, pages 82–115. Cambridge University Press, 2017.

[9] 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.

[10] Brainpool. ECC standard. http://www.ecc-brainpool.org/ecc-standard.htm.

[11] 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.

[12] Craig Costello and Michael Naehrig. Isogenies between (twisted) Edwards and Montgomery curves. https://cryptosith.org/papers/isogenies_tEd2Mont.pdf, 2015. Accessed on 16 September, 2019.

[13] Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic - the case of large characteristic fields. *J. Cryptographic Engineering*, 8(3):227–240, 2018.

[14] NIST Curves. Recommended elliptic curves for federal government use. http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf, 1999.

[15] Harold M. Edwards. A Normal Form for Elliptic Curves. *Bulletin of the American Mathematical Society*, 44:393–422, 2007.

[16] 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.

[17] P. Gaudry and É. Schost. Genus 2 point counting over prime fields. *J. Symb. Comput.*, 47(4):368–400, 2012.

[18] Mike Hamburg. Ed448-goldilocks, a new elliptic curve. *IACR Cryptology ePrint Archive*, 2015:625, 2015.

[19] 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.

[20] Hüseyin Hisil, Berkan Egrice, and Mert Yassi. Fast 4 way vectorized ladder for the complete set of montgomery curves. *IACR Cryptology ePrint Archive*, 2020:388, 2020.

[21] 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.

[22] Sabyasachi Karati and Palash Sarkar. Kummer for Genus One over Prime Order Fields. *Journal of Cryptology*, 2019. https://doi.org/10.1007/s00145-019-09320-4.

[23] Sabyasachi Karati and Palash Sarkar. Kummer for genus one over prime-order fields. *J. Cryptology*, 33(1):92–129, 2020.

[24] Neal Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, 48(177):203–209, 1987.

[25] Adam Langley and Mike Hamburg. Elliptic curves for security. Internet Research Task Force (IRTF), Request for Comments: 7748, https://tools.ietf.org/html/rfc7748, 2016. Accessed on 16 September, 2019.

[26] CFRG/IETF mail archive. https://mailarchive.ietf.org/arch/msg/cfrg/LQIyeCFGgoROzsx_UBf9cjlsS-A.

[27] 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.

[28] Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.

[29] Kaushik Nath and Palash Sarkar. Efficient Arithmetic in (Pseudo-)Mersenne Prime Order Fields. *IACR Cryptology ePrint Archive*, 2018:985, 2018.

[30] Kaushik Nath and Palash Sarkar. Reduction modulo $2^{448} - 2^{224} - 1$. *IACR Cryptology ePrint Archive*, 2019:1304, 2019.

[31] Kaushik Nath and Palash Sarkar. Constant Time Montgomery Ladder. *IACR Cryptology ePrint Archive*, 2020:956, 2020.

[32] Kaushik Nath and Palash Sarkar. Efficient 4-way Vectorizations of the Montgomery Ladder. *IACR Cryptology ePrint Archive*, 2020:378, 2020.

[33] Thomaz Oliveira, Julio López Hernandez, Hüseyin Hisil, Armando Faz-Hernández, and Francisco Rodríguez-Henríquez. How to (pre-)compute a ladder - improving the performance of X25519 and X448. In Carlisle Adams and Jan Camenisch, editors, *Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers*, volume 10719 of *Lecture Notes in Computer Science*, pages 172–191. Springer, 2017.

[34] E. Ozturk, J. Guilford, and V. Gopal. Large integer squaring on Intel architecture processors, intel white paper. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/large-integer-squaring-ia-paper.pdf, 2013.

[35] E. Ozturk, J. Guilford, V. Gopal, and W. Feghali. New instructions supporting large integer arithmetic on Intel architecture processors, intel white paper. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-large-integer-arithmetic-paper.pdf, 2012.

[36] Certicom Research. SEC 2: Recommended elliptic curve domain parameters. http://www.secg.org/sec2-v2.pdf, 2010.

[37] Version 1.3 TLS Protocol. RFC 8446. https://datatracker.ietf.org/doc/rfc8446/?include_text=1, 2018. Accessed on 16 September, 2019.

[38] NUMS: Nothing up my sleeve. https://tools.ietf.org/html/draft-black-tls-numscurves-00.

A Timings of maa-type Implementations

In this section we report the timings of the maa-type implementations. Table 7 contains the timings of variable base scalar multiplication and Table 8 contains the timings of fixed base scalar multiplication. Note that for Curve25519 and M⟦4698⟧, we also consider representations where the number of limbs κ is equal to 5. This corresponds to an unsaturated (or redundant) limb representation, where the limbs store less than 64 bits of information. Similarly, for Curve448 and M⟦4058⟧, we consider unsaturated limb representation using 8 limbs.

Comparing the Haswell entry for the maa-type implementation of Curve448 in Table 8 with the Haswell entry for the mxaa-type implementation of Curve448 in Table 6 we can see that the maa-type implementation performs better than mxaa-type in this case.

Curve	Haswell	Skylake	κ	Strategy	Implementation	Implementation Type
Curve25519	161045	148291	5	64-bit seq	[5]	`maa`, assembly
	179124	147823	4	64-bit seq	[5]	`maa`, assembly
	170381	137453	4	64-bit seq	[33]	`maa`, inline assembly
	167170	128843	4	64-bit seq	this work	`maa`, assembly
Curve448	721044	558740	8	64-bit seq	this work	`maa`, assembly
M⟦4698⟧	154455	132255	5	64-bit seq	this work	`maa`, assembly
	143282	118019	4	64-bit seq	this work	`maa`, assembly
M⟦4058⟧	681257	597240	8	64-bit seq	this work	`maa`, assembly

Table 7: CPU-cycle counts of maa-type implementations for variable base scalar multiplication.

Curve	Haswell	Skylake	κ	Strategy	Implementation	Implementation Type
Curve25519	153754	119958	4	64-bit seq	this work	`maa`, assembly
Curve448	644288	504808	8	64-bit seq	this work	`maa`, assembly
M⟦4698⟧	141965	121265	5	64-bit seq	this work	`maa`, assembly
	133346	109989	4	64-bit seq	this work	`maa`, assembly
M⟦4058⟧	612225	540766	8	64-bit seq	this work	`maa`, assembly

Table 8: CPU-cycle counts of maa-type implementations for fixed base scalar multiplication.