# Selecting Elliptic Curves for Cryptography: An Efficiency and Security Analysis

Joppe W. Bos, Craig Costello, Patrick Longa and Michael Naehrig

Microsoft Research, USA

**Abstract.** We select a set of elliptic curves for cryptography and analyze our selection from a performance and security perspective. This analysis complements recent curve proposals that suggest (twisted) Edwards curves by also considering the Weierstrass model. Working with both Montgomery-friendly and pseudo-Mersenne primes allows us to consider more possibilities which improves the overall efficiency of base field arithmetic. Our Weierstrass curves are backwards compatible with current implementations of prime order NIST curves, while providing improved efficiency and stronger security properties. We choose algorithms and explicit formulas to demonstrate that our curves support constant-time, exception-free scalar multiplications, thereby offering high practical security in cryptographic applications. Our implementation shows that variable-base scalar multiplication on the new Weierstrass curves at the 128-bit security level is about 1.4 times faster than the recent implementation record on the corresponding NIST curve. For practitioners who are willing to use a different curve model and sacrifice a few bits of security, we present a collection of twisted Edwards curves with particularly efficient arithmetic that are up to 1.43, 1.26 and 1.24 times faster than the new Weierstrass curves at the 128-, 192- and 256-bit security levels, respectively. Finally, we discuss how these curves behave in a real world protocol by considering different scalar multiplication scenarios in the transport layer security (TLS) protocol.

## 1 Introduction

The first release of a cryptographic standard specifying elliptic curves for use in practice dates back to 2000 [21]. Nowadays, roughly one out of ten systems on the publicly observable internet offers cipher suites in the Secure Shell (SSH) and Transport Layer Security (TLS) protocols that contain elliptic curve based cryptographic algorithms [16]. Most elliptic curve standards recommend curves for different perceived security levels. These curves are either defined over prime fields or binary extension fields; on the internet, however, the deployed elliptic curves are mostly defined over prime fields [16]. This can be partially explained by the increasing skepticism towards the security of elliptic curves defined over binary extension fields (justified by recent progress on solving the discrete logarithm problem on such curves [25]). Therefore, in this work, we only consider elliptic curves defined over prime fields.

Recently, part of the cryptographic community has been looking for alternatives to the currently deployed elliptic curves that may offer better performance and provide stronger overall security (see for example an evaluation of recent curve candidates in [12]). The urge to change curves has been fueled by the recently leaked NSA documents, which suggest the existence of a back door in the Dual Elliptic Curve Deterministic Random Bit Generator [54]. Although cryptographers have suspected this at least as early as in 2007 [51], these recent revelations have fueled a controversy on whether the widely deployed NIST curves [56] should be replaced by curves with a verifiably deterministic generation. Besides such security concerns, there has been significant progress related to both efficiency and security since the initial standardization of elliptic curve cryptography. Notable examples are algorithms protected against certain side-channel attacks, different "special" prime shapes which allow faster modular arithmetic, and a larger set of curve models from which to choose. For example, in 2007, Edwards [24] discovered an interesting normal form for elliptic curves, now called the Edwards model, which was introduced to cryptographic applications by Bernstein and Lange [11]. A generalization of this curve model, known as the twisted Edwards model [7], facilitates the most efficient curve arithmetic [34]. Such (twisted) Edwards curves also have other attractive properties: they may be selected to support a complete addition law and are compatible with the Montgomery model, which supports efficient Montgomery ladder computations [46]. However, twisted Edwards curves cannot have a prime number of rational points over the base field, and they are therefore incompatible with the *prime-order* Weierstrass curves used in all of the current cryptographic standards [21, 47, 56].

**Related Work.** The NIST curves [56] have been included in numerous standards (e.g. [21, 47]) and are deployed in many security protocols. The most recent speed record on the NIST curve which aims to provide 128-bit security is due to Gueron and Krasnov [30]. Alternatives to the NIST curves have been suggested by the German working group Brainpool [23]; their curve choices followed additional security requirements, one of which demands verifiably pseudo-random curve generation. Another alternative curve has been proposed by Bernstein [5]; this is a Montgomery curve, called Curve25519, which allows efficient computation of ECDH using the Montgomery ladder at the 128-bit security level. It was later shown by Bernstein et al. [9] that a twisted Edwards curve, birationally equivalent to Curve25519, can be used for efficient elliptic curve signature generation and verification. Recently, Bernstein and Lange started a project to select and analyze secure elliptic curves for use in cryptography: see [12] for a list of the security assessments the project performs and the requirements it imposes. A range of curves, targeting different security levels, is also presented in [12], mostly analogous to Curve25519. Following this, several new curves satisfying the requirements from [12], which facilitate both the twisted Edwards and Montgomery form, were proposed by Aranha et al. [3].

**Motivation and Rationale.** The new curves presented in [12, 3] are all efficient and secure elliptic curves ready to be used in cryptography. This prompts the question as to why we should perform an efficiency and security analysis for a set of new curves. It is our opinion that not all options for prime fields and elliptic curve models have been considered in the recent curve proposal projects (either because they are overlooked or do not fit the requirements set by the project). Our goal is to rigorously analyze all of these different aspects from both a security and efficiency perspective, in hope that this paper helps practitioners better understand (and correctly implement) the choices that lie in front of them. Abandoning a set of standard curves demands a judicious selection of new curves, since this cannot be done too frequently if widespread adoption is desired. In that light, it is our opinion that one should consider all of the options available. For example, in contrast to [12, 3], our selection includes prime order Weierstrass curves. Just as the almost-prime order twisted Edwards curves have their practical advantages, we argue that there are also benefits to choosing prime order Weierstrass curves: the absence of small torsion simplifies the point/input validation process, and (over a prime field of fixed length) does not sacrifice any bits of security with respect to attacks on the underlying elliptic curve discrete logarithm problem (ECDLP). In addition, such curves are backwards compatible with current implementations supporting NIST curves over prime fields (i.e., no changes are required in protocols), and could be integrated into existing implementations by simply changing the curve constant and (in some cases) field arithmetic[1].

[1] Cryptographic libraries with support for generic-prime field arithmetic (e.g., using Montgomery arithmetic) are fully compatible with the proposed curves.

We investigate the selection of prime moduli that allow efficient modular arithmetic. As in [5, 34, 41, 15, 12, 3], we study pseudo-Mersenne primes of the form $2^\alpha - \gamma$, but also primes of the form $2^\alpha(2^\beta - \gamma) - 1$ that can be used to accelerate Montgomery arithmetic [45] as used in [31, 15]. Following the deterministic selection requirement from [12], we pick two primes of each shape for a given targeted security level: one prime is selected to be slightly smaller than the other, which sacrifices a small amount of ECDLP security in favor of enhanced performance. Note that, as explained in Section 2, for practical considerations we require all primes to be congruent to 3 modulo 4. These primes are used to construct cryptographically suitable curves focusing on (arguably) the two most relevant curve models: short Weierstrass curves with the curve parameter $a$ set to $-3$ and twisted Edwards curves with the curve parameter $a$ set to $-1$. The prime order Weierstrass curves give full ECDLP security over prime fields of a fixed bitlength, while offering good practical performance. On the other hand, the twisted Edwards curves sacrifice a small amount of ECDLP security but facilitate the fastest realization of curve arithmetic [34]. Both types of curves are selected in a deterministic fashion (see Section 3 for the full details) and offer twist-security [5], a property which is useful in certain scenarios.

An important requirement for implementations of modern cryptographic algorithms is a constant run-time when the algorithm computes on secret data to guard against timing attacks [38]. In particular, this potential threat exists for two basic elliptic curve operations: variable-base and fixed-base scalar multiplication. One solution is to use a complete addition law. However, a complete addition law is typically less efficient compared to the dedicated formulas which can fail for certain inputs. In Section 4 we outline another solution to this problem for the variable-base case. We show that our algorithms which compute on secret data, can never run into any exceptional cases (i. e. produce incorrect results) while using the faster dedicated formulas and ensuring a constant runtime (with the exception of the very last addition; see Section 4.1 for the details). Hence, this solution results in faster implementations compared to the complete solution. In the fixed-base case the situation is more complicated: most efficient algorithms in the literature may potentially run into exceptions. While the use of a complete addition formula suffices to solve the problem on twisted Edwards curves, the high cost of complete additions on Weierstrass curves would degrade performance significantly [18] (see Appendix C.1). To solve this problem, we propose a new formula that works for all possible inputs by exploiting masking techniques. This "complete" addition requires the same number of multiplications and squarings as the unprotected dedicated addition formula and drastically reduces the overhead of protecting scalar multiplication. We comment that the formula is also useful in the context of secure, exception-free multi-scalar multiplications. The reader is referred to Appendix C.1 for more details on the new formula.

We do not claim full security against other attacks such as simple power analysis (SPA); this is left for future work. Nevertheless, we remark that all the selected algorithms have a regular structure as required when implementing countermeasures against certain simple side-channel attacks.

**Summary of contributions.**

- Analysis of a new set of deterministically selected prime-order Weierstrass curves (see Table 1). These are defined over pseudo-Mersenne and Montgomery-friendly primes whose bitlengths match those of the NIST primes. See Sections 2 and 3.
- Analysis of a new set of deterministically selected composite-order twisted Edwards curves (see Table 2 and Section 3). In contrast to existing curve proposals, the selected curves

**Table 1.** Summary of our chosen Weierstrass curves of the form $E_b/\mathbf{F}_p : y^2 = x^3 - 3x + b$ defined over $\mathbf{F}_p$ with quadratic twist $E'_b/\mathbf{F}_p : y^2 = x^3 - 3x - b$ and target security level $\lambda$. The group orders $r = \#E_b(\mathbf{F}_p)$ and $r' = \#E'_b(\mathbf{F}_p)$ are both prime, and $r < p$ for all curves. The value under $\rho$ complexity is an estimate for the actual security of the ECDLP against Pollard's $\rho$ method, it is $\log_2(\sqrt{\pi/4} \cdot \sqrt{r})$ rounded to one decimal.

| target security $\lambda$ | curve name | $p$ | $b$ | $\rho$ complexity |
|---|---|---|---|---|
| 128 | **w-256-mont** | $2^{240}(2^{16} - 88) - 1$ | 85610 | 127.8 |
| | **w-254-mont** | $2^{240}(2^{14} - 127) - 1$ | -12146 | 126.8 |
| | **w-256-mers** | $2^{256} - 189$ | 152961 | 127.8 |
| | **w-255-mers** | $2^{255} - 765$ | -20925 | 127.3 |
| 192 | **w-384-mont** | $2^{376}(2^{8} - 79) - 1$ | 27798 | 191.5 |
| | **w-382-mont** | $2^{368}(2^{14} - 5) - 1$ | -133746 | 190.8 |
| | **w-384-mers** | $2^{384} - 317$ | -34568 | 191.8 |
| | **w-383-mers** | $2^{383} - 421$ | 97724 | 191.3 |
| 256 | **w-512-mont** | $2^{496}(2^{16} - 491) - 1$ | 99821 | 255.8 |
| | **w-510-mont** | $2^{496}(2^{14} - 290) - 1$ | 39053 | 254.8 |
| | **w-512-mers** | $2^{512} - 569$ | 121243 | 255.8 |
| | **w-511-mers** | $2^{511} - 481$ | 555482 | 255.3 |
| | **w-521-mers** | $2^{521} - 1$ | 167884 | 260.3 |

present (simultaneously) minimal parameter $d$ in the twisted Edwards form and minimal parameter $A$ in isogenous Montgomery curves (minimal in absolute value). See Section 3.3.

- A new, (pseudo-)complete addition algorithm for general curves in short Weierstrass form. This algorithm works for *all* pairs of inputs and its execution incurs only a small overhead compared to the dedicated addition law. See Section C.1.
- Efficient fixed- and variable-base scalar multiplication algorithms that are proven to be exception-free and facilitate constant-time implementations. These algorithms can achieve such implementations on all prime-order Weierstrass curves and all composite-order twisted Edwards curves. See Section 4.
- A comprehensive software implementation providing timings for various scenarios; this includes performance estimates for the above curves when used in the context of the TLS protocol. See Sections 5 and 6.

**Proposed Curves.** Tables 1 and 2 show the curves that we have chosen deterministically according to our security and efficiency criteria. The tables show the target security level, which gives a rough estimate for the desired security in each case. Curve names indicate the curve model (**w** for the Weierstrass model and **ed** for the (twisted) Edwards model), the bit length of the underlying base field prime and the type of prime (**mont** for Montgomery-friendly and **mers** for pseudo-Mersenne primes). In Appendix D, we provide the trace of Frobenius $t$ for each curve, so the number of $\mathbf{F}_p$-rational points for the curve $E$ and its quadratic twist $E'$ can be computed as $\#E(\mathbf{F}_p) = p + 1 - t$ and $\#E'(\mathbf{F}_p) = p + 1 + t$. More details on the curve choices and their properties are given in Section 3.

## 2 Modular Arithmetic - Choosing Primes

Over a prime field $\mathbf{F}_p$ (with $p > 3$ prime), the computation of the elliptic curve group operation boils down to numerous computations modulo $p$. In this section we outline the types of primes that we prefer for efficiency and security considerations, and discuss how the primes are uniquely determined from a fixed security level.

**Table 2.** Summary of our chosen twisted Edwards curves of the form $\mathcal{E}_d/\mathbf{F}_p : -x^2 + y^2 = 1 + dx^2y^2$ defined over $\mathbf{F}_p$, where $d = -(A-2)/(A+2)$, and the target security level is $\lambda$. A model for the quadratic twist is $\mathcal{E}'_d/\mathbf{F}_p : -x^2 + y^2 = 1 + (1/d)x^2y^2$. The curve $\mathcal{E}_d$ is birationally equivalent to the Montgomery curve $E_A/\mathbf{F}_p : y^2 = x^3 + Ax^2 + x$ with quadratic twist $E_{-A}/\mathbf{F}_p : y^2 = x^3 - Ax^2 + x$. The group orders are $\#\mathcal{E}_d(\mathbf{F}_p) = 4r$ and $\#\mathcal{E}'_d(\mathbf{F}_p) = 4r'$, where $r$ and $r'$ are both prime, and $4r < p$ for all curves. The value $d_0 = -(A+2)/4 = -1/(d+1)$ defines a curve with the same group order as that given by $d$, i.e. $\#\mathcal{E}_{d_0}(\mathbf{F}_p) = 4r$ and $\#\mathcal{E}'_{d_0}(\mathbf{F}_p) = 4r' = \#\mathcal{E}_{-(d_0+1)}$. The $\rho$ complexity is an estimate for the actual security of the ECDLP against Pollard's $\rho$ method, it is $\log_2(\sqrt{\pi/4} \cdot \sqrt{r})$ rounded to one decimal.

| target security | curve name | $p$ | $A$ | $d_0$ | $\rho$ complexity |
|---|---|---|---|---|---|
| 128 | **ed-256-mont** | $2^{240}(2^{16} - 88) - 1$ | -54314 | 13578 | 126.8 |
| | **ed-254-mont** | $2^{240}(2^{14} - 127) - 1$ | -55790 | 13947 | 125.8 |
| | **ed-256-mers** | $2^{256} - 189$ | -61370 | 15342 | 126.8 |
| | **ed-255-mers** | $2^{255} - 765$ | -240222 | 60055 | 126.3 |
| 192 | **ed-384-mont** | $2^{376}(2^{8} - 79) - 1$ | -113758 | 28439 | 190.5 |
| | **ed-382-mont** | $2^{368}(2^{14} - 5) - 1$ | -2870790 | 717698 | 189.8 |
| | **ed-384-mers** | $2^{384} - 317$ | -1332778 | 333194 | 190.8 |
| | **ed-383-mers** | $2^{383} - 421$ | -2095962 | 523990 | 190.3 |
| 256 | **ed-512-mont** | $2^{496}(2^{16} - 491) - 1$ | -305778 | 76444 | 254.8 |
| | **ed-510-mont** | $2^{496}(2^{14} - 290) - 1$ | -2320506 | 580126 | 253.8 |
| | **ed-512-mers** | $2^{512} - 569$ | -2550434 | 637608 | 254.8 |
| | **ed-511-mers** | $2^{511} - 481$ | -4390390 | 1097597 | 254.3 |
| | **ed-521-mers** | $2^{521} - 1$ | -2201762 | 550440 | 259.3 |

**Primes of the form $2^\alpha - \gamma$.** Selecting primes of a special form to enhance the performance of the modular reduction is not new. The primes standardized in the digital signature standard [56] have a special form allowing fast reduction based on the work by Solinas [52]. Even faster modular reduction can be achieved by selecting primes of the form $p = 2^\alpha - \gamma$, known as pseudo-Mersenne primes. In this case, the value $\alpha$ is determined by the security parameter and is typically a multiple of 64 (or slightly smaller). The integer $\gamma$ is chosen to be a small positive integer, i.e. significantly smaller than $2^{32}$. Given two integers $x$ and $y$ such that $0 \leq x, y < 2^\alpha - \gamma$, one can compute $x \cdot y \bmod (2^\alpha - \gamma)$ by first computing the product and writing this in a radix-$2^\alpha$ system as $x \cdot y = z_h \cdot 2^\alpha + z_\ell$. A first reduction step, based on the shape of the modulus, is $z_h \cdot 2^\alpha + z_\ell \equiv z_\ell + z_h \cdot \gamma \pmod{2^\alpha - \gamma} = z$, where $0 \leq z < (\gamma + 1)2^\alpha$. If this step is repeated, the result is such that $0 \leq z < 2^\alpha + \gamma^2$, which can finally be brought into the desired range by applying an additional correction modulo $p$ using subtractions. A standard way of enhancing the performance is to use a redundant representation: instead of reducing $z$ to the range $[0, 2^\alpha - \gamma)$, one can often more efficiently reduce $z$ to the range $[0, 2^\alpha)$, or to the range $[0, 2^{2s})$ if $\alpha$ is a few bits smaller than $2s$ (at a target security level of $s$ bits). The latter case can be optimized further by computing exclusively in such a redundant form and performing a sole correction at the end of the scalar multiplication.

Given a security level of $s$ bits, we consider the parameter $\alpha \in \{2s, 2s-1\}$. Taking $\alpha = 2s$ makes the prime as large as possible, matching one of the requirements to achieve maximal ECDLP security at the $s$-bit security level. Taking $\alpha = 2s - 1$ sacrifices half a bit of ECDLP security in favor of potential enhancements in efficiency, as described above. Thus, fixing $s$ results in two possible values for $\alpha$ and subsequently two primes of the form $2^\alpha - \gamma$: for a fixed $\alpha$, we choose the smallest $\gamma$ such that $2^\alpha - \gamma$ is both prime and congruent to 3 modulo 4 (the rational behind this congruence condition is discussed below). Following our curve selection criteria, the values $\gamma$ for the curves under analysis are always smaller than $2^{10}$, which makes them attractive for efficient implementation on 16, 32 and 64-bit platforms.

**Primes of the form $2^\alpha(2^\beta - \gamma) - 1$.** Another approach to select primes is inspired by Montgomery arithmetic [45]. The idea behind Montgomery multiplication is to replace the relatively expensive divisions by computationally inexpensive logical shifts when computing the modular reduction. Some computations (and storage) can be avoided when primes of the form $p = 2^\alpha(2^\beta - \gamma) - 1$ are used for positive integers $\alpha$, $\beta$ and $\gamma$ (cf. [39, 37, 1, 31, 15]). When the prime $p$ is two bits short of a multiple of the word size $w$ (i.e. $w \mid \alpha + \beta + 2$), one can avoid a conditional subtraction in every multiplication [57].

There are different ways to construct Montgomery-friendly primes: for example, [31] prefers $\gamma$ to be a power of two, while [15] sets $\beta = 64$ and $\gamma$ as small as possible to specifically target 64-bit platforms. We make choices of $\alpha$, $\beta$ and $\gamma$ such that the modular arithmetic can be implemented efficiently on a wide range of platforms. Given a security level of $s$ bits, we consider $\alpha = 8\delta$ and $\beta \in \{2s - \alpha, 2s - 2 - \alpha\}$, and choose $\gamma$ and $\delta$ as the smallest positive integers such that $p = 2^\alpha(2^\beta - \gamma) - 1$ is prime and $\lceil \log_2(p) \rceil = 2s$ (resp. $\lceil \log_2(p) \rceil = 2s - 2$) in the setting of $\beta = 2s - \alpha$ (resp. $\beta = 2s - 2 - \alpha$). We start with $\delta = 1$ and increment it by 1 (if necessary) until $\gamma$ is found. For instance, for $s = 192$ and $\beta = 2s - \alpha$, we observe that $(\delta, \gamma) = (1, 79)$ results in a prime which can be written as

$$2^{376}(2^8 - 79) - 1 = 2^{352}(2^{32} - 2^{24} \cdot 79) - 1 = 2^{320}(2^{64} - 2^{56} \cdot 79) - 1,$$

for usage on 8-, 32- and 64-bit platforms, respectively. This has the advantage that the reduction step, which has to be computed at every iteration inside the interleaved Montgomery algorithm, can be computed using only a multiply-and-add and an addition instruction. Note that, by construction, primes of this form are always congruent to 3 modulo 4.

**Constant-time modular arithmetic.** One of the measures to guard software implementations against various types of side-channel analysis such as timing attacks [38] is to ensure a constant running time. In practice, this often means writing code which does not contain branches depending on secret data. For instance, the interleaved Montgomery multiplication algorithm requires a conditional subtraction at the end. To remove this, we always compute the subtractions and select (mask) the correct value depending on the conditional flag. In the setting of primes of the shape $2^\alpha - \gamma$, one must always compute the worst-case number of reduction rounds in order to ensure constant runtime.

Besides the "standard" modular operations, there is also the need for constant-time methods to compute the modular inversion and the modular square roots. In order to compute the inversion modulo a prime $p$, one can use Fermat's little theorem: i.e. compute $a^{p-2} \equiv a^{-1} \pmod{p}$. Since our chosen primes all have a special shape, finding efficient addition chains for this exponentiation is not difficult. For the $n$-bit primes considered in this work, we found that we can always compute the modular inversion using at most $1.11\lceil \log_2(p) \rceil$ modular multiplications and modular squarings. If $p \equiv 3 \pmod{4}$, then one can compute a modular square root $x$ (if it exists) of an element $a$ using $x \equiv a^{\frac{p+1}{4}} \pmod{p}$. Since this can be performed efficiently, and in constant-time, we require all of our primes to be congruent to 3 modulo 4.

## 3 Curve Selection

In this section we explain how the curves in Tables 1 and 2 were chosen based on the selection of primes that is outlined in Section 2. In addition to the four primes chosen to target each security level, we also include the Mersenne prime $p = 2^{521} - 1$ to target the 256-bit security level, since Mersenne primes might have a performance benefit. For each chosen prime $p \equiv 3 \pmod 4$, we provide two curves: one is a prime order short Weierstrass curve, while the other is an almost-prime order twisted Edwards curve.

### 3.1 Curve selection for Weierstrass curves

For a fixed prime $p$, a specific curve $E_b : y^2 = x^3 - 3x + b$ is uniquely determined by the curve parameter $b \in \mathbf{F}_p \backslash \{\pm 2, 0\}$. Note that, since $p \equiv 3 \bmod 4$, its non-trivial quadratic twist $E'_b$ has the curve equation $E'_b : y^2 = x^3 - 3x - b$. In order to guarantee *twist-security* [5], we require both the group orders $r = \#E_b(\mathbf{F}_p)$ and $r' = \#E'_b(\mathbf{F}_p)$ to be prime. We have $r = p + 1 - t$ and $r' = p + 1 + t$ for $|t| \leq 2\sqrt{p}$ and demand $|t| > 1$ because curves with $t \in \{0, 1\}$ are weak. Thus, depending on the sign of the trace $t$, either $r > p, r' < p$ or $r < p, r' > p$. To ease implementation, we demand that $r < p$ for all curves, i.e. we choose the curve with positive trace. To leave no room for manipulating the curve choice, we select all curve parameters deterministically, namely by choosing the integer $b$ with the smallest absolute value that yields a curve with the above properties. Based on these considerations, the selection process is completely explained in accordance with the *rigidity* condition of [12]. Specifically, we search for a suitable coefficient $b$ by starting with $b = 1$ and incrementing $b$ by one until both $r$ and $r'$ are prime. For each value of $b$, we use the Schoof-Elkies-Atkin (SEA) point counting algorithm [50] in Magma [17] to compute the trace $t$ of $E_b$, such that $r = p + 1 - t$ and $r' = p + 1 + t$. We use the implementation's 'early abort' feature that abandons the computation when small factors are found either in the curve's or the twist's group order. Because of the curve model for $E'_b$, the search only considers positive values of $b$ and we select the sign of $b$ to ensure that $r < p$. The resulting curves are summarized in Table 1.

### 3.2 Curve selection for twisted Edwards (and Montgomery) curves

For a fixed prime $p$, a specific twisted Edwards curve $\mathcal{E}_d : -x^2 + y^2 = 1 + dx^2y^2$ is uniquely determined by the curve parameter $d \in \mathbf{F}_p \backslash \{0, -1\}$. Let $A = 2\frac{1-d}{d+1}$, and $B = -(A + 2)$. Theorem 3.2 of [7] shows that the twisted Edwards curve $\mathcal{E}$ and the Montgomery curve $By^2 = x^3 + Ax^2 + x$ are birationally equivalent. If $B$ is a square in $\mathbf{F}_p$ (which it is for all our curves), then $\mathcal{E}_d$ is birationally equivalent to $E_A \colon y^2 = x^3 + Ax^2 + x$. As for the Weierstrass curves, we demand $t > 1$ to exclude the weak curves with $t \in \{0, 1\}$ and to ensure that $4r < p$.

Ideally, it would be desirable to have a curve with minimal parameter $d$ in the twisted Edwards form and minimal parameter $A$ in the Montgomery form. Unfortunately, existing curve proposals have been forced to pick one form and optimize it at the expense of the other one. We show in Section 3.3 below, that a search minimizing the absolute value of the parameter $d$ would find curves with the same group orders for curve and twist, where the latter corresponds to $-(d + 1)$. This means that a search for minimal absolute value of $d$ will always find positive $d$ first, which corresponds to negative $A$. Our search thus minimizes the absolute values of $A$ and $d$ at the same time.

For each fixed $p$, we start with $A = -6$ and search for $A \in 2+4\mathbf{Z}$ (subtracting 4 each time) until $\#E_A = 4r$ and $\#E'_A = 4r'$, where $r$ and $r'$ are both prime. Note that the discussion in Section 3.3 also shows that $B = -(A + 2)$ is always a square in $\mathbf{F}_p$, which means that $E'_A : y^2 = x^3 - Ax^2 + x$ is a model for the non-trivial quadratic twist of $E_A$. Again, for each $A$, we use the SEA algorithm [50] in Magma [17] to compute the trace $t$ of $E$, which determines $\#E_A = p+1-t$ and $\#E'_A = p+1+t$. Section 3.3 also shows that $A^2-4$ is non-square in $\mathbf{F}_p$, which simplifies notions of *completeness* on $E$ (see [5]). Furthermore, we check that the curve satisfies all conditions posed by [12], if one of them is not met[2], we continue with the next value for $A$. We note that the cofactors of 4 are minimal when insisting on an $\mathbf{F}_p$-rational twisted Edwards and/or Montgomery form. The resulting curves are summarized in Table 2.

### 3.3 Correspondence between minimal $A$ and $d$ for twisted Edwards curves

Table 2 contains a column with values for the parameter $d_0 = -(A+2)/4$, which can be used for implementing twisted Edwards curves defined over our prime fields. The curve $\mathcal{E}_{d_0}/\mathbf{F}_p : -x^2 + y^2 = 1 + d_0x^2y^2$ has the same number of $\mathbf{F}_p$-rational points as the curve $\mathcal{E}_d/\mathbf{F}_p : -x^2 + y^2 = 1 + dx^2y^2$ with $d = -(A-2)/(A+2)$ and the Montgomery curve $E_A/\mathbf{F}_p : y^2 = x^3 + Ax^2 + x$. Furthermore, the curve $\mathcal{E}_{-(d_0+1)}/\mathbf{F}_p : -x^2 + y^2 = 1 - (d_0+1)x^2y^2$ has the same number of $\mathbf{F}_p$-rational points as the quadratic twist $\mathcal{E}'_d$ and the quadratic twist $E_{-A}$. In this section, we show that this is true in general, and that therefore, the relation between $d_0$ and $A$ shows that the value $d_0$ is the minimal value for $d$ defining $\mathcal{E}_d$ such that all the criteria in our curve selection are satisfied if and only if $A$ is the minimal such value for the Montgomery form. This shows that it is not necessary to search for a new set of twisted Edwards curves if one wants to minimize the parameter $d$ instead of the Montgomery parameter $A$. One can simply use the curve defined by $d_0$.

The following lemma connects the two twisted Edwards curves $\mathcal{E}_d$ and $\mathcal{E}_{d_0}$ via an isogeny whenever $d_0 = -1/(d+1)$. It also gives a condition on $d_0$ which determines whether the map is defined over $F_p$. If this is the case, both curves have the same number of $\mathbf{F}_p$-rational points.

**Lemma 1.** *Let $\mathcal{E}_d : -x^2 + y^2 = 1 + dx^2y^2$ be a twisted Edwards curve defined over a prime field $\mathbf{F}_p$ and let $d_0 = -1/(d+1) \in \mathbf{F}_p$. Then there exists a 4-isogeny $\phi : \mathcal{E}_d \to \mathcal{E}_{d_0}$. If $d_0$ is a square in $\mathbf{F}_p$, the isogeny is defined over $\mathbf{F}_p$, in particular $\#\mathcal{E}_d(\mathbf{F}_p) = \#\mathcal{E}_{d_0}(\mathbf{F}_p)$.*

*Proof.* The isogeny $\phi$ is one of the isogenies described in Section 3 of [2]. This means, it is the composition of maps

$$\phi = \hat{\psi}_{-1,-1/(d+1)} \circ \sigma \circ \psi_{-1,d}.$$

The map $\psi_{-1,d}$ is the 2-isogeny $\psi_{-1,d} : \mathcal{E}_d \to L_{-d}$ to the Legendre form curve $L_{-d} : y^2 = x(x-1)(x+d)$ given in [2, Theorem 3.2], and $\hat{\psi}_{-1,-1/(d+1)} : L_{1/(d+1)} \to \mathcal{E}_{-1/(d+1)}$ is the dual of the corresponding isogeny for $1/(d+1)$. The map $\sigma$ is equal to the isomorphism $\sigma_2\sigma_1 : L_{-d} \to L_{1/(d+1)}$ given in [2, Section 3.2]. The composition $\phi$ is defined over $\mathbf{F}_p$ if $d_0$ and thus $-(d+1)$ is a square in $\mathbf{F}_p$. This proves the lemma. □

The next result uses the previous isogeny to show that the original curve $\mathcal{E}_d$ and its twist $\mathcal{E}'_d$ each have corresponding curves with small parameters $d_0$ and $-(d_0+1)$, respectively, which have the same number of $\mathbf{F}_p$-rational points, provided that both these small parameters are squares in $\mathbf{F}_p$.

[2] The only instance where the first twisted Edwards curve we found did not fulfill all of the safecurves requirements was in the search for **ed-383-mers**: the constant $A = 1629146$ corresponds to a curve-twist pair with $\#E_A = 4r$ and $E'_A = 4r'$, where $r$ and $r'$ are both prime, but the embedding degree of $E_A$ with respect to $r$ is $(r-1)/188$, which fails to meet the minimum requirement of $(r-1)/100$ imposed in [12].

**Lemma 2.** *Let $A \in \mathbf{F}_p \setminus \{-2, 2\}$, $d = -(A-2)/(A+2)$ and $d_0 = -(A+2)/4$ such that both $d_0$ and $-(d_0+1)$ are squares in $\mathbf{F}_p$. Then $\#\mathcal{E}_d(\mathbf{F}_p) = \#\mathcal{E}_{d_0}(\mathbf{F}_p)$. Moreover, $\#\mathcal{E}'_d(\mathbf{F}_p) = \#\mathcal{E}_{-(d_0+1)}(\mathbf{F}_p)$.*

*Proof.* The first part follows from Lemma 1 because $d_0 = -1/(d+1)$. Since the twist $\mathcal{E}'_d = \mathcal{E}_{1/d}$, the second part follows from Lemma 1 with $d$ replaced by $1/d$, which means that $d_0$ is replaced by $-(d_0+1)$. □

Finally, we show that indeed our search criteria, in particular the facts that $p \equiv 3 \pmod 4$ and that both group orders are not divisible by 8, imply that $d_0$ and $-(d_0+1)$ as given in our setting are squares in $\mathbf{F}_p$, which shows that the correspondence above holds.

**Lemma 3.** *Let $p \equiv 3 \pmod 4$, $d_0 \in \mathbf{F}_p$ and let $\mathcal{E}_{d_0} : -x^2 + y^2 = 1 + d_0x^2y^2$ be a twisted Edwards curve such that $\#\mathcal{E}_{d_0}(\mathbf{F}_p) = 4r$ and $\#\mathcal{E}'_{d_0}(\mathbf{F}_p) = 4r'$ for primes $r$ and $r'$. Then $d_0$ and $-(d_0+1)$ are both squares in $\mathbf{F}_p$.*

*Proof.* We first prove that $d_0$ is a square in $\mathbf{F}_p$. Assume that it is not a square. Section 3 in [8] provides an exhaustive description of all points of order 2 and 4 on a twisted Edwards curve. If $d_0$ is not a square, then $-1/d_0$ is a square because $p \equiv 3 \pmod 4$. Then the full 2-torsion is defined over $\mathbf{F}_p$, it consists of the affine point $(0, -1)$ and two points at infinity $((1:0), (\pm\sqrt{-1/d_0}))$ (written as completed points in projective space $\mathbb{P}^1 \times \mathbb{P}^1$, see [8, Section 2.7]). Let $s \in \mathbf{F}_p$ with $s^2 = -1/d_0$, then exactly one of $\pm s$ is a square, assume without loss of generality that it is $s$. Then this value gives 4 affine points $(\pm\sqrt{s}, \pm\sqrt{s})$ (signs chosen independently) of order 4 defined over $\mathbf{F}_p$. The group structure of the 4-torsion on $\mathcal{E}_{d_0}$ that is defined over $\mathbf{F}_p$ is thus $\mathbf{Z}_2 \times \mathbf{Z}_4$ and has order 8. Therefore 8 must divide $\#\mathcal{E}_{d_0}(\mathbf{F}_p)$, which contradicts our assumption that the group order is $4r$ for $r$ prime. Hence, $d_0$ is a square.

We know that the twist $\mathcal{E}'_{d_0}$ is birationally equivalent to $\mathcal{E}_{1/d_0}$, and we have already shown that $d_0$ is a square, so $1/d_0$ is a square. We can apply Lemma 1 with $d_0$ replaced by $1/d_0$, which means that $d = -(d_0+1)$, and obtain that $\#\mathcal{E}_{-(d_0+1)}(\mathbf{F}_p) = \#\mathcal{E}_{1/d_0}(\mathbf{F}_p) = 4r'$. Now looking at the 4-torsion defined over $\mathbf{F}_p$ as above yields that $-(d_0+1)$ is a square in $\mathbf{F}_p$. □

**The minimality of $d_0$.** All our selected twisted Edwards curves satisfy the conditions of the previous two lemmas. Therefore, one can choose to work with the isogenous curves defined by $d_0$ or $-(d_0+1)$, whichever is more convenient. The isogenous curves and their twists have the same orders as the original curves and their twists. Therefore all conditions required in the curve selection are satisfied with the added benefit of a small $d$-value.

We argue that $d_0$ is of minimal absolute value defining a curve that satisfies the search criteria. Assume that $A$ is a coefficient with minimal absolute value that yields a desired curve when minimizing for the Montgomery parameter (like the values for $A$ in our examples). A search that minimizes the absolute value of the parameter $d$ in the twisted Edwards model $\mathcal{E}_d$, must find the value $d_0$ (or $-(d_0+1)$) first since $A = -(4d_0+2)$. Without loss of generality, let $|d_0| < |d_0+1|$, i.e. $d_0 > 0$, otherwise interchange $d_0$ and $-(d_0+1)$. Indeed, assume that a $d_1$ with $|d_1| < |d_0|$ leads to a curve that satisfies all criteria. Let $A_1 = -(4d_1+2)$. By Lemma 3, $d_1$ and $-(d_1+1)$ are squares, then by Lemma 2, $\#\mathcal{E}_{d_1}(\mathbf{F}_p) = \#\mathcal{E}_{\hat{d}_1}(\mathbf{F}_p) = \#E_{A_1}(\mathbf{F}_p)$, where $\hat{d}_1 = -(A_1-2)/(A_1+2)$ and $\#\mathcal{E}'_{d_1}(\mathbf{F}_p) = \mathcal{E}_{-(d_1+1)}(\mathbf{F}_p) = \#\mathcal{E}_{1/\hat{d}_1}(\mathbf{F}_p) = \#E_{-A_1}(\mathbf{F}_p)$. This means that the curve $E_{A_1}$ satisfies the search criteria.

Since we fixed $d_0 > 0$, we have $A < 0$. By assumption, we have $-A = 4d_0+2 = 4|d_0|+2 > 4|d_1|+2$. Now consider the two cases $d_1 > 0$ and $d_1 < 0$. If $d_1 > 0$, then $A_1 = -(4d_1+2) < 0$, and $|A| = -A > 4|d_1| + 2 = 4d_1 + 2 = -A_1 = |A_1|$, contradicting the minimality of $A$. Similarly, if $d_1 < 0$, then $A_1 > 0$ and $|A| = -A > 4|d_1| + 2 > 4|d_1 + 1| + 2 = -4(d_1 + 1) + 2 = -(4d_1 + 2) = A_1 = |A_1|$, again a contradiction. Overall, this means that $d_0$ must be the coefficient with minimal absolute value.

### 3.4 Curve properties

In both families of curves, note that for primes of the form $2^\alpha - \gamma$, the bitlengths of $r$ and $r'$ differ by 1, since $|t| \gg \gamma$ in general; for primes of the form $2^\alpha(2^\beta - \gamma) - 1$, the bitlengths of $r$ and $r'$ are always equal when $\gamma \neq 0$. The curves in Table 2 can be used in different curve models: in the twisted Edwards model, in the Montgomery model for implementing Montgomery ladders, and also in the original Edwards model allowing complete addition formulas [11]. The latter can be seen as follows. Since $p \equiv 3 \pmod 4$, $E_A$ is birationally equivalent to an Edwards curve by [7, Theorem 3.4]. Using the maps discussed in [7, Section 3], one can show that $E_A : y^2 = x^3 + Ax^2 + x$ is birationally equivalent to $\mathcal{E}_{-1/d} : x^2 + y^2 = 1 - (1/d)x^2y^2$. For all of our curves, $d$ is a square in $\mathbf{F}_p$, so $-1/d$ is not a square, which means that the addition law on $\mathcal{E}_{-1/d}$ is complete. All of the curves in Table 2 allow for an efficient map from a subset of their $\mathbf{F}_p$-rational points to bit strings of a certain length, such that they are indistinguishable from uniform random bitstrings of the same length (see [10], which is based on [28]). However, note that curves defined over pseudo-Mersenne primes are more suitable for achieving indistinguishability than those over Montgomery-friendly primes because for the latter primes $p$, the value $(p + 1)/2$ is further away from a power of 2 (see [10, §2.6]). The prime-order Weierstrass curves presented in Table 1 are similar in their basic properties to the NIST curves, as they have the same curve model, share the parameter $a = -3$, and include prime fields of the same bit lengths as the ones for the NIST curves [56]. However, we stress that the curves in Table 1 do not allow any room for manipulations, which can be the case when the curve parameter $b$ is allowed to be chosen "randomly". Our curves are *twist-secure*, do not allow *transfers*, and have large *discriminants* (notions used to guard against certain attacks; e.g., see [12]). The work in [55] shows that indistinguishability can also be achieved for our prime-order Weierstrass curves in Table 1, however the resulting bit strings are twice as large as those that result from applying [10, 28] to the twisted Edwards curves in Table 2.

## 4 Efficient, Constant-time, and Exceptionless Scalar Multiplications

To protect against certain types of side-channel attacks [38], it is essential that scalar multiplications are computed in *constant-time*. This means that the running time of the algorithm for computing a scalar multiplication $kP$ must be independent of the scalar $k$ and the point $P$. Classical curve arithmetic formulas have exceptional cases, i.e. they do not work for all points. Having conditional statements in the code that check for these cases means the algorithms have a variable running time depending on different input cases, but simply leaving them out might lead to exceptional point attacks that produce wrong results or cause other implementation errors. In this section we outline how constant-time algorithms can be achieved efficiently for our chosen Weierstrass and twisted Edwards curves in two different settings: the variable- and fixed-base scenarios. The variable-base scenario refers to the case in which the base point $P$ can be different for each execution of the algorithm. In the fixed-base case, multiples of a public constant point can be precomputed, which allows different optimization possibilities. In Appendix A we present an algorithm for the double-scalar scenario, which carries out a computation of the form $k_1P_1 + k_2P_2$ (see Algorithm 9). This occurs for example in the verification of ECDSA signatures. In this setting the verification algorithm operates on public inputs only, and there is no need for protecting secret inputs against side-channels. Since the implementation does not have to run in constant-time, one can profit from more efficient variable-time algorithms.

We discuss the various cases for implementing scalar multiplication for the different curve models and algorithm choices. We list all algorithms as pseudo-code in Appendix A (scalar multiplication, point validation, precomputation and recoding) and in Appendix B (point operations). The reader is referred to Appendix C for complete details on the selection of explicit formulas. Note that several of these algorithms contain if-statements, which are marked in the pseudo-code according to their nature. For example, some of these statements occur in algorithms that are only run on public inputs and do not need to run in constant time; some of them are implemented in constant time via masking techniques; and some of them are there merely to allow us to represent several algorithms in one pseudo-code algorithm environment and to re-use the different variants in different scenarios. As soon as a specific scenario is chosen, these statements are always executed under the same condition. The remaining if-statements are the ones that when implemented introduce data-dependent branches into the algorithms. They occur only in algorithms for point doubling, point addition and merged point doubling/addition, where they correspond to exceptions, i.e. the exceptional cases for which the given formulas are not valid. But, whenever the implementation needs to be constant-time, the conditions for entering these if-statements are always false such that they are never executed (and can be removed in the code). Below, we argue that indeed no exceptional cases occur and that the proposed algorithms can be implemented to run in constant time (when used as described in the algorithms in Appendix A). Note that the neutral element on Weierstrass curves is the point at infinity, i.e. the point $(0\colon 1\colon 0)$ in projective coordinates, while on twisted Edwards curves the neutral element is the rational point $(0, 1)$, and in the Montgomery ladder the neutral element is $(X\colon Z) = (0\colon 0)$. In this paper, they are all denoted by $\mathcal{O}$.

### 4.1 Weierstrass Scalar Multiplications

Let $E_b/\mathbf{F}_p$ be any of the Weierstrass curves in Table 1, with $r = \#E_b(\mathbf{F}_p)$ prime. Let $k$ be an integer scalar and $P = (x_1, y_1) \in \mathbf{F}_p \times \mathbf{F}_p$. We consider the computation of efficient, constant-time and exception-free scalar multiplications in two scenarios.

**The variable-base scenario.** On input of the scalar $k$ and variable point $P = (x_1, y_1)$, perform the following steps.

1. **Validation:** Validate that $k \in [1, r)$ and that $P = (x_1, y_1) \in E_b(\mathbf{F}_p) \setminus \{\mathcal{O}\}$ by checking that $y_1^2 = x_1^3 - 3x_1 + b$. Otherwise, return false (see Algorithm 2).
2. **Precomputation:** For a fixed window size $2 \leq w < 10$, compute the $2^{w-2}$ multiples $\{P, 3P, \ldots, (2^{w-1}-1)P\}$ of $P$, and store them in a lookup table. This precomputation can be achieved using one point doubling and $2^{w-2} - 1$ point additions[3] (see Algorithm 4).
3. **Scalar recoding:** Convert the scalar $k$ to odd (if even) and recode it into exactly $\lceil \log_2(r)/(w-1) \rceil + 1$ odd, signed, non-zero digits in $\{\pm 1, \pm 3, \ldots, \pm(2^{w-1} - 1)\}$ (see Algorithm 6).
4. **Evaluation:** Compute $kP$ using a fixed window with the precomputed values from the previous step. This requires exactly $(w-1)\lceil \log_2(r)/(w-1) \rceil$ point doublings and $\lceil \log_2(r)/(w-

[3] Except for when $w = 2$, where this comes for free.

1)⌉ point additions, or $(w-2)\lceil \log_2(r)/(w-1)\rceil + 1$ point doublings, $\lceil \log_2(r)/(w-1)\rceil - 1$ point doubling-additions and one addition when $w > 2$. Note that every time an addition is performed, we also negate the selected point in the look-up table, and choose the correct one according to the sign of the digit in the recoded scalar. This is repeated until the last iteration, when crucially, the final addition is performed via a "complete masked" addition (see Appendix C.1). The final result is negated if the original value of $k$ was even.

This can be computed as outlined in Algorithm 1 in Appendix A.

**Proposition 4.** *When computing variable-base scalar multiplications on any of the Weierstrass curves in Table 1 using Algorithm 1 to implement the steps above, no exceptions occur.*

Before proving the proposition, we fix notation to partition the non-zero points in a prime order subgroup of the group $E_b(\mathbf{F}_p)$. For a fixed point $P \in E_b(\mathbf{F}_p) \setminus \{\mathcal{O}\}$, the map $[1, r) \to E_b(\mathbf{F}_p) \setminus \{\mathcal{O}\}$, $k \mapsto kP$ is a bijection. It induces a partition of $E_b(\mathbf{F}_p) \setminus \{\mathcal{O}\} = S_{\text{odd}} \cup S_{\text{even}}$ into two equally sized sets, where $S_{\text{odd}} = \{kP \mid k \in [1, r) \text{ odd}\}$ and $S_{\text{even}} = \{kP \mid k \in [1, r) \text{ even}\}$. Let $T = \{P, 3P, \ldots, (2^{w-1} - 1)P\} \subset S_{\text{odd}}$ and $T^{-1} = \{(r-1)P, (r-3)P, \ldots, (r - (2^{w-1} - 1))P\} \subset S_{\text{even}}$. The set $T^{-1}$ contains the inverses of the points in the set $T$.

*Proof.* To exclude any exceptions in the course of Algorithm 1, we consider all of its doubling, addition and merged doubling/addition operations. First of all, it is easy to see that all doubling and addition steps for building the look-up table are exception-free. Note that the look-up table consists exactly of the points in the set $T$ defined above. The precomputation as shown in Algorithm 4 starts by doubling $P$ with Algorithm 10. The algorithm works for the point at infinity $\mathcal{O}$ when defined as $(0 : Y_1 : 0)$ with $Y_1 \neq 0$, but the case $P = \mathcal{O}$ is excluded by point validation, and it does not have any exceptions since there are no points of order 2 in the group $E(\mathbf{F}_p)$. The points for the look-up table are then computed by adding $2P \in S_{\text{even}}$ to points from $T \subset S_{\text{odd}}$ only, i.e. the input points to the additions are always different and do not include $\mathcal{O}$. Also $-2P = (r-2)P$ is not among these points because $2^{w-1} - 1 < r - 2$ (note $2 \leq w < 10$).

The operations in the evaluation stage depend on the recoding of the scalar $k$, which at this point in the algorithm satisfies $0 < k < r$. Let $t = \lceil \log_2(r)/(w-1)\rceil$, then with notation as in Algorithm 1, the scalar can be written as

$$k = \sum_{i=0}^{t} s_i |k_i| 2^{(w-1)i},$$

where $s_i \in \{-1, 1\}$ and $k_i \in \mathbf{Z}$ with $0 < |k_i| < 2^{w-1}$. The recoding used here guarantees $k_t > 0$ such that $s_t = 1$ and $|k_t| = k_t$. Throughout the evaluation stage, the variable $Q$ is used to denote the running value during the algorithm. At any stage, there is some $z \in [0, r)$ such that $Q = zP$. Let $z_1 > 0$ and $z_2 = 2^{w-1} z_1 \pm z_0$ with $z_0 \in \{1, 3, \ldots, 2^{w-1} - 1\}$, then $z_2 \geq z_1$. If $z_1 > 1$, we even have $z_2 > z_1$. This means that whenever a positive integer is doubled $w - 1$ times and then an integer corresponding to one of the elements in the look-up table is either added or subtracted, the result cannot be smaller than the original integer. Thus, in the evaluation stage of Algorithm 1, after each sequence of $w - 1$ doublings and one addition step, the value $z$ of the running point $Q$ cannot decrease.

The evaluation stage begins with choosing an element from the lookup table $T$ and assigning it to $Q$. After the first assignment, we have $z \in \{1, 3, \ldots, 2^{w-1} - 1\}$. All the doubling operations in Lines 11, 14 and 18 of Algorithm 1 are done using Algorithm 10. Therefore, for the same reasons as explained above there are no exceptions possible in these steps. The last addition in Line 19 is done with a complete addition formula and hence also does not have any exceptional cases. It now suffices to ensure that all remaining addition steps (i.e. in Lines 12 and 15) do not run into exceptions.

First, assume that an exceptional case occurs in one of the additions in Step 15, which computes $Q + R$ for $R \in T \cup T^{-1}$ using Algorithm 12. Note that none of the doubling steps can ever output $\mathcal{O}$ because there are no points of order 2 and $\mathcal{O}$ is never input to any of them since the running value $Q$ always has $1 < z < r$ for all points input to doubling steps prior to any of the additions in Step 15. Thus the only exceptional cases that could occur in this algorithm, are the cases where $Q = \pm R$. This means that either $Q \in T$ or $Q \in T^{-1}$. Since $Q$ is the output of a non-trivial doubling operation, we have $Q \in S_{\text{even}}$ which excludes $Q \in T$ and means that $Q \in T^{-1}$. Therefore, $Q = zP$ with $z \geq r - (2^{w-1} - 1)$. After each addition in Step 15 there are always $w - 1$ doublings that follow. Hence, the minimal value for $z$ that can occur after the exceptional addition and the following doublings is $2^{w-1}(r - 2(2^{w-1} - 1))$. The addition of a table element immediately after these doublings, can bring down this value to the minimal $z_{\text{min}} = 2^{w-1}(r - 2(2^{w-1} - 1)) - (2^{w-1} - 1) = 2^{w-1}r - (2^w + 1)(2^{w-1} - 1)$. This value is larger than $r$, because otherwise, it follows that $r \leq 2^w + 1$, which is not true for any of our curves. Given the observation that a positive integer does not decrease after any sequence of $w - 1$ doublings and a following addition of an integer corresponding to a look-up table element, the scalar $k$ cannot be reached any more as the final value for $z$ after the exceptional addition. This contradicts any exceptions in the additions of Step 15.

Next, assume that an exception occurs in one of the steps in Line 12 of Algorithm 1. This step is a merged doubling and addition step and is computed via Algorithm 11. The algorithm computes $2Q + R$ for $R \in T \cup T^{-1}$ as $(Q + R) + Q$. For the same reasons as above, the input point $Q$ cannot be equal to $\mathcal{O}$. Since $R \in T \cup T^{-1}$, we have $R \neq \mathcal{O}$. The first addition $Q + R$ could have the same exceptions as the additions in Step 15 treated in the previous paragraph. This means that an exception can only be $Q \in T^{-1}$ as above and again we look at the minimal value $z_{\text{min}}$ after carrying out the exceptional addition, the addition of $Q$ and the following $w - 1$ doublings and subsequent addition (also the steps including the merged doubling and addition algorithm can be treated as such). This value is $z_{\text{min}} \geq 2^{w-1} \cdot (2r - 3(2^{w-1} - 1)) - (2^{w-1} - 1) = 2^w r - (3 \cdot 2^{w-1} + 1)(2^{w-1} - 1)$. Again, this value is larger than $r$, because otherwise we would have $r \leq 3 \cdot 2^{w-1} + 1$, which does not hold for our curve parameters. As above this means that the scalar $k < r$ cannot be reached as the final value of $z$, contradicting any exception in the first addition in $(Q + R) + Q$. Finally, we assume that there is an exception in the second addition. We have already excluded $Q = \mathcal{O}$ and $Q + R = \mathcal{O}$. Hence, the only two possibilities for an exception are $Q + R = Q$ or $Q + R = -Q$. The first condition means that $R = \mathcal{O}$ which is not possible since $R \in T \cup T^{-1}$. We are thus left with the condition $2Q = -R$ and hence either $2Q \in T$ or $2Q \in T^{-1}$. Since $2Q \in S_{\text{even}}$, it cannot be in $T$, which leaves $2Q \in T^{-1}$. This means that $2z \geq r - (2^{w-1} - 1)$. The minimal value $z_{\text{min}}$ after the computation $(Q + R) + Q$ and the following $w - 1$ doublings and another addition is $z_{\text{min}} \geq 2^{w-1}(r - 2(2^{w-1} - 1)) - (2^{w-1} - 1) = 2^{w-1}r - (2^w + 1)(2^{w-1} - 1)$. Again, this value is larger than $r$, leaving no way to achieve the scalar $k$ during the remaining computation. This excludes all exceptions in Line 12 and therefore all exceptions in Algorithm 1. □

Given that the recoding always produces a fixed length for the scalar, this means that after a successful validation step, we do not execute any conditional statements.

**The fixed-base scenario.** In this setting, the point $P$ is fixed (e.g., as a public parameter of the system), so multiples of $P$ can be precomputed offline and used to speedup the online computation of $kP$. In terms of performance, it might be difficult to select the "optimal" size of the precomputed table. A larger table with more multiples of $P$ typically means a reduced number of elliptic curve operations at runtime, but such tables might result in cache-misses which can result in a performance penalty. Moreover, when one wants to extract elements from this table in a cache-attack resistant manner, one should access *every* element and mask out the correct value to avoid leaking access patterns. Hence, using a larger table implies an increased access cost for every table-lookup.

This is not the only problem with large precomputed tables. As far as we know, one cannot show (for all inputs) that a current active point in the fixed-base scalar multiplication will not be the same (or have an opposite sign) as one of the many precomputed values. Although this might happen only with extremely low probability, such that honest parties may never encounter this by accident, active adversaries could manipulate such scalar/point combinations to force exceptions. This means that, unlike the variable-base multiplication, the implementation of the group law must cover exceptional cases. One solution is to use complete formulas (which have no exceptional cases). Unfortunately, the complete Weierstrass formulas from [18] (see Appendix C.1) are expensive compared to their incomplete counterparts, and using these would incur a much larger relative penalty than the complete formulas on (twisted) Edwards curves do. Another possible solution is to always compute two candidates for the addition, $C_1 = 2P$ and $C_2 = P + L$, and select (in a constant time manner) $C_1$ if $P = L$, $\mathcal{O}$ if $P = -L$, $L$ if $P = \mathcal{O}$, $P$ if $L = \mathcal{O}$, and $C_2$ otherwise. At a first glance this approach seemingly increases the cost of an addition to be at least that of computing both an addition and an doubling. However, we present a solution which achieves this same behavior *without* increasing the number of modular multiplications or squarings required in a dedicated point addition (see Algorithm 18). The idea is to exploit the similarity in the doubling and addition routines by masking out the correct operands first, and using these as inputs to the arithmetic operations. This allows us to keep the total number of multiplications and squarings exactly the same as when computing just the dedicated point addition. Hence, Algorithm 18 works for *any* input points, does not have any exceptional cases and has roughly the same run-time as a dedicated point addition.

For a scalar $k$ and the fixed point $P = (x_1, y_1)$, we make use of these formulas to perform the following steps.

*Offline computation.*

1. **Point validation:** Validate that $P = (x_1, y_1) \in E_b(\mathbf{F}_p) \setminus \{\mathcal{O}\}$ by checking that $y_1^2 = x_1^3 - 3x_1 + b$. Otherwise, return false (see Algorithm 2).
2. **Precomputation:** For a fixed window size $2 \leq w < 10$, compute $v > 0$ tables of $2^{w-1}$ points (each) for the mLSB-set comb method (see Line 2 of Algorithm 7). Convert all points in the lookup table to affine form.

*Online computation.*

3. **Scalar validation:** Validate that the scalar $k \in [1, r)$. Let the maximum bit-length of all valid scalars be $t = \lceil \log_2(r) \rceil$.
4. **Recoding:** Convert the scalar $k$ to odd (if even) and recode it into the mLSB-set representation (see Algorithm 8).

5. **Evaluation:** Using the precomputed values from the offline precomputation, compute $kP$ with exactly $\lceil \frac{t}{w \cdot v} \rceil - 1$ point doublings and $v \lceil \frac{t}{w \cdot v} \rceil - 1$ point additions[4]. All point additions are computed using the "complete masked" approach in Algorithm 18 in Appendix C.1. The final result is negated if the original value of $k$ was even.

This approach is outlined in Algorithm 7 in Appendix A.

**Proposition 5.** *When computing fixed-base scalar multiplications on any of the Weierstrass curves in Table 1 using Algorithm 7 to implement the steps above, no exceptions occur.*

*Proof.* Following the proof of Proposition 4, point doublings computed via Algorithm 10 do not fail for any rational points in $E_b(\mathbf{F}_p)$ for any of the curves $E_b$ in Table 1. Furthermore, Algorithm 10 also correctly computes doublings at the point at infinity, $\mathcal{O}$. Thus, no exceptions can arise in point doublings; and, since all online additions are implemented using the "complete" masking technique described in Appendix C.1, it follows that no exceptions can arise at any stage of the online computation (offline computations can also make use of this technique if necessary). □

### 4.2 Twisted Edwards Scalar Multiplications

Let $\mathcal{E}_d/\mathbf{F}_p\colon\ -x^2 + y^2 = 1 + dx^2y^2$ be any of the twisted Edwards curves in Table 2, with $\#E(\mathbf{F}_p) = 4r$ for $r$ prime. In a similar vein to [33, 5], we avoid small subgroup attacks by requiring all scalar multiplications to include a cofactor 4. Thus, let the integer $\hat{k}$ be defined as $\hat{k} := 4k$ with $k \in [1, r)$, and let $P = (x_1, y_1)$ be in $\mathbf{F}_p \times \mathbf{F}_p$.

**The variable-base scenario.** On input of $\hat{k}$ and (variable) $P = (x_1, y_1) \in \mathbf{F}_p \times \mathbf{F}_p$, we perform the following steps.

1. **Validation:** Validate that $\hat{k} \in [4 \cdot 1, 4 \cdot 2, \ldots, 4(r-1)]$. Validate that $P = (x_1, y_1) \in \mathcal{E}_d(\mathbf{F}_p) \setminus \{\mathcal{O}\}$ by checking that $-x_1^2 + y_1^2 = 1 + dx_1^2y_1^2$ and that $P \neq (0,1) = \mathcal{O}$ (see Algorithm 3). Otherwise, return false.
2. **Clear torsion:** Compute $Q \leftarrow [4]P$ using two consecutive doublings (as in Algorithm 3).
3. **Revalidation:** Validate that (the projective point) $Q \neq \mathcal{O}$. If not, reject.
4. **Precomputation:** Compute the $2^{w-2}$ odd, positive multiples $\{Q, 3Q, \ldots, (2^{w-1}-1)Q\}$ of $Q$, and store them in a lookup table. This precomputation can be achieved using one point doubling and $2^{w-2} - 1$ point additions[5] (see Algorithm 4).
5. **Scalar recoding:** Using a window size of $2 \leq w < 10$, convert the updated scalar $k := \hat{k}/4 \in [1, r-1]$ to odd (if even) and recode it into exactly $\lceil \log_2(r)/(w-1) \rceil + 1$ odd, signed, non-zero digits in $\{\pm 1, \pm 3, \ldots, \pm(2^{w-1}-1)\}$ (see Algorithm 6).
6. **Evaluation:** Compute $\hat{k}P$ as $kQ$, using exactly $(w-1)\lceil \log_2(r)/(w-1) \rceil$ point doublings and $\lceil \log_2(r)/(w-1) \rceil$ point additions. Note that every time an addition is performed, we also negate the selected point in the look-up table, and choose the correct one according to the sign of the digit in the recoded scalar. This is repeated until the last iteration, when crucially, the final addition is performed using the unified formula in [34, Eq. (5)]. The final result is negated if the original value of $k$ was even.

This computation is given in Algorithm 1 in Appendix A.

[4] We note that this cost increases by a single point addition when $wv \mid t$, since an extra precomputed point is needed in this case.

[5] Again, except for when $w = 2$, where this comes for free.

**Proposition 6.** *When computing variable-base scalar multiplications on any of the twisted Edwards curves in Table 2 using Algorithm 1 to implement the steps above, no exceptions occur.*

*Proof.* The first 3 steps (validation, clear torsion, and revalidation) detailed in Section 4.2 ensure that the point $Q$ has large prime order $r$. Furthermore, only elements of $\langle Q\rangle$ are encountered after the revalidation stage, meaning that Corollary 1 from [34] can be invoked to say that the additions in Algorithm 15 (from [34], but extended according to the representation suggested in [31]) will never fail to add points $P$ and $Q$ of odd order, except when $P = Q$. This corollary also tells us that the formulas for point doubling in Algorithm 14 never fail for points of odd order. Similar to the addition formulas, these doubling formulas, which are from [7], are extended according to [31]. Thus, the proof from this point is identical to the proof of Proposition 4: we partition the elements in $\langle Q\rangle \setminus \{\mathcal{O}\}$ into $S_{\text{odd}}$ and $S_{\text{even}}$ to categorize the elements in the look-up table, and use this to show that the running value that is input into point additions can never be equal to an element in the look-up table, except possibly in the final addition, where we use the formula in [34, Eq. (5)], which is slightly slower, but is exception-free in $\langle Q\rangle$. □

**The fixed-base scenario.** Let $P = (x_1, y_1) \in \mathbf{F}_p \times \mathbf{F}_p$ be a fixed point and let $\hat{k} = 4k$ be an integer scalar, which is a multiple of the cofactor 4. Then perform the following steps.

*Offline computation.*

1. **Validation:** Validate that $\hat{k} \in [4 \cdot 1, 4 \cdot 2, \ldots, 4(r-1)]$. Validate that $P = (x_1, y_1) \in \mathcal{E}_d(\mathbf{F}_p) \setminus \{\mathcal{O}\}$ by checking that $-x_1^2 + y_1^2 = 1 + dx_1^2y_1^2$ and that $P \neq (0,1) = \mathcal{O}$ (see Algorithm 3). Otherwise, return false.
2. **Clear torsion:** Compute $Q \leftarrow [4]P$ using two consecutive doublings (see Algorithm 3).
3. **Revalidation:** Validate that $Q \neq \mathcal{O}$. If not, reject.
4. **Precomputation:** For a fixed window size $2 \leq w < 10$, compute $v > 0$ tables of $2^{w-1}$ points (each) for the mLSB-set comb method (see Line 2 of Algorithm 7) – convert all points in the lookup table to affine form.

   *Online computation.*
5. **Recoding:** Convert the updated scalar $k := \hat{k}/4$ to odd (if even) and recode it into the mLSB-set representation (see Algorithm 8).
6. **Evaluation:** Using the precomputed values from the offline precomputation, compute $\hat{k}P$ as $kQ$ with exactly $\lceil \frac{t}{w \cdot v} \rceil - 1$ point doublings and $v\lceil \frac{t}{w \cdot v} \rceil - 1$ point additions[6]. Every one of these additions is computed using the unified formulas from [34, Eq. (5)]. The final result is negated if the original value of $k$ was even.

Algorithm 7 in Appendix A outlines this computation.

**Proposition 7.** *When computing fixed-base scalar multiplications on any of the twisted Edwards curves in Table 2 using Algorithm 7 to implement the steps above, no exceptions occur.*

*Proof.* As in the proof of Proposition 6, we start by noting that the (updated) point $Q$ has odd order $r$, and that we only compute on elements in $\langle Q\rangle$. The only algorithm we use for online additions corresponds to the formulas in [34, Eq. (5)], which do not fail for any pair of inputs in $\langle Q\rangle$. Additionally, the only algorithm we use for doublings is Algorithm 14 (from [7]), which is also exception-free on all inputs from $\langle Q\rangle$. □

[6] Again, we note that when $wv \mid t$, an extra precomputed point is needed.

### 4.3 The Montgomery Ladder

Let $E_A/\mathbf{F}_p \colon y^2 = x^3 + Ax^2 + x$ be the Montgomery form of any of the curves in Table 2, with $\#E_A(\mathbf{F}_p) = 4r$, for $r$ a large prime. Since the Montgomery ladder is not compatible with the recoding techniques discussed in Section 4, we take the following route to guarantee a fixed length scalar. For all $k \in [1, r-1]$, we use the updated scalar $\hat{k} = 4(\alpha r + k)$, where $\alpha$ is the smallest positive integer such that $\alpha r + 1$ and $(\alpha + 1)r - 1$ have the same bitlength; $\alpha$ is specific to $r$, but for each of the curves in Table 2 we have $\alpha \in \{1, 2, 3\}$. Note that scalar multiplication by $\hat{k}$ corresponds to scalar multiplication by $4k$ on $E_A$, which thwarts small subgroup attacks in the same vein as the twisted Edwards scalar multiplications in Section 4.2.

On input of $\hat{k}$ and $x_1 \in \mathbf{F}_p$, we perform the following steps.

1. **Scalar validation:** First validate that $\hat{k} \in 4\mathbb{Z}$, and then that the integer $\hat{k}/4 \in [\alpha r + 1, (\alpha+1)r - 1]$. Otherwise, reject.
2. **Evaluation:** Process the scalar by inputting $\hat{k}$ and $(x_1 \colon 1)$ into the standard $(X \colon Z)$-only Montgomery ladder routine [46, §10], with constant $(A+2)/4$ in the addition formula. Since $\hat{k} = 4(\alpha r + k)$, this can be done by inputting the fixed-length scalar $\hat{k}/4 = \alpha r + k$ and $(x_1 \colon 1)$ into the Montgomery ladder to give $(X_1 \colon Z_1)$, before finishing with two repeated, standalone Montgomery doublings of $(X_1 \colon Z_1)$ to give $(\hat{X} \colon \hat{Z}) = 4(X_1 \colon Z_1)$
3. **Normalize:** If $\hat{Z} = 0$, return $\mathcal{O}$, otherwise return $\hat{x}_1 = \hat{X}/\hat{Z}$.

Notice that there is no validation of the input coordinate $x_1 \in \mathbf{F}_p$, i.e. that we do not check whether $x_1^3 + Ax_1^2 + x_1$ is a square in $\mathbf{F}_p$, so that $x_1$ corresponds to a point (or points) on $E_A$. Avoiding this check in the presence of twist-security is due to Bernstein (cf. [5]), since even if $x_1$ corresponds to a point on the quadratic twist $E'_A$, the output of the Montgomery ladder corresponds to a scalar multiplication on $E'_A$, because scalar multiplications on both curves use the same constant $(A+2)/4$. In this case, multiplication by $\hat{k} = 4(\alpha r + k)$ on $E'_A$ no longer corresponds to the scalar $4k$, but rather to the scalar $4k'$, where $k' \equiv (\alpha r + k) \bmod r'$ for $\#E'_A(\mathbf{F}_p) = 4r'$. This is not a problem in practice since the cofactor of 4 still clears torsion on the twist, and the twist-security ensures that the discrete logarithm problem has a similar difficulty in $E'_A(\mathbf{F}_p)$ as it does in $E_A(\mathbf{F}_p)$. Following the arguments developed in [4] (see also [5, App. A-B]), it could be possible to prove that no exceptions can occur in Montgomery ladder implementations of the curves in Table 2 that follow Steps 1-3 above, subject to addressing the issues below.

It should first be pointed out that the lack of validation means that there are some scalar/point combinations which could produce exceptions. For example, suppose $k$ is chosen as the unique integer less than $r'$ such that $k \equiv -\alpha r \bmod r'$. If $k$ is also less than $r$, then $\hat{k} := 4(\alpha r + k)$ is a valid scalar according to Step 1 above. But, if an unvalidated $x$-coordinate, say $x'_1$, corresponds to a point $P'_1$ on $E'_A$, then $\hat{k}P_1 = \mathcal{O}$, because $(\alpha r + k) \equiv 0 \bmod r'$; note that outputting $\mathcal{O}$ in Step 3 above could leak information to an attacker. Furthermore, in practice these ladder implementations are often used in conjunction with non-ladder implementations on (most likely a twisted Edwards model of) the same curve – see Section 6. In such a scenario, the refined forms of the scalars in this section do not match the forms of the scalars in Section 4.2, so if the scalars above were to be used on the twisted Edwards form of $E_A$, then Proposition 6 and Proposition 7 no longer provide any guarantees. More specifically, if an implementation synchronizes the inherently larger Montgomery ladder scalars above to also be used on the twisted Edwards curve, then the argument of $\hat{k} \in [4, 8, \ldots, 4(r-1)]$ that was used in the proof of Proposition 6 no longer holds when $\alpha > 0$. Roughly speaking, the fact that $\hat{k}/4$ is now outside the range $[1, r-1]$ means that the running multiple of an input point can now reach the *dangerous* stage of a scalar multiplication (which we handle by using complete additions) before the final addition.

In the Montgomery ladder implementation of Curve25519 [5], and in the complementary Edwards "Ed25519" implementation [9], it seems that the above problems are overcome by restricting the set of permissible scalars to be of a lesser cardinality than the prime subgroup order. Namely, Curve25519 has $r, r' > 2^{252}$, with all scalars being of the form $\hat{k} = 8 \cdot (2^{251} + k)$ for $k \in [0, 2^{251} - 1]$. As well as guaranteeing that all of the possible scalars $\hat{k}$ have the same bitlength, this prevents the existence of a $\hat{k}$ such that $\hat{k} \equiv 0 \bmod r$ or $\hat{k} \equiv 0 \bmod r'$. On the other hand, it also means that for a fixed base point $P$ of order $r$ on Ed25519, less than half of the elements in $\langle P \rangle$ are possible outputs when computing scalar multiplications of $P$.

As one potential alternative, we remark that a hybrid solution which uses both Montgomery and twisted Edwards scalar multiplications could parse scalars differently: $k \in [0, r-1]$ could be modified to $\hat{k} := 4(\alpha r + k)$ in the Montgomery implementation, but modified to $\hat{k} := 4k$ in the twisted Edwards implementation. If, in addition, all $x$-coordinates were validated in Step 1 of the Montgomery ladder routine[7], then this may well be enough to prove that all scalar multiplications compute correctly and without exception: Proposition 6 would then apply directly to the twisted Edwards part, while the techniques in [5, 4] could be used to prove the Montgomery ladder part.

## 5 Implementation Results

To evaluate the performance of the selected curves, we developed a software library[8] that includes support for three scenarios: variable-based, fixed-based and double-scalar multiplication. The library can perform arithmetic on $a = -1$ twisted Edwards, $a = -3$ Weierstrass, and Montgomery curves and supports all of the new curves from Section 3 (see Tables 1 and 2). The implementation of the library is largely in the C-programming language with the modular arithmetic implemented in x64 assembly. We plan to add support for other popular platforms like the ARM architecture, and explore implementation options using vector instructions and methods such as Karatsuba multiplication [36] for larger moduli sizes.

Table 3 shows the performance details of scalar multiplication in the three scenarios of interest. Variable-base scalar multiplication is computed with the fixed-window method (see Algorithm 1 in Appendix A) using window width $w = 6$, except for twisted Edwards at the 256-bit security level which uses $w = 7$. Fixed-base scalar multiplication was computed using the mLSB-set method (see Algorithm 7 in Appendix A) using parameters $w = 6$ and $v = 3$ for the Weierstrass curves and $w = 5$ and $v = 4$ for the twisted Edwards curves. These values correspond to precomputed tables of sizes: 6KB, 9KB and 12KB at the 128-, 192- and 256-bit security levels, respectively. Double-base scalar multiplication was computed using the $w$NAF method with interleaving (see Algorithm 9 in Appendix A) using window width $w_1 = 6$ for the variable base and $w_2 = 7$ for the fixed base. The latter corresponds to precomputed tables with sizes: 2KB, 3KB and 4KB for Weierstrass curves at the 128-, 192- and 256-bit security levels, respectively, and 3KB, 4.5KB and 6KB for twisted Edwards curves at the 128-, 192- and 256-bit security levels, respectively. The results (expressed in

[7] Validating that $x_1 \in \mathbf{F}_p$ corresponds to $E_A$ would incur the small relative cost of an exponentiation and a few multiplications: namely, we reject $x_1$ if $(x_1^3 + Ax_1^2 + x_1)^{(p-1)/2} = -1$.

[8] A version of the library that supports a subset of the curves presented in this work is available at http://research.microsoft.com/en-us/downloads/149804d4-b5f5-496f-9a17-a013b242c02d/.

**Table 3.** Experimental results for variable-base, fixed-base and double-scalar multiplication. The results (rounded to thousand cycles) are the average of $10^4$ runs of the scalar multiplication including the final modular inversion to convert the result to its affine form. These results have been obtained on a 3.4GHz Intel Core i7-2600 Sandy Bridge processor with Intel's Turbo Boost and Hyper-threading disabled. The library was compiled with Visual Studio 2012 on Windows 7 OS.

| security level | curve name | variable-base | fixed-base | double-scalar |
|---|---|---|---|---|
| 128 | **w-256-mont** | 280,000 | 115,000 | 287,000 |
| | **w-256-mers** | 281,000 | 119,000 | 288,000 |
| | **ed-256-mont** | 234,000 | 93,000 | 237,000 |
| | **ed-254-mont** | 196,000 | 76,000 | 198,000 |
| | **ed-256-mers** | 234,000 | 95,000 | 237,000 |
| | **ed-255-mers** | 226,000 | 94,000 | 228,000 |
| | **m-254-mont** | 224,000 | N/A | N/A |
| | **m-255-mers** | 245,000 | N/A | N/A |
| 192 | **w-384-mont** | 795,000 | 281,000 | 812,000 |
| | **w-384-mers** | 744,000 | 277,000 | 761,000 |
| | **ed-384-mont** | 659,000 | 243,000 | 672,000 |
| | **ed-382-mont** | 591,000 | 225,000 | 605,000 |
| | **ed-384-mers** | 617,000 | 231,000 | 624,000 |
| | **ed-383-mers** | 596,000 | 227,000 | 607,000 |
| | **m-382-mont** | 672,000 | N/A | N/A |
| | **m-383-mers** | 666,000 | N/A | N/A |
| 256 | **w-512-mont** | 1,762,000 | 576,000 | 1,821,00 |
| | **w-512-mers** | 1,543,000 | 517,000 | 1,592,000 |
| | **w-521-mers** | 1,881,000 | – | – |
| | **ed-512-mont** | 1,461,000 | 504,000 | 1,507,000 |
| | **ed-510-mont** | 1,338,000 | 466,000 | 1,369,000 |
| | **ed-512-mers** | 1,293,000 | 446,000 | 1,320,000 |
| | **ed-511-mers** | 1,241,000 | 436,000 | 1,274,000 |
| | **ed-521-mers** | 1,552,000 | – | – |
| | **m-510-mont** | 1,496,000 | N/A | N/A |
| | **m-511-mers** | 1,392,000 | N/A | N/A |
| | **m-521-mers** | 1,729,000 | – | – |

terms of computer cycles) were obtained by running and averaging $10^4$ iterations of each computation on an Intel Core i7-2600 (Sandy Bridge) processor with Intel's turbo boost and hyper-threading disabled. The variable- and fixed-base scalar multiplication routines have a constant running time which guards against various types of timing attacks [38, 20], including cache attacks [49] (e.g., see [19] in the asymmetric setting). This means that no conditional branches on secret data or secret indexes for table lookups are allowed in the implementations.

Our results suggest that reducing the size of the pseudo-Mersenne primes does not have a significant effect on the performance: below a factor 1.04 reduction of the running time at the expense of roughly half a bit of ECDLP security. However, using slightly smaller moduli in the setting of the Montgomery-friendly primes does pay off: a reduction of the running time by a factor 1.19, 1.12, and 1.09 at the 128-, 192-, and 256-bit security level, respectively. This performance difference between pseudo-Mersenne and Montgomery-friendly primes can be explained by the fact that the final constant-time conditional subtraction in Montgomery multiplication can be omitted when reducing the modulus size appropriately. The size-reduced Montgomery-friendly primes are the best choice (with respect to performance) at the 128- and 192-bit security levels while the size-reduced pseudo-Mersenne prime is faster for the 256-bit security level. For full-word length moduli, Montgomery-friendly and pseudo-Mersenne primes achieve similar performance at the 128-bit security level, whereas full-word length pseudo-Mersenne moduli are the best option for the 192- and 256-bit security levels. The better performance of pseudo-Mersenne primes at high security levels can be explained by the inherent higher register pressure in our Montgomery-friendly implementations which results in more load and store operations for large moduli sizes. The faster arithmetic operations in the base field translate directly to optimizations in the different scenarios for the scalar multiplication. For the sake of comparison, we also include performance numbers for curves defined over the Mersenne prime $p = 2^{521} - 1$ (at the 256-bit security level). The performance of **w-521-mers** and **ed-521-mers** compared to our corresponding 512-bit Weierstrass and twisted Edwards curves at the 256-bit security level (**w-512-mers** and **ed-512-mers**) degrades by a factor 1.20. This is mainly due to the higher cost of modular multiplication, which performs computations over elements containing one more computer word in comparison with the 512-bit prime options.

In the setting of the variable-base scalar multiplication our twisted Edwards implementation consistently outperforms the Montgomery ladder. This does not come as a surprise since our twisted Edwards curves allow one to use the efficient curve arithmetic from [34] and the efficient fixed-window method which exploits precomputations. Note that the state-of-the-art Montgomery ladder implementation of Curve25519 [5] is 1.15 times faster than our ladder algorithm at the 128-bit security level (when considering the benchmark machine "sandy" [13]). This can largely be explained by the significant level of code optimization that went into the implementation of [5], which includes the fine-tuning of the full curve arithmetic at the assembly level. Still, without resorting to full use of assembly, our best result using twisted Edwards form ($196,000$ cycles) is virtually equivalent to the Montgomery ladder performance numbers from Curve25519 ($194,000$ cycles).

The recent software records for the NIST P-256 curve [30] can compute a variable-base scalar multiplication in 400 thousand cycles on a Sandy Bridge CPU. Our curve **w-256-mont** offers better security properties and results in a 1.43 times reduction of the running time compared to [30]. When switching from prime order Weierstrass curves using full size moduli to composite order twisted Edwards curves with size-reduced moduli one can expect a reduction in the running time by a factor between 1.24 and 1.43 at the price of a slight decrease in ECDLP security.

## 6 Real-World Protocols

Although significant research has been devoted to optimize the most popular ECC operation (the variable-base scalar multiplication), in real-world cryptographic solutions it is often not as simple as computing just a single scalar multiplication with an unknown base. Cryptographic protocols typically require a combination of different types of scalar multiplications including fixed-, variable-base and multiple-scalar operations. In this section we study the TLS protocol, more specifically the computation of the TLS handshake using the ECDHE-ECDSA cipher suite. We outline the impact of using different curve and coordinate systems in practice.

**TLS with perfect forward secrecy.** Support for using elliptic curves in the TLS protocol is specified in RFC 4492 [14]. The cipher suites specified in this RFC use the elliptic curve Diffie-Hellman (ECDH) key exchange, whose keys may either be long-term or ephemeral. We focus our analysis on the latter case (denoted by ECDHE) since it offers perfect forward

**Table 4.** Costs *estimates* for the TLS handshake using the ECDHE-ECDSA cipher suite for different security levels where we consider the elliptic curve scalar multiplications. Costs in cycles are estimated using the performance numbers from Table 3. Estimates for the *total cost* correspond to the full handshake ECDHE-ECDSA involving authentication in both the server and client side. We assume the use of precomputed tables with 96 and 64 points to accelerate fixed-base scalar multiplication on the Weierstrass and twisted Edwards curves, respectively. Similarly, we assume the use of precomputed tables with 32 points to accelerate double-base scalar multiplication (where one base is fixed). For comparison we state performance numbers for NIST P-256 [30] which uses 150KB of storage, and signature performance numbers when using EdDSA [9] (obtained from the benchmark machine "sandy" [13]). We consider that point transmission (T) in the key-exchange can be performed in uncompressed (U) or compressed (C) form.

| security level | curve names | T | estimated cost (in cycles) | | | |
|---|---|---|---|---|---|---|
| | | | ECDHE | ECDSA sign | ECDSA ver | total cost |
| 128 | **w-256-mont** | U | 395,000 | 115,000 | 287,000 | 797,000 |
| | | C | 416,000 | | | 818,000 |
| | **ed-254-mont** | U | 272,000 | 76,000 | 198,000 | 546,000 |
| | | C | 288,000 | | | 562,000 |
| | hybrid<br>**ed-254-mont + m-254-mont** | U | 300,000 | 76,000 | 198,000 | 574,000 |
| | | C | | | | |
| | NIST P-256 [30] | U | 490,000 | 90,000 | 530,000 | 1,110,000 |
| | Ed25519 [9] | C | N/A | 69,000 | 225,000 | N/A |
| 192 | **w-384-mers** | U | 1,021,000 | 277,000 | 761,000 | 2,059,000 |
| | | C | 1,078,000 | | | 2,116,000 |
| | **ed-382-mont** | U | 816,000 | 225,000 | 605,000 | 1,646,000 |
| | | C | 865,000 | | | 1,695,000 |
| | hybrid<br>**ed-382-mont + m-382-mont** | U | 897,000 | 225,000 | 605,000 | 1,727,000 |
| | | C | | | | |
| 256 | **w-512-mers** | U | 2,060,000 | 517,000 | 1,592,000 | 4,169,000 |
| | | C | 2,168,000 | | | 4,277,000 |
| | **ed-511-mers** | U | 1,677,000 | 436,000 | 1,274,000 | 3,387,000 |
| | | C | 1,779,000 | | | 3,489,000 |
| | hybrid<br>**ed-511-mers + m-511-mers** | U | 1,828,000 | 436,000 | 1,274,000 | 3,538,000 |
| | | C | | | | |

secrecy. Besides the usage of elliptic curves in the DH key exchange, TLS certificates contain a public key that the server uses to authenticate itself: this is an ECDSA public key for the case of the ECDHE-ECDSA cipher suite. The TLS handshake, using the ECDHE-ECDSA cipher suite, consists of three main components. The ECDSA signature generation (fixed-base scalar multiplication), ECDSA signature verification (double-base scalar multiplication), and ECDHE (one fixed- and one variable-base scalar multiplication). We consider Weierstrass and twisted Edwards curves separately, with and without point compression. The cost of decompressing a point in Weierstrass and twisted Edwards form is stated in Table 7 (where we follow the approach described in [9] to decompress points on twisted Edwards curves).

When using Weierstrass curves the situation is not complicated: transmitting compressed points costs a single conversion while no additional cost is needed when transmitting uncompressed points. In the setting of twisted Edwards curves there are more possibilities. The simplest approach is to only use the Montgomery form; however, this is expensive since the Montgomery ladder cannot take advantage of the fixed-base setting. One might consider a hybrid solution: computing the fixed-base scalar multiplication using the birationally equivalent twisted Edwards curve while computing the variable-base scalar multiplication using the Montgomery ladder. In such a hybrid solution the protocol should specify if the coordinates are transmitted in (compressed) twisted Edwards or Montgomery coordinates (which are already in compressed form). When using such a hybrid solution in the setting of ECDHE, transmitting the points in Montgomery form is best (see Table 7). The cost for the conversion (between Montgomery and twisted Edwards) is roughly the same as when only using twisted Edwards curves and transmitting compressed points. Our performance numbers suggest that the approach using only twisted Edwards is slightly faster than such a hybrid approach using the Montgomery ladder, while it avoids conversions between coordinate systems. Furthermore, our Montgomery ladder implementations do not include the extra validation step discussed at the end of Section 4.3; if incorporated, this would incur additional overhead.

Table 4 gives the cost estimates for the separate components and total cost of the TLS handshake using the ECDHE-ECDSA cipher suite for different security levels. The results show that the use of twisted Edwards for the ECDHE and full ECDHE-ECDSA computations are approximately a factor 1.45, 1.25 and 1.23 faster in comparison to the Weierstrass curves at the 128-, 192- and 256-bit security levels, respectively. We also include the results from [30] when using NIST P-256. In [30] the fixed-base scalar multiplication is implemented using a relatively large (slightly over 150KB) lookup table for the fixed-base scalar multiplication. It is unclear if this implementation accesses the table-lookup elements in a cache-attack resistant manner and if the dedicated addition formula used takes care of exceptions, and if so if this is done in constant time. This might explain the faster implementation results. In order to compare to the state-of-the-art software implementation of twisted Edwards curves we also include the results from EdDSA using Ed25519 [9] (obtained from the "sandy" benchmark machine" [13]). Note that [9] only computes signatures; when computing ECDH one could use the approach as described in [5] which uses the Montgomery ladder. In order to achieve perfect forward secrecy (ECDHE), the implementation can compute the fixed-base scalar multiplication using the Montgomery ladder (which is slow) or convert the point and compute the fixed-base scalar multiplication using the corresponding twisted Edwards curve (using a hybrid approach).

## 7 Conclusions

In this paper we have presented new elliptic curves for cryptography targeting the 128-, 192-, and 256-bit security levels. By considering different choices for the base field arithmetic, pseudo-Mersenne and Montgomery-friendly primes, we deterministically selected efficient twisted Edwards curves as well as traditional Weierstrass curves. Instead of resorting to the slower complete formulas, we show how to compute efficient scalar multiplications by using constant-time, exceptionless, dedicated group operations. For the cases in which they are not guaranteed to be exceptionless, we have proposed an efficient "complete" addition formula based on masking techniques for Weierstrass curves. Our implementation of the scalar multiplication in the three most-widely deployed scenarios show that our new backwards compatible Weierstrass curves offer enhanced security properties while improving the performance compared to the standard NIST Weierstrass curves. At the expense of at most a few bits of ECDLP security, our new twisted Edwards curves offer a performance increase of a factor 1.2 to 1.4 compared to our new Weierstrass curves. We demonstrated the potential cryptographic impact by showing cost estimates for these curves inside the TLS handshake protocol.

**Acknowledgments.** We thank Niels Ferguson, Thorsten Kleinjung, Dan Shumow and Greg Zaverucha for their valuable feedback, comments, and help.

## References

1. T. Acar and D. Shumow. Modular reduction without pre-computation for special moduli. Technical report, Microsoft Research, 2010.
2. O. Ahmadi and R. Granger. On isogeny classes of edwards curves over finite fields. *Journal of Number Theory*, 132(6):1337 – 1358, 2012.
3. D. F. Aranha, P. S. L. M. Barreto, G. C. C. F. Pereira, and J. E. Ricardini. A note on high-security general-purpose elliptic curves. Cryptology ePrint Archive, Report 2013/647, 2013. `http://eprint.iacr.org/`.
4. D. J. Bernstein. Can we avoid tests for zero in fast elliptic-curve arithmetic?, 2006. `http://cr.yp.to/papers.html#curvezero`.
5. D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography – PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, 2006.
6. D. J. Bernstein. Counting points as a video game, 2010. Slides of a talk given at Counting Points: Theory, Algorithms and Practice, April 19, University of Montreal: `http://cr.yp.to/talks/2010.04.19/slides.pdf`.
7. D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards curves. In S. Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *LNCS*, pages 389–405. Springer, 2008.
8. D. J. Bernstein, P. Birkner, T. Lange, and C. Peters. ECM using edwards curves. *Math. Comput.*, 82(282), 2013.
9. D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.
10. D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *ACM Conference on Computer and Communications Security*, 2013.
11. D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. In K. Kurosawa, editor, *ASIACRYPT*, volume 4833 of *LNCS*, pages 29–50. Springer, 2007.
12. D. J. Bernstein and T. Lange. SafeCurves: choosing safe curves for elliptic-curve cryptography. `http://safecurves.cr.yp.to`, accessed 16 October 2013.
13. D. J. Bernstein and T. Lange (editors). eBACS: ECRYPT Benchmarking of Cryptographic Systems. `http://bench.cr.yp.to`, accessed February 3rd 2014.
14. S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic curve cryptography (ECC) cipher suites for transport layer security (TLS). RFC 4492, 2006.
15. J. W. Bos, C. Costello, H. Hisil, and K. Lauter. Fast cryptography in genus 2. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *LNCS*, pages 194–210. Springer, 2013.
16. J. W. Bos, J. A. Halderman, N. Heninger, J. Moore, M. Naehrig, and E. Wustrow. Elliptic curve cryptography in practice (to appear). In *Financial Cryptography and Data Security*, LNCS. Springer, 2014.
17. W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).
18. W. Bosma and H. W. Lenstra. Complete systems of two addition laws for elliptic curves. *Journal of Number Theory*, 53(2):229–240, 1995.
19. B. B. Brumley and R. M. Hakala. Cache-timing template attacks. In M. Matsui, editor, *ASIACRYPT*, volume 5912 of *LNCS*, pages 667–684. Springer, 2009.
20. D. Brumley and D. Boneh. Remote timing attacks are practical. In S. Mangard and F.-X. Standaert, editors, *Proceedings of the 12th USENIX Security Symposium*, volume 6225 of *LNCS*, pages 80–94. Springer, 2003.
21. Certicom Research. Standards for efficient cryptography 2: Recommended elliptic curve domain parameters. Standard SEC2, Certicom, 2000.
22. D. Chudnovsky and G. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Advances in Applied Mathematics*, 7(4):385–434, 1986.
23. ECC Brainpool. ECC Brainpool Standard Curves and Curve Generation. `http://www.ecc-brainpool.org/download/Domain-parameters.pdf`, 2005.
24. H. M. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44:393–422, July 2007.
25. J.-C. Faugère, L. Perret, C. Petit, and G. Renault. Improving the complexity of index calculus algorithms in elliptic curves over binary fields. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT*, volume 7237 of *LNCS*, pages 27–44. Springer, 2012.
26. A. Faz-Hernández, P. Longa, and A. Sánchez. Efficient and secure algorithms for GLV-based scalar multiplication and their implementation on GLV-GLS curves (extended version). In *Cryptology ePrint Archive, Report 2013/158*, 2013. Available at: `http://eprint.iacr.org/2013/158`.

27. M. Feng, B. Zhu, M. Xu, and S. Li. Efficient comb elliptic curve multiplication methods resistant to power analysis. In *Cryptology ePrint Archive, Report 2005/222*, 2005. Available at: `http://eprint.iacr.org/2005/222`.
28. P.-A. Fouque, A. Joux, and M. Tibouchi. Injective encodings to elliptic curves. In C. Boyd and L. Simpson, editors, *ACISP*, volume 7959 of *LNCS*, pages 203–218. Springer, 2013.
29. R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In J. Kilian, editor, *CRYPTO*, volume 2139 of *LNCS*, pages 190–200. Springer, 2001.
30. S. Gueron and V. Krasnov. Fast prime field elliptic curve cryptography with 256 bit primes. Cryptology ePrint Archive, Report 2013/816, 2013. `http://eprint.iacr.org/`.
31. M. Hamburg. Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309, 2012. `http://eprint.iacr.org/`.
32. M. Hamburg. Twisting edwards curves with isogenies. Cryptology ePrint Archive, Report 2014/027, 2014. `http://eprint.iacr.org/`.
33. D. Hankerson, A. Menezes, and S. Vanstone. *Guide to elliptic curve cryptography.* Springer Verlag, 2004.
34. H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson. Twisted Edwards curves revisited. In J. Pieprzyk, editor, *Asiacrypt 2008*, volume 5350 of *LNCS*, pages 326–343. Springer, Heidelberg, 2008.
35. M. Joye and M. Tunstall. Exponent recoding and regular exponentiation algorithms. In M. Joye, editor, *Proceedings of Africacrypt 2003*, volume 5580 of *LNCS*, pages 334–349. Springer, 2009.
36. A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. In *Soviet physics doklady*, volume 7, page 595, 1963.
37. M. Knežević, F. Vercauteren, and I. Verbauwhede. Speeding up bipartite modular multiplication. In M. Hasan and T. Helleseth, editors, *Arithmetic of Finite Fields – WAIFI 2010*, volume 6087 of *LNCS*, pages 166–179. Springer Berlin / Heidelberg, 2010.
38. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Crypto 1996*, volume 1109 of *LNCS*, pages 104–113. Springer, Heidelberg, 1996.
39. A. K. Lenstra. Generating RSA moduli with a predetermined portion. In K. Ohta and D. Pei, editors, *Asiacrypt'98*, volume 1514 of *LNCS*, pages 1–10. Springer Berlin / Heidelberg, 1998.
40. C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In Y. Desmedt, editor, *CRYPTO*, volume 839 of *LNCS*, pages 95–107. Springer, 1994.
41. P. Longa and C. Gebotys. Efficient techniques for high-speed elliptic curve cryptography. In S. Mangard and F.-X. Standaert, editors, *Proceedings of CHES 2010*, volume 6225 of *LNCS*, pages 80–94. Springer, 2010.
42. P. Longa and A. Miri. New composite operations and precomputation scheme for elliptic curve cryptosystems over prime fields. In R. Cramer, editor, *Proceedings of PKC 2008*, volume 4939 of *LNCS*, pages 229–247. Springer, 2008.
43. N. Meloni. New point addition formulae for ECC applications. In C. Carlet and B. Sunar, editors, *Workshop on Arithmetic of Finite Fields (WAIFI)*, volume 4547 of *LNCS*, pages 189–201. Springer, 2007.
44. B. Möller. Algorithms for multi-exponentiation. In S. Vaudenay and A. M. Youssef, editors, *Selected Areas in Cryptography*, volume 2259 of *LNCS*, pages 165–180. Springer, 2001.
45. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, April 1985.
46. P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.
47. National Security Agency. Fact sheet NSA Suite B Cryptography. `http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml`, 2009.
48. K. Okeya and T. Takagi. The width-$w$ NAF method provides small memory and fast elliptic curve scalars multiplications against side-channel attacks. In M. Joye, editor, *Proceedings of CT-RSA 2003*, volume 2612 of *LNCS*, pages 328–342. Springer, 2003.
49. D. A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: The case of AES. In D. Pointcheval, editor, *CT-RSA*, volume 3860 of *LNCS*, pages 1–20. Springer, 2006.
50. R. Schoof. Counting points on elliptic curves over finite fields. *Journal de théorie des nombres de Bordeaux*, 7(1):219–254, 1995.
51. D. Shumow and N. Ferguson. On the possibility of a back door in the NIST SP800-90 dual ec prng. `http://rump2007.cr.yp.to/15-shumow.pdf`, 2007.
52. J. A. Solinas. Generalized Mersenne numbers. Technical Report CORR 99–39, Centre for Applied Cryptographic Research, University of Waterloo, 1999.
53. J. A. Solinas. Efficient arithmetic on Koblitz curves. *Designs, Codes and Cryptography*, 19(195–249), 2000.

54. The New York Times. Government announces steps to restore confidence on encryption standards. `http://bits.blogs.nytimes.com/2013/09/10/government-announces-steps-to-restore-confidence-on-encryption-standards`, 2013.
55. M. Tibouchi. Elligator squared: Uniform points on elliptic curves of prime order as uniform random strings. Cryptology ePrint Archive, Report 2014/043, 2014. `http://eprint.iacr.org/`.
56. U.S. Department of Commerce/National Institute of Standards and Technology. Digital Signature Standard (DSS). FIPS-186-4, 2013. `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`.
57. C. D. Walter. Montgomery exponentiation needs no final subtractions. *Electronics Letters*, 35(21):1831–1832, 1999.

## A Algorithms for Scalar Multiplication

**Algorithms for variable-base scalar multiplication.** Algorithm 1 computes scalar multiplication for the variable-base scenario using the fixed-window method from [48]. We refer to Sections 4.1 and 4.2 for details on its usage with Weierstrass and twisted Edwards curves, respectively. The computation of this operation mainly consists of four different stages: input and point validation, precomputation, recoding and evaluation. Input and point validation are computed at the very beginning of the execution using Algorithm 2 for Weierstrass curves and Algorithm 3 for twisted Edwards curves. In particular, Algorithm 3 performs two doublings over the input point in twisted Edwards to ensure that subsequent computations are performed in the large prime order subgroup (avoiding small subgroup attacks). We remark that it is the protocol implementer's responsibility to ensure that timing differences during the detection of errors do not leak sensitive information to an attacker. In the precomputation stage, the implementer should first select a window width $2 \leq w < 10$ according to efficiency and/or memory considerations. For example, selecting $w = 6$ for 256-, 384- and 512-bit scalar multiplication was found to achieve optimal performance in our implementations of Weierstrass curves. Precomputation is then computed by successively executing $P + 2P + 2P + \ldots + 2P$ with $2^{w-2} - 1$ point additions and storing the intermediate results. Explicit schemes are given in Algorithms 4 and 5 for $a = -3$ Weierstrass and $a = -1$ twisted Edwards curves, respectively. In the recoding stage, we use a variant of the regular recoding by [35] that ensures fixed length (see Algorithm 6). Since Algorithm 6 only recodes odd integers, we include a conversion step at Step 6 to deal with even values. The corresponding correction is performed at Step 20. These computations should be executed in constant time to protect against timing attacks. For example, a constant time execution of Step 6 could be implemented as follows (assuming a two's complement representation in which $-1 \equiv$ 0xFF...FF, and bitlength($odd$) = bitlength($k$)):

$odd = -(k \text{ AND } 1)$ {If $k$ is even then $odd$ =0xFF...FF else $odd = 0$}
$k' = k - r$
$k = (odd \text{ AND } (k \text{ XOR } k')) \text{ XOR } k'$ {If $odd = 0$ then $k = k - r$}

The main computation in the evaluation stage consists of $t = \lceil \log_2(r)/(w-1) \rceil$ iterations each computing $(w-1)$ doublings and one addition with a value from the precomputed table. For $a = -3$ Weierstrass curves, the use of Jacobian coordinates is a popular choice for efficiency reasons. If this is used, then Algorithm 1 can use an efficient merged doubling-addition formula [42] when $w > 2$ by setting DBLADD = *true*. Other cases, including Weierstrass curves with $w = 2$ or twisted Edwards curves, should use DBLADD = $false$. Note that the

**Algorithm 1** Variable-base scalar multiplication using the fixed-window method.

```
Input: Scalar $k \in [0, r\rangle$ and point $P = (x, y) \in E(\mathbf{F}_p)$, where $\#E(\mathbf{F}_p) = h.r$ with co-factor $h \in \mathbf{Z}^+$ and $r$
   prime.
Output: $kP$.
 1. if $k = 0 \vee k \geq r$ then return ("error: invalid scalar")                                  [if: validation]
 2. Run point validation and compute $T = 4P$ (for $\mathcal{E}_d$) using Algorithm 2 for $E_b$ and Algorithm 3 for $\mathcal{E}_d$. If
    "invalid" return ("error: invalid point"), else set $P = T$ (for $\mathcal{E}_d$).                [if: validation]
    Precomputation Stage:
 3. Fix the window width $2 \leq w < 10 \in \mathbf{Z}^+$.
 4. Compute $P[i] = (2i+1)P$ for $0 \leq i < 2^{w-2}$ using Algorithm 4 for $E_b$ and Algorithm 5 for $\mathcal{E}_d$.
    Recoding Stage:
 5. odd $= k \bmod 2$
 6. if odd $= 0$ then $k = r - k$                                                         [if: masked constant time]
 7. Recode $k$ to $(k_t, \ldots, k_0) = (s_t \cdot |k_t|, \ldots, s_0 \cdot |k_0|)$ using Algorithm 6, where $t = \lceil \log_2(r)/(w-1) \rceil$ and $s_j$ are
    the signs of the recoded digits.
    Evaluation Stage:
 8. $Q = s_t P[(|k_t| - 1)/2]$
 9. for $i = (t-1)$ to 1
10.     if DBLADD $= true \wedge w \neq 2$ then                                             [if: algorithm variant]
11.         $Q = 2^{(w-2)}Q$                                                                       (Use Alg.10)
12.         $Q = 2Q + s_i P[(|k_i| - 1)/2]$                                                        (Use Alg.11)
13.     else
14.         $Q = 2^{(w-1)}Q$                                          (Use Alg.10 for $E_b$ and Alg.14 for $\mathcal{E}_d$)
15.         $Q = Q + s_i P[(|k_i| - 1)/2]$                            (Use Alg.12 for $E_b$ and Alg.15 for $\mathcal{E}_d$)
16.     end if
17. end for
18. $Q = 2^{(w-1)}Q$                                                  (Use Alg.10 for $E_b$ and Alg.14 for $\mathcal{E}_d$)
19. $Q = Q \oplus s_0 P[(|k_0| - 1)/2]$                               (Use Alg.19 for $E_b$ and Alg.17 for $\mathcal{E}_d$)
20. if odd $= 0$ then $Q = -Q$                                                            [if: masked constant time]
21. Convert $Q$ to affine coordinates $(x, y)$.
22. return $Q$.
```

**Algorithm 2** Point validation for the Weierstrass curves $E_b/\mathbf{F}_p : y^2 = x^3 - 3x + b$ in Table 1.

```
Input: Point $P = (x_1, y_1)$.
Output: "Valid" or "invalid" point.
 1. if $P = \mathcal{O}$ then return ("invalid")                                                   [if: validation]
 2. if $x_1 \notin [0, p-1] \vee y_1 \notin [0, p-1]$ then return ("invalid")                      [if: validation]
 3. if $y_1^2 \neq x_1^3 - 3x_1 + b \pmod{p}$ then return ("invalid")                              [if: validation]
 4. return ("valid").
```

evaluation of DBLADD is used to simplify the description of the algorithm. An implementation might choose for having separate functions for twisted Edwards and Weierstrass curves. Following the recommendations from Section 4, the last addition should be performed with a unified formula (denoted by $\oplus$) in order to avoid exceptions and it has been separated from the main loop; see Steps 18 and 19. To achieve constant-time execution, the points from the precomputed table should be extracted by doing a *full* pass over all the points in the lookup table and masking the correct value with the index $(|k_i| - 1)/2$. Finally, a suitable conversion to affine coordinates may be computed at Step 21 (if required).

Algorithm 7 computes scalar multiplication for the fixed-base scenario using the modified LSB-set method [26] (denoted by mLSB-set), which combines the comb method [40] and LSB-set recoding [27]. Refer to Sections 4.1 and 4.2 for details on the use of the method with Weierstrass and twisted Edwards curves, respectively. This operation consists of computations

**Algorithm 3** Combined point validation and torsion clearing for the twisted Edwards curves $\mathcal{E}_d\colon -x^2+y^2=1+dx^2y^2$ in Table 2.

```
Input: Point $P=(x_1,y_1)$.
Output: “Invalid”, or “valid” and a point $T$ of prime order $r$.
1. if $x_1 \notin [0,p-1] \vee y_1 \notin [0,p-1]$ then return (“invalid”)      [if: validation]
2. if $-x_1^2+y_1^2 \neq 1+dx_1^2y_1^2 \pmod p$ then return (“invalid”)      [if: validation]
3. if $(x_1,y_1)=(0,1)$ then return (“invalid”)      [if: validation]
4. Compute $T=4P$
5. if $T=\mathcal{O}$ then return (“invalid”)      [if: validation]
6. return (“valid”) and $T$.
```

**Algorithm 4** Precomputation scheme for the Weierstrass curves $E_b/\mathbf{F}_p: y^2=x^3-3x+b$ in Table 1.

```
Input: Point $P=(x_1,y_1)\in E_b(\mathbf{F}_p)\setminus\{\mathcal{O}\}$ of prime order $r$ and window width $2\le w<10$.
Output: $P[i]=(2i+1)P$ for $0\le i<2^{w-2}$, in Chudnovsky coordinates.
1. Given $P=(x_1,y_1)$, compute $Q=2P$ in Jacobian coordinates ($Q=(\mathbf{X}_2,\mathbf{Y}_2,\mathbf{Z}_2)$) and convert $P$ to
   Chudnovsky coordinates ($P=(X,Y,Z_1,Z_2,Z_3)$) such that the new $P$ and $Q$ have the same $Z$-coordinate
   (read from left to right, top to bottom):
     $t_2=1$,              $t_1=x_1^2$,           $t_1=t_1-t_2$,     $t_2=t_1/2$,   $t_1=t_1+t_2$,
     $Z_2=y_1^2$,          $X=x_1\cdot Z_2$,      $Z_3=Z_2\cdot y_1$, $t_2=t_1^2$    $t_2=t_2-X$,
     $\mathbf{X}_2=t_2-X$, $\mathbf{Z}_2=y_1$,    $Z_1=y_1$,         $Y=Z_2^2$,     $t_2=X-X_2$,
     $t_3=t_1\cdot t_2$,   $\mathbf{Y}_2=t_3-Y$.
2. $P[0]=P$
3. for $i=1$ to $2^{w-2}-1$ do
4.    Given $Q=(X_1,Y_1,Z)$ and $P[i-1]=(X_2,Y_2,Z,Z^2,Z^3)$ compute $P[i]=Q+P[i-1]=$
      $(X_3,Y_3,Z_{3,1},Z_{3,2},Z_{3,3})$ and update the representation of $Q=(\mathbf{X},\mathbf{Y},\mathbf{Z})$ such that $\mathbf{Z}=Z_{3,1}$ (read
      from left to right, top to bottom):
        $t_1=X_2-X_1$,         $Z_{3,1}=Z\cdot t_1$,      $\mathbf{Z}=Z_{3,1}$,      $t_2=t_1^2$,           $Z_{3,2}=Z_{3,1}^2$,
        $t_3=t_1\cdot t_2$,    $\mathbf{X}=X_1\cdot t_2$, $t_1=Y_2-Y_1$,            $X_3=t_1^2$,           $Z_{3,3}=Z_3\cdot Z_{3,2}$,
        $X_3=X_3-t_3$,         $X_3=X_3-\mathbf{X}$,      $X_3=X_3-\mathbf{X}$,     $t_2=\mathbf{X}-X_3$,  $t_1=t_1\cdot t_2$,
        $\mathbf{Y}=Y_1\cdot t_3$, $Y_3=t_1-\mathbf{Y}$.
5. return $P[i]$ ($=(2i+1)P$ for $0\le i<2^{w-2}$).
```

executed *offline*, which involve point validation and precomputing multiples of the known input point, and computations executed *online*, which involve scalar validation, recoding and evaluation stages. As before, point validation for twisted Edwards using Algorithm 3 during the offline phase performs two doublings over the input point to ensure that the computation takes place in the large prime order subgroup. Again, it is the protocol implementer's responsibility to ensure that timing differences during the detection of errors do not leak sensitive information to an attacker. The implementer should choose a window width $2\le w<10$ and a table parameter $v\ge 1$ according to efficiency and/or memory constraints, taking into account that the mLSB-set method requires $v\cdot 2^{w-1}$ precomputed points. For example, selecting $w=6$ and $v=3$ for 256-bit scalar multiplication was found to achieve optimal performance in our implementations of Weierstrass curves when storage is constrained to 6KB. During the online computation, the recoded scalar obtained from Algorithm 8 has a fixed length, which enables a fully regular execution when the representation is set up as described at Step 7. Since Algorithm 8 only recodes odd integers, we include a conversion step at Step 6 to deal with even values. The corresponding correction is performed at Step 13. In the evaluation stage, the main computation consists of $e-1=\lceil\lceil\log_2(r)\rceil/(wv)\rceil-1$ iterations each computing one doubling and $v$ additions with a value from the precomputed table. Following Section 4, the

**Algorithm 5** Precomputation scheme for twisted Edwards curves ($\mathcal{E}_d$).

```
Input: Point $P = (x_1, y_1) \in \mathcal{E}_d(\mathbf{F}_p)$ of prime order $r$ and window width $w \geq 2 \in \mathbf{Z}^+$.
Output: $P[i] = (2i+1)P$, for $0 \leq i < 2^{w-2}$, in extended homogeneous coordinates $(X+Y, Y-X, 2Z, 2T)$.
1. Given $P = (x_1, y_1)$, compute $Q = 2P = (X_2, Y_2, Z_2, T_2)$ (where $Q$ is represented using $(X+Y, Y-X, Z, T)$),
   and update $P = (\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{T})$ in the representation $(X+Y, Y-X, 2Z, 2T)$ (read from left to right, top to
   bottom):
     $\mathbf{X} = x_1^2$,   $t_1 = y_1^2$,   $t_2 = \mathbf{X} + t_1$,   $t_1 = t_1 - \mathbf{X}$,   $\mathbf{Y} = x_1 \cdot y_1$,
     $t_3 = \mathbf{Y} + \mathbf{Y}$,   $Y_2 = t_1 \cdot t_2$,   $T_2 = t_2 \cdot t_3$,   $t_2 = 2$,   $t_2 = t_2 - t_1$,
     $Z_2 = t1 \cdot t_2$,   $t_1 = t2 \cdot t_3$,   $\mathbf{Z} = x_1$,   $\mathbf{T} = x_1 \cdot \mathbf{Y}$,   $X_2 = t_1 + Y_2$,
     $Y_2 = Y_2 - t_1$,   $t_1 = \mathbf{X} + \mathbf{Y}$,   $\mathbf{Y} = \mathbf{Y} - \mathbf{X}$,   $\mathbf{X} = t_1$,   $\mathbf{Z} = \mathbf{Z} + \mathbf{Z}$,
     $\mathbf{T} = \mathbf{T} + \mathbf{T}$.
2. $P[0] = P$
3. for $i = 1$ to $2^{(w-2)} - 1$ do
4.    Given $P[i-1] = (X_2, Y_2, Z_2, T_2)$ (represented using $(X+Y, Y-X, 2Z, 2T)$) and $Q = (X_1, Y_1, Z_1, T_1)$
      (represented using $(X+Y, Y-X, Z, T)$) compute $P[i] = Q + P[i-1]$, where $P[i] = (X_3, Y_3, Z_3, T_3)$ is
      represented as $(X+Y, Y-X, 2Z, 2T)$ (read from left to right, top to bottom):
        $t_1 = T_2 \cdot Z_1$,   $t_2 = T_1 \cdot Z_2$,   $t_3 = t_2 - t_1$,   $t_1 = t_1 + t_2$,   $t_2 = t_1 \cdot t_3$,
        $T_3 = t_2 + t_2$,   $t_2 = X_1 \cdot Y_2$,   $X_3 = Y_1 \cdot X_2$,   $Y_3 = t_2 - X_3$,   $t_2 = X_3 + t_2$,
        $X_3 = Y_3 \cdot t_1$,   $Z_3 = Y_3 \cdot t_2$,   $t_1 = t_3 \cdot t_2$,   $Y_3 = t_1 - X_3$,   $X_3 = X_3 + t_1$,
        $Z_3 = Z_3 + Z_3$.
5. return $P[i]$ $(= (2i+1)P$ for $0 \leq i < 2^{w-2})$.
```

**Algorithm 6** Protected odd-only recoding algorithm for the fixed-window representation.

```
Input: odd integer $k \in [1, r\rangle$ and window width $w \geq 2$, where $r$ is the prime order of the targeted elliptic
   curve (sub)group.
Output: $(k_t, \ldots, k_0)$, where $k_i \in \{\pm 1, \pm 3, \cdots, \pm(2^{w-1} - 1)\}$ and $t = \lceil \log_2(r)/(w-1) \rceil$.
1. $t = \lceil \log_2(r)/(w-1) \rceil$
2. for $i = 0$ to $(t-1)$ do
3.    $k_i = (k \mod 2^w) - 2^{w-1}$
4.    $k = (k - k_i)/2^{w-1}$
5. $k_t = k$
6. return $(k_t, \ldots, k_0)$.
```

additions should be performed with a unified formula (denoted by $\oplus$) to avoid exceptions. Note that, as described in the variable-base case, all the conditional computations using "if" statements as well as the extraction of points from the precomputed table should be executed in constant time in order to protect against timing attacks (with the exception of Step 3, which depends on public parameters; any potential leak through the detection of errors at Step 4 should be assessed by the protocol's implementer). Finally, a suitable conversion to affine coordinates may be computed at Step 14 (if required).

Algorithm 9 computes double-scalar multiplication, which is typically found in signature verification schemes, and uses the width-$w$ non-adjacent form [53] with interleaving [29, 44]. We assume that one of the input points is known in advance ($P_2$) whereas the other one is a variable base ($P_1$). Hence, we distinguish two phases: *offline*, which involves validation of $P_2$ and a precomputation stage using the value $w_2$; and *online*, which involves scalar validation, point validation of $P_1$ and precomputation (using $w_1$), recoding and evaluation stages. Again, point validation for twisted Edwards curves with Algorithm 3 performs two doublings over the input points to ensure computation in the large prime order subgroup. The precomputation for both input points are performed as in the variable-base scenario using Algorithms 4 and 5 for $a = -3$ Weierstrass and $a = -1$ twisted Edwards curves, respectively. However, the

**Algorithm 7** Protected fixed-base scalar multiplication using the modified LSB-set comb method.

**Input:** A point $P = (x, y) \in E(\mathbf{F}_p)$, where $\#E = h.r$ with co-factor $h \in \mathbf{Z}^+$ and $r$ prime, a scalar $k = (k_{t-1}, \ldots, k_0)_2 \in [0, r\rangle$, where $t = \lceil \log_2(r) \rceil$, window width $w \geq 2$, and table parameter $v \geq 1$, such that $e = \lceil t/(wv) \rceil$, $d = ev$ and $\ell = dw$.
**Output:** $kP$.

```
*Offline computation*:
**Precomputation stage:**
Run point validation and compute $T = 4P$ (for $\mathcal{E}_d$) using Algorithm 2 for $E_b$ and Algorithm 3 for $\mathcal{E}_d$. If "invalid" return ("error: invalid point"), else set $P = T$ (for $\mathcal{E}_d$).    [if: validation]
Compute $P[i][j] = 2^{ej}(1 + i_0 2^d + \ldots + i_{w-2} 2^{(w-1)d})P$ for all $0 \leq i < 2^{w-1}$ and $0 \leq j < v$, where $i = (i_{w-2}, \ldots, i_0)_2$.
if $wv \mid t$ then compute $2^{wd}P$.    [if: algorithm variant]
*Online computation*:
if $k = 0 \vee k \geq r$ then return ("error: invalid scalar")    [if: validation]
**Recoding stage:**
odd $= k \bmod 2$
if odd $= 0$ then $k = r - k$    [if: masked constant time]
Pad $k$ with $dw - t$ zeros to the left and convert it to the $m$LSB-set representation using Algorithm 8 such that $k = (c, b_{l-1}, \ldots, b_0)_{m\text{LSB-set}}$. Set the digit columns $\mathbf{T}_{i,j} = |\sum_{m=0}^{w-2} b_{(m+1)d+ei+j} 2^m|$ with signs $s_{i,j} = b_{ei+j}$ for all $0 \leq i < v$ and $0 \leq j < e$.
**Evaluation stage:**
$Q = \sum_{i=0}^{v-1} s_{i,e-1} P[\mathbf{T}_{i,e-1}][i]$
for $i = e - 2$ to $0$ do
    $Q = 2Q$    (Use Alg.10 for $E_b$ and Alg.14 for $\mathcal{E}_d$)
    $Q = Q \oplus \sum_{j=0}^{v-1} s_{j,i} P[\mathbf{T}_{j,i}][j]$    (Use Alg.18 for $E_b$ and Alg.17 for $\mathcal{E}_d$)
if $wv \mid t \wedge c = 1$ then $Q = Q + 2^{wd}P$    [if: masked constant time]
if odd $= 0$ then $Q = -Q$    [if: masked constant time]
Convert $Q$ to affine coordinates $(x, y)$.
return $Q$
```

**Algorithm 8** Protected odd-only recoding algorithm for the modified LSB-set representation.

**Input:** An odd $\ell$-bit integer $k = (k_{\ell-1}, \ldots, k_0)_2 \in [1, r\rangle$, window width $w \geq 2$ and table parameter $v \geq 1$, where $r$ is the prime order of the targeted elliptic curve group such that $e = \lceil t/(wv) \rceil$, $d = ev$ and $\ell = dw$, where $t = \lceil \log_2(r) \rceil$.
**Output:** $(c, b_{\ell-1}, \ldots, b_0)_{m\text{LSB-set}}$, where $\begin{cases} b_i \in \{1, -1\} & \text{if } 0 \leq i < d \\ b_i \in \{0, b_{i \bmod d}\} & \text{if } d \leq i < \ell. \end{cases}$ If $wv \mid t$ then $c \in \{0, 1\}$, otherwise, $c = 0$.

```
$b_{d-1} = 1$
for $i = 0$ to $(d - 2)$ do
    $b_i = 2k_{i+1} - 1$
$c = \lfloor k/2^d \rfloor$
for $i = d$ to $(\ell - 1)$ do
    $b_i = b_{i \bmod d} \cdot c_0$
    $c = \lfloor c/2 \rfloor - \lfloor b_i/2 \rfloor$
return $(c, b_{l-1}, \ldots, b_0)_{m\text{LSB-set}}$.
```

implementer has additional freedom in the selection of $w_2$ since the precomputation for the fixed-base is done offline. For example, we found that using $w_1 = 6$ and $w_2 = 7$ results in optimal performance in our implementations of Weierstrass curves when storage was restricted to 2KB, 3KB and 4KB for 128-, 192- and 256-bit security levels. In the online computation, recoding of the scalars is performed using [33, Algorithm 3.35]. Accordingly, the evaluation stage consists of $\lceil \log_2(r) \rceil + 1$ iterations, each consisting of one doubling and at most two additions (one per precomputed table). As in the variable-base case, for $a = -3$ Weierstrass curves using Jacobian coordinates one may use the merged doubling-addition formula [42] by setting DBLADD = *true*. A suitable conversion to affine coordinates may be computed at Step 39 (if required).

**Algorithm 9** Double-scalar multiplication using the width-$w$ NAF with interleaving. (If-statements are not marked because this algorithm is not assumed to be constant-time.)

```
Input: Scalars $k_1$ and $k_2 \in [0, r\rangle$ and points $P_1$ and $P_2 \in E(\mathbf{F}_p)$, where $\#E = h.r$ with co-factor $h \in \mathbf{Z}^+$ and $r$ prime.
Output: $k_1 P_1 + k_2 P_2$.
  Offline computation:
  Precomputation stage:
  Run point validation over $P_2$ and compute $T = 4P_2$ (for $\mathcal{E}_d$) using Algorithm 2 for $E_b$ and Algorithm 3 for $\mathcal{E}_d$. If "invalid" return ("error: invalid point"), else set $P_2 = T$ (for $\mathcal{E}_d$).
  Fix the window width $w_2 \geq 2 \in \mathbf{Z}^+$.
  Compute $P_2[i] = (2i+1)P_2$ for $0 \leq i < 2^{w_2 - 2}$ using Algorithm 4 for $E_b$ and Algorithm 5 for $\mathcal{E}_d$.
  Online computation:
  if $(k_1 = 0 \lor k_1 \geq r) \lor (k_2 = 0 \lor k_2 \geq r)$ then return ("error: invalid scalar")
  Run point validation over $P_1$ and compute $T = 4P_1$ (for $\mathcal{E}_d$) using Algorithm 2 for $E_b$ and Algorithm 3 for $\mathcal{E}_d$. If "invalid" return ("error: invalid point"), else set $P_1 = T$ (for $\mathcal{E}_d$).
  Precomputation Stage:
  Fix the window width $w_1 \geq 2 \in \mathbf{Z}^+$.
  Compute $P_1[i] = (2i+1)P_1$ for $0 \leq i < 2^{w_1 - 2}$ using Algorithm 4 for $E_b$ and Algorithm 5 for $\mathcal{E}_d$.
  Recoding Stage:
  Recode $k_1$ to $(k_{1,i-1}, k_{1,i-2}, \ldots, k_{1,0})_{w\mathrm{NAF}}$ using [33, Algorithm 3.35] and pad it with $\lceil \log_2(r) \rceil - i + 1$ zeros to the left.
  Recode $k_2$ to $(k_{2,j-1}, k_{2,j-2}, \ldots, k_{2,0})_{w\mathrm{NAF}}$ using [33, Algorithm 3.35] and pad it with $\lceil \log_2(r) \rceil - j + 1$ zeros to the left.
  Evaluation Stage:
  for $i = \lceil \log_2(r) \rceil$ to 0
      if DBLADD = $true$ then
          if $k_{1,i} = 0$ then
              $Q = 2Q$                                  (Use Algorithm 10)
          else if $k_{1,i} > 0$ then
              $Q = 2Q + P_1[k_{1,i}/2]$                 (Use Algorithm 11)
          else if $k_{1,i} < 0$ then
              $Q = 2Q - P_1[(-k_{1,i})/2]$              (Use Algorithm 11)
          end if
          if $k_{2,i} > 0$ then
              $Q = Q + P_2[k_{2,i}/2]$                  (Use Algorithm 13)
          else if $k_{2,i} < 0$ then
              $Q = Q - P_2[(-k_{2,i})/2]$               (Use Algorithm 13)
          end if
      else
          $Q = 2Q$                                      (Use Algorithm 14)
          else if $k_{1,i} > 0$ then
              $Q = Q + P_1[k_{1,i}/2]$                  (Use Algorithm 15)
          else if $k_{1,i} < 0$ then
              $Q = Q - P_1[(-k_{1,i})/2]$               (Use Algorithm 15)
          end if
          if $k_{2,i} > 0$ then
              $Q = Q + P_2[k_{2,i}/2]$                  (Use Algorithm 16)
          else if $k_{2,i} < 0$ then
              $Q = Q - P_2[(-k_{2,i})/2]$               (Use Algorithm 16)
          end if
      end if
  end for
  Convert $Q$ to affine coordinates $(x, y)$.
  return $Q$.
```

# B Algorithms for Point Operations

**Algorithm 10** Point doubling using Jacobian coordinates on Weierstrass curves ($E_b$).

**Input:** $P = (X_1, Y_1, Z_1) \in E_b(\mathbf{F}_p)$ in Jacobian coordinates.
**Output:** $2P = (X_2, Y_2, Z_2) \in E_b(\mathbf{F}_p)$ in Jacobian coordinates.

```
$t_1 = Z_1^2$
$t_2 = X_1 + t_1$
$t_1 = X_1 - t_1$
$t_1 = t_1 \times t_2$
$t_2 = t_1/2$
$t_1 = t_1 + t_2$
$t_2 = Y_1^2$
$t_3 = X_1 \times t_2$
$t_4 = t_1^2$
$t_4 = t_4 - t_3$
$X_2 = t_4 - t_3$
$Z_2 = Y_1 \times Z_1$
$t_2 = t_2^2$
$t_4 = t_3 - X_2$
$t_1 = t_1 \times t_4$
$Y_2 = t_1 - t_2$
return $2P = (X_2, Y_2, Z_2)$.
```

**Algorithm 11** Merged point doubling-addition using Jacobian/Chudnosvky coordinates on Weierstrass curves ($E_b$).

**Input:** $P, Q \in E_b(\mathbf{F}_p)$ such that $P = (X_1, Y_1, Z_1)$ is in Jacobian coordinates and $Q = (X_2, Y_2, Z_2, Z_2^2, Z_2^3)$ is in Chudnosvky coordinates.
**Output:** $2P + Q = (X_4, Y_4, Z_4) \in E_b(\mathbf{F}_p)$ in Jacobian coordinates.

```
if $P = \mathcal{O}$ then return $Q$                                [if: exception]
if $Q = \mathcal{O}$ then use Algorithm 10 to compute and
return $2P$                                                         [if: exception]
$t_1 = Z_1^2$
$t_2 = Z_1 \times t_1$
$t_3 = Z_2^3 \times Y_1$
$t_2 = Y_2 \times t_2$
$t_2 = t_2 - t_3$
$t_4 = Z_2^2 \times X_1$
$t_1 = t_1 \times X_2$
$t_1 = t_1 - t_4$
if $t_1 = 0$ then                                                   [if: exception]
    if $t_2 = 0$ then                                               [if: exception]
        Use Alg. 10 to compute $R = 2P$
        Use Alg. 12 to compute and return $R + Q$
    else return $P$
$Z_4 = Z_1 \times Z_2$
$Z_4 = t_1 \times Z_4$
$t_5 = t_1^2$
$t_1 = t_1 \times t_5$
$X_4 = t_4 \times t_5$
$t_4 = t_2^2$
$t_4 = t_4 - t_1$
$t_4 = t_4 - X_4$
$t_4 = t_4 - X_4$
$t_4 = t_4 - X_4$
if $t_4 = 0$ then return ($\mathcal{O}$)                           [if: exception]
$Y_4 = t_1 \times t_3$
$t_1 = t_2 \times t_4$
$t_1 = t_1 + Y_4$
$t_1 = t_1 + Y_4$
$Z_4 = Z_4 \times t_4$
$t_2 = t_4^2$
$t_4 = t_2 \times t_4$
$t_2 = t_2 \times X_4$
$t_3 = t_1^2$
$t_3 = t_3 - t_4$
$t_3 = t_3 - t_2$
$X_4 = t_3 - t_2$
$t_3 = X_4 - t_2$
$t_4 = t_4 \times Y_4$
$t_1 = t_1 \times t_3$
$Y_4 = t_1 - t_4$
return $2P + Q = (X_4, Y_4, Z_4)$.
```

**Algorithm 12** Point addition using Jacobian/Chudnovsky coordinates on Weierstrass curves ($E_b$).

**Input:** $P, Q \in E_b(\mathbf{F}_p)$ such that $P = (X_1, Y_1, Z_1)$ is in Jacobian coordinates and $Q = (X_2, Y_2, Z_2, Z_2^2, Z_2^3)$ is in Chudnosvky coordinates.

**Output:** $P + Q = (X_3, Y_3, Z_3) \in E_b(\mathbf{F}_p)$ in Jacobian coordinates.

```
if $P = \mathcal{O}$ then return $Q$        [if: exception]
if $Q = \mathcal{O}$ then return $P$        [if: exception]
$t_1 = Z_1^2$
$t_2 = Z_1 \times t_1$
$t_3 = Z_2^3 \times Y_1$
$t_2 = Y_2 \times t_2$
$t_2 = t_2 - t_3$
$t_4 = Z_2^2 \times X_1$
$t_1 = t_1 \times X_2$
$t_1 = t_1 - t_4$
if $t_1 = 0$ then                           [if: exception]
    if $t_2 = 0$ then                       [if: exception]
        Use Alg. 10 to compute and return $2P$.
    else return $(\mathcal{O})$
$Z_3 = Z_1 \times Z_2$
$Z_3 = t_1 \times Z_3$
$t_5 = t_1^2$
$t_1 = t_1 \times t_5$
$t_4 = t_4 \times t_5$
$t_5 = t_2^2$
$t_5 = t_5 - t_1$
$t_5 = t_5 - t_4$
$X_3 = t_5 - t_4$
$t_4 = t_4 - X_3$
$t_4 = t_2 \times t_4$
$t_1 = t_1 \times t_3$
$Y_3 = t_4 - t_1$
return $P + Q = (X_3, Y_3, Z_3)$.
```

**Algorithm 13** Point addition using Jacobian/affine coordinates on Weierstrass curves ($E_b$).

**Input:** $P, Q \in E_b(\mathbf{F}_p)$ such that $P = (X_1, Y_1, Z_1)$ is in Jacobian coordinates and $Q = (x_2, y_2)$ is in affine coordinates.

**Output:** $P + Q = (X_3, Y_3, Z_3) \in E_b(\mathbf{F}_p)$ in Jacobian coordinates.

```
if $P = \mathcal{O}$ then return $Q$        [if: exception]
if $Q = \mathcal{O}$ then return $P$        [if: exception]
$t_1 = Z_1^2$
$t_2 = Z_1 \times t_1$
$t_1 = t_1 \times x_2$
$t_2 = t_2 \times y_2$
$t_1 = t_1 - X_1$
$t_2 = t_2 - Y_1$
if $t_1 = 0$ then                           [if: exception]
    if $t_2 = 0$ then                       [if: exception]
        Use Alg. 10 to compute and return $2P$.
    else return $(\mathcal{O})$
$Z_3 = Z_1 \times t_1$
$t_3 = t_1^2$
$t_4 = t_1 \times t_3$
$t_3 = X_1 \times t_3$
$t_1 = t_3 + t_3$
$X_3 = t_2^2$
$X_3 = X_3 - t_1$
$X_3 = X_3 - t_4$
$t_3 = t_3 - X_3$
$t_3 = t_2 \times t_3$
$t_4 = t_4 \times Y_1$
$Y_3 = t_3 - t_4$
return $P + Q = (X_3, Y_3, Z_3)$.
```

**Algorithm 14** Point doubling using homogeneous/extended homogeneous coordinates on Edwards curves ($\mathcal{E}_d$).

**Input:** $P = (X_1, Y_1, Z_1) \in \mathcal{E}_d(\mathbf{F}_p)$.

**Output:** $2P = (X_2, Y_2, Z_2, T_{2,a}, T_{2,b}) \in \mathcal{E}_d(\mathbf{F}_p)$.

```
$t_1 = X_1^2$
$t_2 = Y_1^2$
$T_{2,b} = t_1 + t_2$
$t_1 = t_2 - t_1$
$t_2 = Y_1 + Y_1$
$T_{2,a} = X_1 \times t_2$
$Y_2 = t_1 \times T_{2,b}$
$t_2 = Z_1^2$
$t_2 = t_2 + t_2$
$t_2 = t_2 - t_1$
$Z_2 = t_1 \times t_2$
$X_2 = t_2 \times T_{2,a}$
return $2P = (X_2, Y_2, Z_2, T_{2,a}, T_{2,b})$.
```

**Algorithm 15** Point addition using extended homogeneous coordinates on Edwards curves ($\mathcal{E}_d$).

**Input:** $P, Q \in \mathcal{E}_d(\mathbf{F}_p)$ such that $P = (X_1, Y_1, Z_1, T_{1,a}, T_{1,b})$ and $Q = (X_2 + Y_2, Y_2 - X_2, 2Z_2, 2T_2)$.
**Output:** $P + Q = (X_3, Y_3, Z_3, T_{3,a}, T_{3,b}) \in \mathcal{E}_d(\mathbf{F}_p)$.

```
if $Q = \mathcal{O}$ then return $P$        [if: exception]
if $P = \mathcal{O}$ then                   [if: exception]
    $t_1 = (X_2 + Y_2) - (Y_2 - X_2)$
    $t_1 = t_1/2$
    $Y_3 = (Y_2 - X_2) + t_1$
    $X_3 = t_1$
    $Z_3 = (2Z_2)/2$
    $T_{3,a} = (2T_2)/2$
    $T_{3,b} = 1$
$T_{3,a} = T_{1,a} \times T_{1,b}$
$t_1 = (2T_2) \times Z_1$
$t_2 = T_{3,a} \times (2Z_2)$
$T_{3,a} = t_2 - t_1$
$T_{3,b} = t_1 + t_2$
$t_2 = X_1 + Y_1$
$t_1 = (Y_2 - X_2) \times t_2$
$t_2 = Y_1 - X_1$
$t_2 = (X_2 + Y_2) \times t_2$
$Z_3 = t_1 - t_2$
$t_1 = t_1 + t_2$
$X_3 = T_{3,b} \times Z_3$
$Z_3 = Z_3 \times t1$
$Y_3 = T_{3,a} \times t_1$
return $P + Q = (X_3, Y_3, Z_3, T_{3,a}, T_{3,b})$.
```

**Algorithm 16** Point addition using extended homogeneous/extended affine coordinates on Edwards curves ($\mathcal{E}_d$).

**Input:** $P, Q \in \mathcal{E}_d(\mathbf{F}_p)$ such that $P = (X_1, Y_1, Z_1, T_{1,a}, T_{1,b})$ and $Q = (x_2 + y_2, y_2 - x_2, 2t_2)$.
**Output:** $P + Q = (X_3, Y_3, Z_3, T_{3,a}, T_{3,b}) \in \mathcal{E}_d(\mathbf{F}_p)$.

```
if $Q = \mathcal{O}$ then return $P$        [if: exception]
if $P = \mathcal{O}$ then                   [if: exception]
    $t_1 = (x_2 + y_2) - (y_2 - x_2)$
    $t_1 = t_1/2$
    $Y_3 = (y_2 - x_2) + t_1$
    $X_3 = t_1$
    $Z_3 = 1$
    $T_{3,a} = (2t_2)/2$
    $T_{3,b} = 1$
$T_{3,a} = T_{1,a} \times T_{1,b}$
$t_1 = (2T_2) \times Z_1$
$t_2 = T_{3,a} + T_{3,a}$
$T_{3,a} = t_2 - t_1$
$T_{3,b} = t_1 + t_2$
$t_2 = X_1 + Y_1$
$t_1 = (Y_2 - X_2) \times t_2$
$t_2 = Y_1 - X_1$
$t_2 = (X_2 + Y_2) \times t_2$
$Z_3 = t_1 - t2$
$t_1 = t_1 + t_2$
$X_3 = T_{3,b} \times Z_3$
$Z_3 = Z_3 \times t1$
$Y_3 = T_{3,a} \times t_1$
return $P + Q = (X_3, Y_3, Z_3, T_{3,a}, T_{3,b})$.
```

**Algorithm 17** Unified point addition using extended homogeneous coordinates on Edwards curves ($\mathcal{E}_d$).

**Input:** $P, Q \in \mathcal{E}_d(\mathbf{F}_p)$ such that $P = (X_1, Y_1, Z_1, T_{1,a}, T_{1,b})$ and $Q = (X_2 + Y_2, Y_2 - X_2, 2Z_2, 2T_2)$.
**Output:** $P + Q = (X_3, Y_3, Z_3, T_{3,a}, T_{3,b}) \in \mathcal{E}_d(\mathbf{F}_p)$.

```
$T_{3,a} = T_{1,a} \times T_{1,b}$
if $2Z_2 = 2$ then                          [if: exception]
    $t_1 = Z_1 + Z_1$     {$Q$ is in affine coordinates}
else
    $t_1 = (2Z_2) \times Z_1$
$t_2 = T_{3,a} \times (2T_2)$
$T_{3,a} = t_2 \times d$
$t_3 = t_1 - T_{3,a}$
$t_1 = t_1 + T_{3,a}$
$t_2 = X_1 + Y_1$
$T_{3,a} = (X_2 + Y_2) \times t_2$
$t_2 = Y_1 - X_1$
$X_3 = (Y_2 - X_2) \times t_2$
$T_{3,b} = T_{3,a} - X_3$
$T_{3,a} = T_{3,a} + X_3$
$X_3 = T_{3,b} \times t_3$
$Z_3 = t_3 \times t_1$
$Y_3 = T_{3,a} \times t_1$
return $P + Q = (X_3, Y_3, Z_3, T_{3,a}, T_{3,b})$.
```

## C Implementing the Group Law

**Weierstrass curves.** It is standard to represent points on $E_b\colon y^2 = x^3 - 3x + b$ using Jacobian coordinates [21, 47, 56]: for non-zero $Z \in \mathbf{F}_p$, the tuple $(X\colon Y\colon Z)$ is used to represent the affine point $(X/Z^2, Y/Z^3)$ on $E_b$. There are many different variants of the Jacobian formulas originally proposed in [22]. In our implementation we use the doubling formula from [41] (see Algorithm 10). Point additions are usually performed between a running point and a point from a (precomputed) 'look-up' table. Typically, it is advantageous to leave the precomputed points in projective form for variable-base computations, and to convert them (offline) to their affine form for fixed-base computations. When elements in the table are stored in affine coordinates, point addition is performed using mixed Jacobian/affine coordinates using, for example, the formula presented in [33] (see Algorithm 13). There are cases in which exceptions in the formulas might arise. This is the case, for example, for fixed-base scalar multiplication. To achieve constant-time execution, we devised a complete formula based on masking that works for point addition, doubling, inverses and the point at infinity (see Algorithm 18). If points from the precomputed table are stored in projective coordinates, we use Chudnovsky coordinates to represent the affine point $(X/Z^2, Y/Z^3) \in E_b$ by the projective tuple $(X : Y : Z : Z^2 : Z^3)$. The corresponding addition formula is given as Algorithm 12. More efficiently, whenever a doubling is followed by an addition (as in the main loop of the variable-base scalar multiplication; see Algorithm 1) one can use a merged doubling-addition formula [42] that is based on the special addition with the same $Z$-coordinate from [43] (see Algorithm 11). The different costs of the point formulas used in our implementation can be found in Table 5. Finally, the exact routine to perform the precomputation for the variable-base scenario is outlined in Algorithm 4. The scheme uses a straightforward variant of the general formulas, including the special addition from [43].

**Twisted Edwards curves.** Hisil et al. [34] derive efficient formulas for additions on (special) twisted Edwards curves [7] by representing affine points $(X/Z, Y/Z)$ on $\mathcal{E}_d\colon -x^2 + y^2 = 1 + dx^2y^2$ by the projective tuple $(X : Y : Z : T)$, where $T = XY/Z$. Hamburg [31] proposes to represent such a projective point using five elements: $(X : Y : Z : T_1 : T_2)$, where $T = T_1T_2$. This has the advantage of avoiding a required look-ahead when computing the elliptic curve scalar multiplication using the techniques from [34]. If the addition formulas are "dedicated" they do not work for doubling but are usually more efficient. The details of the dedicated additions used in our implementation are outlined in Algorithm 15 and 16. For settings that might trigger exceptions in the formulas (e.g., fixed-based scalar multiplication), one can use the unified addition formula proposed by [34] (see Algorithm 17). The algorithm for point doubling on $\mathcal{E}_d$ is given in Algorithm 14: this extends the formula from [7] by using the five element representation as suggested in [31].

When storing precomputed points, we follow the caching techniques described in [34]: we store affine points as $(x + y, y - x, 2t)$ with $t = xy$, or projective points as $(X + Y : Y - X : 2Z : 2T)$ with $T = XY/Z$, both of which can speed up the scalar multiplication computation. Just as in the case of the Weierstrass curves above, it is usually advantageous to leave the precomputed points in projective form for variable-base computations, and to convert them (offline) to their affine form for fixed-base computations. The explicit routine that performs the precomputation for the variable-base scenario is outlined in Algorithm 5. The costs of the different formulas used in our implementation are displayed in Table 5.

**Table 5.** An overview of the number of modular operations required to implement the group law for $a = -3$ Weierstrass, $a = -1$ twisted Edwards and Montgomery curves using different coordinate systems. The Weierstrass point doubling works on Jacobian coordinates while the point addition formula takes as input one Jacobian (Jac) coordinate and the other in either affine (aff) or (projective) Chudnovsky coordinates. We also show a merged double-and-add approach which computes $R = 2P + Q$ where $R$ and $P$ are in Jacobian and $Q$ in Chudnovsky coordinates. The complete addition formulas also include the number of table lookups (denoted by #lut) that are required for their realization. The Edwards doubling uses the five-element projective coordinates $(X : Y : Z : T_1 : T_2)$. The Edwards addition adds a five-element projective coordinate $(X : Y : Z : T_1 : T_2)$ to a four-element projective coordinate $(X + Y : Y - X : 2Z : 2T)$ (proj.) or a three-element extended affine coordinate $(x + y, y - x, 2t)$ (aff.) resulting in a five-element coordinate as a result. The performance of a single step of the Montgomery ladder (which computes a doubling and a differential addition) is stated as well.

| | ref | #mul | #sqr | #mulc | #add | #sub | #div2 | #lut | see |
|---|---|---|---|---|---|---|---|---|---|
| Weierstrass double | [41] | 4 | 4 | 0 | 2 | 5 | 1 | 0 | Algorithm 10 |
| Weierstrass add: | | | | | | | | | |
| Jac + Chud → Jac | [22] | 11 | 3 | 0 | 0 | 7 | 0 | 0 | Algorithm 12 |
| Jac + aff. → Jac | [33] | 8 | 3 | 0 | 1 | 6 | 0 | 0 | Algorithm 13 |
| Weierstrass dbl-add | [42] | 16 | 5 | 0 | 2 | 11 | 0 | 0 | Algorithm 11 |
| (Complete) Jac + aff → Jac | This work | 8 | 3 | 0 | 2 | 8 | 1 | 1 | Algorithm 18 |
| (Complete) Jac + Jac → Jac | This work | 12 | 4 | 0 | 2 | 8 | 1 | 1 | Algorithm 19 |
| Edwards doubling | [7] | 4 | 3 | 0 | 3 | 2 | 0 | 0 | Algorithm 14 |
| Edwards addition proj. | [34] | 8 | 0 | 0 | 3 | 3 | 0 | 0 | Algorithm 15 |
| Edwards addition aff. | [34] | 7 | 0 | 0 | 4 | 3 | 0 | 0 | Algorithm 15 |
| (Unified) Edwards addition proj. | [34] | 9 | 0 | 0 | 3 | 3 | 0 | 0 | Algorithm 17 |
| (Unified) Edwards addition aff. | [34] | 8 | 0 | 0 | 4 | 3 | 0 | 0 | Algorithm 17 |
| Montgomery ladder step double-and-add | [46] | 5 | 4 | 1 | 4 | 4 | 0 | 0 | |

## C.1 Complete Addition Laws

An elliptic curve addition law is said to be *complete* if it correctly computes the group operation regardless of the two input points. Although employing such an addition law on its own can simplify the task of the implementer, it usually incurs a performance penalty. This is because the fastest formulas available for a particular curve model, which work fine for most input pairs, tend to fail on certain inputs. However, it is often the case that implementers can safely exploit the speed of such *incomplete* formulas by correctly dealing with all possible exceptions, or by designing the scalar multiplication routine such that exceptions can never arise. All of the twisted Edwards curves presented in this paper can make use of the complete addition law in [11] by working on the birationally equivalent Edwards model $\mathcal{E}_{-1/d} : x^2 + y^2 = 1 - (1/d)x^2y^2$. However, the complete formulas are slower compared to the fastest formulas on the twisted Edwards curve [34]. But even when working on an Edwards curve with complete formulas, an implementation of the scalar multiplication could still be sped up by mapping to a different curve, while remaining with the complete formulas for all other operations. One could for example follow the approach suggested in [32], and use an isogeny to the twisted Edwards curve $\mathcal{E}_{-1/d-1} : x^2 + y^2 = 1 - (1/d + 1)x^2y^2$; or use the birational equivalence to $\mathcal{E} : -x^2 + y^2 = 1 + dx^2y^2$.

The situation for the prime order Weierstrass curves in this paper is more complicated. As pointed out by Bosma and Lenstra [18], the best that we can do for general elliptic curves is as follows: on input of two points $P_1$ and $P_2$, we must compute two candidate sums, $P_3$ and $P_3'$, for which we can only be guaranteed that at least one of them is a correct projective representation for $P_1 + P_2$. In the case that precisely one of $P_3$ and $P_3'$ correctly corresponds to $P_1 + P_2$, the other candidate has all of its coordinates as zero; although this makes it straightforward to write a constant-time routine for complete additions, it also means that computing complete additions in this way is much more costly than computing incomplete additions.

For the sake of comparison, we present the simplified version of the complete formulas[9] from [18], which are specialized to short Weierstrass curves of the form $E\colon y^2 = x^3 + ax + b$. For two input points $P_1 = (X_1 \colon Y_1 \colon Z_1)$ and $P_2 = (X_2 \colon Y_2 \colon Z_2)$ in homogeneous projective space, the two candidate sums $P_3 = (X_3 \colon Y_3 \colon Z_3)$ and $P_3' = (X_3' \colon Y_3' \colon Z_3')$ are computed as

$$\begin{aligned}
X_3 &= (X_1Y_2 - X_2Y_1)(Y_1Z_2 + Y_2Z_1) - (X_1Z_2 - X_2Z_1)(a(X_1Z_2 + X_2Z_1) + 3bZ_1Z_2 - Y_1Y_2);\\
Y_3 &= -(3X_1X_2 + aZ_1Z_2)(X_1Y_2 - X_2Y_1) + (Y_1Z_2 - Y_2Z_1)(a(X_1Z_2 + X_2Z_1) + 3bZ_1Z_2 - Y_1Y_2);\\
Z_3 &= (3X_1X_2 + aZ_1Z_2)(X_1Z_2 - X_2Z_1) - (Y_1Z_2 + Y_2Z_1)(Y_1Z_2 - Y_2Z_1);\\
X_3' &= -(X_1Y_2 + X_2Y_1)(a(X_1Z_2 + X_2Z_1) + 3bZ_1Z_2 - Y_1Y_2) - (Y_1Z_2 + Y_2Z_1)(3b(X_1Z_2 + X_2Z_1) + a(X_1X_2 - aZ_1Z_2));\\
Y_3' &= Y_1^2Y_2^2 + 3aX_1^2X_2^2 - 2a^2X_1X_2Z_1Z_2 - (a^3 + 9b^2)Z_1Z_2^2 + (X_1Z_2 + X_2Z_1)(3b(3X_1X_2 - aZ_1Z_2) - a^2(X_2Z_1 + X_1Z_2));\\
Z_3' &= (3X_1X_2 + aZ_1Z_2)(X_1Y_2 + X_2Y_1) + (Y_1Z_2 + Y_2Z_1)(Y_1Y_2 + 3bZ_1Z_2 + a(X_1Z_2 + X_2Z_1)). \qquad (1)
\end{aligned}$$

In the case of $a = -3$ short Weierstrass curves, like the prime order curves in this paper, we found that the computations in (1) require at most[10] 22 multiplications, 3 multiplications by $b$, and one multiplication by $b^2 - 3$. The adaptation of the formulas to points in Jacobian coordinates can be achieved in the obvious way at an additional cost of 6 multiplications and 3 squarings: preceding (1), we can transform from Jacobian coordinates to homogeneous coordinates by taking $X_i \leftarrow X_i \cdot Z_i$ and then $Z_i \leftarrow Z_i^3$ for $i = 1, 2$; and, following the correct choosing of $P_3 = (X_3 \colon Y_3 \colon Z_3)$, we can move back to Jacobian coordinates by taking $X_3 \leftarrow X_3 \cdot Z_3$ and then $Y_3 \leftarrow Y_3 \cdot Z_3^2$.

Although the formulas in (1) are mathematically satisfactory, their computation costs around twice as much as an incomplete addition (see Table 5), which renders them far from satisfactory in cryptographic applications. On the other hand, the work-around we present in Algorithm 19 and Algorithm 18, while perhaps not as mathematically elegant, is equivalent for all practical purposes and incurs a much smaller overhead over the incomplete formulas. In particular, there are no additional multiplications or squarings (on top of those incurred during an incomplete addition) required when performing a complete addition via this *masking* approach.

As briefly discussed in Section 4.1, the idea is to exploit the similarity between the sequences of operations computed in a doubling and an addition. On input of $P$ and $Q$, one would ordinarily compute the doubling $2P$ and the (non-unified) addition $P + Q$ and mask out the correct result at the end, depending on whether $P = Q$. However, the detection of $P = Q$ (or not) can be achieved much earlier in projective space using only a few operations that are common to both doublings and non-unified additions – see Line 17 (resp. Line 12) in Algorithm 19 (resp. Algorithm 18). After this detection, the required operation (doubling or addition) is achieved by masking the correct inputs and outputs through a sequence of subsequent computations, those which overlap in the explicit formulas for point doublings and additions. Of course, in the case that one or both of $P$ or $Q$ is $\mathcal{O}$, or that $P = -Q$, these superfluous computations are still computed in constant-time such that the correct result is masked out in a cache-attack resistant manner.

[9] We also corrected some typos in [18] that were pointed out in [6].

[10] We did not optimize (1) agressively; we simply grouped common subexpressions and employed obvious operation scheduling – it is likely that there are faster routes.

**Algorithm 18** Complete (mixed) addition using masking and Jacobian/affine coordinates on prime-order Weierstrass curves $E_b$.

**Input:** $P, Q \in E_b(\mathbf{F}_p)$ such that $P = (X_1, Y_1, Z_1)$ is in Jacobian coordinates and $Q = (x_2, y_2)$ is in affine coordinates.

**Output:** $R = P + Q \in E_b(\mathbf{F}_p)$ in Jacobian coordinates. Computations marked with [*] are implemented in constant time using masking.

```
$T[0] = \mathcal{O}$    $\{T[i] = (\tilde{X}_i, \tilde{Y}_i, \tilde{Z}_i)$ for $0 \leq i < 4\}$
$T[1] = Q$
$t_2 = Z_1^2$
$t_3 = Z_1 \times t_2$
$t_1 = x_2 \times t_2$
$t_4 = y_2 \times t_3$
$t_1 = t_1 - X_1$
$t_4 = t_4 - Y_1$
index $= 3$
if $t_1 = 0$ then                                    [*]
    index $= 0$        $\{R = \mathcal{O}\}$
    if $t_4 = 0$ then index $= 2$    $\{R = 2P\}$    [*]
if $P = \mathcal{O}$ then index $= 1$    $\{R = Q\}$    [*]
mask $= 0$
if index $= 3$ then mask $= 1$                       [*]
{case $P + Q$, else any other case}
$t_3 = X_1 + t_2$
$t_6 = X_1 - t_2$
if mask $= 0$ then $t_2 = Y_1$ else $t_2 = t_1$      [*]
$t_5 = t_2^2$
$t_1 = X_1 \times t_5$
$\tilde{Z}_2 = Z_1 \times t_2$
$\tilde{Z}_3 = \tilde{Z}_2$
if mask $\neq 0$ then $t_3 = t_2$                    [*]
if mask $\neq 0$ then $t_6 = t_5$                    [*]
$t_2 = t_3 \times t_6$
$t_3 = t_2/2$
$t_3 = t_2 + t_3$
if mask $\neq 0$ then $t_3 = t_4$                    [*]
$t_4 = t_3^2$
$t_4 = t_4 - t_1$
$\tilde{X}_2 = t_4 - t_1$
$\tilde{X}_3 = \tilde{X}_2 - t_2$
if mask $= 0$ then $t_4 = \tilde{X}_2$ else $t_4 = \tilde{X}_3$    [*]
$t_1 = t_1 - t_4$
$t_4 = t_3 \times t_1$
if mask $= 0$ then $t_1 = t_5$ else $t_1 = Y_1$      [*]
if mask $= 0$ then $t_2 = t_5$                       [*]
$t_3 = t_1 \times t_2$
$\tilde{Y}_2 = t_4 - t_3$
$\tilde{Y}_3 = \tilde{Y}_2$
$R = P[\text{index}]$  $(= (\tilde{X}_{\text{index}}, \tilde{Y}_{\text{index}}, \tilde{Z}_{\text{index}}))$    [*]
return $R$
```

**Algorithm 19** Complete (projective) addition using masking and Jacobian coordinates on prime-order Weierstrass curves $E_b$.

**Input:** $P, Q \in E_b(\mathbf{F}_p)$ such that $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ are in Jacobian coordinates.

**Output:** $R = P + Q \in E_b(\mathbf{F}_p)$ in Jacobian coordinates. Computations marked with [*] are implemented in constant time using masking.

```
$T[0] = \mathcal{O}$    $\{T[i] = (\tilde{X}_i, \tilde{Y}_i, \tilde{Z}_i)$ for $0 \leq i < 5\}$
$T[1] = Q$
$T[4] = P$
$t_2 = Z_1^2$
$t_3 = Z_1 \times t_2$
$t_1 = X_2 \times t_2$
$t_4 = Y_2 \times t_3$
$t_3 = Z_2^2$
$t_5 = Z_2 \times t_3$
$t_7 = X_1 \times t_3$
$t_8 = Y_1 \times t_5$
$t_1 = t_1 - t_7$
$t_4 = t_4 - t_8$
index $= 3$
if $t_1 = 0$ then                                        [*]
    index $= 0$          $\{R = \mathcal{O}\}$
    if $t_4 = 0$ then index $= 2$      $\{R = 2P\}$      [*]
if $P = \mathcal{O}$ then index $= 1$          $\{R = Q\}$      [*]
if $Q = \mathcal{O}$ then index $= 4$          $\{R = P\}$      [*]
mask $= 0$
if index $= 3$ then mask $= 1$
{case $P + Q$, else any other case}                      [*]
$t_3 = X_1 + t_2$
$t_6 = X_1 - t_2$
if mask $= 0$ then $t_2 = Y_1$ else $t_2 = t_1$           [*]
$t_5 = t_2^2$
if mask $= 0$ then $t_7 = X_1$                          [*]
$t_1 = t_5 \times t_7$
$\tilde{Z}_2 = Z_1 \times t_2$
$\tilde{Z}_3 = Z_2 \times \tilde{Z}_2$
if mask $\neq 0$ then $t_3 = t_2$                        [*]
if mask $\neq 0$ then $t_6 = t_5$                        [*]
$t_2 = t_3 \times t_6$
$t_3 = t_2/2$
$t_3 = t_2 + t_3$
if mask $\neq 0$ then $t_3 = t_4$                        [*]
$t_4 = t_3^2$
$t_4 = t_4 - t_1$
$\tilde{X}_2 = t_4 - t_1$
$\tilde{X}_3 = \tilde{X}_2 - t_2$
if mask $= 0$ then $t_4 = \tilde{X}_2$ else $t_4 = \tilde{X}_3$      [*]
$t_1 = t_1 - t_4$
$t_4 = t_3 \times t_1$
if mask $= 0$ then $t_1 = t_5$ else $t_1 = t_8$           [*]
if mask $= 0$ then $t_2 = t_5$                          [*]
$t_3 = t_1 \times t_2$
$\tilde{Y}_2 = t_4 - t_3$
$\tilde{Y}_3 = \tilde{Y}_2$
$R = T[\text{index}]$   $(= (\tilde{X}_{\text{index}}, \tilde{Y}_{\text{index}}, \tilde{Z}_{\text{index}}))$      [*]
return $R$
```

# D Traces of Frobenius

**Table 6.** The traces of Frobenius $t$ for the curves in Tables 1 and 2. Compute group orders as $\#E(\mathbf{F}_p) = p+1-t$ and $\#E'(\mathbf{F}_p) = p+1+t$ for $E \in \{E_b, \mathcal{E}_d\}$.

| curve $E_b$ | trace |
|---|---:|
| **w-256-mont** | 0x3AE8AEC191AF8B462EF3A1E5867A815 |
| **w-254-mont** | 0x147E7415F25C8A3F905BE63B507207C1 |
| **w-256-mers** | 0x1BC37D8A15D9A39FDF54DFD6B8AE571F |
| **w-255-mers** | 0x79B5C7D7C52D4C2054705367C3A6B219 |
| **w-384-mont** | 0x456480EB358AEDAC85B1232C7583BE25D641B76B4D671145 |
| **w-382-mont** | 0x5914E300B421DEB28C4CDE002717D32E9F54797FC144CFE3 |
| **w-384-mers** | 0x29E150E114A2977E412562C2B3C81D859FB27E0984F19D0B |
| **w-383-mers** | 0x563507EB575EE952604F4BFCABE8550CE6D6803F4485BABD |
| **w-512-mont** | 0x9C757286D118AFD67F9B550F47B6719E20C2C66AF9B128C46C69D70E81670237 |
| **w-510-mont** | 0x46EB93321EAF10CC8B854D62E19A8C272DD216A1CDDCFC0C5FF4DFF6790565D3 |
| **w-512-mers** | 0xA4C35B046B187CE4B03DA712682F4239C4A974C99F832DBC31EAC0C6FBCCA86B |
| **w-511-mers** | 0x724105C0A12627C65D2B01900AE91780572C19A95F06605E0FEFA08C4C462C81 |
| **w-521-mers** | 0xEC68BAF7857C1B89CA8C6EEA5B33425078F28632FC263E5626E091FCE826B47F7 |

| curve $\mathcal{E}_d$ | trace |
|---|---:|
| **ed-256- mont** | 0x13AAD11411E6330DA649B44849C4E1154 |
| **ed-254- mont** | 0x51AB3E4DD0A7413C5430B004EE459CE4 |
| **ed-256-mers** | 0x106556A94BD650E6C691EC643BB752C90 |
| **ed-255-mers** | 0x8C3961E84965F3454ED8B84BEF244F30 |
| **ed-384- mont** | 0x2A4BE076C762D8C9825225944DFC2407E406C7167336DD94 |
| **ed-382- mont** | 0xB394157AB7C8FA209CFA7E8EDF87E5F659DFF2586830167C |
| **ed-384-mers** | 0x4CA0BB84A976997697B17EE9C7182C6EB8A4A3823EF64630 |
| **ed-383-mers** | 0x3BBDA3EC630981110CAA5E0D854D777E40050C4F9160DDE8 |
| **ed-512- mont** | 0xCCC0A98C8F32E3CBBF3E7EBB024842CB2099437935363F81733ADE04D1C927EC |
| **ed-510- mont** | 0xA0C4BB860F4395023A482F564F6E7DFD280CF7DBA06996F4DE9F78C8324AB93C |
| **ed-512-mers** | 0x1606BDFD840951119676E1EC2EDAAE83C8C56803CD1FFC1DAC61CB8D3D283F7A4 |
| **ed-511-mers** | 0x560F2F9F46F87459155B3C6E1CEDD9236AF63E504E83379AC20F45C1CBAF41DC |
| **ed-521-mers** | 0x3FDBE33AA148655EB858032064704A4970AD77413EC18570EF44ABE8020D9AFAC |

# E Costs of Point Conversion

**Table 7.** The cost of converting points when using the curves from Tables 1 and 2. This is used for point decompressing and converting between twisted Edwards and Montgomery (and vice versa). The cost is expressed in the number of exponentiations (exp), multiplications (mul), multiplication by constants (mulc) and squarings (squ). Let $E_A/\mathbf{F}_p : v^2 = u^3 + Au^2 + u$ and $\mathcal{E}_d/\mathbf{F}_p : -x^2 + y^2 = 1 + dx^2y^2$ with $B = -(A+2)$ a square in $\mathbf{F}_p$ and $d = -(A-2)/(A+2)$. Let $(X\colon Y\colon Z)$ be the projective coordinates for $\mathcal{E}$. We follow the approach described in [9] to decompress twisted Edwards points.

| Edwards to Montgomery | | |
|---|---|---|
| conversion | formula | cost |
| $(x, y)$ to $(u)$ | $u = (1+y)(1-y)^{p-2}$ | 1 exp, 1 mul |
| $(y)$ to $(u)$ | $u = (1+y)(1-y)^{p-2}$ | 1 exp, 1 mul |
| $(X : Y : Z)$ to $(u)$ | $u = (Z+Y)(Z-Y)^{p-2}$ | 1 exp, 1 mul |

| Edwards to Edwards | | |
|---|---|---|
| conversion | formula | cost |
| $(y)$ to $(x, y)$ | $a = y^2 - 1$<br>$b = dy^2 + 1$<br>$x = ab(ab^3)^{(p-3)/4}$ | 1 exp, 3 mul, 2 squ |

| Montgomery to Edwards | | |
|---|---|---|
| conversion | formula | cost |
| $(u)$ to $(y)$ | $y = (u-1)(u+1)^{p-2}$ | 1 exp, 1 mul |
| $(u)$ to $(x, y)$ | $x = u\sqrt{B}(u^3 + Au^2 + u)^{(3p-5)/4}$<br>$y = (u-1)(u+1)^{p-2}$ | 2 exp, 2 mul, 1 mulc, 1 squ |

| Weierstrass to Weierstrass | | |
|---|---|---|
| conversion | formula | cost |
| $(x)$ to $(x, y)$ | $y = (x^3 - 3x + b)^{p-2}$ | 1 exp, 1 mul, 1 squ |