Complete addition formulas for prime order elliptic curves

Joost Renes[1], Craig Costello[2], and Lejla Batina[1]

[1] Radboud University, Digital Security, Nijmegen, The Netherlands
{j.renes, lejla}@cs.ru.nl
[2] Microsoft Research, Redmond, USA
craigco@microsoft.com

Abstract. An elliptic curve addition law is said to be *complete* if it correctly computes the sum of *any* two points in the elliptic curve group. One of the main reasons for the increased popularity of Edwards curves in the ECC community is that they can allow a complete group law that is also relatively efficient (e.g., when compared to all known addition laws on Edwards curves). Such complete addition formulas can simplify the task of an ECC implementer and, at the same time, can greatly reduce the potential vulnerabilities of a cryptosystem. Unfortunately, until now, complete addition laws that are relatively efficient have only been proposed on curves of composite order and have thus been incompatible with all of the currently standardized prime order curves.

In this paper we present optimized addition formulas that are complete on *every* prime order short Weierstrass curve defined over a field k with $\mathrm{char}(k) \neq 2, 3$. Compared to their incomplete counterparts, these formulas require a larger number of field additions, but interestingly require fewer field multiplications. We discuss how these formulas can be used to achieve secure, exception-free implementations on *all* of the prime order curves in the NIST (and many other) standards.

1 Introduction

Extending the works of Lange–Ruppert [48] and Bosma–Lenstra [19], Arène, Kohel and Ritzenthaler [4] showed that, under any projective embedding of an elliptic curve E/k, *every* addition law has pairs of exceptional points in $(E \times E)(\bar{k})$. That is, over the algebraic closure of k, there are always pairs of points for which a given elliptic curve addition law does not work.

Fortunately, in elliptic curve cryptography (ECC), we are most often only concerned with the k-rational points on E. In this case it is possible to have a single addition law that is well-defined on all pairs of k-rational points, because its exceptional pairs are found in $(E \times E)(\bar{k})$, but not in $(E \times E)(k)$. A celebrated example of this is the Edwards model [31]; when suitably chosen [12], an Edwards curve has a simple addition law that works for all pairs of k-rational points. This phenomenon was characterized more generally over elliptic curves by Kohel [47], and further generalized to arbitrary abelian varieties in [4]. For our purposes it suffices to state a special case of the more general results in [47, 4]: namely, that every elliptic curve E over a finite field $\mathbb{F}_q$ (with $q \geq 5$) has an $\mathbb{F}_q$-complete addition law corresponding to the short Weierstrass model in $\mathbb{P}^2(\mathbb{F}_q)$.

Addition laws that are $\mathbb{F}_q$-complete are highly desirable in ECC. They can significantly simplify the task of an implementer and greatly reduce the potential vulnerabilities of a cryptosystem. We elaborate on this below.

Our contributions. In Algorithm 1 we present optimized point addition formulas that correctly compute the sum of *any* two points on *any* odd order elliptic curve $E/\mathbb{F}_q\colon y^2 =$

$x^3 + ax + b$ with $q \geq 5$. We do not claim credit for the complete formulas themselves, as these are exactly the formulas given by Bosma and Lenstra two decades ago [19]. What is novel in this paper is optimizing the explicit computation of these formulas for cryptographic application. In particular, Table 1 shows that the computation of the Bosma–Lenstra complete additions can be performed using fewer general field multiplications than the best known (incomplete!) addition formulas on short Weierstrass curves: excluding multiplications by curve constants and field additions, the explicit formulas in this paper compute additions in 12 field multiplications ($12\mathbf{M}$), while the fastest known addition formulas in homogeneous coordinates require 14 field multiplications ($12\mathbf{M} + 2\mathbf{S}$) and the fastest known addition formulas in Jacobian coordinates require 16 field multiplications ($11\mathbf{M} + 5\mathbf{S}$). We immediately note, however, that our explicit formulas incur a much larger number of field additions than their incomplete counterparts. Thus, as is discussed at length below, the relative performance of the complete additions will be highly dependent on the platform and/or scenario. However, we stress that outperforming the incomplete addition formulas is not the point of this paper: our aim is to provide the fastest possible complete formulas for prime order curves.

addition formulas	a	excep. Q in ADD(P,Q)	ADD(P,Q)					excep. in DBL(P)	DBL(P)					ref.
			M	**S**	$\mathbf{m_a}$	$\mathbf{m_b}$	**a**		**M**	**S**	$\mathbf{m_a}$	$\mathbf{m_b}$	**a**	
complete homog.	any	**none**	12	0	3	2	23	**none**	8	3	3	2	15	**this work**
	−3		12	0	0	2	29		8	3	0	2	21	
	0		12	0	0	2	19		6	2	0	1	9	
incomplete homog.	any	$\pm P, \mathcal{O}$	12	2	0	0	7	$\mathcal{O}$	5	6	1	0	12	[27, 15]
	−3		12	2	0	0	7		7	3	0	0	11	[27, 15]
	0		-						-					-
incomplete Jacobian	any	$\pm P, \mathcal{O}$	12	4	0	0	7	none	3	6	1	0	13	[27]
	−3		12	4	0	0	7		4	4	0	0	8	[27, 51]
	0		12	4	0	0	7		3	4	0	0	7	[27, 42]

Table 1. Summary of explicit formulas for the addition law on prime order short Weierstrass elliptic curves $E/k\colon y^2 = x^3 + ax + b$ in either homogeneous (homog.) coordinates or Jacobian coordinates, and the corresponding exceptions (excep.) in both points doublings (DBL) and point additions (ADD). Here the operation counts include multiplications ($\mathbf{M}$), squarings ($\mathbf{S}$), multiplications by a ($\mathbf{m_a}$), multiplications by (small multiples of) b ($\mathbf{m_b}$), and additions ($\mathbf{a}$), all in the ground field k. We note that various trade-offs exist with several of the above formulas, in particular for point doublings in Jacobian coordinates – see [15].

Wide applicability. While the existence of an $\mathbb{F}_q$-complete addition law for prime order Weierstrass curves is not news to mathematicians (or to anyone that has read, e.g., [19, 4]), we hope it might be a pleasant surprise to ECC practitioners. In particular, the benefits of completeness are now accessible to anyone whose task it is to securely implement the prime order curves in the standards. These include:

- The example curves originally specified in the working drafts of the American National Standards Institute (ANSI), versions X9.62 and X9.63 [1, 2].
- The five NIST prime curves specified in the current USA digital signature standard (DSS), i.e., FIPS 186-4 – see [55, 56]. This includes Curve P-384, which is the National Security Agency (NSA) recommended curve in the most recent Suite B fact sheet for both key

exchange and digital signatures [60, 28]; Curve P-256, which is the most widely supported curve in the Secure Shell (SSH) and Transport Layer Security (TLS) protocol [17, §3.2-3.3]; and Curve P-192, which is the most common elliptic curve used in Austria's national e-ID cards [17, §3.4].

– The seven curves specified in the German brainpool standard [30], i.e., `brainpoolPXXXr1`, where XXX $\in \{160, 192, 224, 256, 320, 384, 512\}$.
– The eight curves specified by the UK-based company Certivox [23], i.e., `ssc-XXX`, where XXX $\in \{160, 192, 224, 256, 288, 320, 384, 512\}$.
– The curve `FRP256v1` recommended by the French Agence nationale de la sécurité des systèmes d'information (ANSSI) [3].
– The three curves specified (in addition to the above NIST prime curves) in the Certicom SEC 2 standard [22]. This includes `secp256k1`, which is the curve used in the Bitcoin protocol.
– The recommended curve in the Chinese SM2 digital signature algorithm [25].
– The example curve in the Russian GOST R 34.10 standard [35].

In particular, implementers can now write secure, exception-free code that supports all of the above curves without ever having to look further than Algorithm 1 for curve arithmetic. Moreover, in §5.2 we show how Algorithm 1 can easily be used to securely implement the two composite order curves, Curve25519 [7] and Ed448-Goldilocks [39], recently recommended for inclusion in future versions of TLS by the Internet Research Task Force Crypto Forum Research Group (IRTF CFRG).

Side-channel protection. Real-world implementations of ECC have a number of potential side-channel vulnerabilities that can fall victim to simple timing attacks [46] or exceptional point attacks [43, 32]. One of the main reasons these attacks pose a threat is the *branching* that is inherent in the schoolbook short Weierstrass elliptic curve addition operation. For example, among the dozens of `if` statements in OpenSSL's[3] standard addition function "`ec_GFp_simple_add`", the initial three that check whether the input points are equal, opposite, or at infinity can cause timing variability (and therefore leak secret data) in ECDH or ECDSA. The complete formulas in this paper remove these vulnerabilities and significantly decrease the attack surface of a cryptosystem. As Bernstein and Lange point out [13], completeness "eases implementations" and "avoids simple side-channel attacks".

Although it is possible to use incomplete formulas safely, e.g., by carefully deriving uniform scalar multiplication algorithms that avoid exceptional pairs of inputs, implementing these routines in *constant-time* and in a provably correct way can be a cumbersome and painstaking process [16, §4]. Constant-time ECC implementations typically recode scalars from their binary encoding to some other form that allows a uniform execution path (cf. Okeya-Tagaki [57] and Joye-Tunstall [44]), and these recodings can complicate the analysis of exceptional inputs to the point addition functions. For example, it can be difficult to prove that the running value in a scalar multiplication is never equal to (or the inverse of) elements in the lookup table; if this equality occurs before an addition, the incomplete addition function is likely to fail. Furthermore, guaranteeing exception-free, constant-time implementations of more exotic scalar multiplication routines, e.g., multiscalar multiplication for ECDSA verification, fixed-base scalar multiplications [49], scalar multiplications exploiting endomorphisms [34], or scalar multiplications using common power analysis countermeasures [29, 33], is even more difficult; that is, unless the routine can call complete addition formulas.

---

[3] See `ec_smpl.c` in `crypto/ec/` in the latest release at `http://openssl.org/source/`.

Performance considerations. While the wide applicability and correctness of Algorithm 1 is at the heart of this paper, we have also aimed to cater to implementers that do not want to sacrifice free performance gains, particularly those concerned with supporting a special curve or special family of curves. To that end, Algorithms 2-9 give faster complete addition formulas in the special (and standardized) cases that the Weierstrass curve constant a is $a = -3$ or $a = 0$, and in the special cases of point doublings; Table 1 summarizes the operation counts for all of these scenarios.

As we mentioned above, outperforming the (previously deployed) incomplete addition formulas is not the point of this paper. Indeed, the high number of field additions present in our complete addition algorithms are likely to introduce an overall slowdown in many scenarios. To give an idea of this *performance hit* in a common software scenario, we plugged our complete addition algorithms into OpenSSL's implementation of the five NIST prime curves. Using the `openssl speed` function to benchmark the performance of the existing incomplete formulas and the new complete formulas shows that the latter incurs between a 1.34x and 1.44x slowdown in an average run of the elliptic curve Diffie-Hellman (ECDH) protocol (see Table 2 for the full details). As we discuss below, and in detail in §5.3, this factor slowdown should be considered an upper bound on the difference in performance between the fastest incomplete algorithms and our complete ones.

On the contrary, there are example scenarios where plugging in the complete formulas will result in an unnoticeable performance difference, or possibly even a speedup. For example, compared to the incomplete addition function `secp256k1_gej_add_var` used in the Bitcoin code[4], our complete addition function in Algorithm 7 saves $4\mathbf{S}$ at the cost of $8\mathbf{a} + 1\mathtt{mul\_int}$[5]; compared to Bitcoin's incomplete mixed addition function `secp256k1_gej_add_ge_var`, our complete mixed addition saves $3\mathbf{S}$ at the cost of $3\mathbf{M} + 2\mathbf{a} + 1\mathtt{mul\_int}$; and, compared to Bitcoin's doubling function `secp256k1_gej_double_var`, our formulas save $2\mathbf{S} + 5\mathtt{mul\_int}$ at the cost of $3\mathbf{M} + 3\mathbf{a}$. In this case it is unclear which set of formulas would perform faster, but it is likely to be relatively close and to depend on the underlying field arithmetic and/or target platform. Furthermore, the overall speed is not just dependent on the formulas: the `if` statements present in the Bitcoin code also hamper performance. On the contrary, the complete algorithms in this paper have no `if` statements.

There are a number of additional real-world scenarios where the performance gap between the incomplete and the complete formulas will not be as drastic as the OpenSSL example above. The operation counts in Table 1 and Table 3 suggest that this will occur when the cost of field multiplications and squarings heavily outweighs the cost of field additions. The benchmarks above were obtained on a 64-bit processor, where the $\mathbf{M}/\mathbf{a}$ ratio tends to be much lower than that of low-end (e.g., 8-, 16-, and 32-bit) architectures. For example, field multiplications on wireless sensor nodes commonly require over 10 times more clock cycles than a field addition (e.g., see [50, Table 1] and [59, Table 1]), and in those cases the complete formulas in this paper are likely to be very competitive in terms of raw performance.

In any case, we believe that many practitioners will agree that a small performance difference is a worthwhile cost to pay for branch-free point addition formulas that culminate in much simpler and more compact code, which *guarantees* correctness of the outputs and eliminates several side-channel vulnerabilities. We also note that the Bitcoin curve is not an isolated example of the more favorable formula comparison above: all of the most popular

---

[4] See `https://github.com/bitcoin/bitcoin/tree/master/src/secp256k1`.

[5] `mul_int` denotes the cost of Bitcoin's specialized function that multiplies field elements by small integers.

pairing-friendly curves, including Barreto-Naehrig (BN) curves [5] which have appeared in recent IETF drafts[6], also have $a = 0$. In those cases, our specialized, exception-free formulas give implementers an easy way to correctly implement curve arithmetic in both $\mathbb{G}_1$ and $\mathbb{G}_2$ in the setting of cryptographic pairings. On a related note, we point that the word "prime" in our title can be relaxed to "odd"; the completeness of the Bosma–Lenstra formulas only requires the non-existence of rational two-torsion points (see Sections 2 and 3), i.e., that the group order $\#E(\mathbb{F}_q)$ is not even. BN curves define $\mathbb{G}_2$ as (being isomorphic to) a proper subgroup of a curve $E'/\mathbb{F}_{p^2}$, whose group order $\#E'(\mathbb{F}_{p^2})$ is the product of a large prime with odd integers [5, §3], meaning that our explicit formulas are not only complete in $\mathbb{G}_2 \subset E'(\mathbb{F}_{p^2})$, but also in $E'(\mathbb{F}_{p^2})$.

Related work. Complete addition laws have been found and studied on non-Weierstrass models of elliptic curves, e.g., on the (twisted) Edwards [12, 8] and (twisted) Hessian models [9]. Unfortunately, in all of those scenarios, the models are not compatible with prime order curves and therefore all of the standardized curves mentioned above.

In terms of obtaining a complete and computationally efficient addition algorithm for prime order curves, there has been little success to date. Bernstein and Lange [13] found complete formulas on a non-Weierstrass model that would be compatible with, e.g., the NIST curves, reporting explicit formulas that (ignoring additions and multiplications by curve constants) cost $26\mathbf{M} + 8\mathbf{S}$. Bos *et al.* [16] considered applying the set of two Bosma–Lenstra addition laws to certain prime order Weierstrass curves, missing the observation (cf. [4, Remark 4.4]) that one of the addition laws is enough, and abandoning the high cost of computing both addition laws for an alternative but more complicated approach towards side-channel protection [16, Appendix C]. Brier and Joye [20] developed *unified* formulas[7] for general Weierstrass curves, but these formulas still have exceptions and (again, ignoring additions and multiplications by curve constants) require $11\mathbf{M} + 6\mathbf{S}$, which is significantly slower than our complete algorithms.

Prime order curves can be safe. Several of the standardized prime order curves mentioned above have recently been critiqued in [14], where they were deemed not to meet (some or all of) the four "ECC security" requirements: (i) Ladder, (ii) Twists, (iii) Completeness, and (iv) Indistinguishability.

On the contrary, this paper shows that prime order curves have complete formulas that are comparably efficient. In addition, Brier and Joye [20, §4] extended the Montgomery ladder to all short Weierstrass curves. In particular, when $E/\mathbb{F}_q\colon y^2 = x^3 + ax + b$ is a prime order curve, their formulas give rise to a function `ladder` that computes $x([m]P) = \mathtt{ladder}(x(P), m, a, b)$ for the points $P \in E(\mathbb{F}_{q^2})$ with $(x, y) \in \mathbb{F}_q \times \mathbb{F}_{q^2}$, that is, a function that works for all $x \in \mathbb{F}_q$ and that does not distinguish whether x corresponds to a point on the curve E, or to a point on its quadratic twist $E'\colon dy^2 = x^3 + ax + b$, where d is non-square in $\mathbb{F}_q$. If E is chosen to be twist-secure (this presents no problem in the prime order setting), then for all $x \in \mathbb{F}_q$, the function $\mathtt{ladder}(x, m, a, b)$ returns an instance of the discrete logarithm problem (whose solution is m) on a cryptographically strong curve, just like the analogous function on twist-secure Montgomery curves [7]. Finally, we note that Tibouchi [61] presented a prime-order

---

[6] See `http://datatracker.ietf.org/doc/draft-kasamatsu-bncurves-01`.
[7] These are addition formulas that also work for point doublings.

analogue of the encoding given for certain composite-order curves in [11], showing that the indistinguishability property can also be achieved on prime order curves.

As is discussed in [14], adopting the Brier-Joye ladder (or, in our case, the complete formulas) in place of the fastest formulas presents implementers with a trade-off between "simplicity, security and speed". However, these same trade-offs also exist on certain choices of Edwards curves, where, for example, the fastest explicit formulas are also not complete: the Curve41417 implementation chooses to sacrifice the fastest coordinate system for the sake of completeness [10, §3.1], while the Goldilocks implementation goes to more complicated lengths to use the fastest formulas [37–39]. Furthermore, there is an additional category that is not considered in [14], i.e., the non-trivial security issues related to having a cofactor h greater than 1 [38, §1.1].

Given the complete explicit formulas in this paper, it is our opinion that well-chosen prime order curves can be considered safe choices for elliptic curve cryptography. It is well-known that curves with cofactors offer efficiency benefits in certain scenarios, but to our knowledge, efficiency and/or bandwidth issues are the only valid justifications for choosing a curve with a cofactor $h > 1$.

Organization. Section 2 briefly gives some preliminaries and notation. Section 3 presents the complete addition algorithms. In Section 4 we give intuition as to why these explicit formulas are optimal, or close to optimal, for prime order curves in short Weierstrass form. In Section 5 we discuss how these formulas can be used in practice. The appendix provides Magma [18] scripts that can be used to verify our explicit algorithms and operation counts.

2   Preliminaries

Let k be a field of characteristic not two or three, and $\mathbb{P}^2(k)$ be the homogeneous projective plane of dimension two. Two points $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ in $\mathbb{P}^2(k)$ are equal if and only if there exist $\lambda \in k^\times$ such that $(X_1, Y_1, Z_1) = (\lambda X_2, \lambda Y_2, \lambda Z_2)$.

Let E/k be an elliptic curve embedded in $\mathbb{P}^2(k)$ as a Weierstrass model $E/k \colon Y^2 Z = X^3 + aXZ^2 + bZ^3$. The points on E form an abelian group with identity $\mathcal{O} = (0 : 1 : 0)$. An *addition law* on E is a triple of polynomials (X_3, Y_3, Z_3) such that the map

$$P, Q \mapsto (X_3(P, Q) : Y_3(P, Q) : Z_3(P, Q))$$

determines the group law $+$ on an open subset of $(E \times E)(\bar{k})$, where $\bar{k}$ is the algebraic closure of k. For an extension K of k, a set of such addition laws is said to be K-*complete* if, for any pair of K-rational pair of points (P, Q), at least one addition law in the set is defined at (P, Q).

Lange and Ruppert [48] proved that the space of all addition laws on E has dimension 3, and Bosma and Lenstra [19] proved that a $\bar{k}$-complete set must contain (at least) two addition laws. In other words, Bosma and Lenstra proved that every addition law on E has at least one exceptional pair of inputs over the algebraic closure. More recent work by Arène, Kohel and Ritzenthaler [4] showed that this is true without assuming a Weierstrass embedding of E. That is, they showed that every elliptic curve addition law has exceptional pairs over the algebraic closure, irrespective of the projective embedding.

Following [19], for positive integers μ and ν, we define an *addition law of bidegree* (μ, ν) to be a triple of polynomials

$$X_3, Y_3, Z_3 \in k[X_1, Y_1, Z_1, X_2, Y_2, Z_2]$$

that satisfy the following two properties:

1. The polynomials X_3, Y_3 and Z_3 are homogeneous of degree μ in X_1, Y_1 and Z_1, and are homogeneous of degree ν in X_2 Y_2 and Z_2;
2. Let $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$ be in $E(K)$, where K is an extension of k. Then either X_3, Y_3 and Z_3 are all zero at P and Q, or else $(X_3 : Y_3 : Z_3)$ is an element of $E(K)$ and is equal to $P + Q$. If the first holds, we say that the pair (P, Q) is *exceptional* for the addition law. If there are no exceptional pairs of points, we say that the addition law is *K-complete* (note that this is in line with the definition of a K-complete set of addition laws).

Hereafter, if a single addition law is k-complete, we simply call it *complete*.

3 Complete addition formulas

Let $E/k \colon Y^2 Z = X^3 + aXZ^2 + bZ^3 \subset \mathbb{P}^2$ with $\mathrm{char}(k) \neq 2, 3$. The complete addition formulas optimized in this section follow from the theorem of Bosma and Lenstra [19, Theorem 2], which states that, for any extension field K/k, there exists a 1-to-1 correspondence between lines in $\mathbb{P}^2(K)$ and addition laws of bidegree $(2, 2)$ on $E(K)$. Two points P and Q in $E(K)$ are then exceptional for an addition law if and only if $P - Q$ lies on the corresponding line. When $K = \bar{k}$, the algebraic closure of k, *every* line intersects $E(K)$; thus, one consequence of this theorem is that every addition law of bidegree $(2, 2)$ has an exceptional pair over the algebraic closure.

The addition law considered in this paper is the addition law corresponding to the line $Y = 0$ in $\mathbb{P}^2$ in [19], specialized to the short Weierstrass embedding of E above. For two points $P = (X_1 : Y_1 : Z_1)$, $Q = (X_2 : Y_2 : Z_2)$ on E, the sum $(X_3 : Y_3 : Z_3) = P + Q$ is given by

$$
\begin{aligned}
X_3 ={}& Y_1 Y_2 (X_1 Y_2 + X_2 Y_1) - a X_1 X_2 (Y_1 Z_2 + Y_2 Z_1) \\
& - a (X_1 Y_2 + X_2 Y_1)(X_1 Z_2 + X_2 Z_1) - 3b(X_1 Y_2 + X_2 Y_1) Z_1 Z_2 \\
& - 3b(X_1 Z_2 + X_2 Z_1)(Y_1 Z_2 + Y_2 Z_1) + a^2 (Y_1 Z_2 + Y_2 Z_1) Z_1 Z_2, \\
Y_3 ={}& Y_1^2 Y_2^2 + 3a X_1^2 X_2^2 + 9b X_1 X_2 (X_1 Z_2 + X_2 Z_1) \\
& - 2a^2 X_1 Z_2 (X_1 Z_2 + 2 X_2 Z_1) + a^2 (X_1 Z_2 + X_2 Z_1)(X_1 Z_2 - X_2 Z_1) \\
& - 3ab X_1 Z_1 Z_2^2 - 3ab X_2 Z_1^2 Z_2 - (a^3 + 9b^2) Z_1^2 Z_2^2, \\
Z_3 ={}& 3 X_1 X_2 (X_1 Y_2 + X_2 Y_1) + Y_1 Y_2 (Y_1 Z_2 + Y_2 Z_1) \\
& + a(X_1 Y_2 + X_2 Y_1) Z_1 Z_2 + a(X_1 Z_2 + X_2 Z_1)(Y_1 Z_2 + Y_2 Z_1) \\
& + 3b(Y_1 Z_2 + Y_2 Z_1) Z_1 Z_2.
\end{aligned}
$$

Bosma and Lenstra prove that a pair of points (P, Q) is exceptional for this addition law if and only if $P - Q$ is a point of order two.

Exceptions. Throughout this paper, we fix $q \geq 5$ and assume throughout that $E(\mathbb{F}_q)$ has prime order to exclude $\mathbb{F}_q$-rational points of order two, so that the above formulas are complete. However, we note that the explicit algorithms that are derived in Section 3 will, firstly, be complete for any short Weierstrass curves of odd order, and secondly, also be exception-free for all pairs of points inside odd order subgroups on any short Weierstrass curve. In particular, this means that they can also be used to compute exception-free additions and scalar multiplications on certain curves with an even order. We come back to this in §5.2.

3.1   The general case

Despite the attractive properties that come with completeness, this addition law seems to have been overlooked due to its apparent inefficiency. We now begin to show that these formulas are not as inefficient as they seem, to the point where the performance will be competitive with the fastest, incomplete addition laws in current implementations of prime order curves.

We start by rewriting the above formulas as

$$
\begin{aligned}
X_3 &= (X_1Y_2 + X_2Y_1)(Y_1Y_2 - a(X_1Z_2 + X_2Z_1) - 3bZ_1Z_2) \\
&\quad - (Y_1Z_2 + Y_2Z_1)(aX_1X_2 + 3b(X_1Z_2 + X_2Z_1) - a^2Z_1Z_2), \\
Y_3 &= (3X_1X_2 + aZ_1Z_2)(aX_1X_2 + 3b(X_1Z_2 + X_2Z_1) - a^2Z_1Z_2) + \\
&\quad (Y_1Y_2 + a(X_1Z_2 + X_2Z_1) + 3bZ_1Z_2)(Y_1Y_2 - a(X_1Z_2 + X_2Z_1) - 3bZ_1Z_2), \\
Z_3 &= (Y_1Z_2 + Y_2Z_1)(Y_1Y_2 + a(X_1Z_2 + X_2Z_1) + 3bZ_1Z_2) \\
&\quad + (X_1Y_2 + X_2Y_1)(3X_1X_2 + aZ_1Z_2).
\end{aligned} \tag{1}
$$

The rewritten formulas still appear somewhat cumbersome, but a closer inspection of (1) reveals that several terms are repeated. In Algorithm 1, we show that this can in fact be computed[8] using $12\mathbf{M} + 3\mathbf{m_a} + 2\mathbf{m_{3b}} + 23\mathbf{a}$[9].

Algorithm 1: Complete, projective point addition for arbitrary prime order short Weierstrass curves $E/\mathbb{F}_q \colon y^2 = x^3 + ax + b$.

Require: $P = (X_1 : Y_1 : Z_1)$, $Q = (X_2 : Y_2 : Z_2)$, $E \colon Y^2Z = X^3 + aXZ^2 + bZ^3$,
    and $b_3 = 3 \cdot b$.
Ensure: $(X_3 : Y_3 : Z_3) = P + Q$.

1. $t_0 \leftarrow X_1 \cdot X_2$	2. $t_1 \leftarrow Y_1 \cdot Y_2$	3. $t_2 \leftarrow Z_1 \cdot Z_2$
4. $t_3 \leftarrow X_1 + Y_1$	5. $t_4 \leftarrow X_2 + Y_2$	6. $t_3 \leftarrow t_3 \cdot t_4$
7. $t_4 \leftarrow t_0 + t_1$	8. $t_3 \leftarrow t_3 - t_4$	9. $t_4 \leftarrow X_1 + Z_1$
10. $t_5 \leftarrow X_2 + Z_2$	11. $t_4 \leftarrow t_4 \cdot t_5$	12. $t_5 \leftarrow t_0 + t_2$
13. $t_4 \leftarrow t_4 - t_5$	14. $t_5 \leftarrow Y_1 + Z_1$	15. $X_3 \leftarrow Y_2 + Z_2$
16. $t_5 \leftarrow t_5 \cdot X_3$	17. $X_3 \leftarrow t_1 + t_2$	18. $t_5 \leftarrow t_5 - X_3$
19. $Z_3 \leftarrow a \cdot t_4$	20. $X_3 \leftarrow b_3 \cdot t_2$	21. $Z_3 \leftarrow X_3 + Z_3$
22. $X_3 \leftarrow t_1 - Z_3$	23. $Z_3 \leftarrow t_1 + Z_3$	24. $Y_3 \leftarrow X_3 \cdot Z_3$
25. $t_1 \leftarrow t_0 + t_0$	26. $t_1 \leftarrow t_1 + t_0$	27. $t_2 \leftarrow a \cdot t_2$
28. $t_4 \leftarrow b_3 \cdot t_4$	29. $t_1 \leftarrow t_1 + t_2$	30. $t_2 \leftarrow t_0 - t_2$
31. $t_2 \leftarrow a \cdot t_2$	32. $t_4 \leftarrow t_4 + t_2$	33. $t_0 \leftarrow t_1 \cdot t_4$
34. $Y_3 \leftarrow Y_3 + t_0$	35. $t_0 \leftarrow t_5 \cdot t_4$	36. $X_3 \leftarrow t_3 \cdot X_3$
37. $X_3 \leftarrow X_3 - t_0$	38. $t_0 \leftarrow t_3 \cdot t_1$	39. $Z_3 \leftarrow t_5 \cdot Z_3$
40. $Z_3 \leftarrow Z_3 + t_0$		

Although Algorithm 1 is sufficient for cryptographic implementations, performance gains can be obtained by specializing the point additions to the useful scenarios of mixed additions[10]

---

[8] Notation here is the same as in Table 1, except for $\mathbf{m_{3b}}$ which denotes multiplication by the curve constant $3b$.

[9] We thank Emmanuel Thomé whose careful read-through resulted in a $1\mathbf{m}_a$ saving in all three of the explicit formulas for the general case.

[10] We note that it is not technically correct to call "mixed" additions complete, since $Z_2 = 1$ precludes the second point being the point at infinity. However, this is not a problem in practice as the second point is

(i.e., where $Z_2 = 1$) and/or point doublings (i.e., where $P = Q$). The mixed addition follows the same formulas as for point addition; Algorithm 2 shows this can be done in $11\mathbf{M} + 3\mathbf{m_a} + 2\mathbf{m_{3b}} + 17\mathbf{a}$.

Algorithm 2: Complete, mixed point addition for arbitrary prime order short Weierstrass curves $E/\mathbb{F}_q\colon y^2 = x^3 + ax + b$.

Require: $P = (X_1 : Y_1 : Z_1)$, $Q = (X_2 : Y_2 : 1)$, $E\colon Y^2 Z = X^3 + aXZ^2 + bZ^3$, and $b_3 = 3 \cdot b$.

Ensure: $(X_3 : Y_3 : Z_3) = P + Q$.

1. $t_0 \leftarrow X_1 \cdot X_2$	2. $t_1 \leftarrow Y_1 \cdot Y_2$	3. $t_3 \leftarrow X_2 + Y_2$
4. $t_4 \leftarrow X_1 + Y_1$	5. $t_3 \leftarrow t_3 \cdot t_4$	6. $t_4 \leftarrow t_0 + t_1$
7. $t_3 \leftarrow t_3 - t_4$	8. $t_4 \leftarrow X_2 \cdot Z_1$	9. $t_4 \leftarrow t_4 + X_1$
10. $t_5 \leftarrow Y_2 \cdot Z_1$	11. $t_5 \leftarrow t_5 + Y_1$	12. $Z_3 \leftarrow a \cdot t_4$
13. $X_3 \leftarrow b_3 \cdot Z_1$	14. $Z_3 \leftarrow X_3 + Z_3$	15. $X_3 \leftarrow t_1 - Z_3$
16. $Z_3 \leftarrow t_1 + Z_3$	17. $Y_3 \leftarrow X_3 \cdot Z_3$	18. $t_1 \leftarrow t_0 + t_0$
19. $t_1 \leftarrow t_1 + t_0$	20. $t_2 \leftarrow a \cdot Z_1$	21. $t_4 \leftarrow b_3 \cdot t_4$
22. $t_1 \leftarrow t_1 + t_2$	23. $t_2 \leftarrow t_0 - t_2$	24. $t_2 \leftarrow a \cdot t_2$
25. $t_4 \leftarrow t_4 + t_2$	26. $t_0 \leftarrow t_1 \cdot t_4$	27. $Y_3 \leftarrow Y_3 + t_0$
28. $t_0 \leftarrow t_5 \cdot t_4$	29. $X_3 \leftarrow t_3 \cdot X_3$	30. $X_3 \leftarrow X_3 - t_0$
31. $t_0 \leftarrow t_3 \cdot t_1$	32. $Z_3 \leftarrow t_5 \cdot Z_3$	33. $Z_3 \leftarrow Z_3 + t_0$

For a point $P = (X : Y : Z)$, doubling is computed as

$$
\begin{aligned}
X_3 &= 2XY(Y^2 - 2aXZ - 3bZ^2) \\
&\quad - 2YZ(aX^2 + 6bXZ - a^2Z^2), \\
Y_3 &= (Y^2 + 2aXZ + 3bZ^2)(Y^2 - 2aXZ - 3bZ^2) \\
&\quad + (3X^2 + aZ^2)(aX^2 + 6bXZ - a^2Z^2), \\
Z_3 &= 8Y^3Z.
\end{aligned}
$$

Algorithm 3 shows that this can be computed in $8\mathbf{M} + 3\mathbf{S} + 3\mathbf{m_a} + 2\mathbf{m_{3b}} + 15\mathbf{a}$.

3.2 Special cases of interest

$\boldsymbol{a = -3.}$ Several standards (e.g., [23, 56, 60, 3, 22, 30]) adopt short Weierstrass curves with the constant a being $a = -3$, which gives rise to faster explicit formulas for point doubling[11]. In this case, the complete formulas in (1) specialize to

---

typically taken from a precomputed lookup table consisting of small multiples of the input point $P \neq \mathcal{O}$. For prime order curves, these small multiples can never be at infinity.

[11] When $\mathbb{F}_q$ is a large prime field, $a = -3$ covers 1/2 (resp. 1/4) of the isomorphism classes for $q \equiv 3 \bmod 4$ (resp. $q \equiv 1 \bmod 4$) – see [21, §3].

Algorithm 3: Exception-free point doubling for arbitrary prime order short Weierstrass curves $E/\mathbb{F}_q\colon y^2 = x^3 + ax + b$.

Require: $P = (X : Y : Z)$ on $E\colon Y^2Z = X^3 + aXZ^2 + bZ^3$, and $b_3 = 3 \cdot b$.

Ensure: $(X_3 : Y_3 : Z_3) = 2P$.

1. $t_0 \leftarrow X \cdot X$	2. $t_1 \leftarrow Y \cdot Y$	3. $t_2 \leftarrow Z \cdot Z$
4. $t_3 \leftarrow X \cdot Y$	5. $t_3 \leftarrow t_3 + t_3$	6. $Z_3 \leftarrow X \cdot Z$
7. $Z_3 \leftarrow Z_3 + Z_3$	8. $X_3 \leftarrow a \cdot Z_3$	9. $Y_3 \leftarrow b_3 \cdot t_2$
10. $Y_3 \leftarrow X_3 + Y_3$	11. $X_3 \leftarrow t_1 - Y_3$	12. $Y_3 \leftarrow t_1 + Y_3$
13. $Y_3 \leftarrow X_3 \cdot Y_3$	14. $X_3 \leftarrow t_3 \cdot X_3$	15. $Z_3 \leftarrow b_3 \cdot Z_3$
16. $t_2 \leftarrow a \cdot t_2$	17. $t_3 \leftarrow t_0 - t_2$	18. $t_3 \leftarrow a \cdot t_3$
19. $t_3 \leftarrow t_3 + Z_3$	20. $Z_3 \leftarrow t_0 + t_0$	21. $t_0 \leftarrow Z_3 + t_0$
22. $t_0 \leftarrow t_0 + t_2$	23. $t_0 \leftarrow t_0 \cdot t_3$	24. $Y_3 \leftarrow Y_3 + t_0$
25. $t_2 \leftarrow Y \cdot Z$	26. $t_2 \leftarrow t_2 + t_2$	27. $t_0 \leftarrow t_2 \cdot t_3$
28. $X_3 \leftarrow X_3 - t_0$	29. $Z_3 \leftarrow t_2 \cdot t_1$	30. $Z_3 \leftarrow Z_3 + Z_3$
31. $Z_3 \leftarrow Z_3 + Z_3$		

$$\begin{aligned}
X_3 &= (X_1Y_2 + X_2Y_1)(Y_1Y_2 + 3(X_1Z_2 + X_2Z_1 - bZ_1Z_2)) \\
&\quad - 3(Y_1Z_2 + Y_2Z_1)(b(X_1Z_2 + X_2Z_1) - X_1X_2 - 3Z_1Z_2), \\
Y_3 &= 3(3X_1X_2 - 3Z_1Z_2)(b(X_1Z_2 + X_2Z_1) - X_1X_2 - 3Z_1Z_2) + \\
&\quad (Y_1Y_2 - 3(X_1Z_2 + X_2Z_1 - bZ_1Z_2))(Y_1Y_2 + 3(X_1Z_2 + X_2Z_1 - bZ_1Z_2)), \\
Z_3 &= (Y_1Z_2 + Y_2Z_1)(Y_1Y_2 - 3(X_1Z_2 + X_2Z_1 - bZ_1Z_2)) \\
&\quad + (X_1Y_2 + X_2Y_1)(3X_1X_2 - 3Z_1Z_2).
\end{aligned}$$

These can be computed at a cost of $12\mathbf{M} + 2\mathbf{m_b} + 29\mathbf{a}$ using Algorithm 4. The mixed addition can be done at a cost of $11\mathbf{M} + 2\mathbf{m_b} + 23\mathbf{a}$, as shown in Algorithm 5.

Algorithm 4: Complete, projective point addition for prime order short Weierstrass curves $E/\mathbb{F}_q\colon y^2 = x^3 + ax + b$ with $a = -3$.

Require: $P = (X_1 : Y_1 : Z_1)$, $Q = (X_2 : Y_2 : Z_2)$, $E\colon Y^2Z = X^3 - 3XZ^2 + bZ^3$

Ensure: $(X_3 : Y_3 : Z_3) = P + Q$;

1. $t_0 \leftarrow X_1 \cdot X_2$	2. $t_1 \leftarrow Y_1 \cdot Y_2$	3. $t_2 \leftarrow Z_1 \cdot Z_2$
4. $t_3 \leftarrow X_1 + Y_1$	5. $t_4 \leftarrow X_2 + Y_2$	6. $t_3 \leftarrow t_3 \cdot t_4$
7. $t_4 \leftarrow t_0 + t_1$	8. $t_3 \leftarrow t_3 - t_4$	9. $t_4 \leftarrow Y_1 + Z_1$
10. $X_3 \leftarrow Y_2 + Z_2$	11. $t_4 \leftarrow t_4 \cdot X_3$	12. $X_3 \leftarrow t_1 + t_2$
13. $t_4 \leftarrow t_4 - X_3$	14. $X_3 \leftarrow X_1 + Z_1$	15. $Y_3 \leftarrow X_2 + Z_2$
16. $X_3 \leftarrow X_3 \cdot Y_3$	17. $Y_3 \leftarrow t_0 + t_2$	18. $Y_3 \leftarrow X_3 - Y_3$
19. $Z_3 \leftarrow b \cdot t_2$	20. $X_3 \leftarrow Y_3 - Z_3$	21. $Z_3 \leftarrow X_3 + X_3$
22. $X_3 \leftarrow X_3 + Z_3$	23. $Z_3 \leftarrow t_1 - X_3$	24. $X_3 \leftarrow t_1 + X_3$
25. $Y_3 \leftarrow b \cdot Y_3$	26. $t_1 \leftarrow t_2 + t_2$	27. $t_2 \leftarrow t_1 + t_2$
28. $Y_3 \leftarrow Y_3 - t_2$	29. $Y_3 \leftarrow Y_3 - t_0$	30. $t_1 \leftarrow Y_3 + Y_3$
31. $Y_3 \leftarrow t_1 + Y_3$	32. $t_1 \leftarrow t_0 + t_0$	33. $t_0 \leftarrow t_1 + t_0$
34. $t_0 \leftarrow t_0 - t_2$	35. $t_1 \leftarrow t_4 \cdot Y_3$	36. $t_2 \leftarrow t_0 \cdot Y_3$
37. $Y_3 \leftarrow X_3 \cdot Z_3$	38. $Y_3 \leftarrow Y_3 + t_2$	39. $X_3 \leftarrow t_3 \cdot X_3$
40. $X_3 \leftarrow X_3 - t_1$	41. $Z_3 \leftarrow t_4 \cdot Z_3$	42. $t_1 \leftarrow t_3 \cdot t_0$
43. $Z_3 \leftarrow Z_3 + t_1$		

Algorithm 5: Complete, mixed point addition for prime order short Weierstrass curves $E/\mathbb{F}_q\colon y^2 = x^3 + ax + b$ with $a = -3$.

Require: $P = (X_1 : Y_1 : Z_1)$, $Q = (X_2 : Y_2 : 1)$, $E\colon Y^2 Z = X^3 - 3XZ^2 + bZ^3$

Ensure: $(X_3 : Y_3 : Z_3) = P + Q$;

1. $t_0 \leftarrow X_1 \cdot X_2$	2. $t_1 \leftarrow Y_1 \cdot Y_2$	3. $t_3 \leftarrow X_2 + Y_2$
4. $t_4 \leftarrow X_1 + Y_1$	5. $t_3 \leftarrow t_3 \cdot t_4$	6. $t_4 \leftarrow t_0 + t_1$
7. $t_3 \leftarrow t_3 - t_4$	8. $t_4 \leftarrow Y_2 \cdot Z_1$	9. $t_4 \leftarrow t_4 + Y_1$
10. $Y_3 \leftarrow X_2 \cdot Z_1$	11. $Y_3 \leftarrow Y_3 + X_1$	12. $Z_3 \leftarrow b \cdot Z_1$
13. $X_3 \leftarrow Y_3 - Z_3$	14. $Z_3 \leftarrow X_3 + X_3$	15. $X_3 \leftarrow X_3 + Z_3$
16. $Z_3 \leftarrow t_1 - X_3$	17. $X_3 \leftarrow t_1 + X_3$	18. $Y_3 \leftarrow b \cdot Y_3$
19. $t_1 \leftarrow Z_1 + Z_1$	20. $t_2 \leftarrow t_1 + Z_1$	21. $Y_3 \leftarrow Y_3 - t_2$
22. $Y_3 \leftarrow Y_3 - t_0$	23. $t_1 \leftarrow Y_3 + Y_3$	24. $Y_3 \leftarrow t_1 + Y_3$
25. $t_1 \leftarrow t_0 + t_0$	26. $t_0 \leftarrow t_1 + t_0$	27. $t_0 \leftarrow t_0 - t_2$
28. $t_1 \leftarrow t_4 \cdot Y_3$	29. $t_2 \leftarrow t_0 \cdot Y_3$	30. $Y_3 \leftarrow X_3 \cdot Z_3$
31. $Y_3 \leftarrow Y_3 + t_2$	32. $X_3 \leftarrow t_3 \cdot X_3$	33. $X_3 \leftarrow X_3 - t_1$
34. $Z_3 \leftarrow t_4 \cdot Z_3$	35. $t_1 \leftarrow t_3 \cdot t_0$	36. $Z_3 \leftarrow Z_3 + t_1$

In this case, the doubling formulas become

$$
\begin{aligned}
X_3 &= 2XY(Y^2 + 3(2XZ - bZ^2)) \\
&\quad - 6YZ(2bXZ - X^2 - 3Z^2), \\
Y_3 &= (Y^2 - 3(2XZ - bZ^2))(Y^2 + 3(2XZ - bZ^2)) \\
&\quad + 3(3X^2 - 3Z^2)(2bXZ - X^2 - 3Z^2), \\
Z_3 &= 8Y^3 Z,
\end{aligned}
$$

which can be computed at a cost of $8\mathbf{M} + 3\mathbf{S} + 2\mathbf{m_b} + 21\mathbf{a}$ using Algorithm 6.

Algorithm 6: Exception-free point doubling for prime order short Weierstrass curves $E/\mathbb{F}_q\colon y^2 = x^3 + ax + b$ with $a = -3$.

Require: $P = (X : Y : Z)$ on $E\colon Y^2 Z = X^3 - 3XZ^2 + bZ^3$.

Ensure: $(X_3 : Y_3 : Z_3) = 2P$.

1. $t_0 \leftarrow X \cdot X$	2. $t_1 \leftarrow Y \cdot Y$	3. $t_2 \leftarrow Z \cdot Z$
4. $t_3 \leftarrow X \cdot Y$	5. $t_3 \leftarrow t_3 + t_3$	6. $Z_3 \leftarrow X \cdot Z$
7. $Z_3 \leftarrow Z_3 + Z_3$	8. $Y_3 \leftarrow b \cdot t_2$	9. $Y_3 \leftarrow Y_3 - Z_3$
10. $X_3 \leftarrow Y_3 + Y_3$	11. $Y_3 \leftarrow X_3 + Y_3$	12. $X_3 \leftarrow t_1 - Y_3$
13. $Y_3 \leftarrow t_1 + Y_3$	14. $Y_3 \leftarrow X_3 \cdot Y_3$	15. $X_3 \leftarrow X_3 \cdot t_3$
16. $t_3 \leftarrow t_2 + t_2$	17. $t_2 \leftarrow t_2 + t_3$	18. $Z_3 \leftarrow b \cdot Z_3$
19. $Z_3 \leftarrow Z_3 - t_2$	20. $Z_3 \leftarrow Z_3 - t_0$	21. $t_3 \leftarrow Z_3 + Z_3$
22. $Z_3 \leftarrow Z_3 + t_3$	23. $t_3 \leftarrow t_0 + t_0$	24. $t_0 \leftarrow t_3 + t_0$
25. $t_0 \leftarrow t_0 - t_2$	26. $t_0 \leftarrow t_0 \cdot Z_3$	27. $Y_3 \leftarrow Y_3 + t_0$
28. $t_0 \leftarrow Y \cdot Z$	29. $t_0 \leftarrow t_0 + t_0$	30. $Z_3 \leftarrow t_0 \cdot Z_3$
31. $X_3 \leftarrow X_3 - Z_3$	32. $Z_3 \leftarrow t_0 \cdot t_1$	33. $Z_3 \leftarrow Z_3 + Z_3$
34. $Z_3 \leftarrow Z_3 + Z_3$		

$\boldsymbol{a = 0.}$ Short Weierstrass curves with $a = 0$, i.e., with j-invariant 0, have also appeared in the standards. For example, Certicom's SEC-2 standard [22] specifies three such curves; one of these is `secp256k1`, which is the curve used in the Bitcoin protocol. In addition, in the case that pairing-based cryptography becomes standardized, it is most likely that the curve choices will be short Weierstrass curves with $a = 0$, e.g., BN curves [5].

In this case, the complete additions simplify to

$$
\begin{aligned}
X_3 &= (X_1Y_2 + X_2Y_1)(Y_1Y_2 - 3bZ_1Z_2) \\
&\quad - 3b(Y_1Z_2 + Y_2Z_1)(X_1Z_2 + X_2Z_1), \\
Y_3 &= (Y_1Y_2 + 3bZ_1Z_2)(Y_1Y_2 - 3bZ_1Z_2) + 9bX_1X_2(X_1Z_2 + X_2Z_1), \\
Z_3 &= (Y_1Z_2 + Y_2Z_1)(Y_1Y_2 + 3bZ_1Z_2) + 3X_1X_2(X_1Y_2 + X_2Y_1),
\end{aligned}
$$

which can be computed in $12\mathbf{M} + 2\mathbf{m_{3b}} + 19\mathbf{a}$ via Algorithm 7. The mixed addition is computed in $11\mathbf{M} + 2\mathbf{m_{3b}} + 13\mathbf{a}$ via Algorithm 8.

Algorithm 7: Complete, projective point addition for prime order j-invariant 0 short Weierstrass curves $E/\mathbb{F}_q : y^2 = x^3 + b$.

Require: $P = (X_1 : Y_1 : Z_1)$, $Q = (X_2 : Y_2 : Z_2)$ on $E : Y^2Z = X^3 + bZ^3$
        and $b_3 = 3 \cdot b$.
Ensure: $(X_3 : Y_3 : Z_3) = P + Q$;

1. $t_0 \leftarrow X_1 \cdot X_2$	2. $t_1 \leftarrow Y_1 \cdot Y_2$	3. $t_2 \leftarrow Z_1 \cdot Z_2$
4. $t_3 \leftarrow X_1 + Y_1$	5. $t_4 \leftarrow X_2 + Y_2$	6. $t_3 \leftarrow t_3 \cdot t_4$
7. $t_4 \leftarrow t_0 + t_1$	8. $t_3 \leftarrow t_3 - t_4$	9. $t_4 \leftarrow Y_1 + Z_1$
10. $X_3 \leftarrow Y_2 + Z_2$	11. $t_4 \leftarrow t_4 \cdot X_3$	12. $X_3 \leftarrow t_1 + t_2$
13. $t_4 \leftarrow t_4 - X_3$	14. $X_3 \leftarrow X_1 + Z_1$	15. $Y_3 \leftarrow X_2 + Z_2$
16. $X_3 \leftarrow X_3 \cdot Y_3$	17. $Y_3 \leftarrow t_0 + t_2$	18. $Y_3 \leftarrow X_3 - Y_3$
19. $X_3 \leftarrow t_0 + t_0$	20. $t_0 \leftarrow X_3 + t_0$	21. $t_2 \leftarrow b_3 \cdot t_2$
22. $Z_3 \leftarrow t_1 + t_2$	23. $t_1 \leftarrow t_1 - t_2$	24. $Y_3 \leftarrow b_3 \cdot Y_3$
25. $X_3 \leftarrow t_4 \cdot Y_3$	26. $t_2 \leftarrow t_3 \cdot t_1$	27. $X_3 \leftarrow t_2 - X_3$
28. $Y_3 \leftarrow Y_3 \cdot t_0$	29. $t_1 \leftarrow t_1 \cdot Z_3$	30. $Y_3 \leftarrow t_1 + Y_3$
31. $t_0 \leftarrow t_0 \cdot t_3$	32. $Z_3 \leftarrow Z_3 \cdot t_4$	33. $Z_3 \leftarrow Z_3 + t_0$

The doubling formulas in this case are

$$
\begin{aligned}
X_3 &= 2XY(Y^2 - 9bZ^2), \\
Y_3 &= (Y^2 - 9bZ^2)(Y^2 + 3bZ^2) + 24bY^2Z^2, \\
Z_3 &= 8Y^3Z.
\end{aligned}
$$

These can be computed in $6\mathbf{M} + 2\mathbf{S} + 1\mathbf{m_{3b}} + 9\mathbf{a}$ via Algorithm 9.

4   Some intuition towards optimality

In this section we motivate the choice of the complete formulas in (1) that were taken from Bosma and Lenstra [19], by providing reasoning as to why, among the many possible complete addition laws on prime order curves, we chose the set corresponding to the line $Y = 0$ in $\mathbb{P}^2(k)$ under the straightforward homogeneous projection.

Algorithm 8: Complete, mixed point addition for prime order j-invariant 0 short Weierstrass curves $E/\mathbb{F}_q\colon y^2 = x^3 + b$.

Require: $P = (X_1 : Y_1 : Z_1)$, $Q = (X_2 : Y_2 : 1)$ on $E\colon Y^2 Z = X^3 + bZ^3$
  and $b_3 = 3 \cdot b$.

Ensure: $(X_3 : Y_3 : Z_3) = P + Q$;

1. $t_0 \leftarrow X_1 \cdot X_2$	2. $t_1 \leftarrow Y_1 \cdot Y_2$	3. $t_3 \leftarrow X_2 + Y_2$
4. $t_4 \leftarrow X_1 + Y_1$	5. $t_3 \leftarrow t_3 \cdot t_4$	6. $t_4 \leftarrow t_0 + t_1$
7. $t_3 \leftarrow t_3 - t_4$	8. $t_4 \leftarrow Y_2 \cdot Z_1$	9. $t_4 \leftarrow t_4 + Y_1$
10. $Y_3 \leftarrow X_2 \cdot Z_1$	11. $Y_3 \leftarrow Y_3 + X_1$	12. $X_3 \leftarrow t_0 + t_0$
13. $t_0 \leftarrow X_3 + t_0$	14. $t_2 \leftarrow b_3 \cdot Z_1$	15. $Z_3 \leftarrow t_1 + t_2$
16. $t_1 \leftarrow t_1 - t_2$	17. $Y_3 \leftarrow b_3 \cdot Y_3$	18. $X_3 \leftarrow t_4 \cdot Y_3$
19. $t_2 \leftarrow t_3 \cdot t_1$	20. $X_3 \leftarrow t_2 - X_3$	21. $Y_3 \leftarrow Y_3 \cdot t_0$
22. $t_1 \leftarrow t_1 \cdot Z_3$	23. $Y_3 \leftarrow t_1 + Y_3$	24. $t_0 \leftarrow t_0 \cdot t_3$
25. $Z_3 \leftarrow Z_3 \cdot t_4$	26. $Z_3 \leftarrow Z_3 + t_0$	

Algorithm 9: Exception-free point doubling for prime order j-invariant 0 short Weierstrass curves $E/\mathbb{F}_q\colon y^2 = x^3 + b$.

Require: $P = (X : Y : Z)$ on $E\colon Y^2 Z = X^3 + bZ^3$ and $b_3 = 3 \cdot b$.

Ensure: $(X_3 : Y_3 : Z_3) = 2P$.

1. $t_0 \leftarrow Y \cdot Y$	2. $Z_3 \leftarrow t_0 + t_0$	3. $Z_3 \leftarrow Z_3 + Z_3$
4. $Z_3 \leftarrow Z_3 + Z_3$	5. $t_1 \leftarrow Y \cdot Z$	6. $t_2 \leftarrow Z \cdot Z$
7. $t_2 \leftarrow b_3 \cdot t_2$	8. $X_3 \leftarrow t_2 \cdot Z_3$	9. $Y_3 \leftarrow t_0 + t_2$
10. $Z_3 \leftarrow t_1 \cdot Z_3$	11. $t_1 \leftarrow t_2 + t_2$	12. $t_2 \leftarrow t_1 + t_2$
13. $t_0 \leftarrow t_0 - t_2$	14. $Y_3 \leftarrow t_0 \cdot Y_3$	15. $Y_3 \leftarrow X_3 + Y_3$
16. $t_1 \leftarrow X \cdot Y$	17. $X_3 \leftarrow t_0 \cdot t_1$	18. $X_3 \leftarrow X_3 + X_3$

We do not claim that this choice is truly optimal, since proving that a certain choice of projective embedding and/or complete addition law for any particular prime order curve is faster than *all* of the other choices for that curve seems extremely difficult, if not impossible. We merely explain why, when aiming to write down explicit algorithms that will simultaneously be complete on all prime order short Weierstrass curves, choosing the Bosma–Lenstra formulas makes sense.

Furthermore, we also do not claim that our explicit algorithms to compute the addition law in (1) are computationally optimal. It is likely that trade-offs can be advantageously exploited on some platforms (cf. [41, §3.6]) or that alternative operation scheduling could reduce the number of field additions[12].

4.1   Choice of the line $Y = 0$ for bidegree $(2,2)$ addition laws

Let $L_{(\alpha,\beta,\gamma)}$ denote the line given by $\alpha X + \beta Y + \gamma Z = 0$ inside $\mathbb{P}^2(\mathbb{F}_q)$, and, under the necessary assumption that $L_{(\alpha,\beta,\gamma)}$ does not intersect the curve $E\colon Y^2 Z = X^3 + aXZ^2 + bZ^3 \subset \mathbb{P}^2(\mathbb{F}_q)$, let $A_{(\alpha,\beta,\gamma)}$ denote the complete addition law of bidegree $(2,2)$ corresponding to $L_{(\alpha,\beta,\gamma)}$ given by [19, Theorem 2]. So far we have given optimizations for $A_{(0,1,0)}$, but the question remains as to whether there are other lines $L_{(\alpha,\beta,\gamma)}$ which give rise to even faster addition laws $A_{(\alpha,\beta,\gamma)}$.

---

[12] Our experimentation did suggest that computing (1) in any reasonable way with fewer than 12 generic multiplications appears to be difficult.

We first point out that $L_{(0,1,0)}$ is the only line that does not intersect E independently of a, b and q. It is easy to show that any other line in $\mathbb{P}^2(\mathbb{F}_q)$ that does not intersect E will have a dependency on at least one of a, b and q, and the resulting addition law will therefore only be complete on a subset of prime order curves.

Nevertheless, it is possible that there is a better choice than $A_{(0,1,0)}$ for a given short Weierstrass curve, or that there are special choices of prime order curves that give rise to more efficient complete group laws. We now sketch some intuition as to why this is unlikely. For $A_{(\alpha,\beta,\gamma)}$ to be complete, it is necessary that, in particular, $L_{(\alpha,\beta,\gamma)}$ does not intersect E at the point at infinity $(0:1:0)$. This implies that $\beta \neq 0$. From [48, 19], we know that the space of all addition laws has dimension 3 and that

$$A_{(\alpha,\beta,\gamma)} = \alpha A_{(1,0,0)} + \beta A_{(0,1,0)} + \gamma A_{(0,0,1)},$$

where $A_{(1,0,0)}$, $A_{(0,1,0)}$ and $A_{(0,0,1)}$ are the three addition laws given in [19, pp. 236-239], specialized to short Weierstrass curves. Given that $\beta \neq 0$, our only hope of finding a more simple addition law than $A_{(0,1,0)}$ is by choosing α and/or γ in a way that causes an advantageous cross-cancellation of terms. Close inspection of the formulas in [19] strongly suggests that no such cancellation exists.

Remark 1. Interestingly, both $A_{(1,0,0)}$ and $A_{(0,0,1)}$ vanish to zero when specialized to doubling. This means that any doubling formula in bidegree $(2,2)$ that is not exceptional at the point at infinity is a scalar multiple of $A_{(0,1,0)}$, i.e., the formulas used in this paper.

Remark 2. Although a more efficient addition law might exist for larger bidegrees, it is worth reporting that our experiments to find higher bidegree analogues of the Bosma and Lenstra formulas suggest that this, too, is unlikely. The complexity (and computational cost) of the explicit formulas grows rapidly as the bidegree increases, which is most commonly the case across all models of elliptic curves and projective embeddings (cf. [41]). We could hope for an addition law of bidegree lower than $(2,2)$, but in [19, §3] Bosma and Lenstra prove that this is not possible under the short Weierstrass embedding[13] of E.

4.2   Jacobian coordinates

Since first suggested for short Weierstrass curves by Miller in his seminal paper [52, p. 424], Jacobian coordinates have proven to offer significant performance advantages over other coordinate systems. Given their ubiquity in real-world ECC code, and the fact that their most commonly used sets of efficient point doubling formulas turn out to be exception-free on prime order curves (see Table 1), it is highly desirable to go searching for a Jacobian coordinate analogue of the Bosma–Lenstra (homogeneous coordinates) addition law. Unfortunately, we now show that such addition formulas in Jacobian coordinates must have a higher bidegree, intuitively making them slower to compute.

For the remainder of this section only, let $E \subset \mathbb{P}(2,3,1)(k)$ have odd order. If an addition law $f = (f_X, f_Y, f_Z)$ has f_Z of bidegree (μ,ν), then the bidegrees of f_X and f_Y are $(2\mu,2\nu)$ and $(3\mu,3\nu)$, respectively. Below we show that any complete formulas must have $\mu, \nu \geq 3$.

Consider the addition of two points $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$, using the addition law

$$f(P,Q) = (f_X(P,Q) : f_Y(P,Q) : f_Z(P,Q)),$$

---

[13] Lower bidegree addition laws are possible for other embeddings (i.e., models) of E in the case where E has a k-rational torsion structure – see [47].

with f_Z of bidegree (μ, ν). Suppose that f is complete, and that $\mu < 3$. Then f_Z, viewed as a polynomial in X_1, Y_1, Z_1, has degree $\mu < 3$, and in particular cannot contain Y_1. Now, since $-P = (X_1 : -Y_1 : Z_1)$ on E, it follows that $f_Z(P, Q) = f_Z(-P, Q)$ for all possible Q, and in particular when $Q = P$. But in this case, and given that P cannot have order 2, we have $f_Z(P, Q) \neq 0$ and $f_Z(-P, Q) = 0$, a contradiction. We conclude that $\mu \geq 3$, and (by symmetry) that $\nu \geq 3$. It follows that f_X and f_Y have bidegrees at least $(6, 6)$ and $(9, 9)$, respectively, which destroys any hope of comparable efficiency to the homogeneous Bosma–Lenstra formulas.

5 Using these formulas in practice

In this section we discuss the practical application of the complete algorithms in this paper. We discuss how they can be used for both the prime order curves (§5.1) and composite order curves (§5.2) in the standards. In §5.3, we give performance numbers that shed light on the expected cost of completeness in certain software scenarios, before discussing why this cost is likely to be significantly reduced in many other scenarios, e.g., in hardware.

5.1 Application to prime order curves (or, secure ECC for noobs)

Using Algorithm 1 as a black-box point addition routine, non-experts now have a straightforward way to implement the standardized prime order elliptic curves. So long as scalars are *recoded* correctly, the subsequent scalar multiplication routine will always compute the correct result.

Given the vulnerabilities exposed in already-deployed ECC implementations (see §1), we now provide some implementation recommendations, e.g., for an implementer whose task it is to (re)write a simple and timing-resistant scalar multiplication routine for prime order curves from scratch. The main point is that branches (e.g., `if` statements) inside the elliptic curve point addition algorithms can now be avoided entirely. Our main recommendation is that more streamlined versions of Algorithm 1 should only be introduced to an implementation if they are guaranteed to be exception-free; subsequently, we stress that branching should never be introduced into any point addition algorithms.

Assuming access to branch-free, constant-time field arithmetic in $\mathbb{F}_q$, a first step is to implement Algorithm 1 to be used for *all* point (doubling and addition) operations, working entirely in homogeneous projective space. The natural next step is to implement a basic scalar recoding (e.g., [57, 44]) that gives rise to a fixed, uniform, scalar-independent main loop. This typically means that the main loop repeats the same pattern of a fixed number of doublings followed by a single table lookup/extraction and, subsequently, an addition. The important points are that this table lookup must be done in a cache-timing resistant manner (cf. [45, §3.4]), and that the basic scalar recoding must itself be performed in a uniform manner.

Once the above routine is running correctly, an implementer that is seeking further performance gains can start by viewing stages of the routine where Algorithm 1 can safely be replaced by its specialized, more efficient variants. If the code is intended to support only short Weierstrass curves with either $a = -3$ or $a = 0$, then Algorithm 1 should be replaced by (the faster and more compact) Algorithm 4 or Algorithm 7, respectively. If the performance gains warrant the additional code, then at all stages where the addition function is called to add a point to itself (i.e., the point doubling stages), the respective exception-free point doubling routine(s) in Algorithms 3, 6 and 9 should be implemented and called there instead.

Incomplete short Weierstrass addition routines (e.g., the prior works summarized in Table 1) should only be introduced for further performance gains if the implementer can guarantee that exceptional pairs of points can never be input into the algorithms, and subsequently can implement them without introducing any branches. For example, Bos *et al.* [16, §4.1] proved that, under their particular choice of scalar multiplication algorithm, all-but-one of the point additions in a variable-base scalar multiplication can be performed without exception using an incomplete addition algorithm. The high-level argument used there was that such additions almost always took place between elements of the lookup table and a running value that had just been output from a point doubling, the former being small odd multiples of the input point (e.g., P, $[3]P$, $[5]P$, etc.) and the latter being some even multiple. Subsequently, they showed that the only possible time when the input points to the addition algorithm could coincide with (or be inverses of) each other is in the final addition, ruling out the exceptional points in all prior additions. On the other hand, as we mentioned in §1 and as was encountered in [16, §4.1], it can be significantly more complicated to rule out exceptional input points in more exotic scalar multiplication scenarios like fixed-base scalar multiplications, multiscalar multiplications, or those that exploit endomorphisms. In those cases, it could be that the only option to rule out any exceptional points is to *always* call complete addition algorithms.

Remark 3 (The best of both worlds?). We conclude this subsection by mentioning one more option that may be of interest to implementers who want to combine the fastest complete point addition algorithms with the fastest exception-free point doubling algorithms. Recall from Table 1 that the fastest doubling algorithms for short Weierstrass curves work in Jacobian coordinates and happen to be exception-free in the prime order setting, but recall from §4.2 that there is little hope of obtaining relatively efficient complete addition algorithms in Jacobian coordinates. This prompts the question as to whether the doubling algorithms that take place in $\mathbb{P}(2,3,1)(k)$ can be combined with our complete addition algorithms that take place in $\mathbb{P}^2(k)$. Generically, we can map the elliptic curve point $(X : Y : Z) \in \mathbb{P}(2,3,1)(k)$ to $(XZ : Y : Z^3) \in \mathbb{P}^2(k)$, and conversely, we can map the point $(X : Y : Z) \in \mathbb{P}^2(k)$ to $(XZ : YZ^2 : Z) \in \mathbb{P}(2,3,1)(k)$; both maps cost $2\mathbf{M} + 1\mathbf{S}$. We note that in the first direction there are no exceptions: in particular, the point at infinity $(1 : 1 : 0) \in \mathbb{P}(2,3,1)(k)$ correctly maps to $(0 : 1 : 0) \in \mathbb{P}^2(k)$. However, in the other direction, the point at infinity $(0 : 1 : 0) \in \mathbb{P}^2(k)$ does not correctly map to $(1 : 1 : 0) \in \mathbb{P}(2,3,1)(k)$, but rather to the point $(0 : 0 : 0) \notin \mathbb{P}(2,3,1)(k)$.

For a variable-base scalar multiplication using a fixed window of width w, one option would be to store the precomputed lookup table in $\mathbb{P}^2(k)$ (or in $\mathbb{A}^2(k)$ if normalizing for the sake of complete mixed additions is preferred), and to compute the main loop as follows. After computing each of the w consecutive doublings in $\mathbb{P}(2,3,1)(k)$, the running value is converted to $\mathbb{P}^2(k)$ at a cost of $2\mathbf{M} + 1\mathbf{S}$, then the result of a complete addition (between the running value and a lookup table element) is converted back to $\mathbb{P}(2,3,1)(k)$ at a cost of $2\mathbf{M} + 1\mathbf{S}$. Even for small window sizes that result in additions (and thus the back-and-forth conversions) occurring relatively often, the operation counts in Table 1 suggest that this trade-off will be favorable; and, for larger window sizes, the resulting scalar multiplication will be significantly faster than one that works entirely in $\mathbb{P}^2(k)$.

The only possible exception that could occur in the above routine is when the result of an addition is the point at infinity $(0 : 1 : 0) \in \mathbb{P}^2(k)$, since the conversion back to $\mathbb{P}(2,3,1)(k)$ fails here. Thus, this strategy should only be used if the scalar multiplication routine is

such that the running value is never the inverse of any element in the lookup table, or if the conversion from $\mathbb{P}^2(k)$ to $\mathbb{P}(2,3,1)(k)$ is written to handle this possible exception in a constant-time fashion. In the former case, if (as in [16, §4.1]) this can only happen in the final addition, then the workaround is easy: either guarantee that the scalars cannot be a multiple of the group order (which rules out this possibility), or else do not apply the conversion back to $\mathbb{P}(2,3,1)(k)$ after the final addition.

5.2   Interoperability with composite order curves

The IRTF CFRG recently selected two composite order curves as a recommendation to the TLS working group for inclusion in upcoming versions of TLS: Bernstein's Curve25519 [7] and Hamburg's Goldilocks [39]. The current IETF internet draft[14] specifies the wire format for these curves to be the u-coordinate corresponding to a point (u, v) on the Montgomery model of these curves $E_M/\mathbb{F}_q \colon v^2 = u^3 + Au^2 + u$. Curve25519 has $q = 2^{255} - 19$ with $A = 486662$ and Goldilocks has $q = 2^{448} - 2^{224} - 1$ with $A = 156326$.

Since our complete formulas are likely to be of interest to practitioners concerned with global interoperability, e.g., those investing a significant budget into one implementation that may be intended to support as many standardized curves as possible, we now show that Algorithm 1 can be adapted to interoperate with the composite order curves in upcoming TLS ciphersuites. We make no attempt to disguise the fact that this will come with a significant performance penalty over the Montgomery ladder, but in this case we are assuming that top performance is not the priority.

A trivial map from the Montgomery curve to a short Weierstrass curve is $\kappa \colon E_M \to E$, $(u, v) \mapsto (x, y) = (u - A/3, v)$; here the short Weierstrass curve is $E \colon y^2 = x^3 + ax + b$, with $a = 1 - A^2/3$ and $b = A(2A^2 - 9)/27$.

Thus, a dedicated short Weierstrass implementation can interoperate with Curve25519 (resp. Goldilocks) as follows. After receiving the u-coordinate on the wire, set $x = u - A/3$ (i.e., add a fixed, global constant), and decompress to compute the corresponding y-coordinate on E via the square root $y = \sqrt{x^3 + ax + b}$ as usual; the choice of square root here does not matter. Setting $P = (x, y)$ and validating that $P \in E$, we can then call Algorithm 1 to compute 3 (resp. 2) successive doublings to get Q. This is in accordance with the scalars being defined with 3 (resp. 2) fixed zero bits to clear the cofactor [7]. The point Q is then multiplied by the secret part of the scalar (using, e.g., the methods we just described in §5.1), then normalized to give $Q = (x', y')$, and the Montgomery u-coordinate of the result is output as $u' = x' + A/3$.

Note that the above routine is exception free: Algorithm 1 only fails to add the points P_1 and P_2 when $P_1 - P_2$ is a point of exact order 2. Thus, it can be used for point doublings on all short Weierstrass curves (including those of even order). Furthermore, the point Q is in the prime order subgroup, so the subsequent scalar multiplication (which only encounters multiples of Q) cannot find a pair of points that are exceptional to Algorithm 1.

Finally, we note that although neither Curve25519 or Goldilocks are isomorphic to a Weierstrass curve with $a = -3$, both curves have simple isomorphisms to Weierstrass curves with small a values, e.g., $a = 2$ and $a = 1$, respectively. Making use of this would noticeably decrease the overhead of our complete formulas.

---

[14] See `https://datatracker.ietf.org/doc/draft-irtf-cfrg-curves/`.

5.3   The cost of completeness

In Table 2 we report the factor slowdown obtained when substituting the complete formulas in Algorithms 4–6 for OpenSSL's "ec_GFp_simple_add" and "ec_GFp_simple_dbl" functions inside the OpenSSL scalar multiplication routine for the five NIST prime curves (which all have $a = -3$).

NIST	no. of ECDH operations (per 10s)		factor
curve	complete	incomplete	slowdown
P-192	35274	47431	1.34x
P-224	24810	34313	1.38x
P-256	21853	30158	1.38x
P-384	10109	14252	1.41x
P-521	4580	6634	1.44x

Table 2. Number of ECDH operations in 10 seconds for the OpenSSL implementation of the five NIST prime curves, using complete and incomplete addition formulas. Timings were obtained by running the "openssl speed ecdhpXXX" command on an Intel Core i5-5300 CPU @ 2.30GHz, averaged over 100 trials of 10s each.

We intentionally left OpenSSL's scalar multiplication routines unaltered in order to provide an unbiased upper bound on the performance penalty that our complete algorithms will introduce. For the remainder of this subsection, we discuss why the performance difference is unlikely to be this large in many practical scenarios.

Referring to Table 3 (which, as well as the counts given in Table 1, includes the operation counts for mixed additions), we see that the mixed addition formulas in Jacobian coordinates are $4\mathbf{M} + 1\mathbf{S}$ faster than full additions, while for our complete formulas the difference is only $1\mathbf{M} + 6\mathbf{a}$. Thus, in Jacobian coordinates, it is often advantageous to normalize the lookup table (using one shared inversion [54]) in order to save $4\mathbf{M} + 1\mathbf{S}$ per addition. On the other hand, in the case of the complete formulas, this will not be a favorable trade-off and (assuming there is ample cache space) it is likely to be better to leave all of the lookup elements in $\mathbb{P}^2$. The numbers reported in Table 2 use OpenSSL's scalar multiplication which does normalize the lookup table to use mixed additions, putting the complete formulas at a disadvantage.

As we mentioned in §1, the slowdowns reported in Table 2 (which were obtained on a 64-bit machine) are likely to be significantly less on low-end architectures where the relative cost of field additions drops. Furthermore, in embedded scenarios where implementations must be protected against more than just timing attacks, a common countermeasure is to randomize the projective coordinates of intermediate points [29]. In these cases, normalized lookup table elements could also give rise to side-channel vulnerabilities [33, §3.4–3.6], which would take mixed additions out of the equation. As Table 3 suggests, when full additions are used throughout, our complete algorithms will give much better performance relative to their incomplete counterparts.

Hardware implementations of ECC typically rely on using general field hardware multipliers that are often based on the algorithm of Montgomery [53]. These types of hardware modules use a multiplier for both multiplications and squarings [24, 36], meaning that the squarings our addition algorithms save (over the prior formulas) are full multiplications. Moreover, hardware architectures that are based on Montgomery multiplication can benefit from modular additions/subtractions computed as non-modular operations. The concept is ex-

addition formulas	a	ADD(P,Q)					mADD(P,Q)					DBL(P)				
		M	**S**	m_a	m_b	**a**	**M**	**S**	m_a	m_b	**a**	**M**	**S**	m_a	m_b	**a**
complete homogeneous	any	12	0	3	2	23	11	0	3	2	17	8	3	3	2	15
	-3	12	0	0	2	29	11	0	0	2	23	8	3	0	2	21
	0	12	0	0	2	19	11	0	0	2	13	6	2	0	1	9
incomplete homogeneous	any	12	2	0	0	7	9	2	0	0	7	5	6	1	0	12
	-3	12	2	0	0	7	9	2	0	0	7	7	3	0	0	11
	0			-					-					-		
incomplete Jacobian	any	12	4	0	0	7	8	3	0	0	7	3	6	1	0	13
	-3	12	4	0	0	7	8	3	0	0	7	4	4	0	0	8
	0	12	4	0	0	7	8	3	0	0	7	3	4	0	0	7

Table 3. Operation counts for the prior incomplete addition algorithms and our complete ones, with the inclusion of mixed addition formulas. Credits for the incomplete formulas are the same as in Table 1, except for the additional mixed formulas which are, in homogeneous coordinates, due to Cohen, Miyaji and Ono [27], and in Jacobian coordinates, due to Hankerson, Menezes and Vanstone [40, p. 91].

plained in [6], which is a typical ECC hardware architecture using the "relaxed" Montgomery parameter such that the conditional subtraction (from the original algorithm of Montgomery) can be omitted. In this way, the modular addition/subtraction is implemented not just very efficiently, but also as a time-constant operation. Using this approach implies the only cost to be taken into account is the one of modular multiplication, i.e., modular additions come almost for free. Similar conclusions apply for multiplications with a constant as they can be implemented very efficiently in hardware, assuming a constant is predefined and hence "hardwired". Again, viewing the operation counts in Table 3 suggests that such scenarios are, relatively speaking, likely to give a greater benefit to our complete algorithms.

Finally, we remark that runtime is not the only metric of concern to ECC practitioners; in fact, there was wide consensus (among both speakers and panelists) at the recent NIST workshop[15] that security and simplicity are far more important in real-world ECC than raw performance. While our complete algorithms are likely to be slower in some scenarios, we reiterate that complete formulas reign supreme in all other aspects, including total code size, ease of implementation, and issues relating to side-channel resistance.

References

1. Accredited Standards Committee X9. American National Standard X9.62-1999, Public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA). Draft at `http://grouper.ieee.org/groups/1363/Research/Other.html`, 1999.
2. Accredited Standards Committee X9. American National Standard X9.63-2001, Public key cryptography for the financial services industry: key agreement and key transport using elliptic curve cryptography. Draft at `http://grouper.ieee.org/groups/1363/Research/Other.html`, 1999.

---

[15] See `http://www.nist.gov/itl/csd/ct/ecc-workshop.cfm`.

3. Agence nationale de la sécurité des sysèmes d'information (ANSSI). Mécanismes cryptographiques: Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques. `http://www.ssi.gouv.fr/uploads/2015/01/RGS_v-2-0_B1.pdf`, 2014.

4. C. Arène, D. Kohel, and C. Ritzenthaler. Complete addition laws on abelian varieties. *LMS Journal of Computation and Mathematics*, 15:308–316, 2012.

5. P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. E. Tavares, editors, *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2005.

6. L. Batina, G. Bruin-Muurling, and S. B. Örs. Flexible hardware design for RSA and elliptic curve cryptosystems. In *Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings*, pages 250–263, 2004.

7. 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, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.

8. D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards curves. In S. Vaudenay, editor, *Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings*, volume 5023 of *Lecture Notes in Computer Science*, pages 389–405. Springer, 2008.

9. D. J. Bernstein, C. Chuengsatiansup, D. Kohel, and T. Lange. Twisted Hessian curves. In K. E. Lauter and F. Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, volume 9230 of *Lecture Notes in Computer Science*, pages 269–294. Springer, 2015.

10. D. J. Bernstein, C. Chuengsatiansup, and T. Lange. Curve41417: Karatsuba revisited. In L. Batina and M. Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 316–334. Springer, 2014.

11. D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In A. Sadeghi, V. D. Gligor, and M. Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 967–980. ACM, 2013.

12. D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. In K. Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 29–50. Springer, 2007.

13. D. J. Bernstein and T. Lange. Complete addition laws for elliptic curves. Talk at Algebra and Number Theory Seminar (Universidad Autonomo de Madrid). Slides at `http://cr.yp.to/talks/2009.04.17/slides.pdf`, 2009.

14. D. J. Bernstein and T. Lange. Safecurves: choosing safe curves for elliptic-curve cryptography. URL: `http://safecurves.cr.yp.to/`, Accessed 5 October 2015.

15. D. J. Bernstein and T. Lange. Explicit-Formulas Database. `http://hyperelliptic.org/EFD/index.html`, Date accessed: October 3, 2015.

16. J. W. Bos, C. Costello, P. Longa, and M. Naehrig. Selecting elliptic curves for cryptography: An efficiency and security analysis. *J. Cryptographic Engineering*, 2015. `http://dx.doi.org/10.1007/s13389-015-0097-y`.

17. J. W. Bos, J. A. Halderman, N. Heninger, J. Moore, M. Naehrig, and E. Wustrow. Elliptic curve cryptography in practice. In Christin and Safavi-Naini [26], pages 157–175.

18. W. Bosma, J. J. Cannon, and C. Playoust. The Magma algebra system I: the user language. *J. Symb. Comput.*, 24(3/4):235–265, 1997.

19. W. Bosma and H. W. Lenstra. Complete systems of two addition laws for elliptic curves. *Journal of Number theory*, 53(2):229–240, 1995.

20. E. Brier and M. Joye. Weierstraß elliptic curves and side-channel attacks. In D. Naccache and P. Paillier, editors, *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*, volume 2274 of *Lecture Notes in Computer Science*, pages 335–345. Springer, 2002.

21. E. Brier and M. Joye. Fast point multiplication on elliptic curves through isogenies. In M. P. C. Fossorier, T. Høholdt, and A. Poli, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 15th*

International Symposium, AAECC-15, Toulouse, France, May 12-16, 2003, Proceedings, volume 2643 of *Lecture Notes in Computer Science*, pages 43–50. Springer, 2003.

22. Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0. `http://www.secg.org/sec2-v2.pdf`, 2010.

23. Certivox UK, Ltd.   CertiVox Standard Curves.   `http://docs.certivox.com/docs/miracl/certivox-standard-curves`, Date accessed: September 9, 2015.

24. G. Chen, G. Bai, and H. Chen. A high-performance elliptic curve cryptographic processor for general curves over GF(p) based on a systolic arithmetic unit. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 54(5):412–416, May 2007.

25. Chinese Commerical Cryptography Administration Office.   SM2 Digital Signature Algorithm. See `http://www.oscca.gov.cn/UpFile/2010122214836668.pdf` and `http://tools.ietf.org/html/draft-shen-sm2-ecdsa-02`, 2010.

26. N. Christin and R. Safavi-Naini, editors. *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, volume 8437 of *Lecture Notes in Computer Science*. Springer, 2014.

27. H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In K. Ohta and D. Pei, editors, *Advances in Cryptology - ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Beijing, China, October 18-22, 1998, Proceedings*, volume 1514 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 1998.

28. Committee on National Security Systems (CNSS). Advisory Memorandum: Use of Public Standards for the Secure Sharing of Information Among National Security Systems. `https://www.cnss.gov/CNSS/openDoc.cfm?Q5ww0Xu+7kg/OpTB/R2/MQ==`, 2015.

29. J. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Çetin K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99*, volume 1717 of *LNCS*, pages 292–302. SV, 1999.

30. ECC Brainpool. ECC Brainpool Standard Curves and Curve Generation. `http://www.ecc-brainpool.org/download/Domain-parameters.pdf`, 2005.

31. H. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44(3):393–422, 2007.

32. J. Fan, B. Gierlichs, and F. Vercauteren. To infinity and beyond: Combined attack on ECC using points of low order. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 143–159. Springer, 2011.

33. J. Fan and I. Verbauwhede. An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In D. Naccache, editor, *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, volume 6805 of *Lecture Notes in Computer Science*, pages 265–282. Springer, 2012.

34. R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In J. Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer, 2001.

35. Government Committee of Russia for Standards. Information technology. Cryptographic data security. Signature and verification processes of [electronic] digital signature. See `https://tools.ietf.org/html/rfc5832`, 2001.

36. T. Guneysu and C. Paar. Ultra High Performance ECC over NIST Primes on Commercial FPGAs. In *Cryptographic Hardware and Embedded Systems/93 CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 62–78. Springer Berlin Heidelberg, 2008.

37. M. Hamburg. Twisting Edwards curves with isogenies. Cryptology ePrint Archive, Report 2014/027, 2014. `http://eprint.iacr.org/`.

38. M. Hamburg. Decaf: Eliminating cofactors through point compression. In R. Gennaro and M. Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 705–723. Springer, 2015.

39. M. Hamburg. Ed448-Goldilocks, a new elliptic curve. Cryptology ePrint Archive, Report 2015/625, 2015. `http://eprint.iacr.org/`.

40. D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

41. H. Hisil. *Elliptic curves, group law, and efficient computation*. PhD thesis, Queensland University of Technology, URL: `http://eprints.qut.edu.au/33233/`, 2010.

42. Z. Hu, P. Longa, and M. Xu. Implementing the 4-dimensional GLV method on GLS elliptic curves with j-invariant 0. *Des. Codes Cryptography*, 63(3):331–343, 2012.
43. T. Izu and T. Takagi. Exceptional procedure attack on elliptic curve cryptosystems. In Y. Desmedt, editor, *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, volume 2567 of *Lecture Notes in Computer Science*, pages 224–239. Springer, 2003.
44. M. Joye and M. Tunstall. Exponent recoding and regular exponentiation algorithms. In B. Preneel, editor, *Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings*, volume 5580 of *Lecture Notes in Computer Science*, pages 334–349. Springer, 2009.
45. E. Käsper. Fast elliptic curve cryptography in OpenSSL. In G. Danezis, S. Dietrich, and K. Sako, editors, *Financial Cryptography and Data Security - FC 2011 Workshops, RLCPS and WECSR 2011, Rodney Bay, St. Lucia, February 28 - March 4, 2011, Revised Selected Papers*, volume 7126 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2011.
46. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
47. D. Kohel. Addition law structure of elliptic curves. *Journal of Number Theory*, 131(5):894–919, 2011.
48. H. Lange and W. Ruppert. Complete systems of addition laws on abelian varieties. *Inventiones mathematicae*, 79(3):603–610, 1985.
49. C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In Y. Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 1994.
50. Z. Liu, H. Seo, J. Großschädl, and H. Kim. Efficient implementation of NIST-compliant elliptic curve cryptography for sensor nodes. In S. Qing, J. Zhou, and D. Liu, editors, *Information and Communications Security - 15th International Conference, ICICS 2013, Beijing, China, November 20-22, 2013. Proceedings*, volume 8233 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2013.
51. P. Longa and C. H. Gebotys. Efficient techniques for high-speed elliptic curve cryptography. In S. Mangard and F. Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 80–94. Springer, 2010.
52. V. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology/97CRYPTO/9285 Proceedings*, pages 417–426. Springer, 1986.
53. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.
54. P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.
55. National Institute for Standards and Technology (NIST). Digital Signature Standard. Federal Information Processing Standards Publication 186-2. `http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf`, 2000.
56. National Institute for Standards and Technology (NIST). Digital Signature Standard. Federal Information Processing Standards Publication 186-4. `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`, 2013.
57. K. Okeya and T. Takagi. The width-w NAF method provides small memory and fast elliptic scalar multiplications secure against side channel attacks. In M. Joye, editor, *Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings*, volume 2612 of *Lecture Notes in Computer Science*, pages 328–342. Springer, 2003.
58. J. Renes, C. Costello, and L. Batina. Complete addition formulas for prime order elliptic curves. Cryptology ePrint Archive, Report 2015/1060, 2015. `http://eprint.iacr.org/`.
59. P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: testing the limits of elliptic curve cryptography in sensor networks. In R. Verdone, editor, *Wireless Sensor Networks, 5th European Conference, EWSN 2008, Bologna, Italy, January 30-February 1, 2008, Proceedings*, volume 4913 of *Lecture Notes in Computer Science*, pages 305–320. Springer, 2008.
60. The National Security Agency. Suite B Cryptography (fact sheet). `https://www.nsa.gov/ia/programs/suiteb_cryptography/`, 2015.

61. M. Tibouchi. Elligator squared: Uniform points on elliptic curves of prime order as uniform random strings. In Christin and Safavi-Naini [26], pages 139–156.

A   Magma verification code

The following Magma [18] code is included to allow easy verification of the complete addition and exception-free doubling formulas. The test loop at the bottom of each page chooses random odd order curves over very small primes in order to be able to exhaust all pairs of points on those curves.

A.1   Addition and doubling formulas for arbitrary a

```
ADD:=function(X1,Y1,Z1,X2,Y2,Z2,a,b3)
    t0 := X1*X2; t1 := Y1*Y2; t2 := Z1*Z2;
    t3 := X1+Y1; t4 := X2+Y2; t3 := t3*t4;
    t4 := t0+t1; t3 := t3-t4; t4 := X1+Z1;
    t5 := X2+Z2; t4 := t4*t5; t5 := t0+t2;
    t4 := t4-t5; t5 := Y1+Z1; X3 := Y2+Z2;
    t5 := t5*X3; X3 := t1+t2; t5 := t5-X3;
    Z3 := a*t4;  X3 := b3*t2; Z3 := X3+Z3;
    X3 := t1-Z3; Z3 := t1+Z3; Y3 := X3*Z3;
    t1 := t0+t0; t1 := t1+t0; t2 := a*t2;
    t4 := b3*t4; t1 := t1+t2; t2 := t0-t2;
    t2 := a*t2;  t4 := t4+t2; t0 := t1*t4;
    Y3 := Y3+t0; t0 := t5*t4; X3 := t3*X3;
    X3 := X3-t0; t0 := t3*t1; Z3 := t5*Z3;
    Z3 := Z3+t0;
    return X3,Y3,Z3;
end function;


DBL := function(X,Y,Z,a,b3)
    t0 := X^2;    t1 := Y^2;    t2 := Z^2;
    t3 := X*Y;    t3 := t3+t3; Z3 := X*Z;
    Z3 := Z3+Z3; X3 := a*Z3;   Y3 := b3*t2;
    Y3 := X3+Y3; X3 := t1-Y3; Y3 := t1+Y3;
    Y3 := X3*Y3; X3 := t3*X3; Z3 := b3*Z3;
    t2 := a*t2;   t3 := t0-t2; t3 := a*t3;
    t3 := t3+Z3; Z3 := t0+t0; t0 := Z3+t0;
    t0 := t0+t2; t0 := t0*t3; Y3 := Y3+t0;
    t2 := Y*Z;    t2 := t2+t2; t0 := t2*t3;
    X3 := X3-t0; Z3 := t2*t1; Z3 := Z3+Z3;
    Z3 := Z3+Z3;
    return X3,Y3,Z3;
end function;


while true do
    repeat q:=RandomPrime(6); until q gt 3; Fq:=GF(q);
    repeat repeat a:=Random(Fq); b:=Random(Fq); until not (4*a^3+27*b^2 eq 0);
        E:=EllipticCurve([Fq|a,b]); b3 := 3*b; until IsOdd(#E);
    for P in Set(E) do
        repeat Z1:=Random(Fq); until Z1 ne 0;
        X1:=P[1]*Z1; Y1:=P[2]*Z1; Z1:=P[3]*Z1;
        X3,Y3,Z3:=DBL(X1,Y1,Z1,a,b3); assert P+P eq E![X3,Y3,Z3];
        for Q in Set(E) do
            repeat Z1:=Random(Fq); Z2:=Random(Fq); until Z1*Z2 ne 0;
            X1:=P[1]*Z1; Y1:=P[2]*Z1; Z1:=P[3]*Z1;
            X2:=Q[1]*Z2; Y2:=Q[2]*Z2; Z2:=Q[3]*Z2;
            X3,Y3,Z3:=ADD(X1,Y1,Z1,X2,Y2,Z2,a,b3); assert P+Q eq E![X3,Y3,Z3];
        end for;
    end for; "complete";
end while;
```

A.2   Addition and doubling formulas for $a = -3$

```
ADD:=function(X1,Y1,Z1,X2,Y2,Z2,b)
    t0 := X1*X2; t1 := Y1*Y2; t2 := Z1*Z2;
    t3 := X1+Y1; t4 := X2+Y2; t3 := t3*t4;
    t4 := t0+t1; t3 := t3-t4; t4 := Y1+Z1;
    X3 := Y2+Z2; t4 := t4*X3; X3 := t1+t2;
    t4 := t4-X3; X3 := X1+Z1; Y3 := X2+Z2;
    X3 := X3*Y3; Y3 := t0+t2; Y3 := X3-Y3;
    Z3 := b*t2;  X3 := Y3-Z3; Z3 := X3+X3;
    X3 := X3+Z3; Z3 := t1-X3; X3 := t1+X3;
    Y3 := b*Y3;  t1 := t2+t2; t2 := t1+t2;
    Y3 := Y3-t2; Y3 := Y3-t0; t1 := Y3+Y3;
    Y3 := t1+Y3; t1 := t0+t0; t0 := t1+t0;
    t0 := t0-t2; t1 := t4*Y3; t2 := t0*Y3;
    Y3 := X3*Z3; Y3 := Y3+t2; X3 := t3*X3;
    X3 := X3-t1; Z3 := t4*Z3; t1 := t3*t0;
    Z3 := Z3+t1;
    return X3,Y3,Z3;
end function;


DBL := function(X,Y,Z,b)
    t0 := X^2;    t1 := Y^2;    t2 := Z^2;
    t3 := X*Y;    t3 := t3+t3; Z3 := X*Z;
    Z3 := Z3+Z3; Y3 := b*t2;   Y3 := Y3-Z3;
    X3 := Y3+Y3; Y3 := X3+Y3; X3 := t1-Y3;
    Y3 := t1+Y3; Y3 := X3*Y3; X3 := X3*t3;
    t3 := t2+t2; t2 := t2+t3; Z3 := b*Z3;
    Z3 := Z3-t2; Z3 := Z3-t0; t3 := Z3+Z3;
    Z3 := Z3+t3; t3 := t0+t0; t0 := t3+t0;
    t0 := t0-t2; t0 := t0*Z3; Y3 := Y3+t0;
    t0 := Y*Z;    t0 := t0+t0; Z3 := t0*Z3;
    X3 := X3-Z3; Z3 := t0*t1; Z3 := Z3+Z3;
    Z3 := Z3+Z3;
    return X3,Y3,Z3;
end function;


while true do
    repeat q:=RandomPrime(6); until q gt 3; Fq:=GF(q); a:=Fq!-3;
    repeat
        repeat b:=Random(Fq); until not (4*a^3+27*b^2 eq 0);
        E:=EllipticCurve([Fq|a,b]);
    until IsOdd(#E);
    for P in Set(E) do
        repeat Z1:=Random(Fq); until Z1 ne 0;
        X1:=P[1]*Z1; Y1:=P[2]*Z1; Z1:=P[3]*Z1;
        X3,Y3,Z3:=DBL(X1,Y1,Z1,b);
        assert P+P eq E![X3,Y3,Z3];
        for Q in Set(E) do
            repeat Z1:=Random(Fq); Z2:=Random(Fq); until Z1*Z2 ne 0;
            X1:=P[1]*Z1; Y1:=P[2]*Z1; Z1:=P[3]*Z1;
            X2:=Q[1]*Z2; Y2:=Q[2]*Z2; Z2:=Q[3]*Z2;
            X3,Y3,Z3:=ADD(X1,Y1,Z1,X2,Y2,Z2,b);
            assert P+Q eq E![X3,Y3,Z3];
        end for;
    end for;
    "complete";
end while;
```

A.3   Addition and doubling formulas for $a = 0$

```
ADD:=function(X1,Y1,Z1,X2,Y2,Z2,b3)
    t0 := X1*X2; t1 := Y1*Y2; t2 := Z1*Z2;
    t3 := X1+Y1; t4 := X2+Y2; t3 := t3*t4;
    t4 := t0+t1; t3 := t3-t4; t4 := Y1+Z1;
    X3 := Y2+Z2; t4 := t4*X3; X3 := t1+t2;
    t4 := t4-X3; X3 := X1+Z1; Y3 := X2+Z2;
    X3 := X3*Y3; Y3 := t0+t2; Y3 := X3-Y3;
    X3 := t0+t0; t0 := X3+t0; t2 := b3*t2;
    Z3 := t1+t2; t1 := t1-t2; Y3 := b3*Y3;
    X3 := t4*Y3; t2 := t3*t1; X3 := t2-X3;
    Y3 := Y3*t0; t1 := t1*Z3; Y3 := t1+Y3;
    t0 := t0*t3; Z3 := Z3*t4; Z3 := Z3+t0;
    return X3,Y3,Z3;
end function;


DBL := function(X,Y,Z,b3)
    t0 := Y^2;    Z3 := t0+t0; Z3 := Z3+Z3;
    Z3 := Z3+Z3; t1 := Y*Z;    t2 := Z^2;
    t2 := b3*t2; X3 := t2*Z3; Y3 := t0+t2;
    Z3 := t1*Z3; t1 := t2+t2; t2 := t1+t2;
    t0 := t0-t2; Y3 := t0*Y3; Y3 := X3+Y3;
    t1 := X*Y;    X3 := t0*t1; X3 := X3+X3;
    return X3,Y3,Z3;
end function;


while true do
    repeat
        repeat
            q:=RandomPrime(6);  Fq:=GF(q); a:=Fq!0; b:=Random(Fq);
        until not (4*a^3+27*b^2 eq 0) and q gt 5;
        E:=EllipticCurve([Fq|a,b]); b3:=3*b;
    until IsOdd(#E);
    for P in Set(E) do
        repeat Z1:=Random(Fq); until Z1 ne 0;
        X1:=P[1]*Z1; Y1:=P[2]*Z1; Z1:=P[3]*Z1;
        X3,Y3,Z3:=DBL(X1,Y1,Z1,b3);
        assert P+P eq E![X3,Y3,Z3];
        for Q in Set(E) do
            repeat Z1:=Random(Fq); Z2:=Random(Fq); until Z1*Z2 ne 0;
            X1:=P[1]*Z1; Y1:=P[2]*Z1; Z1:=P[3]*Z1;
            X2:=Q[1]*Z2; Y2:=Q[2]*Z2; Z2:=Q[3]*Z2;
            X3,Y3,Z3:=ADD(X1,Y1,Z1,X2,Y2,Z2,b3);
            assert P+Q eq E![X3,Y3,Z3];
        end for;
    end for;
    "complete";
end while;
```