Elliptic Curve Arithmetic for Cryptography

Srinivasa Rao Subramanya Rao

August 2017

A Thesis submitted for
the degree of Doctor of Philosophy
of The Australian National University

Declaration

I confirm that this thesis is a result of my own work, except as cited in the references. I also confirm that I have not previously submitted any part of this work for the purposes of an award of any degree or diploma from any University.

Srinivasa Rao Subramanya Rao
March 2017.

Previously Published Content

During the course of research contributing to this thesis, the following papers were published by the author of this thesis. This PhD thesis contains material from these papers:

1. Srinivasa Rao Subramanya Rao, *A Note on Schoenmakers Algorithm for Multi Exponentiation*, 12th International Conference on Security and Cryptography (Colmar, France), Proceedings of SECRYPT 2015, pages 384-391.

2. Srinivasa Rao Subramanya Rao, *Interesting Results Arising from Karatsuba Multiplication - Montgomery family of formulae*, in Proceedings of Sixth International Conference on Computer and Communication Technology 2015 (Allahabad, India), pages 317-322.

3. Srinivasa Rao Subramanya Rao, *Three Dimensional Montgomery Ladder, Differential Point Tripling on Montgomery Curves and Point Quintupling on Weierstrass' and Edwards Curves*, 8th International Conference on Cryptology in Africa, Proceedings of AFRICACRYPT 2016, (Fes, Morocco), LNCS 9646, Springer, pages 84-106.

4. Srinivasa Rao Subramanya Rao, *Differential Addition in Edwards Coordinates Revisited and a Short Note on Doubling in Twisted Edwards Form*, 13th International Conference on Security and Cryptography (Lisbon, Portugal), Proceedings of SECRYPT 2016, pages 336-343

and

5. Srinivasa Rao Subramanya Rao, *An improved EllipticNet Algorithm for Tate Pairing on Weierstrass' Curves, Faster Point Arithmetic and Pairing on Selmer Curves and a Note on Double Scalar Multiplication*, 7th International Conference on Applications and Technologies in Information Security(Cairns, Australia), Proceedings of ATIS 2016 Communications in Computer and Information Science, Volume 651, pages 93-105.

Acknowledgments

I would like to sincerely thank Prof Richard Brent for his guidance and support in my research, without which, this thesis would not have been possible.

I wish to thank all of my colleagues at the Mathematical Sciences Institute(MSI) with whom I may have interacted and learnt from, at various points in time. I also thank the ANU for providing me with resources, including financial support at various points in time.

I also thank all of the excellent teachers that I was blessed with since my school days. They inspired me to understand that asking questions is more important than the answers to those questions.

I also thank my parents Sulochana Bai and Subramanya Rao for their unconditional support through out my life and thanks should go to my brother Krishna as well, for his sometimes conditional and sometimes unconditional support. I would also thank my wife Gayathri Srinivas, who has been more than a useful and helpful distraction, almost always.

Abstract

The advantages of using *public key cryptography* over *secret key cryptography* include the convenience of better key management and increased security. However, due to the complexity of the underlying number theoretic algorithms, public key cryptography is slower than conventional secret key cryptography, thus motivating the need to speed up public key cryptosystems.

A mathematical object called an *elliptic curve* can be used in the construction of public key cryptosystems. This thesis focuses on speeding up *elliptic curve cryptography* which is an attractive alternative to traditional public key cryptosystems such as RSA. Speeding up elliptic curve cryptography can be done by speeding up *point arithmetic* algorithms and by improving *scalar multiplication* algorithms. This thesis provides a speed up of some point arithmetic algorithms. The study of *addition chains* has been shown to be useful in improving scalar multiplication algorithms, when the scalar is fixed. A special form of an addition chain called a *Lucas chain* or a *differential addition chain* is useful to compute scalar multiplication on some elliptic curves, such as *Montgomery curves* for which differential addition formulae are available. While single scalar multiplication may suffice in some systems, there are others where a double or a triple scalar multiplication algorithm may be desired. This thesis provides triple scalar multiplication algorithms in the context of differential addition chains. *Precomputations* are useful in speeding up scalar multiplication algorithms, when the elliptic curve point is fixed. This thesis focuses on both speeding up point arithmetic and improving scalar multiplication in the context of precomputations toward double scalar multiplication. Further, this thesis revisits pairing computations which use elliptic curve groups to compute *pairings* such as the Tate pairing. More specifically, the thesis looks at Stange's algorithm to compute pairings and also pairings on Selmer curves. The thesis also looks at some aspects of the underlying *finite field* arithmetic.

Contents

Chapter 1

Cryptography and Elliptic Curves

This chapter provides an overview of the use of elliptic curves in cryptography. We first provide a brief background to public key cryptography and the Discrete Logarithm Problem, before introducing elliptic curves and the elliptic curve analogue of the Discrete Logarithm Problem.

1.1 Cryptography and the Discrete Logarithm Problem

While the history of cryptography is probably as old as the history of our species and has used diverse techniques, contemporary cryptography is the design, development and analysis of mathematical techniques for secure communication in the presence of adversaries [51]. The book [92] provides a very readable account of cryptography.

Typically Alice and Bob communicate with each other over an unsecured communication channel in the presence of an adversary Eve who intends to subvert any security services available to Alice and Bob. Cryptographic systems can be broadly classified into *secret key cryptosystems* and *public key cryptosystems*. If Alice and Bob intend to use a secret key cryptosystem,

they agree upon a secret key k which is known only to Alice and Bob and is a secret to the rest of the world. Given a plaintext p, Alice would use a function $f(p, k)$ to produce the ciphertext c and this is transmitted to Bob. Bob uses the function $g(c, k)$ to recover the plaintext p. While efficiency is the main advantage of secret key cryptosystems, the key used by Alice and Bob should first be exchanged using a secure communication channel. This would require a second secret key to ensure secrecy of the first and this would then require a third secret key to ensure secrecy of the second and so on. Even if this problem were to be resolved by using a physically secure channel such as a trusted courier, if n entities need to communicate amongst themselves and if each of these n communications is to be a secret, each communicating entity should ensure that the $n-1$ secret keys are appropriately exchanged with the other $n-1$ entities with whom it communicates. Moreover, a *digital signature scheme* where a participant cannot deny previous commitments cannot be easily constructed using secret key cryptography techniques. The above problems are satisfactorily overcome by using *public key cryptography*, initially introduced in [39].

In public key cryptography, Alice and Bob have two keys each, called a key pair. Each key pair consists of a private and a public key and it is not easy to derive the private key from the public key. The relationship between the private and the public key must be such that it is computationally difficult to derive the private key, given that the public key is known to the whole world. The computational difficulty usually depends on the assumed intractability of number theoretic problems such as integer factorization, the discrete logarithm problem and its elliptic curve analog. These number theoretic problems are assumed to be intractable (in the absence of quantum computers), but intractability is an open problem. They are conjectured to be neither in complexity class P nor in *NP-Complete*.

Now we set up the framework for the discrete logarithm problem. We begin by defining a Group. A *Group* is a set G which comes with an operation, say $+$, usually called the *Group Law*, which satisfies the following properties:

(i) Closure: If x and $y \in G$, then $x + y \in G$.
(ii) Associativity: If x, y and $z \in G$, then $(x + y) + z = x + (y + z)$.
(iii) Identity: There exists an element $0 \in G$ such that $x + 0 = x = 0 + x$ for all $x \in G$.
(iv) Inverse: For every element $x \in G$, there exists $-x \in G$, such that $x + (-x) = 0 = (-x) + x$, the identity element of G.

Some of the conditions above may be redundant. If G is a group such that $a + b = b + a$ for all $a, b \in G$, then G is called a *commutative* or an *Abelian* group. For instance, if p is a prime number, $\mathbf{F}_p = \{0, 1, 2, ...p - 1\}$ is the set of integers modulo p and if $+$ is the addition modulo p operation, then $(\mathbf{F}_p, +)$ is a finite Abelian group where the identity element equals 0. Further, if $*$ is the multiplication modulo p operation, then $(\mathbf{F}_p^*, *)$ is a multiplicative group where $\mathbf{F}_p^* = \mathbf{F}_p \backslash \{0\}$ and the identity element equals 1. The triple $(\mathbf{F}_p, +, *)$ is usually written as $\mathbf{F}_p$ in the literature and is a *Finite Field*, where for all $x, y, z \in \mathbf{F}_p$, $(x + y) * z = (x * z) + (y * z)$. The *order* of a group is the number of elements in the group (if finite) or ∞ (if number of elements is infinite). Thus the order of $(\mathbf{F}_p, +)$ is p while the order of $(\mathbf{F}_p^*, *)$ is $p - 1$.

Now, we formulate the Discrete Logarithm Problem. Assume that the group operation is $*$. Given a fixed element $a \in G$ and x a positive integer, it is possible to compute $b \in G$, where $b = a^x$ using a polynomial time algorithm (a^x is defined as $a^x = a * a^{(x-1)}$ if $x > 0$ and $a^0 = 1$). However, if it is the other way around, that is, given $a, b \in G$, the problem of determining the least possible (or any) value of x such that $b = a^x$ is called the Discrete Logarithm Problem (DLP) and using current state of the art algorithms, cannot be computed in polynomial time for groups such as $(\mathbf{F}_p^*, *)$. While it is easy to construct a polynomial time algorithm to compute the DLP in groups such as the additive group $\mathbb{Z}/n\mathbb{Z}$, the best currently available algorithms for the DLP in the multiplicative group of a finite field with medium or large characteristic run in subexponential time (subexponential in the bit-length of the group order). Recent research has shown that it is possible to construct better

than sub-exponential algorithms to compute the DLP in finite fields of small characteristic [6]. The computational complexity of the DLP depends on the properties of the underlying group such as the order of the group and the prime factorisation of the order. The order of a is the order of the subgroup $\langle a \rangle$ generated by $a \in G$. Groups suitable for public key cryptosystems should be such that the order of the fixed/base element a should be either a large prime or the product of a large prime and a very small integer [63, Section 5]. It can be conjectured that the DLP in $(\mathbf{F}_p^*, *)$ where $p-1$ has large prime factors is as hard as factoring integers of size about the same as that of p [83]. A very accessible introduction to computation of discrete logarithms is [87].

The success of public key cryptosystems such as the Diffie-Hellman key exchange algorithm and the ElGamal encryption algorithm rely on the assumed intractability of the DLP in a finite cyclic group, sometimes called the *generalized discrete logarithm problem* in the literature [77], which can be stated as follows: Given a finite cyclic group G with a generator a and group order n and some $b \in G$, compute the integer x, $1 \leq x \leq (n-1)$ such that $a^x = b$. (If b is the identity element in G, then $x = 0$). In the Diffie-Hellman key exchange algorithm, Alice and Bob intend to exchange a secret key between themselves. As part of the initial setup, they choose a finite cyclic group G with order n and generator a. Then Alice chooses an integer x at random ($1 \leq x \leq (n-1)$) and without revealing the value of x to anybody, computes a^x and transmits it to Bob over a public channel. Bob chooses an integer y at random ($1 \leq y \leq (n-1)$) and without revealing the value of y to anybody, computes a^y and transmits it to Alice. Alice computes $(a^y)^x$ and Bob computes $(a^x)^y$, thus resulting in both Alice and Bob sharing the secret key a^{xy} between themselves. An intruder Eve with access to both a^x and a^y, may desire to compute a^{xy}. The problem that Eve is confronted with is sometimes called the Generalized Diffie-Hellman problem (GDHP) in the literature [77]. An efficient algorithm to solve the DLP will enable Eve to solve the GDHP, as she can compute x and y and thus compute a^{xy}, resulting in the compromise of the cryptosystem. An efficient algorithm to solve the DLP will not only result in the compromise of the Diffie-Hellman key

exchange algorithm, but also compromise cryptosystems such as the ElGamal encryption system. It is conjectured that solving the GDHP is equivalent in difficulty to solving the DLP. However, this conjecture has not been proved yet.

1.2 Elliptic Curves

An elliptic curve is a mathematical object that can be studied from a variety of perspectives: Number Theory, Diophantine Problems, Algebra, Algebraic Geometry and so on. In the foreword to his book on elliptic curves [65], Lang writes

> *It is possible to write endlessly on elliptic curves (This is not a threat).*

Recently, elliptic curves played a major role in graduating the celebrated *Fermat's Last Theorem* from the status of a conjecture to a theorem, leading to a renewed interest in elliptic curves and their study. The use of elliptic curves in cryptography has led to a commercial interest in elliptic curves. *Thus it is possible to write endlessly and beyond on elliptic curves.* In this thesis, however, we confine ourselves to computational aspects of elliptic curve cryptography. We now proceed to provide a very short introduction to elliptic curves and their utility in public key cryptography.

1.2.1 What are Elliptic Curves?

An introduction to elliptic curves can be found in Poonen's article [88]. Curves of the form $\{(x, y) : f(x, y) = 0\}$ where $f(x, y)$ is a polynomial in two variables are called *plane curves*. The degree of the curve is the maximum of $(i + j)$ when cx^iy^j is any monomial occurring in f with $c \neq 0$. While plane curves of degree 1 are called *lines*, plane curves of degree 2 are called *conic sections* or *conics*. Plane curves of degree 3 are called *cubics*. The general form of a cubic is

$$c_1x^3 + c_2x^2y + c_3xy^2 + c_4y^3 + c_5x^2 + c_6xy + c_7y^2 + c_8x + c_9y + c_{10} = 0 \quad (1.1)$$

Nonsingular cubic curves with a distinguished rational point defined over the base field are called *Elliptic curves*. Equation (1.1) can be transformed into an equation with the following form [36]:

$$y^2 + a_1xy + a_3y = f(x) \tag{1.2}$$

where $f(x)$ is a polynomial of degree 3 and $f(x)$ has no multiple roots. In a finite field F with identity element 1, the smallest value of n such that

$$\underbrace{1 + 1 + \cdots + 1}_{\text{(n-1) additions}} = 0$$

is called the characteristic of the field. Over a finite field K whose characteristic is not equal to 2 or 3, one can transform any elliptic curve into one of the form

$$y^2 = x^3 + Ax + B \tag{1.3}$$

where A and $B \in K$ [51]. Any curve of the above form is an elliptic curve (i.e., without multiple roots) if and only if $(4A^3 + 27B^3) \neq 0$.

1.2.2 Why the name *Elliptic Curve*?

A brief historical introduction to elliptic curves is provided in [102]. For about one and a half millennia, from the time of Diophantus to Newton, mathematical properties of certain cubic equations that are today known as elliptic curves were seen to be generalizations of those of conics. However the advent of calculus helped highlight marked differences between conics and elliptic curves. While conic sections can be parameterized by rational functions, elliptic curves cannot be parameterized by rational functions. The simplest functions that can parameterize elliptic curves are *elliptic functions* encountered in calculus as the inverses of so called *elliptic integrals*. Elliptic integrals are called so, as a typical example is the integral for the arc length of an ellipse. Thus the name *elliptic curve*.

Interesting introductory expositions to elliptic curves can be found in [21, 22, 66].

1.3 Group Law on Elliptic Curves and the ECDLP

If we denote the elliptic curve in equation (1.3) as E, we can define the addition of two points P and Q on E, denoted as $P + Q$, as follows: Draw a line connecting P and Q and intersecting E at another point denoted by $\overline{P + Q}$, which in turn is reflected through the x-axis to obtain the point $P+Q$. $\overline{P + Q}$ exists provided P and Q are distinct and not on a vertical line, and is unique because the equation is a cubic. This is shown below when the field under consideration is $\mathbb{R}$.

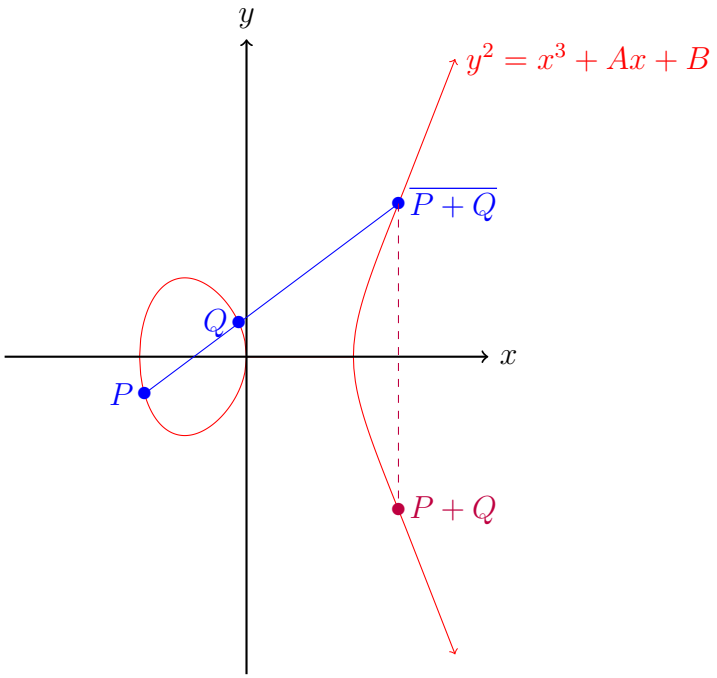


Figure 1.1: Elliptic Curve Point Addition

For point doubling, $(P = Q)$, the line connecting P and Q becomes the tangent at P as shown below.

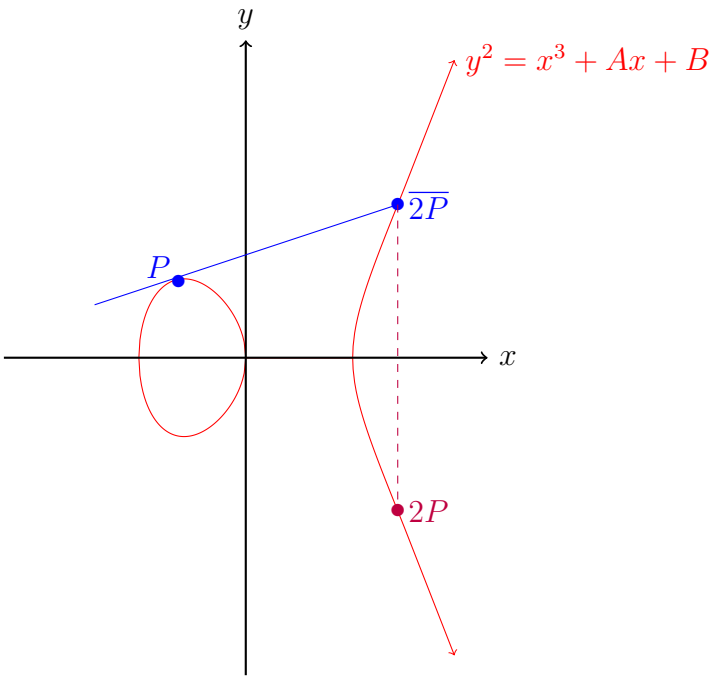


Figure 1.2: Elliptic Curve Point Doubling

Along with all other points $(x, y) \in K^2$, where K is the field in which E is defined, the inclusion of the point (∞, ∞), denoted by $\mathcal{O}$ is necessary in order to obtain a group.

We will take the lines through (∞, ∞) to be vertical. Thus when $P + \mathcal{O}$ is computed, the line through $P = (x, y)$ and $\mathcal{O}$ should intersect E at $(x, -y)$. Similarly when $(x, -y) + \mathcal{O}$ is computed, the line through $P = (x, -y)$ and $\mathcal{O}$ should intersect E at (x, y). Thus $P + \mathcal{O} = P$ and $P + (-P) = \mathcal{O}$. We can also show that if P, Q and R are points on E, then $(P+Q)+R = P+(Q+R)$. Also, $P + Q = Q + P$. Thus the operation of point addition satisfies the properties of Closure, Associativity, Identity, Inverse and Commutativity thereby ensuring that the set of points on E with the addition operation $'+'$ constitutes an Abelian group.

It was possible to draw the elliptic curves shown in Figures 1.1 and 1.2 above using Janoski's document [55].

The earliest use of elliptic curves in areas related to Cryptology was not in traditional encryption and decryption but in constructing a sub-exponential algorithm for integer factorization. This algorithm was introduced by Lenstra in [70] and further enhanced by Brent and Montgomery. For instance, see [15] [16], [80]. A good reference on elliptic curve factorization is the paper [43].

We can now set up the Elliptic curve analogue of the DLP. If P and Q are two points on an elliptic curve and given that $P = xQ$ for some integer x (xQ is defined as $xQ = (x-1)Q + Q$ if $x > 0$ and $xQ = \mathcal{O}$ when $x = 0$), then the problem of finding x given P and Q on E is the Elliptic Curve Discrete Logarithm Problem(ECDLP). One of the advantages of using the ECDLP as compared to integer factorization or just the DLP is that while there are sub-exponential algorithms to solve the integer factorization problem and the DLP, there are no known subexponential algorithms (subexponential in the bit-length of the curve group order) to solve the ECDLP. (Some instances of the ECDLP may not be hard. For instance, when the group order can be written as a product of only small primes, it may be easy to solve the ECDLP using the Pohlig-Hellman algorithm). This advantage translates into smaller key sizes in an ECDLP based system as compared to key sizes in integer factorization based systems such as RSA, for the same level of security. Smaller key sizes make it suitable for elliptic curve cryptosystems to be deployed in constrained environments such as mobile platforms. The notions of exponential and subexponential algorithms can be formalized using the *big-O* notation [86]. However, in this thesis, we focus on operation counts to compare point arithmetic algorithms rather than comparing algorithms from the perspective of classical computational complexity.

While it should be difficult for an adversary Eve to solve the ECDLP, it is desirable for Alice and Bob to be able to encrypt, decrypt, digitally sign and verify digital signatures as quickly as possible. In other words, computation of quantities such as $P = xQ$ or $P = xQ + yR$, (where P, Q and R are points on an elliptic curve and x and y are integers) and the group operations

in the underlying group in which the ECDLP is setup should be made as fast as possible. This thesis focuses on speeding up elliptic curve arithmetic including scalar multiplication algorithms. Faster elliptic curve arithmetic not only facilitates faster cryptography but may also enhance the performance of cryptanalytic methods.

1.4 Roadmap

The rest of the thesis is structured as follows:

In Chapter 2, we provide an introduction to point arithmetic formulae for various forms of elliptic curves such as Weierstrass, Montgomery, Edwards, Huff and Selmer curves with point representations such as affine, projective coordinates etc. We motivate the need for fast quintupling algorithms and then (i) show that Giorgi's quintupling algorithm for Weierstrass curves can be derived from Mishra's and Dimitov's algorithm and (ii) provide new quintupling formulae for Edwards curves.

In Chapter 3, we provide new differential point tripling formulae for Montgomery curves and then provide faster differential arithmetic algorithms for Generalized Edwards and a variant of Binary Edwards curves.

In Chapter 4, we provide new algorithms for triple scalar multiplication in the context of differential addition chains and compare these algorithms with the straight-forward method for triple scalar multiplication. In this chapter, we also provide a context in which the differential tripling formulae for Montgomery curves can be used.

Making use of results from Chapter 2, we review two algorithms for double scalar multiplication in Chapter 5. One is based on conjugate addition and the other on co-Z addition. Whilst reviewing the algorithm based on co-Z addition, we show that some results published in the literature are incorrect and in this process provide a new algorithm for double scalar multiplication.

In Chapter 6, we provide an improvement to Stange's Elliptic Net algorithm for the Tate pairing and then improve the Tate pairing computation on Selmer curves.

In Chapter 7, we show that the 3-way Karatsuba algorithm can be derived from the 2-way Karatsuba algorithm and then provides a new family of formulae to multiply two quadratics.

Chapter 2

Arithmetic on Elliptic Curves and some improvements

In the previous chapter, we looked at the formulation of the group law for the set of points on an elliptic curve. In this chapter, we look at some formulae that are used to compute the group operation, that is the formulae related to elliptic curve point arithmetic. There are various forms of elliptic curves, such as the Weierstrass curves, Montgomery curves etc and then there are various forms of point representation on these elliptic curves, such as affine coordinates, projective coordinates etc. In this chapter, we also review some formulae for Weierstrass and Edwards curves. We start with Weierstrass curves.

2.1 Weierstrass Curves

Assume that K is a finite field, $char(K) \neq 2, 3$. An elliptic curve of the form

$$E : y^2 = x^3 + a_4x + a_6 \text{ where } a_4 \text{ and } a_6 \in K \text{ and } (4a_4 + 27a_6) \neq 0$$

is called a Weierstrass curve. Various point representation systems can be used to perform point arithmetic operations on elliptic curves. We will first look at the point representation system in which a curve point is represented

in the most natural way, which is (x, y). This is the affine coordinate system and in this system $-(x, y) = (x, -y)$.

2.1.1 Affine Coordinates

If $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are any two distinct points on E, $P \neq \mathcal{O}, Q \neq \mathcal{O}$ and $P \neq \pm Q$, then $P + Q = (x_3, y_3)$ is given by

$$\left.\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ \text{and} \qquad y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}\right\} \qquad \text{where} \qquad \lambda = \frac{(y_1 - y_2)}{(x_1 - x_2)}$$

If $P = Q = (x_1, y_1)$ and $P \neq -P$, then $2P = (x_3, y_3)$ is defined as

$$\left.\begin{aligned} x_3 &= \lambda^2 - 2x_1 \\ \text{and} \qquad y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}\right\} \qquad \text{where} \qquad \lambda = \frac{3x_1^2 + a_4}{2y_1}$$

Taking I to be the cost of one inversion, M to be the cost of one multiplication and S to be the cost of one squaring in the field K, then

$$\begin{aligned} &\text{Addition requires } 1I + 2M + 1S \text{ operations} \\ &\text{and Doubling requires } 1I + 2M + 2S \text{ operations.} \end{aligned}$$

Unless otherwise stated, the cost of the *field* operations I, M and S are considered, while the cost of a field addition/subtraction is ignored, since it is small compared to $M/S/I$. Though a squaring is clearly a special case of a multiplication, where the operands are equal, it is useful to distinguish between these two operations, as it is possible to tune a squaring to provide a tangible speedup over a multiplication. The best possible speedup factor is 2 [17, Exercise 1.17]. A good rule of thumb is to take the cost of a squaring to be about 2/3 that of a multiplication [17, Section 1.3.6, Figure 1.2]. Clearly, these comparisons depend on the underlying implementation of these operations. and in some implementations, $S = 0.8M$. Computing an inverse in K is an expensive operation and typically $I > 30M$. We denote the cost of multiplication by a small constant c by M_c. It is useful to distinguish

between M and M_c as special purpose algorithms can be used for constant multiplication [17, Section 1.3.7].

Next, we look at projective coordinates. The use of projective coordinates (and its many variants) avoids the use of the inverse operation, whereas when affine coordinates are used, it is not easy to do away with the inverse operation.

2.1.2 Projective Coordinates

The homogeneous projective form of the Weierstrass equation can be written as

$$E : Y^2Z = X^3 + a_4XZ^2 + a_6Z^3$$

In this system, (x, y) is replaced with any triple $(X, Y, Z) = (x\zeta, y\zeta, \zeta)$ where $\zeta \in K^*$. From a triple (X, Y, Z), the affine coordinates can be written as $\text{Affine}(X, Y, Z) = (x = X/Z, y = Y/Z)$. The negative of $-(X, Y, Z)$ is $(X, -Y, Z)$ while the identity element $\mathcal{O}$ corresponds to $(0, 1, 0)$ (This is a special case having $Z = 0$, otherwise $Z \neq 0$). If points $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ correspond to affine points $(X_1/Z_1, Y_1/Z_1)$ and $(X_2/Z_2, Y_2/Z_2)$ respectively, $P \neq \pm Q$, $P \neq \mathcal{O}$ and $Q \neq \mathcal{O}$, then the point $P+Q = (X_3, Y_3, Z_3)$ can be computed as follows:

$$\begin{aligned}
\text{Computing} \quad & A = Y_2Z_1 - Y_1Z_2, \\
& B = X_2Z_1 - X_1Z_2 \\
\text{and} \quad & C = A^2Z_1Z_2 - B^3 - 2B^2X_1Z_2, \\
\text{we get} \quad & X_3 = BC, \\
& Y_3 = A(B^2X_1Z_2 - C) - B^3Y_1Z_2 \\
\text{and} \quad & Z_3 = B^3Z_1Z_2
\end{aligned}$$

Similarly, if $2P = (X_3 : Y_3 : Z_3)$,

$$\begin{aligned} \text{Computing} \quad & A = a_4 Z_1^2 + 3X_1^2, \\ & B = Y_1 Z_1, \\ & C = X_1 Y_1 B \\ \text{and} \quad & D = A^2 - 8C \\ \text{we get} \quad & X_3 = 2BD, \\ & Y_3 = A(4C - D) - 8Y_1^2 B^2 \\ \text{and} \quad & Z_3 = 8B^3 \end{aligned}$$

Addition requires $(12M + 2S)$ operations and doubling requires $(7M + 5S)$ operations

Since every addition or a doubling in affine coordinates requires one inversion operation, the number of inversions increases with the number of point additions or doublings required. Use of projective coordinates avoids this problem, but at the cost of additional memory, as Z-coordinates for all of the points being processed need to be stored as well. In this context, it may be appropriate to quote the research problem 7.25 posed by Crandall and Pomerance in [35] which states

" . . . One looks longingly at expressions $x_3 = \lambda^2 - x_1 - x_2$, $y_3 = \lambda(x_1 - x_3) - y_1$, *in the realization that if only inversion were 'free', the affine approach would be superior. However, known inversion methods are quite expensive. One finds in practice that inversion times tend to be one or two orders of magnitude greater than multiply-mod times . . . it is very hard to bring down the cost of inversion(modulo a typical cryptographic prime* $p \approx 2^{200}$*) to 20 multiplies . . . ".*

Continuing with the above research question, the authors of [35] motivate readers to think about primes of special forms, use look up tables and specialized gcd algorithms for modular inversion with the aim of reducing the impact of the high computational costs of inversion in affine coordinates.

Whilst one way of overcoming the costs of inversion is to reduce the cost of computing the inversion, a complementary way to do this is to try to reduce the number of inversions required. In recent years, there has been some research in this direction.

To begin with, the authors in [45] introduced a new method to reduce the number of operations required to be computed when affine coordinates are used. This method may be called the $2P + Q$ method or "Double-Add" method. We next look at this method.

2.1.3 $2P + Q$/Double-Add Method

If $P = (x_1, y_1), Q = (x_2, y_2)$ are two points on $E : y^2 = x^3 + a_4x + a_6$, then $2P + Q = (x_4, y_4)$ can be computed as below:
We could first compute $P + Q = (x_3, y_3)$ as

$$\left.\begin{aligned} x_3 &= \lambda_1^2 - x_1 - x_2 \\ \text{and} \qquad y_3 &= (x_1 - x_3)\lambda_1 - y_1 \end{aligned}\right\} \qquad \text{where} \qquad \lambda_1 = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

To obtain $2P+Q = (x_4, y_4)$, $\quad P+Q = (x_3, y_3)$ can be added to $P = (x_1, y_1)$. Thus

$$\left.\begin{aligned} x_4 &= \lambda_2^2 - x_1 - x_3 \\ y_4 &= (x_1 - x_4)\lambda_2 - y_1 \end{aligned}\right\} \qquad \text{where} \qquad \lambda_2 = \frac{(y_3 - y_1)}{(x_3 - x_1)}$$

Now λ_2 can be written as

$$\lambda_2 = \frac{y_3 - y_1}{x_3 - x_1} = \frac{(x_1 - x_3)\lambda_1 - y_1}{(x_3 - x_1)} = -\lambda_1 - \frac{2y_1}{(x_3 - x_1)}$$

A straightforward computation of $P+Q = (x_3, y_3)$ would require $(1I+2M+1S)$ and similarly computing $(P + Q) + P$ would require another $(1I + 2M + 1S)$, thus requiring a total of $(2I + 4M + 2S)$. However, in the above equation for λ_2, y_3 need not be computed thus saving $1M$. Thus $(2P+Q)$ can be computed

using $(2I + 3M + 2S)$ operations. This can be extended to computing $3P$ and $(3P + Q)$ as well.

$2P + Q$	requires	$(2I + 3M + 2S)$	operations
$3P$	requires	$(2I + 3M + 3S)$	operations
$3P + Q$	requires	$(3I + 4M + 3S)$	operations

Thus whilst the *2P+Q* method reduces the number of multiplications required, it does not reduce the number of inversions required. However, in a subsequent paper [27], Ciet, Joye, Lauter and Montgomary reduce the number of inversions required by using the trick of simultaneous inverse computation. This can be called the *Enhanced 2P+Q* method, which we describe below:

2.1.4 Enhanced $2P + Q$ Method

As seen previously, whilst computing $(2P + Q)$ as $((P + Q) + P)$, y_3 need not be computed. It turns out that x_3 need not be computed as well. $2P + Q = (x_4, y_4)$ can be computed directly as follows:

x_4 can be equated to $(\lambda_2 - \lambda_1)(\lambda_1 + \lambda_2) + x_2$

Letting $d = (x_2 - x_1)^2(2x_1 + x_2) - (y_2 - y_1)^2$, we see that $d = (x_2 - x_1)^2(x_1 - x_3)$

Defining $D = d(x_2 - x_1)$ and $I = D^{-1}$ we have

$$\frac{1}{x_2 - x_1} = dI \qquad \text{and} \qquad \frac{1}{(x_1 - x_3)} = (x_2 - x_1)^3 I$$

Counting the number of operations required to compute x_4 and y_4 results in $(1I + 9M + 2S)$ thus reducing the number of inversions required. This

method could be extended to compute $3P, (3P+Q), 4P$ and $(4P+Q)$.

$2P+Q$	requires	$(1I+9M+2S)$	operations
$3P$	requires	$(1I+7M+4S)$	operations
$3P+Q$	requires	$(2I+9M+4S)$	operations
$4P$	requires	$(1I+9M+9S)$	operations
$4P+Q$	requires	$(2I+11M+4S)$	operations

Whilst the ideas in [45, 27] were further extended and generalized in a 2007 paper [37] by Dahmen, Okeya and Schapers(DOS), where the authors precompute all odd points $3P, 5P \ldots (2k-1)P,\ k \geq 2$ in affine coordinates for elliptic curves over a prime field, whilst requiring $(10k-11)M+(4k)S+1I$ operations to compute these points and they use $2(k-1)$ registers, the authors in [67] used the same ideas to provide a faster affine point Quadrupling scheme for Weierstrass curves over a prime field costing $(1I+8M+8S)$. The DOS algorithm was then adapted by the authors in [42] to precompute $2P, 3P, 5P \ldots (2k-1)P,\ k \geq 2$ in affine coordinates for elliptic curves over a field of char 2 and their algorithm required $(11k-13)M+(2k)S+1I$ operations.

There are other ways using which the number of inversion computations can be reduced. The story of reducing the number of inversions in affine representation did not begin with [27]. One of the earliest attempts at doing this can be seen in [29]. Using the well known Montgomery inversion trick, the authors in [29] first compute $2P$, and then subsequently compute $(3P, 4P), (5P, 7P, 8P) \ldots, ((k.2^{-2}+1)P, \ldots, (k.2^{-1}-1)P, (k.2^{-1}P)),$ $((k.2^{-1}+1)P, \ldots, (k-1)P)$ thus reducing the number of inverses required to be computed from k to $(\lceil \log_2 k \rceil + 1)$.

Further, Okeya, Takagi and Vuillaume in [85] introduced the idea of computing $P \pm Q$ simultaneously, whilst saving one inversion in the process. Then, at ECC 2008, Michael Scott motivated the further use of this idea in precomputation schemes, not confined to affine coordinates. Taking this useful idea further, Longa and Gebotys in [72] provide new precomputation schemes for projective coordinates on Weierstrass, Jacobi Quartic and Edwards

curves. A precomputation algorithm structurally similar to that of in [72] was constructed by Le and Tan in [68]. We revisit some of these precomputation schemes further in Chapter 5 of this thesis.

The motivation in using the $2P + Q$ or the enhanced $2P + Q$ method or the DOS algorithm to precompute $3P, 5P \ldots (2k - 1)P$ is to trade field inversions for multiplications when elliptic curve points, for efficiency reasons, are represented in affine coordinates. The $2P + Q$ method can also be used to efficiently compute $3P$ [27]. These methods can be used in cryptographic protocols where the elliptic curve points such as P and Q are not fixed, and thus the precomputations cannot be performed offline [37].

Next we look at Jacobian coordinates, a derivative of projective coordinate point representation.

2.1.5 Jacobian Coordinates

The weighted projective form of the Weierstrass equation is written as

$$E : Y^2 = X^3 + a_4 X Z^4 + a_6 Z^6$$

In this system, (x, y) is replaced with any triple $(X, Y, Z) = (x\lambda^2, y\lambda^3, \lambda)$ where $\lambda \in K^*$. From a triple (X, Y, Z), the affine coordinates can be written as $\text{Affine}(X, Y, Z) = (x = X/Z^2, y = Y/Z^3)$. The negative of $-(X : Y : Z)$ is $(X : -Y : Z)$ while the identity element $\mathcal{O}$ corresponds to $(1, 1, 0)$. If points $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ correspond to affine points $(X_1/Z_1^2, Y_1/Z_1^3)$ and $(X_2/Z_2^2, Y_2/Z_2^3)$ respectively, $P \neq \pm Q$, $P \neq \mathcal{O}$, $Q \neq \mathcal{O}$, and $P \neq \pm Q$, then the point $P + Q = (X_3, Y_3, Z_3)$ can be computed as

follows:

$$\begin{aligned} \text{Setting} \quad & A = X_1 Z_1^2, \\ & B = X_2 Z_1^2, \\ & C = Y_1 Z_2^3, \\ & D = Y_2 Z_1^3 \text{ and} \\ & E = (B - A), F = (D - C) \\ \text{we can write} \quad & X_3 = -E^3 - 2AE^2 + F^2, \\ & Y_3 = -CE^3 + F(AE^2 - X_3) \text{ and} \\ & Z_3 = Z_1 Z_2 E \end{aligned}$$

Similarly, if $2P = (X_3, : Y_3, Z_3)$, we can compute $2P$ as follows:

$$\begin{aligned} \text{Setting} \quad & A = 4X_1 Y_1^2 \text{ and} \\ & B = 3X_1^2 + a_4 Z_1^4 \\ \text{we can write} \quad & X_3 = -2A + B^2 \\ & Y_3 = -8Y_1^4 + B(A - X_3) \\ & Z_3 = 2Y_1 Z_1 \end{aligned}$$

Addition requires $(12M + 4S)$ operations and doubling requires $(4M + 6S)$ operations.

Other derivatives of projective coordinate systems are *Chudnovsky Jacobian coordinates* and *Modified Jacobian coordinates*. In Chudnovsky Jacobian coordinates, an affine point $(X/Z^2, Y/Z^3)$ is represented as a quintuple (X, Y, Z, Z^2, Z^3). Point Addition requires $(11M + 3S)$ operations and point doubling requires $(5M + 6S)$ operations and this representation ensures faster addition and doubling when compared to Projective coordinate point representation. In Modified Jacobian coordinates, an affine point $(X/Z^2, Y/Z^3)$ is represented as $(X, Y, Z, a_4 Z^4)$ The operation counts for modified Jacobian co-ordinates are $(13M + 6S)$ for point addition and $(4M + 4S)$ for point doubling.

Until now, in this chapter, we looked at some formulae for point arithmetic on the Weierstrass curve. We will return to other formulae for Weierstrass

curves in Section 2.6.2 of this chapter and then in Chapter 5. We now turn to other forms of elliptic curves.

2.2 Montgomery Curves

In a landmark paper [80], an elliptic curve of the form

$$E_m : By^2 = x^3 + Ax^2 + x$$

was introduced by Peter Montgomery. Over a field K where $char(K) \neq 2$, curve parameters A, $(B \neq 0)$ for E_m satisfy A, $B \in K$, and $B(A^2 - 4) \neq 0$. A Montgomery curve over K can be written in Weierstrass form, whilst it is not always possible for a Weierstrass curve to be written in Montgomery form as the order of a Montgomery curve is divisible by 4. The Weierstrass curve $y^2 = x^3 + a_4x + a_6$ over K can be written in Montgomery form if and only if the polynomial $x^3 + a_4x + a_6$ has a root x_p in K and $(3x_p^2 + a_4)$ is a square in K [28]. If $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are points on E_m and $x_1 \neq x_2$, then the x-coordinate of $P + Q = (x_3, y_3)$ can be computed (provided the x-coordinate of $P - Q = (x_4, y_4)$ is known) as follows:

$$x_3 = \frac{1}{x_4}.\frac{(x_2x_1 - 1)^2}{(x_1 - x_2)^2}, \quad \text{which requires} \quad (1I + 2M + 2S) \text{ operations.}$$

The inverse operation can be avoided if projective coordinates are used. We provide further details in Chapter 3 of this thesis. The formulae for Montgomery curves are different to those of Weierstrass curves, in the sense that, x coordinates suffice for point arithmetic (x-coordinate only arithmetic). It is attractive due to its low operation count and lower memory requirement. However, the difference of the two points being added should be known in advance. The *x-coordinate* only formulae can be generalized to certain other forms of elliptic curves[18]. We provide a list of such generalizations in Sec 3.3 of this thesis.

2.3 Edwards Curves

In 2007, Edwards [44] introduced a new form of an elliptic curve, now known as *Edwards curves*. An Edwards curve, defined over a field K where $char(K) \neq 2$, is given by the following equation [11]

$$E_E : x^2 + y^2 = 1 + dx^2y^2, \quad \text{where} \quad d \in K \setminus \{0, 1\}$$

If $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are two point on E_E, then $P + Q = (x_3, y_3)$ is given by

$$x_3 = \frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2} \quad \text{and} \quad y_3 = \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2}$$

The formulae for doubling and tripling are as follows:

$$\begin{aligned}
\text{If } 2P = (x_3, y_3), \quad \text{then} \quad x_3 &= \frac{2x_1y_1}{x_1^2 + y_1^2} \\
\text{and} \quad y_3 &= \frac{y_1^2 - x_1^2}{2 - (x_1^2 + y_1^2)} \\
\text{If } 3P = (x_4, y_4), \quad \text{then} \quad x_4 &= \frac{(x_1^2 + y_1^2)^2 - (2y_1)^2}{4(x_1^2 - 1)x_1^2 - (x_1^2 - y_1^2)^2} \\
\text{and} \quad y_4 &= \frac{(x_1^2 + y_1^2)^2 - (2x_1)^2}{-4(y_1^2 - 1)y_1^2 + (x_1^2 - y_1^2)^2}
\end{aligned}$$

The group operation on an Edwards' curve(a plane quartic) is not defined using the chord-tangent construction given earlier in section 1.3. A proof of the group law on an Edwards curve is given in [12]. Not all elliptic curves can be written in Edwards form over K, as an Edwards curve always has points of order 4, see [12]. To avoid computing inverses, the equation for E_E could be homogenized and then transformed [12] to a curve of the form $(X^2+Y^2)Z^2 = c^2(Z^4+dX^2Y^2)$ such that $c, d \in K \setminus \{0, 1\}$ and $dc^4 \neq 1$. Using projective coordinates, on an Edwards curve,

$$\begin{aligned}
\text{Addition requires} \quad & (10M + 1S) \quad \text{operations} \\
\text{Doubling requires} \quad & (3M + 4S) \quad \text{operations} \\
\text{Tripling requires} \quad & (9M + 4S) \quad \text{operations}
\end{aligned}$$

Using an efficient (computationally) birational equivalence from the Edwards form to the Montgomery form [23], the operation count for doubling on a Montgomery form elliptic curve can be reduced from $(2M + 2S + 1M_c)$ to $(1M + 3S + 3M_c)$ when the curve parameter d is a square in K. While not all Montgomery curves can be written in Edwards form, every Montgomery curve can be written in a generalized form of Edwards curves called *twisted Edwards* curves (a curve of the form $ax^2 + y^2 = 1 + dx^2y^2$ over a field K where non zero elements $a, d \in K$ and $char(K) \neq 2$) [10].

2.4 Huff's Model

Another model of the elliptic curve introduced by Huff in 1948 has been adapted for use in cryptography [57]. An elliptic curve in Huff's form (defined over a field k) is given by

$$E_H : aX(Y^2 - X^2) = bY(X^2 - Z^2) \quad \text{where} \quad a^2 \neq b^2$$

If $P_1 = (X_1 : Y_1 : Z_1)$ and $P_2 = (X_2 : Y_2 : Z_2)$ are two points on E_H and $P_3 = (X_3 : Y_3 : Z_3) = P_1 + P_2$ then

$$\begin{aligned} X_3 &= (X_1Z_2 + X_2Z_1)(Y_1Y_2 + Z_1Z_2)^2(Z_1Z_2 - X_1X_2) \\ Y_3 &= (Y_1Z_2 + Y_2Z_1)(X_1X_2 + Z_1Z_2)^2(Z_1Z_2 - Y_1Y_2) \\ Z_3 &= (Z_1^2Z_2^2 - X_1^2X_2^2)(Z_1^2Z_2^2 - Y_1^2Y_2^2) \end{aligned}$$

The above computation of addition on an Huff's elliptic curve can be achieved by an algorithm costing $12M$. The same formulae can be used for doubling as well and can be performed in $(7M + 5S)$. When $S > 0.75M$, a dedicated doubling algorithm costing $(10M + 1S)$ was presented in [57].

2.5 Selmer Curves

In [110], the authors consider a new model of an elliptic curve called *Selmer Curves* that was so named by Ian Connell [30]. The authors in [110] also provide explicit point addition and doubling formulae for Selmer curves.

Following [30], we provide the following definition.

A Selmer curve over K is defined by an equation of the form

$$ax^3 + by^3 = c$$

where $a, b, c \in K$ and $abc \neq 0$. Along with point arithmetic formulae, we will further look at Selmer curves in the context of pairings in Chapter 6.

The costs for point addition and doubling for various coordinate systems/curve forms reviewed in this chapter until now can be summarized as below:

Table 2.1: Point Addition and Doubling Summary

Coordinate system/ Elliptic curve form	Addition	Doubling
Affine	$1I + 2M + 1S$	$1I + 2M + 2S$
Projective	$12M + 2S$	$7M + 5S$
Jacobian	$12M + 4S$	$4M + 6S$
Chudnovsky	$11M + 3S$	$5M + 6S$
Modified Jacobian	$13M + 6S$	$4M + 4S$
Edwards Curve	$10M + 1S$	$3M + 4S$
Huff's Curve	$12M$	$7M + 5S$

2.6 Scalar Multiplication

One of the most important aspects of elliptic curve cryptography is that of Scalar multiplication. This is the quantity nP, where n is an integer and P is a point on an Elliptic curve defined over a finite field. The quantity nP is given by

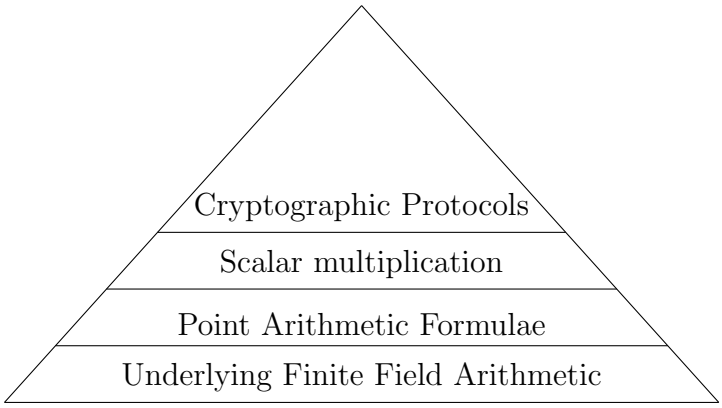
$$[n]p = \underbrace{P + P + \cdots + P}_{\text{(n-1) point additions}}$$

and consumes a significant amount of time in typical elliptic curve cryptographic schemes and thus has received attention in terms of

implementing elliptic curve cryptosystems. Efficient implementation of elliptic curve cryptography is dependent on the following:

(i) scalar multiplication technique
(ii) point arithmetic formulae and
(iii) field arithmetic.

This hierarchy can be visualized as below:



The techniques used in different layers of the hierarchy depicted above are not independent of each other. In this chapter, we have looked at point arithmetic formulae. The scalar multiplication technique used in a particular implementation can influence the choice of point arithmetic formuale used. When the scalar n is fixed, the scalar multiplication technique focuses on an *addition chain* (see Section 4.1) for n. When the point P is fixed, Yao's method [109] can be used, where precomputations are of primary importance. In a more generic setting, the well known Double-and-Add algorithm which is analogous to the square-and-multiply method can be used to compute nP using l doublings and about m additions where l is the bit length of n and m is the hamming weight of n. The Double-and-Add algorithm produces an addition chain for n. The Double-and-Add algorithm is also known as the binary method in the literature and can be generalized to an *m-ary* method. The 2^s-ary method breaks up the binary

representation of n into window lengths of s and a *sliding window* algorithm that makes use of suitable precomputed values can be utilized to compute nP.

One of the techniques of computing a scalar multiplication when n is fixed is to write n in the Double Base Number System format as depicted below.

2.6.1 Double Base Number System

The Double Base Number System (DBNS) introduced initially by Dimitrov and Cooklev in [40] was utilized later in the context of elliptic curves in [41]. With this system, the scalar n is written as

$$n = \sum_{i=1}^{l} s_i 2^{a_i} 3^{b_i} \qquad \text{or} \qquad n = \sum_{i=1}^{l} s_i 2^{a_i} 5^{b_i} \qquad \text{where } s_i = \pm 1 \ .$$

The above idea can be generalized to a triple base number system where an integer n is represented as

$$n = \sum_{i=1}^{l} s_i 2^{a_i} 3^{b_i} 5^{c_i} \text{ where } s_i = \pm 1.$$

Double and Triple base number system representations, though very short, are not immediately suitable for use in scalar multiplication algorithms. However, if we could somehow ensure that the three exponents are all simultaneously decreasing, i.e., $a_1 \geq a_2 \geq \ldots a_l$ and $b_1 \geq b_2 \geq \ldots b_l$ and $c_1 \geq c_2 \geq \ldots c_l$, then using Horner's rule, a scalar multiplication algorithm can be developed to compute nP. The simultaneously decreasing exponents can be computed using *greedy algorithms*. An example of such a greedy algorithm was provided by Mishra and Dimitrov in [79, Algorithm 1].

From the double, triple and quintuple base representations of a scalar k with simultaneously decreasing exponents as depicted above, it is clear that in addition to fast point addition and doubling, fast point tripling and quintupling algorithms are highly desirable, as this would speed up the

computation of nP when the DBNS is employed. Thus, there has been a keen interest in obtaining faster point tripling and quintupling algorithms amongst researchers.

2.6.2 Quintupling Formulae for Weierstrass Curves

In [79] the authors provide a fast quintupling algorithm for Affine coordinates on binary Weierstrass curves. In the same paper, the authors also propose a fast quintupling algorithm in Affine and Projective Jacobian coordinates for Weierstrass curves over $\mathbb{F}_p$. In Jacobian coordinates, the cost of the quintupling algorithm provided in [79] is $(15M + 10S)$. Another algorithm costing $(7M + 16S)$ was provided by Giorgi et al in [49]. The authors in [79] take into account the multiplication by the curve parameter a while computing the cost of their algorithm, whereas the authors in [49] do not take into account the multiplication by the curve parameter a while computing the costs of their algorithm. Thus for comparison purposes we can take the cost of the algorithm in [49] to be $(8M + 16S)$. In [74], Longa and Miri provide a quintupling algorithm (Jacobian coordinates, Weierstrass curve over $\mathbb{F}_p$) with costs equal to $(10M + 14S)$. When the curve parameter $a = -3$, Mishra's algorithm in [79] costs $(15M + 8S)$, Giorgi's algorithm in [49] costs $(7M + 16S)$ and Longa's algorithm in [74] costs $(11M + 11S)$. Thus while Longa's algorithm in [74] performs better than Mishra's algorithm in [79], Giorgi's algorithm in [49] is the best option.

Giorgi's $8M + 16S$ point quintupling algorithm in [49] was derived using an automaton implementing a directed acyclic graph structure looking for common subexpressions in the formulae and executing several arithmetic transformations. However using the simple transformation

$$2XY = (X + Y)^2 - X^2 - Y^2 \tag{2.1}$$

we show that the complexity of Mishra's algorithm can be reduced to that of Giorgi's algorithm, as shown below.

Mishra and Dimitrov's Algorithm for Quintupling on Weierstrass Curves

We recall the equation for Weierstrass curve over a prime field K given by

$$H_w(K) : y^2 = x^3 + ax + b$$

where $a, b \in K$, $4a^3 + 27b^2 \neq 0$ and the point $P = (X : Y : Z)$ corresponds to the point $(X/Z^2, Y/Z^3)$ in Jacobian coordinates. Given that P is a point on H_w and if $5P = 5(X : Y : Z) = (X_5, Y_5, Z_5)$, Mishra and Dimitrov, using Division Polynomials, provide the following formulae in [79] to compute X_5, Y_5 and Z_5.

$$\begin{aligned} X_5 &= XV^2 - 2YUW, \\ Y_5 &= Y(E^3(12VL^2 - V^2 - 16L^4) - 64TL^5) \text{ and} \\ Z_5 &= ZV \end{aligned}$$

where

$$\begin{aligned} T &= 8Y^4; && (\text{Cost} = 2S), \\ M &= 3X^2 + aZ^4; && (\text{Cost} = 3S + 1M), \\ E &= 12XY^2 - M^2; && (\text{Cost} = 1S + 1M), \\ 2L &= 2ME - 2T; && (\text{Cost} = 1M), \\ U &= 4YL; && (\text{Cost} = 1M), \\ V &= 4TL - E^3; && (\text{Cost} = 1S + 2M), \\ N &= V - 4L^2; && (\text{Cost} = 1S), \\ 2W &= 2EN; && (\text{Cost} = 1M), \\ X_5 &= 4(X.V^2 - 2Y.U.W); && (\text{Cost} = 3M + 1S), \\ Y_5 &= 8Y.[E^3.(12V.L^2 - V^2 \\ &\qquad - 16(L^2)^2) - 64(TL.(L^2)^2)]; && (\text{Cost} = 4M + 1S) \\ \text{and } Z_5 &= 2Z.V; && (\text{Cost} = 1M). \end{aligned}$$

Thus the total cost of computing the Quintupling formulae is $(15M + 10S)$. Now using equation(2.1), $12XY^2$, $4YL$, $12V.L^2$, ME and ZV can be

computed as

$$\begin{aligned}
12X.Y^2 &= 6[(X+Y^2)^2 - X^2 - Y^4]; \text{ (Cost} = 1S \text{ traded for } 1M), \\
4YL &= 2[(Y+L)^2 - Y^2 - L^2]; \quad \text{(Cost} = 1S \text{ traded for } 1M), \\
12V.L^2 &= 6[(V+L^2) - V^2 - L^4]; \quad \text{(Cost} = 1S \text{ traded for } 1M), \\
2ME &= [(M+E)^2 - M^2 - E^2]; \quad \text{(Cost} = 1S \text{ traded for } 1M) \text{ and} \\
2ZV &= [(Z+V)^2 - Z^2 - V^2]; \quad \text{(Cost} = 1S \text{ traded for } 1M) \ .
\end{aligned}$$

Thus the cost of the Mishra and Dimitrov Quintupling algorithm can be reduced from $15M + 10S$ to $10M + 15S$. It turns out that one multiplication can further be eliminated from the Mishra and Dimitrov Algorithm. Indeed, in the computation of X_5, YUW is computed where $U = 4YL$ and thus $YUW = 4Y^2LW$. Thus we could alter U to be equal to $4Y^2L$ instead of $4YL$. Now, we could write $4Y^2L$ as

$$\begin{aligned}
U = 4Y^2L &= 2[(Y^2+L)^2 - Y^4 - L^2]; \text{ and thus} \\
X_5 &= 4(XV^2 - 2UW); \text{ (New Cost} = 2M + 1S) \ .
\end{aligned}$$

Thus the cost of the modified Mishra and Dimitrov Algorithm can be reduced to $9M + 15S$ which is just slightly better than the $10M + 14S$ cost of the Longa and Miri Quintupling Algorithm. Further we could compute N^2 as

$$N^2 = (V - 4L^2)^2 = V^2 + 16L^4 - 8VL^2 \ .$$

Now N^2 could be computed without using any extra squarings or multiplications, as V^2, L^4 and VL^2 are computed for other steps in the algorithm, as shown above. Thus $2W = 2EN$ could be computed as

$$2W = [(E+N)^2 - E^2 - N^2] \ .$$

Now, E^2 is also computed in another step in the algorithm, thus effectively replacing $1M$ with a $1S$. Thus, we have reduced the cost of the modified Mishra and Dimitrov algorithm to $8M + 16S$ using equation (2.1).

The costs for point quintupling on Jacobian coordinates can be summarized as below:

Table 2.2: Jacobian Coordinates Quintupling

Algorithm	a need not be equal to -3	$a = -3$
Mishra and Dimitrov [79]	$15M + 10S$	$15M + 8S$
Giorgi [49]	$8M + 16S$	$7M + 16S$
Longa and Miri [74]	$10M + 14S$	$11M + 11S$
Modified Mishra and Dimitrov [98]	$8M + 16S$	

2.6.3 Quintupling Formulae for Edwards Curves Revisited

In [11], Bernstein et al, amongst other things, provide two fast algorithms for point quintupling on Edwards curves defined over $\mathbb{F}_p$. As in Section 2.3, an Edwards curve E_d defined over a field K is given by $x^2 + y^2 = 1 + dx^2y^2$ where $d \in K\backslash\{0, 1\}$.

The two quintupling algorithms provided in [11], (we call them Algorithm A and Algorithm B for convenience) cost $(17M + 7S)$ and $(14M + 11S)$ respectively. The authors in [11] conclude that Algorithm A performs better if the S/M ratio i.e., $S/M > 0.75$ while Algorithm B performs better if $S/M < 0.75$. When $S/M = 0.75$, both algorithms share the same complexity. Here we modify Algorithm B slightly to provide an alternate algorithm (Algorithm C). If the affine point $(X_1/Z_1, Y_1/Z_1)$ represents the point (X_1, Y_1, Z_1) on the homogenized equation of E_d, and if $(X_5, Y_5, Z_5) = 5(X_1, Y_1, Z_1)$, the new quintupling algorithm (Algorithm C) is as below:

$A = X_1^2$; $B = Y_1^2$; $C = Z_1^2$; $D = A + B$; $E = 2C - D$; $F = A^2$; $G = B^2$; $H = F + G$; $I = D^2 - H$; $J = E^2$; $K = G - F$; $L = K^2$; $M = 2I.J$; $N = L + M$; $O = L - M$; $P = N.O$; $Q = (E + K)^2 - J - L$; $R = 2(2JH - L)$; $S = Q.R$; $T = 4Q.O.(D - C)$; $U = R.N$; $V = U + T$;

$W = U - T;\ X_5 = 2X_1.(P + B.S).W;\ Y_5 = 2Y_1.(P - A.S).V;\ Z_5 = Z_1.V.W$

Algorithm A and Algorithm B were verified by the authors in [11]. The only difference between Algorithm C above and Algorithm B in [11] is in the computation of R. In [11], R was computed as

$$R = ((D + E)^2 - J - H - I)^2 - 2N$$

In Algorithm C we employ $R = 2(2JH - L)$, as we can rewrite R as follows:

$$\begin{aligned}
R &= ((D + E)^2 - J - H - I)^2 - 2N \\
&= \Bigg\{ \big[(X_1^2 + Y_1^2) + (2Z_1^2 - (X_1^2 + Y_1^2)\big]^2 \\
&\qquad - \big[2Z_1^2 - (X_1^2 + Y_1^2)\big]^2 - \big[X_1^4 + Y_1^4\big] - 2X_1^2Y_1^2 \Bigg\}^2 - 2N \\
&= \big[2(X_1^2 + Y_1^2)(2Z_1^2 - (X_1^2 + Y_1^2))\big]^2 \\
&\qquad - 2\big[(Y_1^4 - X_1^4)^2 + 4(X_1^2Y_1^2)\{2Z_1^2 - (X_1^2 + Y_1^2)\}^2\big] \\
&= 4\big[2Z_1^2 - (X_1^2 + Y_1^2)\big]^2\big\{X_1^4 + Y_1^4\big\} - 2\big[(Y_1^4 - X_1^4)^2\big] \\
&= 4JH - 2L \\
&= 2(2JH - L)
\end{aligned}$$

Algorithm C above for quintupling costs $(15M + 9S)$ and is better than both the quintupling algorithms provided in [11] as long as $M > S$ and $M < 2S$ and irrespective of whether $S/M < 0.75$ or $S/M > 0.75$.

Table 2.3: Edwards Curve Quintupling Formulae Summary

Algorithm	Quintupling Costs
Algorithm A [11]	$17M + 7S$
Algorithm B [11]	$14M + 11S$
Algorithm C [98]	$15M + 9S$

2.7 Side Channel Attacks

Side channel attacks are a group of techniques using which an adversary can obtain the key (or a part of the key) by using information related to power consumption, time taken for a cryptographic operation, electromagnetic radiation etc [18, 31, 48, 85]. For example, if kP were computed where k is a scalar and also a secret with P being an Elliptic curve point, and if the double-and-add algorithm were to be used for scalar multiplication, then it is possible to deduce information about the key from a power/timing analysis corresponding to the computation of kP as there is a difference between the point doubling and point addition operations. A countermeasure to this type of attack is the double-and-always-add algorithm of Coron [31]. Another counter measure is the Montgomery Ladder which ensures that there is no relation between the Hamming weight of the secret k and the execution time of the algorithm. Implementing an *Atomic Block* is one way to overcome side channel attacks. Atomic blocks consist of structuring various point arithmetic operations using a homogeneous sequence of operations. An introduction to atomic blocks can be found in [28, Chapter 29].

From the previous sections of this chapter, it may appear that we should always look for algorithms with the least operation count cost. However there are circumstances where the most cost-effective option is not always the best option in every situation, especially so when it comes to overcoming side channel attacks. For example, in [1], Abarzua and Theriault, while designing side-channel resistant atomic blocks for Weierstrass elliptic curves in Jacobian Coordinates over prime fields, use a $(9M + 7S)$ point tripling formulae due to Dimitirov, Imbert and Mishra [41], as the more economical $(7M + 7S)$ tripling algorithm due to Longa and Miri [73] could not fit nicely into their atomic block pattern. Thus, there may be a trade-off between efficiency and security, as the most efficient algorithm may not lend itself to side-channel resistant methods.

Chapter 3

Differential Arithmetic on Elliptic Curves

In Chapter 2 of this thesis, we introduced $x-coordinate$ only formulae for Montgomery curves with the points in affine representation. This $x-coordinate$ only arithmetic is usually known as *differential arithmetic*. In some cases, $y-coordinate$ only arithmetic or $w-coordinate$ arithmetic are used and such formulae are also known as differential arithmetic formuale. We first provide Projective coordinate formulae for Montgomery curves and then provide $x-coordinate$ only tripling formulae for Montgomery curves. We then speed up differential addition formulae for Generalised Edwards coordinates and Binary Edwards curves.

3.1 Introduction to Differential Arithmetic

The projective form of the Montgomery curve defined over a finite field $\mathbb{F}_p$ can be written as

$$E_m : BY^2Z = X^3 + AX^2Z + XZ^2 \text{ where } A, B \in \mathbb{F}_p \text{ and } A \neq \pm 2, B \neq 0$$

Let $P = (X_1, Y_1, Z_1)$ and $nP = (X_n, Y_n, Z_n)$. We know that the set of points on E_m form an Abelian group and the identity element (0,1,0) in this group is denoted as $\mathcal{O}$. The sum $(n+m)P = nP + mP$ can be computed using the

formulae below:
Addition: $(n \neq m)$:

$$X_{m+n} = Z_{m-n}((X_m - Z_m)(X_n + Z_n) + (X_m + Z_m)(X_n - Z_n))^2$$
$$Z_{m+n} = X_{m-n}((X_m - Z_m)(X_n + Z_n) - (X_m + Z_m)(X_n - Z_n))^2$$

Doubling: $(n = m)$:

$$\begin{aligned} X_{2n} &= (X_n + Z_n)^2(X_n - Z_n)^2 \\ 4X_nZ_n &= (X_n + Z_n)^2 - (X_n - Z_n)^2 \\ Z_{2n} &= 4X_nZ_n((X_n - Z_n)^2 + ((A+2)/4)(4X_nZ_n)) \\ &= 4X_nZ_n((X_n + Z_n)^2 + ((A-2)/4)(4X_nZ_n)) \end{aligned}$$

Point addition requires $4M + 2S$ operations and a point doubling requires $3M + 2S$ operations. If $Z_{m-n} = 1$, then point addition requires $3M + 2S$ operations. The well known Montgomery ladder that can be used to compute nP is as below:

Algorithm 3.1: Left-to-Right Montgomery ladder for scalar multiplication

---

INPUT: A point P on E_m and a positive integer $n = (n_t \dots n_0)_2$ with $n_t = 1$
OUTPUT: The point nP

$P_1 \leftarrow P$ and $P_2 \leftarrow 2P$
for $i = t - 1$ down to 0 do
if $n_i = 0$ then
  $P_2 \leftarrow P_2 + P_1$ (P); $P_1 \leftarrow 2P_1$
else
  $P_1 \leftarrow P_2 + P_1$ (P); $P_2 \leftarrow 2P_2$
end if
end for
return P_1

In all algorithms in this thesis, whenever the difference between two points is required to compute the sum of those points, the difference is indicated in brackets immediately after the addition formula. In Algorithm 3.1, we have

$$P_1 \leftarrow P_2 + P_1 \ (P) \tag{3.1}$$

The notation in equation (3.1) means that when P_2 is added to P_1 and the result is stored in P_1 the difference required between these two points is P i.e., $P_2 - P_1 = P$. Clearly, the above algorithm is a Left-to-Right algorithm. If $(n_t \dots n_1 n_0)_2$ is the binary representation of n and $(n_t = 1)$, to compute $[n]P$, we proceed as follows: we hold $\{m_i P, (m_i + 1)P\}$ for $m_i = (n_t \dots n_i)_2$. If $n_i = 0, m_i P = 2m_{i+1}P$ and $(m_i + 1)P = (m_{i+1} + 1)P + m_{i+1}P$ else $m_i P = (m_{i+1} + 1)P + m_{i+1}P$ and $(m_i + 1)P = 2(m_{i+1} + 1)P$. Beginning from $\{P, 2P\}$, Algorithm 3.1 computes $\{nP, (n + 1)P\}$.

Until now, in this chapter, we have had a look at x-coordinate only point addition and doubling formulae. We proceed to look at x-coordinate only tripling formulae for Montgomery curves.

3.2 Differential Tripling Formulae for Montgomery Curves

Let $P_1 = (X_1, Y_1, Z_1)$, $P_2 = (X_2, Y_2, Z_2)$ and $P_3 = (X_3, Y_3, Z_3)$ be points on a Montgomery curve E_m with $P_2 = 2P_1$ and $P_3 = 3P_1$. Then the tripling can be computed as

$$\begin{aligned} X_3 &= X_1\big(\big(X_1^2 - Z_1^2\big)^2 - \big(X_1^2 + Z_1^2 + AX_1Z_1\big)\,(2Z_1)^2\big)^2 \text{ and} \\ Z_3 &= Z_1\big(\big(X_1^2 - Z_1^2\big)^2 - \big(X_1^2 + Z_1^2 + AX_1Z_1\big)\,(2X_1)^2\big)^2 \ . \end{aligned}$$

The above tripling formulae require $6M + 5S$ operations. As the Tripling formulae is *x-coordinate* only formulae, we use the term *Differential Tripling* to distinguish the new formulae from usual tripling formulae where both x and y coordinates are computed.

We derive the above differential point tripling formulae for Montgomery curves. Let $P_1 = (X_1, Y_1, Z_1)$, $P_2 = (X_2, Y_2, Z_2)$ and $P_3 = (X_3, Y_3, Z_3)$ be points on a Montgomery curve E_m with $P_2 = 2P_1$ and$P_3 = 3P_1$. We can write $P_3 = 3P_1 = 2P_1 + P_1 = P_2 + P_1$. Then

$$X_2 = (X_1 + Z_1)^2(X_1 - Z_1)^2 \tag{3.2}$$
$$Z_2 = 4X_1Z_1((X_1 - Z_1)^2 + ((A + 2)/4)(4X_1Z_1)) \tag{3.3}$$
$$X_3 = Z_1[(X_1 - Z_1)(X_2 + Z_2) + (X_1 + Z_1)(X_2 - Z_2)]^2 \tag{3.4}$$
$$Z_3 = X_1[(X_1 - Z_1)(X_2 + Z_2) - (X_1 + Z_1)(X_2 - Z_2)]^2 \tag{3.5}$$

From equations (3.2), (3.3), (3.4) and (3.5) we can write

$$\begin{aligned}
X_3 =& Z_1\Big[(X_1 - Z_1) \\
&\left\{(X_1 + Z_1)^2 (X_1 - Z_1)^2 + 4X_1Z_1 (X_1 - Z_1)^2 + ((A + 2)/4) (4X_1Z_1)^2\right\} \\
&+ (X_1 + Z_1) \\
&\left\{(X_1 + Z_1)^2 (X_1 - Z_1)^2 - 4X_1Z_1 (X_1 - Z_1)^2 - ((A + 2)/4) (4X_1Z_1)^2\right\}\Big]^2 \\
=& Z_1\Big[\big((X_1 + Z_1)(X_1 - Z_1)\big)^2\{2X_1\} - 4X_1Z_1 (X_1 - Z_1)^2 \{2Z_1\} \\
&\qquad - \big((A + 2)/4\big)(4X_1Z_1)^2\{2Z_1\}\Big]^2 \\
=& 4X_1^2Z_1\Big[\big((X_1 + Z_1)(X_1 - Z_1)\big)^2 - 4Z_1^2 (X_1 - Z_1)^2 \\
&\qquad - \big((A + 2)/4\big)\left(16X_1Z_1^3\right)\Big]^2 \\
=& 4X_1^2Z_1\big(\left(X_1^2 - Z_1^2\right)^2 - \left(X_1^2 + Z_1^2 + AX_1Z_1\right)(2Z_1)^2\big)^2
\end{aligned}$$

Similarly,

$$\begin{aligned}
Z_3 =& X_1\Big[(X_1 - Z_1) \\
&\big\{(X_1 + Z_1)^2 (X_1 - Z_1)^2 + 4X_1Z_1 (X_1 - Z_1)^2 + ((A+2)/4)(4X_1Z_1)^2\big\} \\
&- (X_1 + Z_1) \\
&\big\{(X_1 + Z_1)^2 (X_1 - Z_1)^2 - 4X_1Z_1 (X_1 - Z_1)^2 - ((A+2)/4)(4X_1Z_1)^2\big\}\Big]^2 \\
=& X_1\big[\big((X_1 + Z_1)(X_1 - Z_1)\big)^2\{-2Z_1\} - 4X_1Z_1 (X_1 - Z_1)^2 \{2X_1\} \\
&\qquad - \big((A+2)/4\big)(4X_1Z_1)^2\{2X_1\}\big]^2 \\
=& 4X_1Z_1^2\big[-\big((X_1 + Z_1)(X_1 - Z_1)\big)^2 + 4X_1^2 (X_1 - Z_1)^2 \\
&\qquad + \big((A+2)/4\big)\big(16X_1^3Z_1\big)\big]^2 \\
=& 4X_1Z_1^2\big(-\big(X_1^2 - Z_1^2\big)^2 + \big(X_1^2 + Z_1^2 + AX_1Z_1\big)(2X_1)^2\big)^2 \\
=& 4X_1Z_1^2\big(\big(X_1^2 - Z_1^2\big)^2 - \big(X_1^2 + Z_1^2 + AX_1Z_1\big)(2X_1)^2\big)^2
\end{aligned}$$

Dividing both X_3 and Z_3 by $4X_1Z_1$ we get, when $(X_1, Y_1) \neq (0, 0)$

$$\begin{aligned}
X_3 &= X_1\big(\big(X_1^2 - Z_1^2\big)^2 - \big(X_1^2 + Z_1^2 + AX_1Z_1\big)(2Z_1)^2\big)^2 \\
Z_3 &= Z_1\big(\big(X_1^2 - Z_1^2\big)^2 - \big(X_1^2 + Z_1^2 + AX_1Z_1\big)(2X_1)^2\big)^2
\end{aligned}$$

The formulae for X_3 and Y_3 derived above can be computed using the following $6M + 5S$ algorithm:

$$
\begin{aligned}
&T_1 \leftarrow X_1;\ T_2 \leftarrow Z_1 && \\
&T_1 \leftarrow T_1^2 && (= X_1^2) \\
&T_2 \leftarrow T_2^2 && (= Z_1^2) \\
&T_3 \leftarrow (T_1 - T_2)^2 && (= (X_1^2 - Z_1^2)^2) \\
&T_4 \leftarrow X_1 Z_1 && (= X_1 Z_1) \\
&T_4 \leftarrow A.T_4 && (= AX_1 Z_1) \\
&T_5 \leftarrow T_2 + T_2 + T_2 + T_2 && (= 4Z_1^2) \\
&T_6 \leftarrow T_1 + T_1 + T_1 + T_1 && (= 4X_1^2) \\
&T_4 \leftarrow T_1 + T_2 + T_4 && (= X_1^2 + Z_1^2 + AX_1 Z_1) \\
&T_7 \leftarrow T_4.T_5 && (= (X_1^2 + Z_1^2 + AX_1 Z_1)(4Z_1^2)) \\
&T_8 \leftarrow T_4.T_6 && (= (X_1^2 + Z_1^2 + AX_1 Z_1)(4X_1^2)) \\
&T_1 \leftarrow (T_3 - T_7)^2 && \left(= \left(\left(X_1^2 - Z_1^2\right)^2 - \left(X_1^2 + Z_1^2 + AX_1 Z_1\right)(2Z_1)^2\right)^2\right) \\
&T_2 \leftarrow (T_3 - T_8)^2 && \left(= \left(\left(X_1^2 - Z_1^2\right)^2 - \left(X_1^2 + Z_1^2 + AX_1 Z_1\right)(2X_1)^2\right)^2\right) \\
&X_3 \leftarrow X_1.T_1 && \left(= X_1\left(\left(X_1^2 - Z_1^2\right)^2 - \left(X_1^2 + Z_1^2 + AX_1 Z_1\right)(2Z_1)^2\right)^2\right) \\
&Z_3 \leftarrow Z_1.T_2 && \left(= Z_1\left(\left(X_1^2 - Z_1^2\right)^2 - \left(X_1^2 + Z_1^2 + AX_1 Z_1\right)(2X_1)^2\right)^2\right)
\end{aligned}
$$

If P_3 is computed as $2P_1 + P_1$ (i.e.,using a point doubling followed by a point addition), we need $(4M + 2S) + (3M + 2S) = 7M + 4S$, while the number of field additions/subtractions required are the same in both the cases (using either the differential tripling formula or using $P_3 = 2P_1 + P_1$). Thus the above tripling formulae are efficient as $(6M + 5S) < (7M + 4S)$. When $Z_1 = 1$ the above tripling formulae only needs $(3M + 4S)$ operations whereas, if $3P$ were to be computed as $2P + P$, (that is, a doubling followed by an addition with $Z_1 = 1$), then $(3M + 2S) + (3M + 2S) = (6M + 4S)$ operations would be required thus resulting in a saving of $3M$. The differential tripling formulae cannot be readily used in the binary Montgomery Ladder. However in the next chapter, we describe situations in which the tripling formuale can be utilized.

3.3 Differential Arithmetic Generalized

The point addition and doubling formulae for Montgomery curves provided in Section 3.1 were the first differential addition formulae published in the literature. The idea of differential addition has since been extended to other forms of elliptic curves such as

- Lopez and Dahab's extension [75] to Weierstrass curves over $\mathbb{F}_{2m}$.
- Two independently developed extensions to Weierstrass curves over $\mathbb{F}_p$
  - one due to Fisher, Giraud, Knudsen and Seifert[48]
  and
  - the other due to Brier and Joye [18].
- Bernstein, Lange and Farashahi's [13] extension to Binary Edwards curves.
- Justus and Loebenberger's extension [58] to Generalized Edwards curves over $\mathbb{F}_q$ ($\text{char}(\mathbb{F}_q) \neq 2$).
- Farashahi and Joye's extension [47] to Generalized Binary Hessian curves.
- Devigne and Joye's extension [38] to Binary Huff curves.
- Hutter, Joye and Sierra's [52] extension to Weierstrass curves over $\mathbb{F}_p$ in homogeneous projective coordinates where the points to be added share the same Z-coordinate.
- Wu, Tang and Feng's [108] extension to a new model of Binary Edwards curves.

The Montgomery ladder for scalar multiplication that was initially proposed and utilized for Montgomery curves can be adapted to the examples listed above. Differential addition formulae have been published for the examples above. For instance, [28, Section 13.3.4] provides the differential addition formulae for Binary Weistrass' curves. However, the addition formula is

correct only when $Z_{m-n} = 1$. The text does not mention this. If $Z_{m-n} \neq 1$, then the formulae in [28, Section 13.3.4] should read as follows:

$$Z_{m+n} = Z_{m-n}\big[X_m Z_n + X_n Z_m\big]^2$$
$$X_{m+n} = X_{m-n}\big[X_m X_n + X_n Z_m\big]^2 + Z_{m-n}(X_m Z_n)(X_n Z_m)$$

Thus addition should take $6M + 1S$ operations; When $Z_{m-n} = 1$, this would reduce to $4M + 1S$.

3.3.1 Differential Arithmetic on Generalized Edwards' Curves revisited

In Section 3.3.2, we try to speed up some of the formulae proposed in [58] and the affine w-Coordinate differential addition proposed in [108]. In the remainder of this section, we review some of the differential addition formulae for Generalized Edwards curves as provided in [58] and the affine w-Coordinate differential addition formulae for a new model of Binary elliptic curve as provided in [108].

Generalized Edwards curves over a finite field F_p are given by (curve parameters $c, d \in F_p$)

$$E_{c,d} : x^2 + y^2 = c^2(1 + dx^2y^2)$$

It turns out that the differential addition formulae for generalized Edwards curves use y-only coordinates instead of x-only coordinates for Montgomery curves. Let $P = (x_1, y_1)$ be a point on $E_{c,d}$. In projective coordinates, P can be written as $P = (X_1, Y_1, Z_1)$ we write $nP = (X_n : Y_n : Z_n)$. If $c, d \neq 0$, $dc^4 \neq 1$ and d is not a square in $GF(p)$, the sum $(n + m)P = nP + mP$, as provided in [58], is reproduced below:

A. Differential Addition for Generalised Edwards Coordinates:
$(m > n)$
(Operation count given by authors in [58] is $6M + 4S$).

$$Y_{m+n} = Z_{m-n}(Y_m^2(Z_n^2 - c^2 d Y_n^2) + Z_m^2(Y_n^2 - c^2 Z_n^2))$$
$$Z_{m+n} = Y_{m-n}(dY_m^2(Y_n^2 - c^2 Z_n^2) + Z_m^2(Z_n^2 - c^2 d Y_n^2))$$

B. Differential Doubling for Generalised Edwards Coordinates:
$(n = m)$
(Operation count given by authors in [58] is $1M + 4S$).

$$Y_{2n} = -c^2 d Y_n^4 + 2Y_n^2 Z_n^2 - c^2 Z_n^4$$
$$Z_{2n} = dY_n^4 - 2c^2 d Y_n^2 Z_n^2 + Z_n^4$$

In [58], point tripling formula are provided as well. We reproduce them below:

C. Tripling for Generalised Edwards Coordinates:
(Operation count given by authors in [58] is $4M + 7S$).

$$\begin{aligned} Y_{3n} = Y_n(c^2(3Z_n^4 - dY_n^4)^2 - \\ Z_n^4(8c^2 Z_n^4 + (Y_n^2(c^3 d + c^{-1}) - 2cZ_n^2)^2 - \\ c^{-2}(c^4 d + 1)^2 Y_n^4)) \\ Z_{3n} = Z_n(c^2(Z_n^4 - 3dY_n^4)^2 + \\ dY_n^4(4c^2 Z_n^4 - (Y_n^2(c^3 d + c^{-1}) - 2cZ_n^2)^2 + \\ c^{-2}((c^4 d + 1)^2 - 12c^4 d)^2 Y_n^4)) \end{aligned}$$

In [58], an alternate parameterization is provided by the authors, where only the squares of the points $(Y_m : Z_m)$, $(Y_n : Z_n)$ and $(Y_{m-n} : Z_{m-n})$ are utilized. We call this *Squares Only* or SQO parametrization. The authors provide addition, doubling and tripling formulae for this parametrization. Here we reproduce the doubling and the tripling formulae from [58] for SQO parametrization.

D. SQO Doubling for Generalised Edwards Coordinates: $n = m$
(Operation count given in [58] is $5S$).

$$Y_{2n}^2 = ((1 - c^2 d)Y_n^4 + (1 - c^2)Z_n^4 - (Y_n^2 - Z_n^2)^2)^2$$
$$Z_{2n}^2 = (dc^2(Y_n^2 - Z_n^2)^2 - d(c^2 - 1)Y_n^4 + (c^2 d - 1)Z_n^4)^2$$

E. SQO Tripling for Generalised Edwards Coordinates: $m = 2n$
(Operation count in [58] is $4M + 7S$).

$$\begin{aligned} Y_{3n}^2 = Y_n^2(c^2(3Z_n^4 - dY_n^4)^2 - \\ Z_n^4(8c^2 Z_n^4 + (Y_n^2(c^3 d + c^{-1}) - 2cZ_n^2)^2 - \\ c^{-2}(c^4 d + 1)^2 Y_n^4))^2 \end{aligned}$$
$$\begin{aligned} Z_{3n}^2 = Z_n^2(c^2(Z_n^4 - 3dY_n^4)^2 + \\ dY_n^4(4c^2 Z_n^4 - (Y_n^2(c^3 d + c^{-1}) - 2cZ_n^2)^2 + \\ c^{-2}((c^4 d + 1)^2 - 12c^4 d)Y_n^4))^2 \end{aligned}$$

In [108], the authors propose a new model of Binary Edwards curve given by

$$S_t : x^2 y + xy^2 + txy + x + y = 0$$

where $(x, y) \in K^2$ and K is a field of char 2. Further, in section 6 of their paper, the authors construct differential addition formula for S_t. We reproduce the approach and the formulae here.

F. Affine w-coordinate Differential Addition and Doubling for a new model of Binary Edwards Curves proposed in [108]:
(Operation count for addition and doubling as given in [108] is $1I + 2M + 2S + 1M_c$ and $1I + 1M + 2S + 1M_c$ respectively.)

Utilizing the idea of w-coordinate differential addition that was initially proposed by the authors in [13] for Binary Edwards curves, the authors in [108] propose w-coordinate differential addition and doubling for the elliptic curve S_t, i.e., they present formulae to compute $w(P+Q)$ and $w(2P)$ from $w(P)$, $w(Q)$ and $w(Q-P)$. If $P = (x, y)$ is a point on S_t, then the w-function is defined as $w(P) = xy$. If $P = (x_2, y_2)$, $Q = (x_3, y_3)$, $Q - P = (x_1, y_1)$, $2P = (x_4, y_4)$ and $Q + P = (x_5, y_5)$, we write $w_i = x_i y_i$ for $i = 1, 2, 3, 4, 5$. Then $w_2 = w(P)$, $w_4 = w(2P)$, $w_5 = w(P + Q)$, $w_1 = w(Q - P)$ and $w_3 = w(Q)$. The affine differential addition formulae on S_t, as developed and presented in [108] are as follows:

$$w_4 = \frac{1 + w_2^4}{t^2 w_2^2} \text{ and } w_5 = w_1 + \frac{t^2 w_2 w_3}{w_2^2 + w_3^2}$$

The efficiency of arithmetic in characteristic 2 fields is significantly different compared to fields of odd characteristic. For instance, in binary fields, squaring is much faster than multiplication and has a complexity similar to that of modular addition [106].

3.3.2 Alternate Algorithms and Newer Operation Counts

In this section, recalling results from [97], we show that the operation counts in formulae (B-F) of Section 3.3.1 can be improved. For clarity in comparison, the subsections that describe and compare our improvements to (B-F) of Section 3.3.1 are labeled as (BB-FF) respectively.

BB. Differential Doubling for Generalised Edwards Coordinates:
The operation count of formula (B) in Section 3.3.1 is $1M + 4S$ as the formula

can be computed using the following algorithm:

$$
\begin{array}{lll}
A \leftarrow Y_n^2 & (=Y_n^2) & S \\
B \leftarrow Z_n^2 & (=Z_n^2) & S \\
D \leftarrow A * B & (=Y_n^2 Z_n^2) & M \\
A \leftarrow A^2 & (=Y_n^4) & S \\
B \leftarrow B^2 & (=Z_n^4) & S \\
Y_{2n} = -c^2 dA + & (=-c^2 dY_n^4 + & 2M_c \\
\quad 2D - c^2 B & \quad 2Y_n^2 Z_n^2 - c^2 Z_n^4) & \\
Z_{2n} = dA - 2c^2 dD + B & (=dY_n^4 - & 2M_c \\
 & \quad 2c^2 dY_n^2 Z_n^2 + Z_n^4) &
\end{array}
$$

Thus the total complexity, if one takes into consideration the cost M_c of multiplication by a constant other than 1 or 2 or 3, is $(1M + 4M_c + 4S)$. The formulae (B) in Section 3.3.1 can be rewritten as

$$
\begin{aligned}
Y_{2n} &= -c^2 dY_n^4 + 2Y_n^2 Z_n^2 - c^2 Z_n^4 = 2Y_n^2 Z_n^2 - c^2 (Z_n^4 + dY_n^4) \\
Z_{2n} &= dY_n^4 - 2c^2 dY_n^2 Z_n^2 + Z_n^4 = -c^2 d(2Y_n^2 Z_n^2) + (Z_n^4 + dY_n^4)
\end{aligned}
$$

The rewritten formulae above can be computed using the algorithm below:

$$
\begin{array}{lll}
A \leftarrow Y_n^2 & (=Y_n^2) & S \\
B \leftarrow Z_n^2 & (=Z_n^2) & S \\
E \leftarrow A^2 & (=Y_n^4) & S \\
F \leftarrow B^2 & (=Z_n^4) & S \\
G \leftarrow (A+B)^2 & (=2Y_n^2 Z_n^2) & S \\
\quad -E-F & & \\
Y_{2n} = G - & (=2Y_n^2 Z_n^2 - & 2M_c \\
\quad c^2(F+dE) & \quad c^2(Z_n^4 + dY_n^4)) & \\
Z_{2n} = (-c^2 d)G + & (=-c^2 d(2Y_n^2 Z_n^2) + & 1M_c \\
\quad (F+dE) & \quad (Z_n^4 + dY_n^4)) &
\end{array}
$$

Thus the new complexity is $5S + 3M_c$. As $1S < 1M$, the new complexity $5S + 3M_c$ is less than the older complexity $(1M + 4M_c + 4S)$

CC. Tripling for Generalised Edwards Coordinates:
The operation count of formula (C) in Section 3.3.1 is $4M+7S$. In addition to this, by inspection, one can count $8M_c$ operations are required to compute the requisite formula. Thus the total complexity of formula(C) is $4M + 7S + 8M_c$ From [58, Section 3.1], we have

$$y_3 = \frac{y(c^2d^2y^8 - 6c^2dy^4 + 4(c^4d + 1)y^2 - 3c^2)}{-3c^2d^2y^8 + 4d(c^4d + 1)y^6 - 6c^2dy^4 + c^2} .$$

Writing $y = Y/Z$ in projective coordinates, the above formula can be written as

$$\frac{Y_{3n}}{Z_{3n}} = \frac{Y_n}{Z_n} . \frac{(c^2d^2Y_n^8 - 6c^2dY_n^4Z_n^4 + 4(c^4d + 1)Y_n^2Z_n^6 - 3c^2Z_n^8)}{-3c^2d^2Y_n^8 + 4d(c^4d + 1)Y_n^6Z_n^2 - 6c^2dY_n^4Z_n^4 + c^2Z_n^8} .$$

Then

$$Y_{3n} = Y_n[c^2d^2Y_n^8 - 6c^2dY_n^4Z_n^4 + 4(c^4d + 1)Y_n^2Z_n^6 - 3c^2Z_n^8]$$

and

$$Z_{3n} = Z_n[-3c^2d^2Y_n^8 + 4d(c^4d + 1)Y_n^6Z_n^2 - 6c^2dY_n^4Z_n^4 + c^2Z_n^8]$$

The above rewritten formulae can now be computed using the algorithm

below:

$$
\begin{array}{lll}
A \leftarrow Y_n^2 & (=Y_n^2) & S \\
B \leftarrow Z_n^2 & (=Z_n^2) & S \\
E \leftarrow A^2 & (=Y_n^4) & S \\
F \leftarrow B^2 & (=Z_n^4) & S \\
G \leftarrow (A+B)^2 & (=(Y_n^2+Z_n^2)^2 & S \\
\quad -E-F & \quad -Y_n^4-Z_n^4=2Y_n^2Z_n^2) & \\
H \leftarrow G^2 & (=4Y_n^4Z_n^4) & S \\
J \leftarrow E^2 & (=Y_n^8) & S \\
K \leftarrow F^2 & (=Z_n^8) & S \\
M \leftarrow (G+F)^2 & [=(2Y_n^2Z_n^2+Z_n^4)^2-Z_n^8 & S \\
\quad -K-H & \quad -4Y_n^4Z_n^4]=4Y_n^2Z_n^6 & \\
N \leftarrow (G+E)^2 & [=(2Y_n^2Z_n^2+Y_n^4)^2 & S \\
\quad -J-H & \quad -Y_n^8-4Y_n^4Z_n^4]=4Y_n^6Z_n^2 &
\end{array}
$$

Finally,
$Y_{3n} \leftarrow Y_n[(c^2d^2)J-(\frac{3}{2}c^2d)H+(c^2d+1)M-(3c^2)K]$
which costs $1M+3M_c$ and
$Z_{3n} \leftarrow Z_n[(-3c^2d^2)J-d(c^4d+1)N-(\frac{3}{2}c^2d)H+(c^2)K]$
which costs $1M+2M_c$. In the above, once $(c^2)K$ is computed, the cost of computing $(3c^2)K$ is ignored. The complexity of the new algorithm is $(10S+2M+5M_c)$. If $3S < 2M+3M_c$, then the new complexity of $(10S+2M+5M_c)$ is less than the older complexity of $(7S+4M+8M_c)$. In [8], $2M=3S$ and thus $3S<2M+3M_c$.

DD. SQO Doubling for Generalised Edwards Coordinates:
By inspecting formula(D) in Section 3.3.1 and taking into consideration that we are provided with X_{2n}^2 and Y_{2n}^2, we can see that the total complexity of the formula(D) is $(5S+5M_c)$. We can improve upon this. Using the doubling

formula(BB) in this section, we can write

$$Y_{2n}^2 = \big[2Y_n^2 Z_n^2 - c^2(Z_n^4 + dY_n^4)\big]^2$$
$$Z_{2n}^2 = \big[-c^2 d(2Y_n^2 Z_n^2) + (Z_n^4 + dY_n^4)\big]^2$$

Given that only squares of the coordinates are stored, the above formula can be computed using the following algorithm:

$A \leftarrow (Y_n^2)^2$	$(= Y_n^4)$	$1S$
$B \leftarrow (Z_n^2)^2$	$(= Z_n^4)$	$1S$
$E \leftarrow (Y_n^2 + Z_n^2)^2 - A - B$	$(= 2Y_n^2 Z_n^2)$	$1S$
$Y_{2n}^2 \leftarrow \big[E -$	$(= \big[2Y_n^2 Z_n^2$	$1S + 2M_c$
$\quad c^2(B + dA)\big]^2$	$\quad - c^2(Z_n^4 + dY_n^4)\big]^2$	
$Z_{2n}^2 \leftarrow \big[-c^2 dE +$	$(= \big[-c^2 d(2Y_n^2 Z_n^2)$	$1S + M_c$
$\quad (B + dA)^2\big]^2$	$\quad + (Z_n^4 + dY_n^4)\big]^2$	

The complexity of the new algorithm is $(5S + 3M_c)$ while the old complexity was $(5S + 5M_c)$

EE. SQO Tripling for Generalised Edwards Coordinates:
By inspecting formula(E) in Section 3.3.1, we can see that the total complexity of formula(E) is $(4M + 7S + 8M_c)$. The algorithm used to compute Y_{3n} and Z_{3n} in formula(CC) of this section can be adapted to compute the requisite formulae. The first two steps can be omitted as squares are already available and the last two steps can be replaced with

$$Y_{3n}^2 \leftarrow Y_n^2\big[(c^2d^2)J - (\frac{3}{2}c^2 d)H + (c^2 d + 1)M - (3c^2)K\big]^2$$
$$Z_{3n}^2 \leftarrow Z_n^2\big[(-3c^2d^2)J - d(c^4 d + 1)N - (\frac{3}{2}c^2 d)H + (c^2)K\big]^2$$

The complexity of this algorithm is the same as that of formula(CC) which is $(10S + 2M + 5M_c)$. We can take $2M = 3S$ [8]. Thus $3S < (2M + 3M_c)$ and the new algorithm with complexity $(10S + 2M + 5M_c)$ is better than the old algorithm with complexity $(7S + 4M + 8M_c)$.

The improvements for Generalized Edwards Coordinates can be summarized in the table below:

Table 3.1: Differential Arithmetic on Generalized Edwards Coordinates

Point arithmetic on Generalized Edwards coordinates	Previous [58]	New [97]
Differential Doubling	$1M + 4S + 4M_c$	$5S + 3M_c$
Tripling	$4M + 7S + 8M_c$	$2M + 10S + 5M_c$
SQO Doubling	$5S + 5M_c$	$5S + 3M_c$
SQO Tripling	$4M + 7S + 8M_c$	$2M + 10S + 5M_c$

FF. Affine w-coordinate Differential Addition and Doubling for a new model of Binary Edwards Curves proposed in [108]:
The operation count of computing w_4 in formula (F) in Section 3.3.1 is $1I + 1M + 1M_c + 2S$ as the formula can be computed using the algorithm below:

$$\begin{array}{lll} A = w_2^2 & (= w_2^2) & 1S \\ B = A^2 & (= w_2^4) & 1S \\ C = t^2 A & (= t^2 w_2^2) & 1M_c \\ D = C^{-1} & \left(= \dfrac{1}{t^2 w_2^2}\right) & 1I \\ w_4 = (1+B)D & \left(= \dfrac{1+w_2^4}{t^2 w_2^2}\right) & 1M \end{array}$$

Now w_4 can be rewritten as

$$w_4 = \left(\frac{1}{t^2}\right)\left(\frac{1}{w_2^2} + w_2^2\right) \text{ and}$$

w_4 can be computed using the following algorithm:

$$\begin{array}{lll} A = w_2^2 & (= w_2^2) & 1S \\ B = \dfrac{1}{A} & \left(= \dfrac{1}{w_2^2}\right) & 1I \\ w_4 = \left(\dfrac{1}{t^2}\right)(A+B) & \left(= \left(\dfrac{1}{t^2}\right)\left(\dfrac{1}{w_2^2} + w_2^2\right)\right) & M_c \end{array}$$

Thus the complexity of the new doubling algorithm is $1I + 1S + 1M_c$ resulting in a saving of $1M + 1S$. The formulae(F) for differential addition(w_5) in the previous section costs $1I + 2M + 2S + 1M_c$. Considering that w_2^2 is computed both in the differential addition and doubling steps, w_2^2 can be computed just once. Thus the new total cost of a differential addition and doubling is $2I + 2M + 2S + 2M_c$ or $1I + 5M + 2S + 2M_c$ with Montgomery's Inversion trick, as compared to the previous total cost of $1I + 6M + 4S + 2M_c$ resulting in an overall saving of $1M + 2S$.

The improvements for the Binary Edwards curve defined in [108] can be summarized as below:

Table 3.2: Differential Arithmetic for Binary Edwards curve

Binary Edwards	Previous [108]	New [97]
Differential Doubling	$1I$+$1M$+$1M_c$+$2S$	$1I + 1S + 1M_c$
Differential Addition and Doubling	$1I$+$6M$+$4S$+$2M_c$	$1I$+$5M$+$2S$+$2M_c$

Chapter 4

Multi Exponentiation and Differential Chains

In this chapter, we motivate the need for computing Multi-exponentiation and the construction of Schoenmakers' algorithm for Double Scalar multiplication. We further provide two Triple Scalar multiplication algorithms.

4.1 Addition Chains and Exponentiation

Multi-exponentiation (also known as simultaneous exponentiation) in an Abelian group is a commonly used computation in cryptography, for example in signature verification algorithms and identification schemes (Chapter 7 and Chapter 9 in [103]). A straightforward method to compute the sum of products $n_1x_1 + n_2x_2 \in G$ (where $(G, +)$ is an Abelian group and x_1, $x_2 \in (G, +)$ and the exponents $n_1, n_2 \in Z$) is to compute n_1x_1 and n_2x_2 separately and then add them. The Strauss-Shamir method ([7], [46], Algorithm 9.23 in [28]) scans the corresponding bit representations of n_1 and n_2 simultaneously from left to right and makes use of precomputed group elements to compute $n_1x_1 + n_2x_2$, thus reducing the number of additions required to compute the desired sum. The Joint Sparse Form [93] introduced by Solinas in 2001 makes use of signed representations of the exponents to improve the Strauss-Shamir method and are useful in groups where inverses of group elements can be

computed efficiently such as elliptic curve groups. The problem of minimizing the number of multiplications whilst computing a product, such as n_1x_1 or n_2x_2, can be reduced to minimizing the number of additions in an abstraction known as an addition chain. A finite sequence of integers $a_0, a_1, \ldots a_r$ is called an addition chain (section 4.63 in [62]) for a_r if for each element a_i, there exists a_j and a_k in the sequence such that

$$a_i = a_j + a_k, \text{ for some } k \leq\ j < i \tag{4.1}$$

for all $i = 1, 2, \ldots, r$. Addition chains can be used to efficiently compute either a single exponentiation or multi-exponentiation (by using Strauss's method). Addition chains are applicable both in the context of multiplicative groups and additive groups such as Elliptic curve groups over a finite field.

We know that the arithmetic on a Montgomery curve relies on x-coordinate only arithmetic and also requires the *difference* of two group elements (points) to be known prior to the computation of addition of these two elements. Thus ordinary addition chains and improvements of these chains cannot be directly utilized for scalar multiplication on Montgomery curves. A special form of an addition chain called Lucas chains is useful in this context. A Lucas chain is a restricted variant of an addition chain where the indices in equation (4.1) above are such that either $j = k$ or the difference $a_k - a_j$ is already part of the chain. A special case of Lucas chains occur when either $j = k$ or $a_k - a_j = a_0 = 1$ and this is called the binary chain. A Lucas chain is also known as a *Differential Addition Chain* in the literature [9].

4.2 Montgomery's PRAC

While the Montgomery ladders in the previous sections are instances of binary chain algorithms to compute scalar multiplication, Montgomery's PRAC [81] algorithm, whilst computing scalar multiplication, is an example of a *Lucas chain* that is not a binary chain. In [81], the author proposes a Continued Fraction method for scalar multiplication and calls this algorithm CFRC. He then proposes a better algorithm than CFRC and calls it a

Practical algorithm(PRAC). The PRAC algorithm permits the exponent (the scalar) to be either prime or composite. Independent of the scalar being prime or composite, Montgomery provides a list of transformations that can be applied to the scalar in the course of performing the computation. This list can be found in Table 4 of [81]. The PRAC algorithm, in the process of constructing a Lucas chain for n, begins with $(d, e) = (n, \lceil n/\phi + 1/2 \rceil)$ where ϕ is the golden ratio. The algorithm iteratively uses a list of 9 transformations. At each iteration, the value of the pair (d, e) is reduced as specified in the transformation and this continues until $d = 1$ [43]. We reproduce the 7th and 8th transformations here for convenience:

Condition	**Action(s)**
$d \equiv -e \ (mod\ 3)$	$d \leftarrow (d - 2e)/3$ and $T_1 \leftarrow f(A, B, C)$ and $(A, B) \leftarrow (X_3(A)), f(T_1, A, B))$
$d \equiv \quad e \ (mod\ 3)$	$d \leftarrow (d - e)/3$ and $(T_1, T_2) \leftarrow (f(A, B, C), f(A, C, B))$ and $(A, B, C) \leftarrow (X_3(A), T_1, T_2)$

The above two transformations motivate the use of point tripling.

When PRAC is used with a composite exponent, the transformations suggested by Montgomery were employed in [99, Algorithm 3.33]. We reproduce the first two steps of Stam's algorithm here:

ALGORITHM 3.33 (Montgomery's PRAC Algorithm)
Given a base v and an exponent n, this algorithm computes v_n

1. [Make d odd] Let f_2 be the highest power of 2 dividing n. Set $d \leftarrow (n/2^{f_2})$ and $A \leftarrow \delta^{f_2}(v)$.

2. [$d \neq 0 \pmod 3$] Let f_3 be the highest power of 3 dividing n. Set $d \leftarrow (d/3^{f_3})$ and $A \leftarrow \tau^{f_3}(A)$.

$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$

Here δ is a doubling and τ is a tripling. As seen in step 2 of the algorithm above, if the exponent is a multiple of three and if f_3 is the highest power of 3 dividing the exponent n, then the algorithm requires f_3 triplings. For instance, if $n = 108 = 2^2 * 3^3$, doublings are carried out twice and triplings thrice. The dedicated differential tripling formulae introduced in the previous chapter can be used in this context. The tripling formula can also be used in the Supersingular Isogeny Diffie-Hellman key exchange (SIDH) algorithm [54, 34] which is a post-quantum cryptographic algorithm that enables Alice and Bob to exchange a secret key.

4.3 Algorithms for Multiexponentiation

The Strauss-Shamir method for simultaneous scalar multiplication cannot be immediately used in the context of Differential Addition Chains(DACs). However, this technique can be adapted to DACs as shown by Schoenmakers, who constructed the first algorithm to produce two dimensional (double scalar) DACs in 2000. This algorithm was published in [99] by Stam in 2003. Akishita's algorithm to construct two dimensional DACs was published in 2001 [2]. Both Shoenmakers' and Akishita's algorithms produce two dimensional binary chains. Bernstein proposed new algorithms to construct two dimensional DACs in 2006 along with a summary of previously known algorithms [8]. These included binary chains as well as Euclidean chain algorithms (algorithms using the Euclidean GCD scheme to construct DACs). In [5], Azarderakhsh and Karabina propose another DAC.

A natural question to ask is, if one can construct triple scalar multiplication analogues of the two dimensional DACs listed above. A practical motivation to construct such multi scalar multiplication algorithms arises in the implementation of some digital signature and identification schemes and their elliptic curve analogues. [77, Chapter 11] covers some of these signature schemes. The Okamoto Identification scheme [103, Section 9.3] requires a triple scalar multiplication operation to be performed by the signature verifier. Triple scalar multiplication can also be utilized in the accelerated

verification of ElGamal like signatures [3]. The need for higher order analogues can be seen in the batch verification of multiple signatures [25]. In [60], the authors propose three methods for randomized batch verification of ECDSA signatures, one of which is based on Montgomery ladders. Simultaneous scalar multiplication in the context of DAC could be utilized to achieve improved running times in the Montgomery ladder signature verification method. Interest in higher order DACs also arises from the recent interest in standardizing Montgomery curves [8] such as Curve25519. Further motivation to construct higher order DACs is found in [8], where the author presents new double scalar DAC algorithms and writes

Perhaps 3-dimensional versions of the ideas in this paper will also save time in the recent elliptic-curve-signature-verification algorithm .... I will leave this exploration to future research.

This exploration can be extended to other double scalar multiplication algorithms too, such as Akishita's and Schoenmakers'. In 2006, Brown extended Bernstein's ideas to dimensions ≥ 2 [19], but this method is patented [20].

Before we look at 3-dimensional extensions, we will introduce some 2-dimensional algorithms to produce binary chains.

4.4 Schoenmakers' Algorithm

In this section, we motivate the construction of the Schoenmakers' algorithm [94, 99]. Towards this end, we define a set of four points

$$G_i = \left\{ \begin{array}{l} m_i P + n_i Q, \\ m_i P + (n_i + 1) Q \\ (m_i + 1) P + n_i Q \\ (m_i + 1) P + (n_i + 1) Q \end{array} \right\}$$

for $m_i = (k_t \dots k_i)_2$, $n_i = (l_t \dots l_i)_2$. Below we show the construction of elements of G_i from G_{i+1} for all four combinations of (k_i, l_i):

1. $(k_i, l_i) = (0, 0)$: here $m_i = 2m_{i+1}$ and $n_i = 2n_{i+1}$.

$$\begin{aligned}
m_iP + n_iQ &= 2(m_{i+1}P + n_{i+1}Q) \\
(m_i + 1)P + n_iQ &= ((m_{i+1} + 1)P + n_{i+1}Q) + (m_{i+1}P + n_{i+1}Q) \\
m_iP + (n_i + 1)Q &= (m_{i+1}P + (n_{i+1} + 1)Q) + (m_{i+1}P + n_{i+1}Q) \\
(m_i + 1)P + (n_i + 1)Q &= ((m_{i+1} + 1)P + n_{i+1}Q) + (m_{i+1}P + (n_{i+1} + 1)Q)
\end{aligned}$$

2. $(k_i, l_i) = (1, 0)$: here $m_i = 2m_{i+1} + 1$ and $n_i = 2n_{i+1}$.

$$\begin{aligned}
m_iP + n_iQ &= ((m_{i+1} + 1)P + n_{i+1}Q) + (m_{i+1}P + n_{i+1}Q) \\
(m_i + 1)P + n_iQ &= 2((m_{i+1} + 1)P + n_{i+1}Q) \\
m_iP + (n_i + 1)Q &= ((m_{i+1} + 1)P + n_{i+1}Q) + (m_{i+1}P + (n_{i+1} + 1)Q) \\
((m_i + 1)P + (n_i + 1)Q &= ((m_{i+1} + 1)P + (n_{i+1} + 1)Q) + ((m_{i+1} + 1)P + n_{i+1}Q)
\end{aligned}$$

3. $(k_i, l_i) = (0, 1)$: here $m_i = 2m_{i+1}$ and $n_i = 2n_{i+1} + 1$.

$$\begin{aligned}
m_iP + n_iQ &= (m_{i+1}P + (n_{i+1} + 1)Q) + (m_{i+1}P + n_{i+1}Q) \\
(m_i + 1)P + n_iQ &= ((m_{i+1} + 1)P + n_{i+1}Q) + (m_{i+1}P + (n_{i+1} + 1)Q) \\
m_iP + (n_i + 1)Q &= 2(m_{i+1}P + (n_{i+1} + 1)Q) \\
((m_i + 1)P + (n_i + 1)Q &= ((m_{i+1} + 1)P + (n_{i+1} + 1)Q) + (m_{i+1}P + (n_{i+1} + 1)Q)
\end{aligned}$$

4. $(k_i, l_i) = (1, 1)$: here $m_i = 2m_{i+1} + 1$ and $n_i = 2n_{i+1} + 1$.

$$\begin{aligned}
m_iP + n_iQ &= ((m_{i+1} + 1)P + n_{i+1}Q) + (m_{i+1}P + (n_{i+1} + 1)Q) \\
(m_i + 1)P + n_iQ &= ((m_{i+1} + 1)P + (n_{i+1} + 1)Q) + ((m_{i+1} + 1)P + n_{i+1}Q) \\
m_iP + (n_i + 1)Q &= ((m_{i+1} + 1)P + (n_{i+1} + 1)Q) + (m_{i+1}P + (n_{i+1} + 1)Q) \\
((m_i + 1)P + (n_i + 1)Q &= 2((m_{i+1} + 1)P + (n_{i+1} + 1)Q)
\end{aligned}$$

The four elements in G_i, $0 \leq i \leq t$ give rise to the easy double scalar multiplication binary chain. The easy double scalar multiplication algorithm is as follows:

Algorithm 4.2: L-R Easy Double scalar multiplication algorithm

INPUT: Points P and Q on E_m; positive integers $k = (k_t \dots k_0)_2$ and $l = (l_t \dots l_0)_2$;
Precompute $(P - Q)$
OUTPUT: The point $[k]P + [l]Q$

[Initialize]
$P_1 \leftarrow 0$; $P_2 \leftarrow P$; $P_3 \leftarrow Q$; $P_4 \leftarrow P + Q$
[Loop through the scalar bits simultaneously]
for $i = t$ down to 0 do
if $(k_i, l_i) = (0, 0)$ then
$P_2 \leftarrow P_2 + P_1$ (P);
$P_3 \leftarrow P_3 + P_1$ (Q);
$P_4 \leftarrow P_4 + P_1$ (P+Q);
$P_1 \leftarrow 2 * P_1$
else if $(k_i, l_i) = (1, 0)$ then
$P_1 \leftarrow P_2 + P_1$ (P);
$P_3 \leftarrow P_2 + P_3$ (P-Q);
$P_4 \leftarrow P_4 + P_2$ (Q);
$P_2 \leftarrow 2 * P_2$
else if $(k_i, l_i) = (0, 1)$ then
$P_1 \leftarrow P_3 + P_1$ (Q);
$P_2 \leftarrow P_2 + P_3$ (P-Q);
$P_4 \leftarrow P_4 + P_3$ (P);
$P_3 \leftarrow 2 * P_3$
else if $(k_i, l_i) = (1, 1)$ then
$P_1 \leftarrow P_4 + P_1$ (P+Q);
$P_2 \leftarrow P_4 + P_2$ (Q);
$P_3 \leftarrow P_4 + P_3$ (P);
$P_4 \leftarrow 2 * P_4$
end for

$[m_0P + n_0Q$ is in $P_1]$
return P_1

However, to compute $m_0P + n_0Q$, it is not necessary to use all the four elements of G_i. If we omit $(m_i + 1)P + (n_i + 1)Q$ in G_i, it is still possible to compute the rest of the elements in all of the G_i, $0 \leq i \leq t$. Amongst the above set of formulae, for the cases where $(k_i, l_i) = (0, 0)$ or $(1, 0)$ or $(0, 1)$, no change is required, except omitting $(m_i + 1)P + (n_i + 1)Q$. However, when $(k_i, l_i) = (1, 1)$, $(m_i+1)P+n_iQ$ and $m_iP+(n_i+1)Q$ may have to be computed differently from the formula above, as they depend on $(m_{i+1}+1)P+(n_{i+1}+1)Q$ in G_{i+1}. Therefore when $(k_i, l_i) = (1, 1)$, we use

$$\begin{aligned}(m_i + 1)P + n_iQ &= ((m_{i+1} + 1)P + (n_{i+1} + 1)Q) + ((m_{i+1} + 1)P + n_{i+1}Q) \\ &= ((m_{i+1} + 1)P + n_{i+1}Q) + (m_{i+1}P + (n_{i+1} + 1)Q) + P\end{aligned}$$

The difference between the first two terms in the above rewritten equation is $(P - Q)$. The difference between the first two terms taken together which is $((m_{i+1} + 1)P + n_{i+1}Q + m_{i+1}P + (n_{i+1} + 1)Q)$ and P
can also be expressed in terms of elements in G_{i+1} i.e.,

$$\begin{aligned}((m_{i+1} + 1)P + n_{i+1}Q) - (m_{i+1}P + (n_{i+1} + 1)Q) &= (P - Q) \\ ((m_{i+1} + 1)P + n_{i+1}Q + m_{i+1}P + (n_{i+1} + 1)Q) - P & \\ = 2m_{i+1}P + (2n_{i+1} + 1)Q & \\ = (m_{i+1}P + (n_{i+1} + 1)Q) + (m_{i+1}P + n_{i+1}Q) &\end{aligned}$$

Similarly,

$$\begin{aligned}m_iP + (n_i + 1)Q &= (m_{i+1} + 1)P + (n_{i+1} + 1)Q) + (m_{i+1}P + (n_{i+1} + 1)Q) \\ &= ((m_{i+1} + 1)P + n_{i+1}Q) + (m_{i+1}P + (n_{i+1} + 1)Q) + Q\end{aligned}$$

As before, the difference between the first two terms in the above rewritten equation is $(P-Q)$. The difference between the first two terms taken together which is

$$((m_{i+1} + 1)P + n_{i+1}Q + m_{i+1}P + (n_{i+1} + 1)Q) \text{ and } Q$$

can also be expressed in terms of elements in G_{i+1} i.e.,

$$\begin{aligned}((m_{i+1}+1)P + n_{i+1}Q + m_{i+1}P + (n_{i+1}+1)Q) - Q \\ &= (2m_{i+1}+1)P + 2n_{i+1}Q \\ &= ((m_{i+1}+1)P + n_{i+1}Q) + (m_{i+1}P + n_{i+1}Q)\end{aligned}$$

If we denote the reduced G_i i.e., G_i without $(m_i+1)P + (n_i+1)Q$ as G_i' and the elements
$(m_iP + n_iQ)$, $((m_i+1)P + n_iQ)$ and $(m_iP + (n_i+1)Q)$ as $P_1[i]$, $P_2[i]$ and $P_3[i]$ respectively, then for the case $(k_i, l_i) = (1,1)$, the elements of G_i' can be computed as follows:

$$\begin{aligned}P_1[i] &\leftarrow P_2[i+1] + P_3[i+1] && (P - Q) \\ P_2[i] &\leftarrow P_1[i] + P && (P_3[i+1] + P_1[i+1]) \\ P_3[i] &\leftarrow P_1[i] + Q && (P_2[i+1] + P_1[i+1])\end{aligned}$$

Thus in order to compute $P_2[i]$ and $P_3[i]$, we need to first compute $P_3[i+1] + P_1[i+1]$ and $P_2[i+1] + P_1[i+1]$. The difference between $P_3[i+1]$ and $P_1[i+1]$ is Q and the difference between $P_2[i+1]$ and $P_1[i+1]$ is P and thus it is possible to compute both $P_3[i+1] + P_1[i+1]$ and $P_2[i+1] + P_1[i+1]$. Rules for the other cases can be constructed from the formulae above. For example when $(k_i, l_i) = (0,0)$,

$$\begin{aligned}P_1[i] &\leftarrow 2P_1[i+1] \\ P_2[i] &\leftarrow P_2[i+1] + P_1[i+1] && (P) \\ P_3[i] &\leftarrow P_3[i+1] + P_1[i+1] && (Q)\end{aligned}$$

As stated previously, Schoenmakers' algorithm was designed in 2000 and published in [99]. The derivation is not available in [99]. In [9, Section 4], the author specifies which one of the four elements of G_i is eliminated. We fill this gap by motivating the construction here and this helps us in constructing the three-dimensional analogue of Schoenmakers' algorithm in the next section. Doing away with the array notation for P_1, P_2 and P_3 above, we can present Schoenmakers' algorithm for double scalar multiplication as follows:

Algorithm 4.3: L-R Schoenmakers' Double scalar multiplication algorithm

INPUT: Points P and Q on E_m; positive integers $k = (k_t \dots k_0)_2$ and $l = (l_t \dots l_0)_2$;
Precompute $(P - Q)$
OUTPUT: The point $[k]P + [l]Q$

[Initialize]
$P_1 \leftarrow 0$; $P_2 \leftarrow P$; $P_3 \leftarrow Q$

[Loop through the scalar bits simultaneously]
for $i = t$ down to 0 do
P_1Store $\leftarrow P_1$; P_2Store $\leftarrow P_2$; P_3Store $\leftarrow P_3$;
if $(k_i, l_i) = (0, 0)$ then
    $P_1 \leftarrow 2 * P_1$Store ;
    $P_2 \leftarrow P_2$Store $+P_1$Store (P) ;
    $P_3 \leftarrow P_3$Store $+P_1$Store (Q)
else if $(k_i, l_i) = (1, 0)$ then
    $P_1 \leftarrow P_2$Store $+P_1$Store (P) ;
    $P_2 \leftarrow 2 * P_2$Store ;
    $P_3 \leftarrow P_2$Store $+P_3$Store (P-Q)
else if $(k_i, l_i) = (0, 1)$ then
    $P_1 \leftarrow P_3$Store $+P_1$Store (Q) ;
    $P_2 \leftarrow P_2$Store $+P_3$Store (P-Q) ;
    $P_3 \leftarrow 2 * P_3$Store
else if $(k_i, l_i) = (1, 1)$ then
    $P_1 \leftarrow P_2$Store $+P_3$Store (P-Q) ;
    P_2Partial $\leftarrow P_3$Store $+P_1$Store (Q)
    P_3Partial $\leftarrow P_2$Store $+P_1$Store (P) ;
    $P_2 \leftarrow P_1 + P$ (P_2Partial) ;
    $P_3 \leftarrow P_1 + Q$ (P_3Partial)
end if

end for

$[m_0P + n_0Q$ is in $P_1]$
return P_1

We now compare Schoenmakers' algorithm for double scalar multiplication (Algorithm 4.3) with the straightforward method of achieving the same. Since in the For loop in Algorithm 4.3, (k_i, l_i) can take on any of the four values of $(0,0), (1,0), (1,0)$ and $(1,1)$ with equal probability, the average cost per bit of the two scalars taken simultaneously can be computed as follows:

when $(k_i, l_i) \neq (1,1)$:
three point additions are required.
i.e, $3(3M + 2S) = (9M + 6S)$ operations.
when $(k_i, l_i) = (1,1)$:
five point additions are required.
out of these five, three require $(3M + 2S)$ operations.
the other two require $4M + 2S$ operations (as Z-coordinate of $P_2Partial$, $P_3Partial \neq 1$).
resulting in a total of $3(3M+2S)+2(4M+2S) = (17M+10S)$ operations
Thus on the average

$$\frac{(3(9M + 6S) + 3(3M + 2S) + 2(4M + 2S))}{4} \quad = \quad (11M + 7S)$$

operations would be required to run Algorithm 4.3 for every bit of the two scalars taken together.

The straightforward method of computing $[k]P + [l]Q$ constitutes computing $[k]P$ and $[l]Q$ separately, recovering the Y-coordinates of $[k]P$ and $[l]Q$ and adding up $[k]P$ and $[l]Q$ in projective coordinates ([2, Section 2.1]). If we take the bit lengths of scalars k and l to be the same and equal to $|k|$, then this method requires $(12|k| + 28)M + (8|k|S)$. If one ignores the complexity of recovering the Y-coordinates and the complexity of adding up $[k]P$ and

$[l]Q$, then the complexity of the straightforward method per bit of the scalar multiple is $((12|k|)M + (8|k|)S)/|k| = 12M + 8S$ operations. This can also be inferred from the fact that the total complexity to compute nP using the binary ladder is $(6M + 4S)(|n| - 1)$ for Montgomery curves where $|n|$ is the bit length of n [refer to Remarks 13.36, page 288 in [28]]. Thus, on the average, Schoenmakers' algorithm performs better than the straightforward method for double scalar multiplication.

The best case per bit cost of running the Schoenmakers' double scalar multiplication algorithm is $(9M + 6S)$, and this occurs when none of the $(k_i, l_i) \neq (1, 1)$. Under these circumstances Schoenmakers' algorithm will perform better than the straightforward algorithm.

The worst case per bit cost to run Algorithm 4.3 is $(17M + 10S)$, and this occurs when all of the $(k_i, l_i) = (1, 1)$. Thus, under worst case conditions, the straightforward algorithm is better than the Schoenmakers' algorithm for double scalar multiplication.

In comparing the costs here, we have not taken into consideration the costs of the precomputation in the Schoenmakers' algorithm and at the same time we have not taken into consideration the cost of recovering the Y-Coordinates and adding up $[k]P$ and $[l]Q$ in the straightforward method as these are small constant time costs and do not impact the comparison above. The cost per bit is summarized in the table below.

Table 4.1: 2-Dimensional Exponentiation

Algorithm	Cost per bit
Straight forward algorithm	$12M + 8S$
Schoenmakers' algorithm (average case)	$11M + 7S$
Akishita's algorithm	$9M + 6S$

4.5 Schoenmakers' Algorithm for Triple Scalar Multiplication

We now extend Schoenmakers' ideas for triple scalar multiplication. We do not explicitly derive the algorithm as the derivation is very similar to that of the double scalar multiplication algorithm. However, we do note that in every G'_i, where the G'_i is analogous to that used in the double scalar multiplication case, $G'_i = \{m_iP + n_iQ + S_iR$, $(m_i + 1)P + n_iQ + S_iR$, $m_iP + (n_i + 1)Q + S_iR$, $m_iP + n_iQ + (S_i + 1)R$, $(m_i + 1)P + (n_i + 1)Q + S_iR\}$. Taking $(m_iP + n_iQ + S_iR)$, $((m_i + 1)P + n_iQ + S_iR)$, $(m_iP + (n_i + 1)Q + S_iR)$, $(m_iP + n_iQ + (S_i + 1)R)$ and $((m_i + 1)P + (n_i + 1)Q + S_iR)$ to be P_1 P_2, P_3, P_4 and P_5 respectively, the algorithm for triple scalar multiplication is as follows:

Algorithm 4.4: L-R Schoenmakers' triple scalar multiplication algorithm

---

INPUT: Points P, Q and R on E_m;
Positive integers $k = (k_t \dots k_0)_2$, $l = (l_t \dots l_0)_2$, $s = (s_t \dots s_0)_2$
Precompute $(P + Q)$, $(P - Q)$, $(P - R)$, $(Q - R)$, $(P + Q - R)$
OUTPUT: The point $[k]P + [l]Q + [s]R$

[Initialize]
$P_1 \leftarrow 0$;$P_2 \leftarrow P$; $P_3 \leftarrow Q$; $P_4 \leftarrow R$; $P_5 \leftarrow P + Q$
for $i = t$ down to 0 do
P_1Store $\leftarrow P_1$;
P_2Store $\leftarrow P_2$;
P_3Store $\leftarrow P_3$;
P_4Store $\leftarrow P_4$;
P_5Store $\leftarrow P_5$;
if $(k_i, l_i, s_i) = (0, 0, 0)$ then
    $P_1 \leftarrow 2 * P_1$Store ;
    $P_2 \leftarrow P_2$Store $+P_1$Store (P) ;
    $P_3 \leftarrow P_3$Store $+P_1$Store (Q) ;

$P_4 \leftarrow P_4\text{Store} + P_1\text{Store}$ (R) ;
$P_5 \leftarrow P_5\text{Store} + P_1\text{Store}$ (P+Q)
else if $(k_i, l_i, s_i) = (0, 0, 1)$ then
$P_1 \leftarrow P_4\text{Store} + P_1\text{Store}$ (R) ;
$P_2 \leftarrow P_2\text{Store} + P_4\text{Store}$ (P-R) ;
$P_3 \leftarrow P_3\text{Store} + P_4\text{Store}$ (Q-R) ;
$P_4 \leftarrow 2 * P_4\text{Store}$;
$P_5 \leftarrow P_5\text{Store} + P_4\text{Store}$ (P+Q-R)
else if $(k_i, l_i, s_i) = (0, 1, 0)$ then
$P_1 \leftarrow P_3\text{Store} + P_1\text{Store}$ (Q) ;
$P_2 \leftarrow P_2\text{Store} + P_3\text{Store}$ (P-Q) ;
$P_3 \leftarrow 2 * P_3\text{Store}$;
$P_4 \leftarrow P_3\text{Store} + P_4\text{Store}$ (Q-R) ;
$P_5 \leftarrow P_5\text{Store} + P_3\text{Store}$ (P)
else if $(k_i, l_i, s_i) = (0, 1, 1)$ then
$P_1 \leftarrow P_3\text{Store} + P_4\text{Store}$ (Q-R) ;
$P_2 \leftarrow P_5\text{Store} + P_4\text{Store}$ (P+Q-R) ;
$P_3\text{Partial} \leftarrow P_4\text{Store} + P_1\text{Store}$ (R);
$P_4\text{Partial} \leftarrow P_3\text{Store} + P_1\text{Store}$ (Q) ;
$P_5\text{Partial} \leftarrow P_2\text{Store} + P_4\text{Store}$ (P-R) ;
$P_3 \leftarrow P_1 + Q$ (P_3Partial) ;
$P_4 \leftarrow P_1 + R$ (P_4Partial) ;
$P_5 \leftarrow P_2 + Q$ (P_5Partial)
else if $(k_i, l_i, s_i) = (1, 0, 0)$ then
$P_1 \leftarrow P_2\text{Store} + P_1\text{Store}$ (P) ;
$P_2 \leftarrow 2 * P_2\text{Store}$;
$P_3 \leftarrow P_2\text{Store} + P_3\text{Store}$ (P-Q) ;
$P_4 \leftarrow P_2\text{Store} + P_4\text{Store}$ (P-R) ;
$P_5 \leftarrow P_5\text{Store} + P_2\text{Store}$ (Q)
else if $(k_i, l_i, s_i) = (1, 0, 1)$ then
$P_1 \leftarrow P_2\text{Store} + P_4\text{Store}$ (P-R);
$P_3 \leftarrow P_5\text{Store} + P_4\text{Store}$ (P+Q-R) ;
$P_2\text{Partial} \leftarrow P_4\text{Store} + P_1\text{Store}$ (R);

P_4Partial $\leftarrow P_2$Store $+P_1$Store (P) ;
P_5Partial $\leftarrow P_3$Store $+P_4$Store (Q-R) ;
$P_2 \leftarrow P_1 + P$ (P_2Partial) ;
$P_4 \leftarrow P_1 + R$ (P_4Partial) ;
$P_5 \leftarrow P_3 + P$ (P_5Partial)
else if $(k_i, l_i, s_i) = (1, 1, 0)$ then
$P_1 \leftarrow P_5$Store $+P_1$Store (P+Q) ;
$P_2 \leftarrow P_5$Store $+P_2$Store (Q) ;
$P_3 \leftarrow P_5$Store $+P_3$Store (P) ;
$P_4 \leftarrow P_5$Store $+P_4$Store (P+Q-R);
$P_5 \leftarrow 2 * P_5$Store
else if $(k_i, l_i, s_i) = (1, 1, 1)$ then
$P_1 \leftarrow P_5$Store $+P_4$Store (P+Q-R) ;
P_2Partial $\leftarrow P_3$Store $+P_4$Store (Q-R) ;
P_3Partial $\leftarrow P_2$Store $+P_4$Store (P-R) ;
P_4Partial $\leftarrow P_2$Store $+P_3$Store (P-Q) ;
P_5Partial $\leftarrow P_4$Store $+P_1$Store (R) ;
$P_2 \leftarrow P_1 + P$ (P_2Partial);
$P_3 \leftarrow P_1 + Q$ (P_3Partial);
$P_4 \leftarrow P_1 + R$ (P_4Partial);
$P_5 \leftarrow P_1 + (P + Q)$ (P_5Partial)
end if
end for
return P_1

We now compute the per bit average cost of the above algorithm.
When $(k_i, l_i, s_i) = (0, 0, 0)$ or $(0, 0, 1)$ or $(0, 1, 0)$ or $(1, 0, 0)$ or $(1, 1, 0)$:
five point additions are required.
i.e, $5(3M + 2S) = (15M + 10S)$ operations.
When $(k_i, l_i, s_i) = (0, 1, 1)$ or $(1, 0, 1)$:
eight point additions are required.
Out of these eight, five require $(3M + 2S)$ operations each

i.e., $5(3M + 2S) = 15M + 10S$.
The other three require $4M + 2S$ operations each
i.e., $3(4M + 2S) = 12M + 6S$.
resulting in a total of $(27M + 16S)$ operations.

When $(k_i, l_i, s_i) = (1, 1, 1)$:
nine point additions are required.
Out of these nine, five require $(3M + 2S)$ operations
each i.e., $5(3M + 2S) = 15M + 10S$.
The other four require $4M + 2S$ operations each
i.e., $4(4M + 2S) = 16M + 8S$.
resulting in a total of $(31M + 18S)$ operations.

Thus, on average,

$$\frac{(5(15M + 10S) + 2(27M + 16S) + (31M + 18S))}{8} \quad = \quad (20M + 12.5S)$$

operations are required to run Algorithm 4.4 for every bit of the exponent.

The cost per bit when the straightforward algorithm is used is $6(3M + 2S) = (18M + 12S)$ operations. Thus, on the average, the straightforward algorithm performs better than Schoenmakers' algorithm for triple scalar multiplication.

In the best case, the per bit cost of running Algorithm 4.4 is $5(3M + 2S) = (15M + 10S)$ and this occurs when $(k_i, l_i, s_i) = (0, 0, 0)$ or $(0, 0, 1)$ or $(0, 1, 0)$ or $(1, 0, 0)$ or $(1, 1, 0)$. Under these circumstances, Schoenmakers' algorithm performs better than the straightforward algorithm.

The worst case per bit cost of Schoenmakers' algorithm is $31M + 18S$, and this occurs when all of $(k_i, l_i, s_i) = (1, 1, 1)$. Thus under worst case conditions, the straightforward algorithm would perform better than the Schoenmakers' algorithm for triple scalar multiplication.

The best case, average and worst case comparisons between the Schoenmakers'

algorithm and the straightforward method can be summarized as in the table below. The table lists the better option between Schoenmakers' algorithm and the straightforward method under best case, average and worst case conditions.

While the straightforward algorithm is *uniform*, where three differential point additions and one point doubling are required for every possible bit combination in the scalar, the Schoenmakers' algorithm does not have a uniform structure and is thus susceptible to side-channel attacks, when used in protocols where the scalar is a secret.

	Double scalar Schoenmaker vs Straightforward	Triple scalar Schoenmaker vs Straightforward
Best Case	Schoenmakers	Schoenmakers
Average	Schoenmakers	Straightforward
Worst	Straightforward	Straightforward

Thus, there is a need to construct other triple scalar algorithms not dependent on Schoenmakers' Algorithm. Next, we extend the Akishita's algorithm to triple scalar multiplication. Our results in this chapter are independent of Brown's results in [19].

Closely following the approach in [2] and letting $|n|$ denote the bit length of n, computation of nP on a Montgomery curve E_m requires $(6|n| - 3)M + (4|n| - 2)S$ operations. To compute the x-coordinate of $kP + lQ + uR$ on E_m, using the straightforward method, we require the following steps:

1. Compute kP using the Montgomery ladder.
2. Recover Y-coordinate of kP.
3. Compute lQ using the Montgomery ladder.
4. Recover Y-coordinate of lQ.
5. Compute uR using the Montgomery ladder.
6. Recover Y-coordinate of uR.
7. Compute $kP + lQ + uR$ in projective coordinates.
8. Compute x-coordinate(affine) of $kP + lQ + uR$.

We will assume the bit length of all three scalars k, l and u to be the same. The algorithm for recovery of the Y-coordinate is described in [84] and this costs $(12M + S)$ operations. Then, the computational cost of step 1, 3 and 5 together is $3\big[(6|k| - 3)M + (4|k| - 2)S\big]$. Steps 2, 4 and 6 together cost $3(12M + S)$. Consistent with [2], the cost of projective addition is $10M + 2S$, and thus the total cost of Step 7 is $2(10M + 2S)$ while step 8 costs $M + I$ where I denotes a field inversion. Thus, the cost of computing the x-coordinate of $kP+lQ+sR$ is $(18|k|+48)M+(12|k|+3)S+I$.

4.6 Three-Dimensional Scalar Multiplication on a Montgomery Curve

Akishita's algorithm[2] computes two dimensional differential scalar multiplication by proceeding as in the case of Schoenmakers' algorithm (Section 4.4). There is a difference, however. In the case of Schoenmaker's algorithm, the total number of differential point additions and doublings is 3 when $(k_i, l_i) \neq (1, 1)$ and 5 when $(k_i, l_i) = (1, 1)$, whereas in Akishita's algorithm, by performing some lookahead, the total number of differential point additions and doublings is reduced to 3 for all possible bit patterns in the scalar.

In this section, we extend Akishita's ideas [2] to compute $kP + lQ + uR$. We define a set of 8 points

$$G_i = \left\{ \begin{array}{l} m_iP + n_iQ + s_iR \\ m_iP + n_iQ + (s_i + 1)R \\ m_iP + (n_i + 1)Q + s_iR \\ m_iP + (n_i + 1)Q + (s_i + 1)R \\ (m_i + 1)P + n_iQ + s_iR \\ (m_i + 1)P + n_iQ + (s_i + 1)R \\ (m_i + 1)P + (n_i + 1)Q + s_iR \\ (m_i + 1)P + (n_i + 1)Q + (s_i + 1)R \end{array} \right\}$$

for $m_i = (k_t \dots k_i)_2$, $n_i = (l_t \dots l_i)_2$ and $s_i = (u_t \dots u_i)_2$ where $(k_t \dots k_1 k_0)_2$, $(l_t \dots l_1 l_0)_2$ and $(u_t \dots u_1 u_0)_2$ are binary representations of k, l and u respectively; $m_i = 2m_{i+1}$ or $m_i = (2m_{i+1} + 1)$ depending on whether $k_i = 0$ or $k_i = 1$. Similar relationships hold for n_i and s_i i.e., if $l_i = 0$, $n_i = 2n_{i+1}$ else $n_i = (2n_{i+1} + 1)$; if $u_i = 0$, $s_i = 2s_{i+1}$ else $s_i = (2s_{i+1} + 1)$. Each of the 8 elements in G_i can be written in terms of the elements in G_{i+1}. For instance, when $(k_i, l_i, u_i) = (0, 1, 0)$ we can write $m_i = 2m_{i+1}$, $n_i = (2n_{i+1} + 1)$ and $s_i = 2s_{i+1}$. In this case, as examples, we show a couple of elements of G_i written in terms of elements in G_{i+1} as follows:

$$m_iP + n_iQ + s_iR = (m_{i+1}P + n_{i+1}Q + s_{i+1}R) + (m_{i+1}P + (n_{i+1} + 1)Q + s_{i+1}R)$$

and

$$\begin{aligned}(m_i + 1)P + (n_i + 1)Q + s_iR = (m_{i+1}P + (n_{i+1} + 1)Q + s_{i+1}R) + \\ ((m_{i+1} + 1)P + (n_{i+1} + 1)Q + s_{i+1}R) \ .\end{aligned}$$

We can write the other six elements of G_i similarly, in terms of elements of G_{i+1}. However, whilst computing the elements in G_i, we do not want to be using all of the eight elements in G_{i+1} towards computing $(kP + lQ + uR)$, because this would be more expensive than the straightforward computation of $kP + lQ + uR$. Straightforward computation using the binary ladder requires

two such elements to be processed for each bit in the binary representation of a scalar and thus a total of six elements need to be processed for every bit in the three scalars taken at a time. Hence, to make our method more cost effective than the straightforward method, the number of elements in each G_{i+1} should be less than 6. It turns out that it is enough to have five elements in each of the G_{i+1} to achieve our goal of computing $(kP + lQ + uR)$. For example, if $(k_i, l_i, u_i) = (0, 0, 0)$, it suffices to have the following five elements in G_{i+1}:

$$\left\{\begin{array}{l} m_{i+1}P + n_{i+1}Q + s_{i+1}R \\ m_{i+1}P + n_{i+1}Q + (s_{i+1}+1)R \\ m_{i+1}P + (n_{i+1}+1)Q + s_{i+1}R \\ (m_{i+1}+1)P + n_{i+1}Q + s_{i+1}R \\ (m_{i+1}+1)P + (n_{i+1}+1)Q + s_{i+1}R \end{array}\right\}. \tag{4.2}$$

If $(k_i, l_i, u_i) = (0, 1, 0)$, the following 5 elements in G_{i+1} suffice:

$$\left\{\begin{array}{l} m_{i+1}P + n_{i+1}Q + s_{i+1}R \\ m_{i+1}P + (n_{i+1}+1)Q + s_{i+1}R \\ m_{i+1}P + (n_{i+1}+1)Q + (s_{i+1}+1)R \\ (m_{i+1}+1)P + (n_{i+1}+1)Q + s_{i+1}R \\ (m_{i+1}+1)P + (n_{i+1}+1)Q + (s_{i+1}+1)R \end{array}\right\}. \tag{4.3}$$

Next, we need to construct rules for computing elements of G_i from G_{i+1}. For this, we take into consideration the values of k_{i-1}, l_{i-1} and u_{i-1} in addition to k_i, l_i and u_i . We show this with an example. Let $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 0, 0, 0, 1, 0)$. Then $m_i = 2m_{i+1}$, $n_i = 2n_{i+1}$, $s_i = 2s_{i+1}$. The five elements of G_{i+1} are the same as those depicted in equation (4.2) above. The five elements of G_i are

$$\left\{\begin{array}{l} m_iP + n_iQ + s_iR \\ m_iP + (n_i+1)Q + s_iR \\ m_iP + (n_i+1)Q + (s_i+1)R \\ (m_i+1)P + (n_i+1)Q + s_iR \\ (m_i+1)P + (n_i+1)Q + (s_i+1)R \end{array}\right\}. \tag{4.4}$$

These five elements of G_i can be computed from those of G_{i+1} as follows:

$$
\begin{aligned}
&m_iP + n_iQ + s_iR = (m_{i+1}P + n_{i+1}Q + s_{i+1}R) + (m_{i+1}P + n_{i+1}Q + s_{i+1}R),\\
&m_iP + (n_i+1)Q + s_iR\\
&\quad = (m_{i+1}P + n_{i+1}Q + s_{i+1}R) + (m_{i+1}P + (n_{i+1}+1)Q + s_{i+1}R),\\
&m_iP + (n_i+1)Q + (s_i+1)R\\
&\quad = (m_{i+1}P + n_{i+1}Q + (s_{i+1}+1)R) + (m_{i+1}P + (n_{i+1}+1)Q + s_{i+1}R),\\
&(m_i+1)P + (n_i+1)Q + s_iR\\
&\quad = (m_{i+1}P + n_{i+1}Q + s_{i+1}R) + ((m_{i+1}+1)P + (n_{i+1}+1)Q + s_{i+1}R) \text{ and}\\
&(m_i+1)P + (n_i+1)Q + (s_i+1)R\\
&\quad = (m_{i+1}P + n_{i+1}Q + (s_{i+1}+1)R) + ((m_{i+1}+1)P + (n_{i+1}+1)Q + s_{i+1}R).
\end{aligned}
$$

If elements of G_{i+1} are listed as T_0Tmp, T_1Tmp, T_2Tmp, T_3Tmp and T_4Tmp in the same order as in equation (4.2) above, and the elements of G_i are listed as T_0, T_1, T_2, T_3 and T_4 in the same order as in equation (4.4) above, then the following rules enable us to compute elements of G_i from those of G_{i+1}:

$$
\begin{aligned}
&T_0 \leftarrow 2T_0Tmp,\\
&T_1 \leftarrow T_2Tmp + T_0Tmp \ \ (Q),\\
&T_2 \leftarrow T_2Tmp + T_1Tmp \ \ (Q-R),\\
&T_3 \leftarrow T_4Tmp + T_0Tmp \ \ (P+Q) \text{ and}\\
&T_4 \leftarrow T_4Tmp + T_1Tmp \ \ (P+Q-R) \ .
\end{aligned}
$$

As in the case of the formulae in the Montgomery ladder, the values in the brackets beside the formula above give the difference between points being added as these differences would be required for differential addition point arithmetic. While we derived the Montgomery ladder rules when $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,0,0,0,1,0)$, similar rules can be derived for the other 63 possible binary combinations of $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1})$. Whilst we do not explicitly derive these rules here, we list below the five element set G_{i+1} for all combinations of (k_i, l_i, u_i) that was used in the construction of the 3 dimensional extension of Akishita's algorithm.

Table 4.2: Five element set G_{i+1}

$\underline{(k_i, l_i, u_i) = (0,0,0):}$	$\underline{(k_i, l_i, u_i) = (0,0,1):}$
$m_{i+1}P + n_{i+1}Q + s_{i+1}R$	$m_{i+1}P + n_{i+1}Q + s_{i+1}R$
$m_{i+1}P + n_{i+1}Q + (s_{i+1}+1)R$	$m_{i+1}P + n_{i+1}Q + (s_{i+1}+1)R$
$m_{i+1}P + (n_{i+1}+1)Q + s_{i+1}R$	$m_{i+1}P + (n_{i+1}+1)Q + (s_{i+1}+1)R$
$(m_{i+1}+1)P + n_{i+1}Q + s_{i+1}R$	$(m_{i+1}+1)P + n_{i+1}Q + (s_{i+1}+1)R$
$(m_{i+1}+1)P + (n_{i+1}+1)Q + s_{i+1}R$	$(m_{i+1}+1)P+(n_{i+1}+1)Q+(s_{i+1}+1)R$
$\underline{(k_i, l_i, u_i) = (0,1,0):}$	$\underline{(k_i, l_i, u_i) = (0,1,1):}$
$m_{i+1}P + n_{i+1}Q + s_{i+1}R$	$m_{i+1}P + n_{i+1}Q + s_{i+1}R$
$m_{i+1}P + (n_{i+1}+1)Q + s_{i+1}R$	$m_{i+1}P + n_{i+1}Q + (s_{i+1}+1)R$
$m_{i+1}P + (n_{i+1}+1)Q + (s_{i+1}+1)R$	$m_{i+1}P + (n_{i+1}+1)Q + s_{i+1}R$
$(m_{i+1}+1)P + (n_{i+1}+1)Q + s_{i+1}R$	$m_{i+1}P + (n_{i+1}+1)Q + (s_{i+1}+1)R$
$(m_{i+1}+1)P+(n_{i+1}+1)Q+(s_{i+1}+1)R$	$(m_{i+1}+1)P+(n_{i+1}+1)Q+(s_{i+1}+1)R$
$\underline{(k_i, l_i, u_i) = (1,0,0):}$	$\underline{(k_i, l_i, u_i) = (1,0,1):}$
$m_{i+1}P + n_{i+1}Q + s_{i+1}R$	$m_{i+1}P + n_{i+1}Q + s_{i+1}R$
$(m_{i+1}+1)P + n_{i+1}Q + s_{i+1}R$	$m_{i+1}P + n_{i+1}Q + (s_{i+1}+1)R$
$(m_{i+1}+1)P + n_{i+1}Q + (s_{i+1}+1)R$	$(m_{i+1}+1)P + n_{i+1}Q + s_{i+1}R$
$(m_{i+1}+1)P + (n_{i+1}+1)Q + s_{i+1}R$	$(m_{i+1}+1)P + n_{i+1}Q + (s_{i+1}+1)R$
$(m_{i+1}+1)P+(n_{i+1}+1)Q+(s_{i+1}+1)R$	$(m_{i+1}+1)P+(n_{i+1}+1)Q+(s_{i+1}+1)R$
$\underline{(k_i, l_i, u_i) = (1,1,0):}$	$\underline{(k_i, l_i, u_i) = (1,1,1):}$
$m_{i+1}P + n_{i+1}Q + s_{i+1}R$	$m_{i+1}P + n_{i+1}Q + (s_{i+1}+1)R$
$m_{i+1}P + (n_{i+1}+1)Q + s_{i+1}R$	$m_{i+1}P + (n_{i+1}+1)Q + (s_{i+1}+1)R$
$(m_{i+1}+1)P + n_{i+1}Q + s_{i+1}R$	$(m_{i+1}+1)P + n_{i+1}Q + (s_{i+1}R+1)$
$(m_{i+1}+1)P + (n_{i+1}+1)Q + s_{i+1}R$	$(m_{i+1}+1)P + (n_{i+1}+1)Q + s_{i+1}R$
$(m_{i+1}+1)P+(n_{i+1}+1)Q+(s_{i+1}+1)R$	$(m_{i+1}+1)P+(n_{i+1}+1)Q+(s_{i+1}+1)R$

We present the 3-dimensional Montgomery ladder in the Appendix (Algorithm A.1). We now analyze Algorithm A.1 when applied to Montgomery curves. As in the previous section, we will take the bit lengths of all the three scalars to be the same. Computing $P + Q$ and $P - Q$ in affine coordinates costs $4M + 2S + I$. Similarly points $((P + R), (P - R))$, $((Q + R), (Q - R))$ and $((P + Q + R), (P + Q - R))$ need to be precomputed as well in affine coordinates. Thus the total cost of the precomputation steps in Algorithm A.1 is $4 * (4M + 2S + I) = 16M + 8S + 4I$. The cost of a point addition in the above ladder would be $3M + 2S$, as the difference of the points added is in affine form (*i.e.*, $Z = 1$). In the For loop of the above algorithm, either point addition formulae are required four times and point doubling once or alternatively, the point addition formula is required five times per bit of the scalar k. Thus, the cost for every bit of k is $5 * (3M + 2S) = 15M + 10S$ and the total cost of the for loop in the above algorithm is $15(|k| - 1)M + 10(|k| - 1)S$. The finalization step after the for loop costs $3M + 2S$. Computation of the x-coordinate by $x = X/Z$ costs $M + I$. Thus the total cost of the above algorithm is $(15|k| + 5)M + 10|k|S + 5I$. If $|k| = 160$, $S/M = 0.8$ and $I/M = 30$, the cost of the above algorithm is $3835M$.

For the same set of parameters, the cost of the straightforward algorithm as calculated in Section-2 is $(18|k| + 48)M + (12|k| + 3)S + I = 4496M$. Thus simultaneous triple scalar multiplication results in about 15% improvement over the straightforward method. When $|k| = 256$, the improvement is approximately 22% as the three dimensional Montgomery ladder costs $6043M$ and the straightforward method costs $7761M$.

As in the case of the one dimensional Montgomery ladder (Algorithm 3.1), the three dimensional Montgomery ladder (Algorithm A.1) can be adapted to work with differential addition extensions to various other forms of elliptic curves(examples listed previously in Section 3.3 of this thesis) and not limited to Montgomery curves alone.

The usage of temporary variables can be improved in the above algorithm (Algorithm A.1). Some operations towards the end of the computation can be eliminated. In the last iteration of the for loop in the above algorithm computation of T_2 and T_4 can be done away with, thus resulting in a further saving of at least $6M$ and $4S$ operations. Further, the cost of some finite field additions can be done away with by combining some point additions. For example if one has to compute $T_3 \leftarrow T_4\text{Tmp} + T_0\text{Tmp}$ (P+Q) and $T_4 \leftarrow T_4\text{Tmp}+T_1\text{Tmp}$ (P+Q-R) where $T_0\text{Tmp}=(X_0, Y_0, Z_0)$, $T_1\text{Tmp}=(X_1, Y_1, Z_1)$, $P + Q = (X_2, Y_2, Z_2)$, $(P + Q - R) = (X_3, Y_3, Z_3)$, $T_4\text{Tmp}=(X_4, Y_4, Z_4)$, $T_3=(X_5, Y_5, Z_5)$ and $T_4=(X_6, Y_6, Z_6)$ then

$$
\begin{aligned}
X_5 &= Z_2[(X_0 - Z_0)(X_4 + Z_4) + (X_0 + Z_0)(X_4 - Z_4)]^2 \text{ and} \\
X_6 &= Z_3[(X_1 - Z_1)(X_4 + Z_4) + (X_1 + Z_1)(X_4 - Z_4)]^2 \text{ while} \\
Z_5 &= X_2[(X_0 - Z_0)(X_4 + Z_4) - (X_0 + Z_0)(X_4 - Z_4)]^2 \text{ and} \\
Z_6 &= X_3[(X_1 - Z_1)(X_4 + Z_4) - (X_1 + Z_1)(X_4 - Z_4)]^2 \ .
\end{aligned}
$$

Thus one could group the computations of T_3 and T_4 together, thereby computing $(X_4 + Z_4)$ and $(X_4 - Z_4)$ just once, thus saving 2 field additions. In general the addition of points $T_2 + T_0$, $T_1 + T_0$ can save 2 field additions and can be extended to saving n field additions whilst computing $T_n + T_0$, $T_{n-1}+T_0, \ldots, T_1+T_0$. Similar benefits can be obtained when one combines the point addition and doubling operations together. These enhancements can be utilized to improve the performance of the 3 dimensional Montgomery Ladder.

The 3-dimensional Montgomery ladder is not a uniform algorithm and thus, as in the case of Schoenmakers' algorithm, can be susceptible to side-channel attacks.

Chapter 5

Precomputation of Elliptic Curve Points for Jacobian Coordinates for Double Scalar Multiplication

In this chapter we review two precomputation schemes from the literature for double scalar multiplication. As a prelude, we recall from the literature, the formulae for point arithmetic for Jacobian coordinate point representation on Weierstrass curves including a review of *Conjugate addition* and *Co-Z addition*. Of the two precomputation schemes we review here, one is dependent on *Conjugate addition* [72] and the other on *Co-Z addition*[71]. We find that some of the results based on *Co-Z addition* in [71] are incorrect. We construct new double scalar multiplication algorithms for precomputation based on *Conjugate addition* and then show that precomputation algorithms for elliptic curve double scalar multiplication based on *Co-Z addition* are not necessarily faster than those based on *Conjugate addition*.

5.1 Point Arithmetic Formulae for Jacobian Coordinates on Elliptic Curves

In Section 2.1.5, we looked at point addition and doubling formulae for Jacobian coordinates. Longa and Miri in [73] improved the operation counts for Jacobian coordinates. We recall some of their formulae and operation counts.

If $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, Z_1)$ then $P_3 = P_1 + P_2 = (X_3, Y_3, Z_3)$ is given by

$$\begin{aligned} X_3 &= \alpha^2 - (4\beta^3 + 8Z_2^2 X_1 \beta^2), \\ Y_3 &= \alpha(Z_2^2 X_1 \beta^2 - X_3) - Z_2^3 Y_1 \alpha^3, \\ Z_3 &= \theta\beta, \end{aligned} \tag{5.1}$$

where

$$\begin{aligned} \alpha &= 2(Z_1^3 Y_2 - Z_2^3 Y_1), \\ \beta &= Z_1^2 X_2 - Z_2^2 X_1, \\ \theta &= (Z_1 + Z_2)^2 - Z_1^2 - Z_2^2 \ . \end{aligned}$$

The cost of computing point addition using the above formulae is $11M + 5S$ which is an improvement on the $12M + 4S$ scheme depicted in Section (2.1). If $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, 1)$ (called *Mixed Addition*), then $P_3 = P_1 + P_2 = (X_3, Y_3, Z_3)$ is given by

$$\begin{aligned} X_3 &= \alpha^2 - 4\beta^3 - 8X_1\beta^2, \\ Y_3 &= \alpha(4X_1\beta^2 - X_3) - 8Y_1\beta^3, \text{ and} \\ Z_3 &= (Z_1 + \beta)^2 - Z_1^2 - \beta^2, \end{aligned} \tag{5.2}$$

where $\alpha = 2(Z_1^3 Y_2 - Y_1)$ and $\beta = Z_1^2 X_2 - X_1$. Thus the cost of computing addition of two points when one of the points is in affine form is $7M + 4S$. If both the points P_1 and P_2 are in affine form, then the cost is further reduced to $4M + 2S$.

We now recall the fast Point Tripling formulae as given in [73]. If $P = (X_1, Y_1, Z_1)$ then $3P = (X_3, Y_3, Z_3)$ can be computed as follows

$$\begin{aligned} X_3 &= 16Y_1^2(2\beta - 2\alpha) + 4X_1\omega^2, \\ Y_3 &= 8Y_1\big[2\alpha - 2\beta)(4\beta - 2\alpha) - \omega^3\big], \\ Z_3 &= (Z_1 + \omega)^2 - Z_1^2 - \omega^2, \end{aligned}$$

where

$$\begin{aligned} 2\alpha &= (\theta + \omega)^2 - \theta^2 - \omega^2, \\ 2\beta &= 16Y_1^4, \\ \theta &= 3X_1^2 + aZ_1^4, \\ \omega &= 6(\big[(X_1 + Y_1^2)^2 - X_1^2 - Y_1^4\big]) - \theta^2 \ . \end{aligned}$$

The cost of computing the above tripling formulae is $(6M + 10S)$. When $a = -3$, the tripling formulae is the same as that above except for $\theta = 3X_1^2 + aZ_1^4$ which can be written as $\theta = 3(X_1 + Z_1^2)(X_1 - Z_1^2)$. The cost of tripling in this case is $(7M + 7S)$. When the point P is in affine form, i.e., $Z_1 = 1$ and $3P$ is given in projective form, the cost of tripling (called *Mixed Tripling*) is reduced to $(5M + 7S)$.

In [74], the authors develop so called *Doubling-Tripling formulae* useful in the computation of $6P + Q$ where P and Q are elliptic curve points. The authors develop a fast algorithm to compute $6P$ by first doubling, that is computing $2P$ and then tripling this to yield $6P$. They provide a $9M + 15S$ algorithm to compute $6P$ as some of the terms computed during doubling can be reused whilst tripling. However, there is a minor error in their algorithm and we provide a corrected version of the doubling-tripling algorithm. If $P = (X_1, Y_1, Z_1)$, $2P = (X_2, Y_2, Z_2)$ can be computed as below:

$X_2 = A^2 - 2B, Y_2 = A.(B - X_2) - 8D, Z_2 = (Y_1 + Z_1)^2 - C - E,$
where
$A = 3G + H, B = 2[(X_1 + C)^2 - G - D], C = Y_1^2, D = C^2,$
$E = Z_1^2, F = E^2, G = X_1^2, H = a.F$

Further $6P$ can be computed by tripling $2P$ as follows:

$X_3 = I.T + X, Y_3 = 8Y_2.(V - W), Z_3 = 2Z_2.P$
where
$I = Y_2^2, J = I^2, K = 16D.H, L = X_2^2, M = 3L + K, N = M^2,$

$P = [(X_2 + I)^2 - L - J] - N, R = P^2, S = (M + P)^2 - N - R, T = 16J - S,$
$U = 16J + T, V = -T.U, W = P.R, X = 4X_2.R$

This algorithm is the same as in [74], except for the computation of N and P. We next look at *Conjugate Addition*.

5.2 Conjugate Addition

We recall from Section 2.1.4 of this thesis, the idea of simultaneously computing $P \pm Q$ with a reduced operation count. In [72], the authors provided algorithms to add elliptic curve points with the form $P \pm Q$ and called it *Conjugate Addition*. The previous section provided point addition formulae for $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, Z_2)$. Given that $P_1 + P_2 = P_3 = (X_3, Y_3, Z_3)$ is computed using Equation (5.1), $P_1 - P_2 = P_4 = (X_4, Y_4, Z_4)$ can be computed as follows:

$$\begin{aligned} X_4 &= \gamma^2 - (4\beta^3 + 8Z_2^2 X_1 \beta^2), \\ Y_4 &= \gamma(Z_2^2 X_1 \beta^2 - X_4) - Z_2^3 Y_1 \beta^3, \\ Z_4 &= Z_3, \end{aligned}$$

where

$$\begin{aligned} \gamma &= -2(Z_1^3 Y_2 + Z_2^3 Y_1), \\ \beta &= Z_1^2 X_2 - Z_2^2 X_1 \ . \end{aligned}$$

All the terms in the above formulae for P_4 are already computed whilst computing P_3 except for γ^2 and the product of γ and $(Z_2^2 X_1 \beta^2 - X_4)$. Thus the cost of computing P_4 after that of P_3 is reduced to $1M + 1S$.

5.2.1 Conjugate Mixed Addition

When one of the points being added is in affine form, that is, when $P_1 = (X_1, Y_1, Z_1)$, $P_2 = (X_2, Y_2, 1)$ and $P_1 + P_2 = P_3 = (X_3, Y_3, Z_3)$, and given that P_3 is computed using Equation (5.2), $P_1 - P_2 = P_4 = (X_4, Y_4, Z_4)$ can be computed as follows:

$$\begin{aligned} X_4 &= \gamma^2 - (4\beta^3 + 8X_1\beta^2), \\ Y_4 &= \gamma(4X_1\beta^2 - X_4) - 8Y_1\beta^3, \\ Z_4 &= Z_3, \end{aligned}$$

where

$$\begin{aligned} \gamma &= -2(Z_1^3 Y_2 + Z_2^3 Y_1), \\ \beta &= Z_1^2 X_2 - Z_2^2 X_1 \ . \end{aligned}$$

All the terms in the above formulae for for P_4 are already computed whilst computing P_3 except for γ^2 and the product of γ and $(4X_1\beta^2 - X_4)$. Thus, the cost of computing P_4 after that of P_3 is $1M + 1S$. Similarly when both points P_1 and P_2 are in affine form, the cost of computing their difference would cost $1M + 1S$ after their sum is computed.

5.3 Co-Z Addition

In [76], Meloni proposed a new approach to point addition that is well suited to Euclidean addition chains. If the two points to be added $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, Z_2)$ have the same Z-coordinate, that is $Z_1 = Z_2 = Z$, then the following formulae can be utilized to compute $P_3 = P_1 + P_2 = (X_3, Y_3, Z_3)$:

$$\begin{aligned} X_3 &= D - B - C, \\ Y_3 &= (Y_2 - Y_1)(B - X_3) - Y_1(C - B), \\ Z_3 &= Z(X_2 - X_1), \end{aligned}$$

where

$$\begin{aligned} A &= (X_2 - X_1)^2, \\ B &= X_1 A, \\ C &= X_2 A, \text{ and} \\ D &= (Y_2 - Y_1)^2 \ . \end{aligned} \tag{5.3}$$

The operation count of the above algorithm is $5M + 2S$. Now, it so happens that P_1 and P_3 can have the same Z coordinate as
$P_1 = (X_1, Y_1, Z_1) \sim (X_1(X_2 - X_1)^2, Y_1(X_2 - X_1)^3, Z(X_2 - X_1))$
and the expressions
$X_1 A = X_1(X_2 - X_1)^2, Y_1(C - B) = Y_1(X_2 - X_1)^3, Z_3 = Z(X_2 - X_1)$
have already been computed. Thus, P_1 has been updated with the Z-coordinate of $P_1 + P_2$ without using any extra computations.

5.3.1 Point Tripling with *Co-Z* Update

Using the idea of *Co-Z Addition*, the authors in [71] construct a

(i) *point tripling with update* algorithm, and
(ii) *Co-Z addition with update* algorithm,

before utilizing them in constructing a precomputation table that can then be used to compute a double-scalar multiplication. We look at the precomputation scheme in the next section, whilst in this section we revisit the point tripling with update algorithm. If $P_1 = (X_1, Y_1, Z_1)$ and $P_3 = 3P_1 = (X_3, Y_3, Z_3)$, then P_3 is computed by first computing $2P$ such that P and $2P = (X_2, Y_2, Z_2)$ has the same Z coordinate and then using Meloni's scheme above to compute $3P$. $2P$ can be computed as follows:

$$\begin{aligned} X_2 &= -2A + B^2, \\ Y_2 &= -8Y_1^4 + B(A - X_2^3), \text{ and} \\ Z_2 &= 2Y_1 Z_1, \end{aligned}$$

where

$$A = 4X_1Y_1^2, \tag{5.4}$$
$$B = 3X_1^2 + aZ_1^4 \ .$$

P_1 can be updated as follows:

$$X_1 = A,$$
$$Y_1 = 8Y_1^4, \text{ and}$$
$$Z_1 = Z_2 \ .$$

Now P_2 and P_1 have the same Z coordinate, and this requires $4M + 6S$ operations to compute as per the above scheme. Further, $3P$ can be computed using equation (5.3) which requires $(5M + 2S)$ operations, thus requiring a total of $9M + 8S$ operations for the point tripling with update algorithm.

Now, rewriting $Z_2 = 2Y_1Z_1 = (Y_1 + Z_1)^2 - Y_1^2 - Z_1^2$ and $A = 4X_1Y_1^2 = 2((X_1 + Y_1^2)^2 - X_1^2 - Y_1^4)$, in Equation (5.4) we can replace $2M$ with $2S$ and thus the operation count of the above *tripling with update* algorithm can be reduced to $(7M + 10S)$.

5.4 Precomputation of Elliptic Curve points to compute $kP + lQ$

5.4.1 Jacobian Coordinates, $a = -3$

After providing algorithms for Conjugate addition in [72], Longa and Gebotys used it to generate precomputed tables of the form $c_iP + d_iQ$ where $c_i, d_i \in \{1, 3 \ldots m\}$ which can then be used by various algorithms to compute double scalar multiplication, i.e., $kP + lQ$ where k and l are scalars. In Table 4 of their paper, the cost of computing $3P, 3Q, P+Q, P-Q, 3P+Q, 3P-Q, P+3Q, P-3Q, 3P+3Q$ and $3P-3Q$ for standard Jacobian coordinates with curve parameter $a = -3$, was given as $42M + 32S$. This can be improved to

$41M + 31S$, resulting in a further saving of $1M + 1S$, as shown below. [72, Table 1] lists costs for point arithmetic using Jacobian representation.

Result	Operation	Cost (when $a = -3$)
$3P, 3Q$	2 Numbers of Mixed Tripling	$(5M+7S)+(5M+7S)$
$P \pm Q$	Mixed Addition(both affine) + Conjugate Mixed Addition	$(4M+2S)+(1M+1S)$
$3P \pm Q$	Mixed Addition(one affine) + Conjugate Mixed Addition	$(7M+4S)+(1M+1S)$
$P \pm 3Q$	Mixed Addition(one affine) + Conjugate Mixed Addition	$(7M+4S)+(1M+1S)$
$3P \pm 3Q$	Addition with stored values + Conjugate Mixed Addition	$(9M+3S)+(1M+1S)$

As in Section 5.1 addition of points in Projective coordinates (Weierstrass curves) costs $11M + 5S$. But with the square and cube of the Z-coordinate of one of the points available, the point addition cost can be reduced to $(10M + 4S)$. If the square and cube of the Z-coordinate of both the points are available, the point addition cost can be reduced to $(9M + 3S)$. In the above scheme, when $3P + Q$ is computed, both Z_{3P}^2 and Z_{3P}^3 are computed and do not have to be recomputed when $3P + 3Q$ is computed. Further, when $P + 3Q$ is computed, Z_{3Q}^2 and Z_{3Q}^3 is computed and thus does not have to be recomputed when $3P + 3Q$ is computed. Thus, the new total cost of computing $3P, 3Q, P + Q, P - Q, 3P + Q, 3P - Q, P + 3Q, P - 3Q, 3P + 3Q$ and $3P - 3Q$ for standard Jacobian coordinates can be reduced to $41M + 31S$.

5.4.2 Jacobian Coordinates, $a \neq -3$

In [71], Lin and Zhang propose to improve upon the cost of Longa and Gebotys' algorithm for precomputation and thus propose new algorithms to compute $c_iP + d_iQ$ where $c_i, d_i \in \{1, 3\}$ and $c_i, d_i \in \{1, 3, 5\}$ and the curve parameter a need not be equal to -3. Their algorithm for $c_i, d_i \in \{1, 3\}$

utilizes *Co-Z* point arithmetic formulae proposed in [76] and costs $50M + 36S$. Their scheme is as follows:

Lin and Zhang Scheme to compute $c_iP + d_iQ$ where $c_i, d_i \in \{1, 3\}$

Operation	Cost
$P + Q$ and $P - Q$	$11M + 5S$
$2P = (P + Q) + (P - Q)$, $2Q = (P + Q) - (P - Q)$ and $2P, 2Q, P + Q, P - Q$ $co - Z$	$7M + 3S$
$3P + Q = (P + Q) + 2P$, $3P - Q = (P - Q) + 2P$, $3Q + P = (P + Q) + 2Q$, $3Q - P = (Q - P) + 2Q$	$4(5M + 2S)$
$3(P + Q)$ and $3(P - Q)$	$2(6M + 10S)$
Total	$50M + 36S$

The authors in [71] then provide a comparison of the costs of their algorithm with that of Longa and Gebotys, as in the table below:

Algorithm to compute c_iP+d_iQ	Cost when $c_i, d_i \in \{1, 3\}$
Longa and Gebotys [72]	$56M + 40S$
Lin and Zhang [71]	$50M + 36S$

From the cost comparison table, the algorithm of [71] should perform better than the Longa and Gebotys algorithm. However, this is incorrect because we can structure the Longa and Gebotys algorithm when $a \neq -3$ and $c_i, d_i \in \{1, 3\}$ (the number of values each scalar can assume, *i.e.*, the *window* size=2) as follows:

Result	Operation	Cost (when $a \neq -3$)
$3P, 3Q$	2 Numbers of Mixed Tripling	$(6M+7S)+(6M+7S)$
$P \pm Q$	Mixed Addition(both affine)+ Conjugate Mixed Addition	$(4M+2S)+(1M+1S)$
$3P \pm Q$	Mixed Addition(one affine)+ Conjugate Mixed Addition	$(7M+4S)+(1M+1S)$
$P \pm 3Q$	Mixed Addition(one affine)+ Conjugate Mixed Addition	$(7M+4S)+(1M+1S)$
$3P \pm 3Q$	Addition with stored values+ Conjugate Mixed Addition	$(9M+3S)+(1M+1S)$

Thus the total cost of Longa and Gebotys algorithm when $a \neq -3$ is $43M + 31S$ and thus it is more efficient than the Lin and Zhang algorithm which costs $50M + 36S$. Moreover, Longa and Gebotys algorithm computes $3P$ and $3Q$ whereas the algorithm due to Lin and Zhang [71] does not. However, it is not required to compute $3P$ and $3Q$.

The authors in [71] further provide an algorithm to compute $c_iP + d_iQ$, where $c_i, d_i \in \{1, 3, 5\}$, which we reproduce below

Lin and Zhang Scheme to compute $c_iP + d_iQ$ where $c_i, d_i \in \{1, 3, 5\}$

Result	Operation	Cost
$P \pm Q$	$P + Q$ and $P - Q$	$11M + 5S$
	$2P = (P + Q) + (P - Q)$,	
	$2Q = (P + Q) - (P - Q)$	
	and $2P, 2Q, P + Q, P - Q$ $co - Z$	7M+3S
$3P \pm Q$	$3P + Q = (P + Q) + 2P$,	
	$3P - Q = (P - Q) + 2P$,	
$3Q \pm P$	$3Q + P = (P + Q) + 2Q$,	
	$3Q - P = (Q - P) + 2Q$,	$4(5M+2S)$
$3(P + Q)$	$3P + 3Q = 2(P + Q)$	$4M + 6S$
	$+(P + Q)$,	$5M + 2S$
	λ	$1M$
$5(P + Q)$	$5(P + Q) = 3(P + Q)$	
	$+2(P + Q)$	$5M + 2S$
	Adjusting $3(P + Q), 2P$	$1M + 1S$
	and $2Q$ to be $co - Z$	$4M$
$5P + 3Q$	$5P + 3Q = 3(P + Q) + 2P$	
$5Q + 3P$	$5Q + 3P = 3(P + Q) + 2Q$	$2(5M+2S)$

Result	Operation	Cost
$3P-3Q$	$3P-3Q=2(P-Q)$	$4M+6S$
	$+(P-Q)$,	$5M+2S$
	λ	$1M$
$5(P-Q)$	$5(P-Q)=3(P-Q)$	
	$+2(P-Q)$	$5M+2S$
	Adjusting $3(P-Q), 2P$	$1M+1S$
	and $2Q$ to be $co-Z$	$4M$
$5P-3Q$	$5P-3Q=3(P+Q)+2P$	
	$3P-5Q=3(P+Q)+(-2Q)$	$2(5M+2S)$
$5Q-3P$	$5Q-3P=-(3P-5Q)$	
	Total	$98M+46S$

The comparison table for the operation counts as given in [71] is as below:

Algorithm to compute c_iP+d_iQ	Cost when $c_i, d_i \in \{1,3,5\}$
Longa and Gebotys [72]	$129M+95S$
Lin and Zhang [71]	$98M+46S$

The authors in [71] also conclude that their scheme becomes more efficient than that of Longa and Gebotys in [72], as the *window* size increases.
Longa and Gebotys' scheme can be extended when $a \neq -3$ and $c_i, d_i \in \{1,3,5\}$ as follows: As suggested in [72], we can start by performing $P \to 2P \to 4P$ and then obtaining $3P$ and $5P$ using $4P \pm P$. Similarly, we can compute $Q \to 2Q \to 4Q$ and then obtain $3Q$ and $5Q$ using $4Q \pm Q$. We structure the complete algorithm as follows:

Result	Operation	Cost (when $a \neq -3$)
$2P$	Doubling	$(1M + 5S)$
$4P$	Doubling	$(2M + 8S)$
$4P \pm P$	Mixed Addition(one affine) + Conjugate Mixed Addition	$(7M + 4S) + (1M + 1S)$
$2Q$	Doubling	$(1M + 5S)$
$4Q$	Doubling	$(2M + 8S)$
$4Q \pm Q$	Mixed Addition(one affine) + Conjugate Mixed Addition	$(7M + 4S) + (1M + 1S)$
$P \pm Q$	Mixed Addition(both affine) + Conjugate Mixed Addition	$(4M + 2S) + (1M + 1S)$
$3P \pm Q$	Mixed Addition(one affine) + Conjugate Mixed Addition	$(7M + 4S) + (1M + 1S)$
$P \pm 3Q$	Mixed Addition(one affine) + Conjugate Mixed Addition	$(7M + 4S) + (1M + 1S)$
$5P \pm Q$	Mixed Addition(one affine) + Conjugate Mixed Addition	$(7M + 4S) + (1M + 1S)$
$P \pm 5Q$	Mixed Addition(one affine) + Conjugate Mixed Addition	$(7M + 4S) + (1M + 1S)$
$3P \pm 3Q$	Addition with stored values + Conjugate Mixed Addition	$(9M + 3S) + (1M + 1S)$
$3P \pm 5Q$	Addition with stored values + Conjugate Mixed Addition	$(9M + 3S) + (1M + 1S)$
$5P \pm 3Q$	Addition with stored values + Conjugate Mixed Addition	$(9M + 3S) + (1M + 1S)$
$5P \pm 5Q$	Addition with stored values + Conjugate Mixed Addition	$(9M + 3S) + (1M + 1S)$

Thus the total cost of Longa and Gebotys' algorithm to compute $c_iP + d_iQ$ when $a \neq -3$ and $c_i, d_i \in \{1, 3, 5\}$ is $99M + 75S$. In [71], Lin and Zhang provide an algorithm to do the same and the cost of their algorithm is $98M + 46S$. However, Lin and Zhang's algorithm does not compute $5P \pm Q$ and $P \pm 5Q$ and thus is incomplete. Therefore, it may not be appropriate

to compare the costs between the two algorithms. If we add the cost of computing $5P \pm Q$ and $P \pm 5Q$ to the cost of the incomplete algorithm of Lin and Zhang, the cost exceeds that of our adaptation of Longa and Gebotys's algorithm shown above. Thus, the *Co-Z point arithmetic* based algorithms do not always provide better performance when compared with those constructed using only *Conjugate addition arithmetic*, at least not in the two cases studied by the authors in [71].

Chapter 6

Pairing based cryptography

6.1 Introduction

Pairing based cryptography was first introduced by Joux in his one round Tripartite Diffie-Hellman key exchange scheme [56]. The Weil and the Tate pairings are two well-known examples of pairings which are usually computed using Miller's algorithm, that was first described in 1986 and subsequently published in 2004 [78]. Stange [100] proposed an alternate algorithm to compute the Tate pairing by using Elliptic Nets which are a generalization of integer sequences satisfying certain properties that were first studied by Ward [104].

While Miller's algorithm and Stange's Elliptic Net algorithm are both $O(\log n)$ algorithms, the Elliptic Net algorithm is slower, owing to a difference in the constants, though it is only somewhat slower than an optimized Miller's algorithm, especially at higher embedding degrees [100]. There are numerous papers published on the optimization of Miller's algorithm [32], however, there has not been much research published in the literature to optimize Stange's algorithm. This motivates the need to consider optimizations of Stange's algorithm. In this chapter, we provide an improved version of Stange's algorithm to compute the Tate pairing. This improvement may not make Stange's method faster than that of Miller's, but is an improvement

worth considering, as Stange's algorithm is the only viable alternative to Miller's algorithm for Pairing computation. This improvement is also applicable to
(a) an algorithm in [59] that computes elliptic curve scalar multiplication using an adapted version of Stange's algorithm and
(b) the improved version of Kanayama's version in [24].

In [110], the authors propose efficient formulae and algorithms for point arithmetic on a new model of elliptic curves called Selmer Curves. They also provide an algorithm for Tate pairing on these curves. We provide an improved algorithm for point arithmetic and Tate pairing on Selmer curves.

6.2 Stange's Elliptic Net Algorithm to compute the Tate Pairing

Elliptic divisibility sequences are integer sequences $h_0, h_1, h_2, ..., h_n$, satisfying the following two properties:

1. For all positive integers $m > n$,

$$h_{m+n}h_{m-n} = h_{m+1}h_{m-1}h_n^2 - h_{n+1}h_{n-1}h_m^2 \tag{6.1}$$

2. h_n divides h_m whenever n divides m.

6.2.1 Stange's Algorithm for Tate Pairing

Before we outline Stange's algorithm, we must define the Tate pairing. Let E be an elliptic curve defined over a field L containing the m-th roots of unity, where $m \in \mathbb{Z}$. Let $E(L)[m] = \{P \in E(L) | mP = \mathcal{O}\}$ and $mE(L) = \{mP | P \in E(L)\}$. Further let $P \in E(L)[m]$ and $Q \in E(L)/mE(L)$. Since $mP = \mathcal{O}$, there is a rational function f with divisor $div(f_P) = m(P) - m(\mathcal{O})$. If we choose another divisor D_Q defined over L such that $D_Q \sim (Q) - (\mathcal{O})$

and with support disjoint from $div(f_P)$, the Tate pairing is the mapping

$_{Tm} : E(L)[m] \times E(L)/mE(L) \to L^*/(L^*)^m$ defined by $_{Tm}(P, Q) = f_P(D_Q)$.

The Tate pairing as well as the Weil Pairing can be computed using the Miller's algorithm and is a $O(\log m)$ algorithm. As stated above, whilst constructing an algorithm to compute the Tate pairing using Elliptic Nets, Stange provided a theorem useful in constructing the Tate pairing, which we repeat below for convenience:

Theorem A [100]: *Fix a positive $m \in \mathbb{Z}$. Let E be an elliptic curve defined over a finite field L containing the m-th roots of unity. Let $P, Q \in E(L)$, with $[m]P = \mathcal{O}$. Choose $S \in E(L)$ such that $S \notin \{\mathcal{O}, Q\}$. Then there exists an elliptic net $W : \mathbb{Z}^\eta \to L$ and $\mathbf{p}$, $\mathbf{q}$, $\mathbf{s} \in \mathbb{Z}^\eta$ such that the quantity*

$$T_m(P, Q) = \frac{W(\mathbf{s} + m\mathbf{p} + \mathbf{q})W(\mathbf{s})}{W(\mathbf{s} + m\mathbf{p})W(\mathbf{s} + \mathbf{q})}$$

is exactly the Tate pairing $T_m =_{Tm} : E(L)[m] \times E(L)/mE(L) \to L^*/(L^*)^m$.

In the above, the elliptic net W, introduced by Stange as a generalization of Elliptic divisibility sequences, is a map $W : A \to R$ satisfying the following recurrence relation for $p, q, r, s \in A$, where R is an integral domain and A is a finitely generated free abelian group.

$$\begin{aligned} &W(p+q+s)W(p-q)W(r+s)W(r) \\ &\quad + W(q+r+s)W(q-r)W(p+s)W(p) \\ &\quad + W(r+p+s)W(r-p)W(q+s)W(q) = 0 \ . \end{aligned} \tag{6.2}$$

Further Stange provides the following result

Theorem B [100]: *Let E be an elliptic curve defined over a finite field L, m a positive integer, $P \in E(L)[m]$ and $Q \in E(L)$. If W_p is the elliptic net associated to E, P, then*

$$_{Tm}(P, P) = \frac{W_P(m+2)W_P(1)}{W_P(m+1)W_P(2)} \ .$$

Further if $W_{P,Q}$ is the elliptic net associated to E, P, Q, then

$$_{Tm}(P,Q) = \frac{W_{P,Q}(m+1,1)W_{P,Q}(1,0)}{W_{P,Q}(m+1,0)W_{P,Q}(1,1)} \ . \tag{6.3}$$

Using Shipsey's algorithm [89] for computing terms of an elliptic divisibility sequence and the above theorems, Stange provides a scheme that can be used to compute the Tate pairing by calculating the terms $W(m,0)$ and $W(m,1)$ of an elliptic net. Stange's scheme is a $O(\log m)$ algorithm and can be outlined as follows:

Let a block centered on k consist of the following two vectors:

(i) 8 consecutive terms of the sequence $W(i,0)$ centered on terms $W(k,0)$ and $W(k+1,0)$ called the first vector and

(ii) 3 consecutive terms $W(i,1)$ centered on the term $W(k,1)$ called the second vector.

		(k-1,1)	(k,1)	(k+1,1)			
(k-3,0)	(k-2,0)	(k-1,0)	(k,0)	(k+1,0)	(k+2,0)	(k+3,0)	(k+4,0)

Figure 6.1: Block centred on k

For a block V centered on k, Stange proposes two algorithms, *Double(V)*, that constructs the block centered on $2k$ and *DoubleAdd(V)* that constructs the block centered on $2k+1$. While the first vectors of *Double(V)* and *DoubleAdd(V)* are calculated in terms of $W(2,0)$ and the terms of V, using the following instances of (6.2), where $i = k-1, \ldots, k+3$

$$W(2i-1,0) = W(i+1,0)W(i-1,0)^3 - W(i-2,0)W(i,0)^3 \text{ and} \tag{6.4}$$

$$W(2i,0) = (W(i,0)W(i+2,0)W(i-1,0)^2 - W(i,0)W(i-2,0)W(i+1,0)^2)/W(2,0) \ . \tag{6.5}$$

The second vectors are computed, in terms of $W(1,1), W(1,1), W(2,1)$ and

the terms of V, using the following instances of (6.2) below:

$$W(2k-1,1) = (W(k+1,1)W(k-1,1)W(k-1,0)^2 - W(k,0)W(k-2,0)W(k,1)^2)/W(1,1), \tag{6.6}$$

$$W(2k,1) = W(k-1,1)W(k+1,1)W(k,0)^2 - W(k-1,0)W(k+1,0)W(k,1)2, \tag{6.7}$$

$$W(2k+1,1) = (W(k-1,1)W(k+1,1)W(k+1,0)^2 - W(k,0)W(k+2,0)W(k,1)2)/W(-1,1), \tag{6.8}$$

$$W(2k+2,1) = (W(k+1,0)W(k+3,0)W(k,1)^2 - W(k-1,1)W(k+1,1)W(k+2,0)^2)/W(2,-1). \tag{6.9}$$

Algorithm 6.1 below calculates $W(m,1)$ and $W(m,0)$ for any positive integer m.

Algorithm 6.1: Elliptic Net Algorithm

INPUT: Initial terms $a = W(2,0), b = W(3,0), c = W(4,0), d = W(2,1)$, $e = W(-1,1), f = W(2,-1), g = W(1,1)$ of an elliptic net satisfying $W(1,0) = W(0,1) = 1$ and integer $m = (d_k d_{k-1} \dots d_0)_2$ with $d_k = 1$
OUTPUT: Elliptic net elements $W(m,0)$ and $W(m,1)$.

1: $V \leftarrow [[-a,-1,0,1,a,b,c,a^3c-b^3];[1,g,d]]$
2: **for** $i = k-1$ down to 1 **do**
3: **if** $d_i = 0$ **then**
4: $V \leftarrow Double(V)$
5: **else**
6: $V \leftarrow DoubleAdd(V)$
7: **end if**
8: **end for**
9: **return** $V[0,3], V[1,1]$ //terms $W(m,0)$ and $W(m,1)$ respectively

---

The Tate pairing is now computed using Equation (6.3). We recall that a Weierstrass form elliptic curve E over a finite field $\mathbb{F}_q$ of char not 2 or 3 is given by

$$y^2 = x^3 + Ax + B$$

and points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ on $E(\mathbb{F}_q)$ with $Q \neq \pm P$, the values of a, b, c, d, e, f, g must be calculated as required inputs for the Elliptic Net Algorithm, which are the terms of the elliptic net associated to E, P, Q. The necessary formulae are given by the functions $\Psi_{m,0}$ called *division polynomials* ([90, p. 105] and [91, p. 477]). We have

$$W(1,0) = 1, \tag{6.10}$$
$$W(2,0) = 2y_1, \tag{6.11}$$
$$W(3,0) = 3x_1^4 + 6Ax_1^2 + 12Bx_1 - A^2, \tag{6.12}$$
$$W(4,0) = 4y_1(x_1^6 + 5Ax_1^4 + 20Bx_1^3 - 5A^2x_1^2 - 4ABx_1 - 8B^2 - A^3), \tag{6.13}$$
$$W(0,1) = W(1,1) = 1, \tag{6.14}$$
$$W(2,1) = 2x_1 + x_2 - \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 \tag{6.15}$$
$$W(-1,1) = x_1 - x_2 \text{ and} \tag{6.16}$$
$$W(2,-1) = (y_1 + y_2)^2 - (2x_1 + x_2)(x_1 - x_2)^2. \tag{6.17}$$

If P has order m and if a, b, c, d, e, f, g are given by (6.11) – (6.17) above, the output of Algorithm 6.1 can be used to compute the Tate pairing using Equation (6.3) above. Factoring out some common subexpressions that may occur frequently, Stange provides an optimised version of the Double and DoubleAdd algorithm as follows:

Algorithm 6.2: Double and DoubleAdd

INPUT: Block V centred at k of an elliptic net satisfying
$W(1,0) = W(0,1) = 1$, values $\alpha = W(2,0)^{-1}, E = W(-1,1)^{-1}$,
$F = W(2,-1)^{-1}, G = W(1,1)^{-1}$ and boolean add
OUTPUT: Block centered at $2k$ if $add = 0$ and centred at $2k+1$ if $add = 1$.

1. $S \leftarrow [0,0,0,0,0,0]$
2. $P \leftarrow [0,0,0,0,0,0]$
3. $S_0 \leftarrow V[1,1]^2$
4. $P_0 \leftarrow V[1,0]V[1,2]$
5: **for** $i = 0$ to 5 **do**
6: $S[i] \leftarrow V[0,i+1]^2$
7: $P[i] \leftarrow V[0,i]V[0,i+2]$
8: **end for**
9: **if** $add == 0$ **then**
10: **for** $i = 1$ to 4 **do**
11: $V[0,2i-2] \leftarrow S[i-1]P[i] - S[i]P[i-1]$
12: $V[0,2i-1] \leftarrow (S[i-1]P[i+1] - S[i+1]P[i-1])\alpha$
13: **end for**
14. $V[1,0] \leftarrow (S_0P[3] - S[3]P_0)G$
15. $V[1,1] \leftarrow S[3]P_0 - S_0P[3]$
16. $V[1,2] \leftarrow (S[4]P_0 - S_0P[4])E$
17: **else**
18: **for** $i = 1$ to 4 **do**
19: $V[0,2i-2] \leftarrow (S[i-1]P[i+1] - S[i+1]P[i-1])\alpha$
20: $V[0,2i-1] \leftarrow S[i]P[i+1] - S[i+1]P[i]$
21: **end for**
22. $V[1,0] \leftarrow S[3]P_0 - S_0P[3]$
23. $V[1,1] \leftarrow (S[4]P_0 - S_0P[4])E$
24. $V[1,2] \leftarrow (S_0P[5] - S[5]P_0)F$
25: **end if**
26: **return** V

6.2.2 Improvement to Stange's Algorithm

Stange calculates the cost of the Double step in the above scheme to be $6S + (6n + 26)M + S_n + \frac{3}{2}M_n$, while that of the DoubleAdd steps is $6S + (6n + 26)M + S_n + 2M_n$ where M and S are the costs of a multiplication and squaring in $\mathbb{F}_q$ respectively while M_n and S_n are the costs of a multiplication and squaring in $\mathbb{F}_{q^n}$ respectively. Here for the integer m and finite field $\mathbb{F}_q$, the embedding degree n is the least integer such that $m|(q^n - 1)$. Usually for the Tate pairing, a curve is defined over $\mathbb{F}_q$ of embedding degree $n > 1$, while $P \in E(\mathbb{F}_q)$ and $Q \in E(\mathbb{F}_{q^n})$. Now Lines 10-13 of Algorithm 6.2 costs $4 * 5M = 20M$ and can be replaced with the following block of code

$$
\begin{aligned}
&A \leftarrow (P[1] + P[2])(S[1] - S[2]);\\
&B \leftarrow (P[1] - P[2])(S[1] + S[2]);\\
&C \leftarrow (P[1] + P[3])(S[1] - S[3]);\\
&D \leftarrow (P[1] - P[3])(S[1] + S[3]);\\
&E \leftarrow 2(P[2] - P[3])(S[2] + S[3]);\\
&F \leftarrow (P[3] + P[4])(S[3] - S[4]);\\
&G \leftarrow (P[3] - P[4])(S[3] + S[4]);\\
&H \leftarrow (P[3] + P[5])(S[3] - S[5]);\\
&I \leftarrow (P[3] - P[5])(S[3] + S[5]);\\
&J \leftarrow 2(P[4] - P[5])(S[4] + S[5]);\\
&V[0,0] \leftarrow (A - B)/2;\\
&V[0,1] \leftarrow (C - D)\alpha/2;\\
&V[0,2] \leftarrow ((C + D) - (A + B + E))/2;\\
&V[0,3] \leftarrow (S[2]P[4] - S[4]P[2])\alpha;\\
&V[0,4] \leftarrow (F - G)/2;\\
&V[0,5] \leftarrow (H - I)\alpha/2;\\
&V[0,6] \leftarrow ((H + I) - (F + G + J))/2;\\
&V[0,7] \leftarrow (S[4]P[6] - S[6]P[4])\alpha;
\end{aligned}
$$

The above block of code costs $18M$, thus saving us $2M$. Similarly, Lines 18-21 can be replaced with the following block of code which again costs $18M$:

$$
\begin{aligned}
&A \leftarrow (P[2] + P[3])(S[2] - S[3]);\\
&B \leftarrow (P[2] - P[3])(S[2] + S[3]);\\
&C \leftarrow (P[2] + P[4])(S[2] - S[4]);\\
&D \leftarrow (P[2] - P[4])(S[2] + S[4]);\\
&E \leftarrow 2(P[3] - P[4])(S[3] + S[4]);\\
&F \leftarrow (P[4] + P[5])(S[4] - S[5]);\\
&G \leftarrow (P[4] - P[5])(S[4] + S[5]);\\
&H \leftarrow (P[4] + P[6])(S[4] - S[6]);\\
&I \leftarrow (P[4] - P[6])(S[4] + S[6]);\\
&J \leftarrow 2(P[5] - P[6])(S[5] + S[6]);\\
&V[0,0] \leftarrow (S[1]P[3] - S[3]P[1])\alpha;\\
&V[0,1] \leftarrow (A - B)/2;\\
&V[0,2] \leftarrow (C - D)\alpha/2;\\
&V[0,3] \leftarrow ((C + D) - (A + B + E))/2;\\
&V[0,4] \leftarrow (S[3]P[5] - S[5]P[3])\alpha;\\
&V[0,5] \leftarrow (F - G)/2;\\
&V[0,6] \leftarrow (H - I)\alpha/2;\\
&V[0,7] \leftarrow ((H + I) - (F + G + J))/2;
\end{aligned}
$$

By inspection, we can see that the two blocks of code above costs $18M$ each. Thus the cost of the double step in Stange's algorithm can be reduced to $6S + (6n + 24)M + S_n + \frac{3}{2}M_n$, (typically $n \leq 12$) while that of the DoubleAdd step can be reduced to $6S + (6n + 24)M + S_n + 2M_n$, as summarized in the table below.

Algorithm for Tate Pairing	Double	DoubleAdd
Optimised Millers [64]	$4S + (n + 7)M + S_n + M_n$	$7S+(2n+19)M+S_n+2M_n$
Elliptic Net Algorithm [100]	$6S+(6n+26)M+S_n+\frac{3}{2}M_n$	$6S+(6n+26)M+S_n+2M_n$
Improved Elliptic Net Algorithm [96]	$6S+(6n+24)M+S_n+\frac{3}{2}M_n$	$6S+(6n+24)M+S_n+2M_n$

While this improvement does not make the elliptic net algorithm competitive with the Miller's algorithm, it is an improvement worth considering as the elliptic net algorithm is the only viable alternative to the Miller's algorithm. A comparison of the elliptic net algorithm with Miller's algorithm for a range of values for the embedding degree n can be found in [101].

Stange's algorithm was adapted by Kanayama et al to compute elliptic curve scalar multiplication [59]. Their Double and DoubleAdd steps costs $26M + 6S$ and this can be reduced to $24M + 6S$ as a result of adapting the optimization to Stange's algorithm presented in this chapter. Kanayama's algorithm was then further optimized by Chen at al in [24] using one of the optimizations outlined by Stange, which was not utilised by the authors in [59]. Using this optimization and then replacing four multiplications with four squarings in Stange's algorithm, the authors in [59] reduce the cost of the both the Double and DoubleAdd steps in Kanayama's algorithm to $18M + 10S$. Using our optimization in this chapter, the cost of the Double and DoubleAdd steps can be reduced to $16M + 10S$ each. These costs are summarized in the table below:

Elliptic Net Algorithm for Scalar Multiplication	Double	DoubleAdd
Kanayama's Algorithm [59]	$26M + 6S$	$26M + 6S$
Improvement due to Chen [100]	$18M + 10S$	$18M + 10S$
Further Improvement [96]	$16M + 10S$	$16M + 10S$

6.3 Selmer Curves

We recall the definition of Selmer curves from Section 2.5

If K be a field of char $\neq 2$ or 3, a Selmer curve over K is defined by a homogeneous cubic equation of the form $aX^3 + bY^3 = cZ^3$, or in affine coordinates, $ax^3 + by^3 = c$, where $a, b, c \in K$ and $abc \neq 0$.

6.3.1 Point Arithmetic on Selmer Curves

Assume $P_1 = (X_1 : Y_1 : Z_1)$, $P_2 = (X_2 : Y_2 : Z_2)$ and let
$P_1 + P_2 = P_3 = (X_3 : Y_3 : Z_3)$, then

$$X_3 = X_1 Z_1 Y_2^2 - X_2 Z_2 Y_1^2$$
$$Y_3 = Y_1 Z_1 X_2^2 - Y_2 Z_2 X_1^2$$
$$Z_3 = X_1 Y_1 Z_2^2 - X_2 Y_2 Z_1^2$$

Below, we replicate the algorithm to compute the above formula as provided in [110], which costs $12M$.

$A = X_1 Y_2;\ B = X_2 Y_1;\ C = Y_1 Z_2;\ D = Y_2 Z_1;\ E = Z_1 X_2;\ F = Z_2 X_1;$
$X_3 = AD - BC;\ Y_3 = BE - AF;\ Z_3 = CF - DE.$

Now, we provide the formulae and algorithm for point doubling on Selmer curves as given in [110]. Assume $P_1 = (X_1 : Y_1 : Z_1)$, $2P_1 = P_3 = (X_3 : Y_3 : Z_3)$, then

$$X_3 = Y_1(2X_1^3 + Y_1^3)$$
$$Y_3 = X_1(X_1^3 + 2Y_1^3)$$
$$Z_3 = Z_1(X_1^3 - Y_1^3).$$

The above doubling formula can be computed using the following algorithm, which costs $5M + 2S$:

$A = X_1^2;\ B = Y_1^2;\ C = AX_1;\ D = BY_1;$
$X_3 = Y_1(2C + D);\ Y_3 = X_1(C + 2D);\ Z_3 = Z_1(C - D).$

Below we provide a new algorithm for point addition on Selmer Curves:

$A = X_1 Y_2;\ B = X_2 Y_1;\ C = Y_1 Z_2;\ D = Y_2 Z_1;\ E = Z_1 X_2;\ F = Z_2 X_1;$
$G = (B + D)(A - C);\ H = (B - D)(A + C);\ I = (B + F)(A - E);$
$J = (B - F)(A + E);\ K = (D - F)(C + E);$
$2X_3 = (G - H);\ 2Y_3 = (J - I);\ \ 2Z_3 = (I + J) - (G + H + 2K)\ .$

This algorithm costs $11M$, thus saving us $1M$. It should be noted that we have actually computed $2X_3, 2Y_3, 2Z_3$. If $Z_2 = 1$, then the new algorithm costs $9M$.

6.3.2 Cost of Tate Pairing on Selmer Curves

The authors in [110] use Miller's algorithm to compute the Tate pairing on Selmer curves. Using the same notation as in the previous section, the total cost of a Miller addition step (ADD) as given in [110] is $M_n + (n+12)M$, where M_n and M denote multiplication in $\mathbb{F}_{p^n}$ and $\mathbb{F}_p$ respectively. If $Z_2 = 1$ then the authors in [110] show that the cost of mixed Miller addition (mADD) is reduced to $M_n + (n+10)M$. If we use the new algorithm for point addition presented above, then the new total cost of a Miller addition step (ADD) is $M_n + (n+11)M$. If $Z_2 = 1$, then the new cost of mixed Miller addition (mADD) is $M_n + (n+9)M$. The authors in [110] show that Selmer curves are very competitive with the fastest formulae, by comparing the cost of computing the Tate pairing on other forms of Elliptic curves. They summarize the cost of Tate pairing computation as shown in the table below, whilst not including the common cost $1M_n + nM$ in Miller addition step and $1M_n + 1S + nM$ in Miller doubling step. T_1 is the scenario when $S = 0.8M$ and T_2 is the scenario when $S = M$. As per their analysis, computation of Tate pairing would be the fastest on Selmer curves in the T_2 scenario. With our new algorithm in this chapter[96], computation of Tate pairing on Selmer curves would be the fastest under both scenarios, T_1 and T_2.

Table 6.1: **Comparison of costs of various algorithms to compute the Tate Pairing**

	DBL	T_1	T_2	mADD	T_1	T_2
$\mathcal{J}$, [4][53]	$1\mathbf{m} + 11\mathbf{s} + 1\mathbf{m}_d$	$9.8\mathbf{m}$	$12\mathbf{m}$	$6\mathbf{m} + 6s$	$10.8\mathbf{m}$	$12\mathbf{m}$
$\mathcal{J}, a = -3$, [4]	$6\mathbf{m} + 5s$	$10\mathbf{m}$	$11\mathbf{m}$	$6\mathbf{m} + 6s$	$10.8\mathbf{m}$	$12\mathbf{m}$
$\mathcal{J}, a = 0$,[4]	$3\mathbf{m} + 8s$	$9.4\mathbf{m}$	$11\mathbf{m}$	$6\mathbf{m} + 6s$	$10.8\mathbf{m}$	$12\mathbf{m}$
$\mathcal{P}, a = 0, b = c^2$,[33]	$3\mathbf{m} + 5s$	$7\mathbf{m}$	$8\mathbf{m}$	$9\mathbf{m} + 2s + 1\mathbf{m}_d$	$10.6\mathbf{m}$	$11\mathbf{m}$
$\mathcal{E}$,[47]	$6\mathbf{m} + 5s$	$10\mathbf{m}$	$11\mathbf{m}$	$12\mathbf{m}$	$12\mathbf{m}$	$12\mathbf{m}$
$\mathcal{H}$, [50]	$3\mathbf{m} + 6s + 3\mathbf{m}_d$	$7.8\mathbf{m}$	$9\mathbf{m}$	$10\mathbf{m}$	$10\mathbf{m}$	$10\mathbf{m}$
$\mathcal{H}u$, [57]	$11\mathbf{m} + 6s$	$15.8\mathbf{m}$	$17\mathbf{m}$	$13\mathbf{m}$	$13\mathbf{m}$	$13\mathbf{m}$
$\mathcal{J}a$, [111]	$4\mathbf{m} + 8s + 1\mathbf{m}_d$	$10.4\mathbf{m}$	$12\mathbf{m}$	$16\mathbf{m} + 1s + 4\mathbf{m}_d$	$16.8\mathbf{m}$	$17\mathbf{m}$
$\mathcal{S}$, [110]	$5\mathbf{m} + 3s$	$7.4\mathbf{m}$	$8\mathbf{m}$	$10\mathbf{m}$	$10\mathbf{m}$	$10\mathbf{m}$
$\mathcal{S}$, [96]	$5\mathbf{m} + 3s$	$7.4\mathbf{m}$	$8\mathbf{m}$	$9\mathbf{m}$	$9\mathbf{m}$	$9\mathbf{m}$

$\mathcal{J}$: Weierstrass curves (Jacobian coordinates), $\mathcal{E}$: Edwards curves,
$\mathcal{P}$: Weierstrass curves (projective coordinates), $\mathcal{H}$: Hessian curves,
$\mathcal{H}u$: Huff curves, $\mathcal{J}a$: Jacobi quartic curves, $\mathcal{S}$: Selmer curves.
T_1: the total cost of DBL or mADD when s = $0.8\mathbf{m}$
T_2: the total cost of DBL or mADD when s = $\mathbf{m}$

Chapter 7

Some Results Arising from Karatsuba Multiplication

We know that improving Finite Field Arithmetic is crucial to improving implementations of elliptic curve cryptography. One of the frequently used arithmetic operations in cryptography is multiplication and any optimization of this operation facilitates the faster execution of cryptographic algorithms. The literature is abundant with ideas to multiply two integers in a variety of ways - for instance, *Karatsuba's algorithm*, the *Toom-Cook family* and fast multiplication via *FFT*. Though Toom-Cook or the FFT methods are more efficient asymptotically, compared to Karatsuba's algorithm, it is beneficial to extend and optimize the latter as it outperforms the former on number sizes used in current day cryptosystems [28].

The similarity between multi-precision arithmetic and polynomial arithmetic makes it easy to adapt Karatsuba's algorithm to multiply polynomials. Indeed, it is simpler in the context of polynomial multiplication because of the lack of *carries*. Karatsuba's algorithm has been generalized by Weimerskirch and Paar [105] to multiply two polynomials of arbitrary degree. In [82], Montgomery gives formulae to multiply polynomials of degree 2, degree 4, degree 5 and degree 6. Montgomery's formulae for degree 5 and degree 6 require fewer multiplications and additions and are more efficient than the Karatsuba generalizations obtained by Weimerskirch and Paar. Though the origin of the Montgomery family of formulae for degree 2 polynomials appears cryptic at first, they can be derived from Karatsuba's algorithm as shown in this chapter. We also present new families of

formulae to multiply degree 2 polynomials, distinct from the Montgomery family.

7.1 Review of Karatsuba's Algorithm

Karatsuba's algorithm performs the multiplication of two numbers of size $2n$ using three multiplications of two numbers of size n each. The algorithm is applied recursively until n is sufficiently small. Thus the time complexity is $O(n^{\alpha})$ where $\alpha = \ln 3/\ln 2 \simeq 1.57$. An interesting history of this algorithm, along with Kolmogorov's conjecture appears in [61]. This algorithm devised by Karatsuba in 1963 appears to be the first multiplication algorithm that has sub-quadratic complexity.

If two numbers A and B of the same length are to be multiplied, then these two numbers can be split as follows:

$$A = xA_1 + A_0 \qquad \text{and} \qquad B = xB_1 + B_0$$

where x is a power of the base in use (for binary numbers, the base is 2) and x is about half the length of the two numbers. The conventional school book multiplication algorithm requires 4 multiplications and 3 additions. If $A(x) = xA_1 + A_0$ and $B(x) = xB_1 + B_0$ were polynomials over a ring R, then computing $A(x)B(x)$ would require 4 multiplications and 1 addition.

$$A(x)B(x) = x^2A_1B_1 + x(A_0B_1 + A_1B_0) + A_0B_0$$

Karatsuba's algorithm makes use of the fact that AB can be written as

$$A(x)B(x) = x^2A_1B_1 + x[(A_1 + A_0)(B_0 + B_1) - A_0B_0 - A_1B_1] + A_0B_0 \quad (7.1)$$

or

$$A(x)B(x) = x^2A_1B_1 + x[(A_1 - A_0)(B_0 - B_1) + A_0B_0 + A_1B_1] + A_0B_0 \quad (7.2)$$

The Karatsuba's algorithm(referred to as 2-way KA hereafter) requires 3 multiplications and 4 additions.

The 3-way KA multiplies two degree-2 polynomials as follows

$$A(x) = x^2A_2 + xA_1 + A_0 \qquad \text{and} \qquad B(x) = x^2B_2 + xB_1 + B_0$$

By the conventional school book multiplication method, AB can be written as

$$A(x)B(x) = x^4A_2B_2 + x^3(A_2B_1 + A_1B_2) + x^2(A_2B_0 + A_1B_1 + A_0B_2) + x(A_1B_0 + A_0B_1) + A_0B_0$$

This method requires 9 multiplications and 4 additions.

A Karatsuba-like formula[82] can be written as follows

$$\begin{aligned} A(x)B(x) = {} & x^4A_2B_2 + x^3[(A_1 + A_2)(B_1 + B_2) - A_1B_1 - A_2B_2] \\ & + x^2[(A_0 + A_2)(B_0 + B_2) - A_2B_2 - A_0B_0 + A_1B_1] \\ & + x[(A_0 + A_1)(B_0 + B_1) - A_1B_1 - A_0B_0] + A_0B_0 \end{aligned} \quad (7.3)$$

or

$$\begin{aligned} A(x)B(x) = {} & x^4A_2B_2 + x^3[(A_2 - A_1)(B_1 - B_2) + A_1B_1 + A_2B_2] \\ & + x^2[(A_2 - A_0)(B_0 - B_2) + A_2B_2 + A_0B_0 + A_1B_1] \\ & + x[(A_1 - A_0)(B_0 - B_1) + A_1B_1 + A_0B_0] + A_0B_0 \end{aligned} \quad (7.4)$$

The above set of formulae ((7.3) and (7.4)) are the 3-way KA formula and require 6 multiplications and 13 additions.

7.2 3-way KA from 2-way KA

Chung and Hassan [26] depict the 3-way KA as a special case of Montgomery's family of 3-way multiplication algorithms. They further suggest that the 3-way KA does not appear to be a special case of Toom-Cook even though, many fast multiplication algorithms are related to the Toom-Cook multiplication algorithm. The Toom-Cook algorithm relies on the fact that the product of two polynomials $A(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0$ and $B(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1x + b_0$ given by $C(x) = A(x)B(x)$ is recovered by its values at $(2n-1)$ distinct values of x. Here we show that the 3-way KA can in fact be derived by repeated applications of the 2-way KA. We know that

$$\begin{aligned} A(x)B(x) = {} & x^4A_2B_2 + x^3(A_2B_1 + A_1B_2) \\ & + x^2(A_2B_0 + A_1B_1 + A_0B_2) + x(A_1B_0 + A_0B_1) + A_0B_0 \end{aligned} \quad (7.5)$$

The 2-way KA relies on the fact that $A_0B_1 + A_1B_0$, can be written as

$$(A_1 - A_0)(B_0 - B_1) + A_0B_0 + A_1B_1$$

In general, the 2-way KA is based on the following identity:

$$A_iB_j + A_jB_i = (A_i - A_j)(B_j - B_i) + A_jB_j + A_iB_i \tag{7.6}$$

Using (7.6), Equation (7.5) can be written as (in this section, the term underlined by the bracket is rewritten repeatedly using the above identity)

$$\begin{aligned}
A(x)B(x) &= x^4A_2B_2 + x^3\underbrace{(A_2B_1 + A_1B_2)} + x^2(A_2B_0 + A_1B_1 + A_0B_2) \\
&\qquad + x(A_1B_0 + A_0B_1) + A_0B_0 \\
&= (x^4 + x^3)A_2B_2 + x^3[(A_2 - A_1)(B_1 - B_2)] + (x^2 + x^3)A_1B_1 \\
&\qquad + x^2\underbrace{(A_2B_0 + A_0B_2)} + x(A_1B_0 + A_0B_1) + A_0B_0 \\
&= (x^4 + x^3)A_2B_2 + x^3[(A_2 - A_1)(B_1 - B_2)] + (x^2 + x^3)A_1B_1 \\
&\qquad + x^2[(A_2 - A_0)(B_0 - B_2)] \\
&\qquad + x^2A_2B_2 + x\underbrace{(A_1B_0 + A_0B_1)} + A_0B_0 + x^2A_0B_0 \\
&= (x^4 + x^3 + x^2)A_2B_2 + x^3[(A_2 - A_1)(B_1 - B_2)] + (x^3 + x^2)A_1B_1 \\
&\qquad + x^2[(A_2 - A_0)(B_0 - B_2)] + (x^2 + 1)A_0B_0 \\
&\qquad + x[(A_1 - A_0)(B_0 - B_1)] + xA_1B_1 + xA_0B_0 \\
&= (x^4 + x^3 + x^2)A_2B_2 + x^3[(A_2 - A_1)(B_1 - B_2)] + (x^3 + x^2 + x)A_1B_1 \\
&\qquad + x^2[(A_2 - A_0)(B_0 - B_2)] + (x^2 + x + 1)A_0B_0 \\
&\qquad + x[(A_1 - A_0)(B_0 - B_1)] \\
&= x^4A_2B_2 + x^3[(A_2 - A_1)(B_1 - B_2) + A_2B_2 + A_1B_1] \\
&\qquad + x^2[(A_2 - A_0)(B_0 - B_2) + A_2B_2 + A_0B_0 + A_1B_1] \\
&\qquad + x[(A_1 - A_0)(B_0 - B_1) + A_0B_0 + A_1B_1] + A_0B_0
\end{aligned} \tag{7.7}$$

Thus the 3-way KA is the result of repeated applications of 2-way KA.
In fact, the entire Montgomery family of formulae to multiply two polynomials of degree 2, can be obtained from the Karatsuba algorithm. For this, we first show the extension of Karatsuba's algorithm to multiply polynomials of degree 3.

If $A(x) = (x^3A_3 + x^2A_2 + xA_1 + A_0)$ and $B(x) = (x^3B_3 + x^2B_2 + xB_1 + B_0)$ then,

$$\begin{aligned} A(x)B(x) &= (x^3A_3 + x^2A_2 + xA_1 + A_0)(x^3B_3 + x^2B_2 + xB_1 + B_0) \\ &= [x^2(xA_3 + A_2) + (xA_1 + A_0)][x^2(xB_3 + B_2) + (xB_1 + B_0)] \\ &= (x^2M_1 + M_0)(x^2N_1 + N_0) \end{aligned}$$

where $M_1 = xA_3 + A_2, M_0 = xA_1 + A_0, N_1 = xB_3 + B_2, N_0 = xB_1 + B_0$

$$\begin{aligned} &= x^4M_1N_1 + x^2(\underbrace{M_1N_0 + M_0N_1}) + M_0N_0 \\ &= x^4M_1N_1 + x^2[(M_1 + M_0)(N_0 + N_1) - M_0N_0 - M_1N_1] + M_0N_0 \end{aligned}$$

Now

$$\begin{aligned} M_0N_0 &= (xA_1 + A_0)(xA_1 + B_0) \\ &= x^2A_1B_1 + x(\underbrace{A_0B_1 + B_0A_1}) + A_0B_0 \\ &= x^2A_1B_1 + x[(A_0 + A_1)(B_1 + B_0) - A_0B_0 - A_1B_1] + A_0B_0 \end{aligned}$$

Also

$$\begin{aligned} (M_1 + M_0)(N_0 + N_1) &= (xA_3 + A_2 + xA_1 + A_0)(xB_3 + B_2 + xB_1 + B_0) \\ &= [x(A_1 + A_3) + (A_0 + A_2)][x(B_1 + B_3) + (B_0 + B_2)] \\ &= x^2(A_1 + A_3)(B_1 + B_3) \\ &\quad + x[\underbrace{(A_1 + A_3)(B_0 + B_2) + (B_1 + B_3)(A_0 + A_2)}] \\ &\quad + (A_0 + A_2)(B_0 + B_2) \end{aligned} \tag{7.8}$$

$$\begin{aligned} &= x^2(A_1 + A_3)(B_1 + B_3) + x[(A_1 + A_3) + (A_0 + A_2)][(B_0 + B_2) + (B_1 + B_3)] \\ &\quad - x(A_1 + A_3)(B_1 + B_3) - x(A_0 + A_2)(B_0 + B_2) + (A_0 + A_2)(B_0 + B_2) \end{aligned}$$

M_1N_1 remains to be evaluated:

$$\begin{aligned} M_1N_1 = (xA_3 + A_2)(xB_3 + B_2) &= x^2A_3B_3 + x(\underbrace{A_3B_2 + A_2B_3}) + A_2B_2 \\ &= x^2A_3B_3 + x[(A_3 + A_2)(B_2 + B_3) - A_2B_2 - A_3B_3] + A_2B_2 \end{aligned}$$

Defining

$$\begin{aligned}
D_0 &= A_0B_0 & D_4 &= (A_0 + A_1)(B_1 + B_0)\\
D_1 &= A_1B_1 & D_5 &= (A_0 + A_2)(B_2 + B_0)\\
D_2 &= A_2B_2 & D_6 &= (A_1 + A_3)(B_3 + B_1)\\
D_3 &= A_3B_3 & D_7 &= (A_3 + A_2)(B_2 + B_3)\\
D_8 &= [(A_1 + A_3) + (A_0 + A_2)][(B_1 + B_3) + (B_0 + B_2)]
\end{aligned}$$

we can write

$$\begin{aligned}
M_0N_0 &= x^2D_1 + x(D_4 - D_0 - D_1) + D_0\\
M_1N_1 &= x^2D_3 + x(D_7 - D_2 - D_3) + D_2\\
\text{and } (M_1 + M_0)(N_0 + N_1) &= x^2D_6 + x(D_8 - D_6 - D_5) + D_5
\end{aligned}$$

The product AB can now be written as

$$\begin{aligned}
A(x)B(x) =& x^4[x^2D_3 + x(D_7 - D_2 - D_3) + D_2]\\
&+ x^2\{[x^2D_6 + x(D_8 - D_6 - D_5) + D_5]\\
&\quad - [x^2D_1 + x(D_4 - D_0 - D_1) + D_0]\\
&\quad\quad - [x^2D_3 + x(D_7 - D_2 - D_3) + D_2]\}\\
&+ [x^2D_1 + x(D_4 - D_0 - D_1) + D_0]
\end{aligned}$$

$$\begin{aligned}
A(x)B(x) = x^6D_3 &+ x^5(D_7 - D_2 - D_3) + x^4(D_6 - D_1 - D_3 + D_2)\\
&+ x^3(D_8 - D_6 - D_5 - D_4 + D_0 + D_1 + D_2 - D_7 + D_3)\\
&+ x^2(D_5 - D_0 - D_2 + D_1) + x(D_4 - D_0 - D_1) + D_0
\end{aligned} \tag{7.9}$$

The above method obtained from 2-way KA can be called 4-way KA and was known to Winograd in 1980 [107]. The 4-way KA is effectively a formula to multiply two polynomials of degree 3.

Another special case of Montgomery's family of formulae for multiplying quadratics can be obtained from the 4-way KA by assigning $A_3 = 0$ and $B_3 = 0$. Thus equation (7.9) reduces to

$$\begin{aligned}
A(x)B(x) = x^4D_2 &+ x^3(D_8 - D_5 - D_4 + D_0)\\
&+ x^2(D_5 - D_0 - D_2 + D_1) + x(D_4 - D_0 - D_1) + D_0
\end{aligned}$$

because

$$\begin{aligned} D_3 &= A_3B_3 = 0 \\ D_7 &= (A_3 + A_2)(B_2 + B_3) = A_2B_2 = D_2 \\ D_6 &= (A_1 + A_3)(B_3 + B_1) = A_1B_1 = D_1 \end{aligned}$$

i.e.,

$$\begin{aligned} A(x)B(x) = x^4D_2 + x^3[&(A_0 + A_1 + A_2)(B_0 + B_1 + B_2) \\ &- (A_0 + A_2)(B_0 + B_2) - (A_0 + A_1)(B_1 + B_0) + A_0B_0 \\ &+ x^2[(A_0 + A_2)(B_2 + B_0) - A_0B_0 - A_2B_2 + A_1B_1] \\ &+ x[(A_0 + A_1)(B_1 + B_0) - A_0B_0 - A_1B_1] + A_0B_0] \end{aligned}$$

which is a special case of Montgomery's family of formula. Now we have the identity

$$\begin{aligned} A_0B_0 + A_1B_1 + A_2B_2 - (A_0 + A_1)(B_0 + B_1) - (A_0 + A_2)(B_0 + B_2) \\ - (A_1 + A_2)(B_1 + B_2) + (A_0 + A_1 + A_2)(B_0 + B_1 + B_2) = 0 \end{aligned} \quad (7.10)$$

Multiplying (7.10) by any polynomial C with integer coefficients, and adding it to (7.3), we obtain the entire Montgomery family of formulae for multiplying 2 quadratics.

7.3 New Family of Formulae to multiply two quadratics

A new family of formulae to multiply two quadratics is as follows [95]:

$$\begin{aligned} A(x)B(x) = &A_0B_0(C + 1 - x^2 - x) \\ &+ A_1B_1(3C - x + x^2 - x^3) \\ &+ A_2B_2(C + x^4 - x^3 - x^2) \\ &+ (A_0 + A_1)(B_0 + B_1)(-C + x) \\ &+ (A_0 + A_2)(B_0 + B_2)(C + x^2) \\ &+ (A_1 + A_2)(B_1 + B_2)(-C + x^3) \\ &+ (A_1 - A_0 - A_2)(B_0 + B_2 - B_1)C \end{aligned} \quad (7.11)$$

This follows from

$$\begin{aligned} A(x)B(x) = x^4 A_2B_2 &+ x^3[(A_1 - A_0 - A_2)(B_0 + B_2 - B_1) + (A_0 + A_2)(B_2 + B_0) \\ &- (A_0 + A_1)(B_0 + B_1) + A_0B_0 + 2A_1B_1] \\ &+ x^2[(A_0 + A_2)(B_0 + B_2) - A_2B_2 + A_1B_1 - A_0B_0] \\ &+ x[(A_0 + A_1)(B_0 + B_1) - A_1B_1 - A_0B_0] + A_0B_0 \end{aligned} \tag{7.12}$$

and

$$\begin{aligned} [A_0B_0 + 3A_1B_1 + A_2B_2 &- (A_0 + A_1)(B_0 + B_1) + (A_0 + A_2)(B_0 + B_2) \\ &- (A_1 + A_2)(B_1 + B_2) + (A_1 - A_0 - A_2)(B_0 + B_2 - B_1)] = 0 \end{aligned} \tag{7.13}$$

Multiplying equation (7.13) by C and adding it to (7.4), we obtain the above family of formulae.
Equation(7.12) is obtained by writing $(A_1 + A_3)(B_0 + B_2) + (B_4 + B_3)(A_0 + A_2)$ as $[(A_1 + A_3) - (A_0 + A_2)][(B_0 + B_2) - (B_1 + B_2)] + (A_1 + A_3)(B_1 + B_2) + (A_0 + A_2)(B_0 + B_2)$ in equation (7.8).
Yet another family of formulae to multiply two quadratics is

$$\begin{aligned} A(x)B(x) = &A_0B_0(C + 1 + x + x^2) \\ &+ A_1B_1(-C + x + x^2 + x^3) \\ &+ A_2B_2(-C + x^2 + x^3 + x^4) \\ &+ (A_1 - A_0)(B_0 - B_1)(-C + x) \\ &+ (A_0 - A_2)(B_2 - B_0)(C + x^2) \\ &+ (A_1 - A_2)(B_2 - B_1)(-C + x^3) \\ &+ (A_0 - A_1 - A_2)(B_0 - B_1 - B_2)C \ , \end{aligned} \tag{7.14}$$

where C is any polynomial with integer coefficients.
The efficiency of the new family of formulae introduced in this section to multiply two quadratics is similar to that of the Montgomery family of formulae to multiply two quadratics [82].

7.4 Extension of 2-way KA and 3-way KA to multiply three numbers

The 2-way KA can be extended to multiply 3 integers. If A, B and C are written as $A = xA_1 + A_0, B = xB_1 + B_0, C = xC_1 + C_0$ then

$$\begin{aligned} ABC &= (xA_1 + A_0)(xB_1 + B_0)(xC_1 + C_0) \\ &= x^3A_1B_1C_1 + x^2(A_1B_1C_0 + A_1B_0C_1 + A_0B_1C_1) \\ &\quad + x(A_1B_0C_0 + A_0B_0C_1 + A_0B_1C_0) + A_0B_0C_0 \ . \end{aligned} \tag{7.15}$$

The above conventional school book algorithm requires 8 multiplications and 4 additions.
Using the 2-way KA, we can write the product ABC as

$$ABC = x^3D_0 + x^2[D_0 + D_1 + D_2 + D_3] + x[D_1 + D_4 + D_5 + D_2] + D_5 \tag{7.16}$$

where

$$\begin{aligned} D_0 &= A_1B_1C_1, & D_3 &= (A_1 - A_0)(B_0 - B_1)C_1, \\ D_1 &= A_1B_1C_0, & D_4 &= (A_1 - A_0)(B_0 - B_1)C_0, \\ D_2 &= A_0B_0C_1, & D_5 &= A_0B_0C_0, \end{aligned}$$

and this requires 6 multiplications and 8 additions. The above equation can also be written as

$$ABC = x^3D_0 + x^2[D_5 + 2D_0 + D_3 + D_6] + x[D_0 + 2D_5 + D_4 + D_6] + D_5 \tag{7.17}$$

where $D_6 = (A_0B_0 - A_1B_1)(C_1 - C_0)$

The product ABC can also be computed by first computing AB and then computing ABC. The product ABC can be written as

$$\begin{aligned} ABC &= \underbrace{(x^2A_1B_1 + x(A_0B_1 + A_1B_0) + A_0B_0)}_{\text{requires 3 multiplications and 4 additions using 2-way KA}} \quad (xC_1 + C_0) \\ &= \underbrace{x^2A_1B_1(xC_1 + C_0)}_{\text{requires 2 multiplications}} + \underbrace{\Big\{x(A_0B_1 + A_1B_0) + A_0B_0\Big\}(xC_1 + C_0)}_{\text{requires 3 multiplications and 4 additions using 2-way KA}} \end{aligned}$$

Thus ABC can be computed using 8 multiplications and 9 additions which is more expensive than 6 multiplications and 8 additions that is incurred using the

three-term product computed using equation (7.16).

Using equation (7.17) to multiply two numbers, of size $2n$ each, one would require 3 multiplications of two numbers of size n each and 5 multiplications of two numbers, where one number is of size $2n$ and the other of size n.

Multiplication of two numbers each of size n	Multiplication of two numbers (one of size $2n$ and other of size n)
$G_0 = A_0B_0$	$D_5 = G_0C_0$
$G_1 = A_1B_1$	$D_0 = G_1C_1$
$G_{01} = (A_1 - A_0)(B_0 - B_1)$	$D_4 = G_{01}C_0$
	$D_3 = G_{01}C_1$
	$D_6 = (G_0 - G_1)(C_1 - C_0)$

The 3-way KA also can be extended to multiply 3 integers. If A, B and C can be written as

$$\begin{aligned} A &= x^2A_2 + xA_1 + A_0 \\ B &= x^2B_2 + xB_1 + B_0 \\ C &= x^2C_2 + xC_1 + C_0 \end{aligned}$$

then

$$\begin{aligned} ABC &= (x^2A_2 + xA_1 + A_0)(x^2B_2 + xB_1 + B_0)(x^2C_2 + xC_1 + C_0) \\ &= x^6A_2B_2C_2 + x^5(A_2B_2C_1 + A_2B_1C_2 + A_1B_2C_2) \\ &\quad + x^4(A_2B_2C_0 + A_2B_1C_1 + A_2B_0C_2 \\ &\qquad + A_1B_2C_1 + A_1B_1C_2 + A_0B_2C_2) \\ &\quad + x^3(A_2B_1C_0 + A_2B_0C_1 + A_1B_2C_0 \\ &\qquad + A_1B_1C_1 + A_1B_0C_2 + A_0B_2C_1 + A_0B_1C_2) \\ &\quad + x^2(A_2B_0C_0 + A_1B_1C_0 + A_1B_0C_1 + A_0B_2C_0 + A_0B_1C_1 + A_0B_0C_2) \\ &\quad + x(A_1B_0C_0 + A_0B_1C_0 + A_0B_0C_1) + A_0B_0C_0 \ . \end{aligned} \tag{7.18}$$

The above conventional school book multiplication algorithm needs 27 multiplications and 20 additions. If we define the auxiliary products

$$
\begin{aligned}
D_0 &= A_0B_0C_0 & D_9 &= (A_2 - A_0)(B_0 - B_2)C_2 \\
D_1 &= A_0B_0C_1 & D_{10} &= (A_2 - A_0)(B_0 - B_2)C_1 \\
D_2 &= A_0B_0C_2 & D_{11} &= (A_2 - A_0)(B_0 - B_2)C_0 \\
D_3 &= A_1B_1C_0 & D_{12} &= (A_2 - A_1)(B_1 - B_2)C_2 \\
D_4 &= A_1B_1C_1 & D_{13} &= (A_2 - A_1)(B_1 - B_2)C_1 \\
D_5 &= A_1B_1C_2 & D_{14} &= (A_2 - A_1)(B_1 - B_2)C_0 \\
D_6 &= A_2B_2C_0 & D_{15} &= (A_1 - A_0)(B_0 - B_1)C_2 \\
D_7 &= A_2B_2C_1 & D_{16} &= (A_1 - A_0)(B_0 - B_1)C_1 \\
D_8 &= A_2B_2C_2 & D_{17} &= (A_1 - A_0)(B_0 - B_1)C_0
\end{aligned}
$$

then

$$
\begin{aligned}
ABC =& x^6D_8 + x^5(D_8 + D_{12} + D_5 + D_7) \\
&+ x^4(D_9 + D_8 + D_2 + D_5 + D_{13} + D_4 + D_6 + D_7) \\
&+ x^3(D_{10} + D_2 + D_{15} + D_{14} + D_5 + D_7 + D_1 + D_6 + D_3 + D_4) \\
&+ x^2(D_{11} + D_2 + D_1 + D_6 + D_{16} + D_4 + D_0 + D_3) \\
&+ x(D_{17} + D_3 + D_0 + D_1) + D_0 \ .
\end{aligned} \tag{7.19}
$$

This requires 18 multiplications and 29 additions. In the above formula, the following pairs of auxiliary products can be combined: $(D_1, D_3), (D_2, D_6)$ and (D_5, D_7)

$$
\begin{aligned}
D_1 + D_3 &= (A_1B_1 - A_0B_0)(C_0 - C_1) + A_1B_1C_1 + A_0B_0C_0 \\
D_2 + D_6 &= (A_0B_0 - A_2B_2)(C_2 - C_0) + A_0B_0C_0 + A_2B_2C_2 \\
D_5 + D_7 &= (A_1B_1 - A_2B_2)(C_2 - C_1) + A_1B_1C_1 + A_2B_2C_2
\end{aligned}
$$

Assigning

$$
\begin{aligned}
E_{10} &= (A_1B_1 - A_0B_0)(C_0 - C_1) \\
E_{02} &= (A_0B_0 - A_2B_2)(C_2 - C_0) \\
E_{12} &= (A_1B_1 - A_2B_2)(C_2 - C_1)
\end{aligned}
$$

we have

$$\begin{aligned} D_1 + D_3 &= E_{10} + D_4 + D_0 \\ D_2 + D_6 &= E_{02} + D_0 + D_8 \\ D_5 + D_7 &= E_{12} + D_4 + D_8 \end{aligned}$$

Thus

$$\begin{aligned} ABC = x^6 D_8 &+ x^5[2D_8 + D_{12} + D_4 + E_{12}] \\ &+ x^4[D_9 + 3D_8 + E_{02} + D_0 + E_{12} + 2D_4 + D_{13}] \\ &+ x^3[D_{10} + E_{02} + 2D_0 + 2D_8 \\ &\qquad + D_{15} + D_{14} + E_{12} + 3D_4 + E_{10}] \\ &+ x^2[D_{11} + E_{02} + 3D_0 + D_8 + E_{10} + 2D_4 + D_{16}] \\ &+ x[D_{17} + E_{10} + D_4 + 2D_0] + D_0 \end{aligned} \tag{7.20}$$

Using this formula to multiply 3 numbers of size $3n$ each, one would require 12 multiplications of 3 numbers of size n each, 3 multiplications of 2 numbers, where one is of the size $2n$ and the other of size n and 3 multiplications of 2 numbers of size n each.

The above formula can also be realized using 6 multiplications of 2 numbers, each of size n and $(9 + 6)$ multiplications of two numbers, one of size $2n$ and other of size n, as shown in the table below

Multiplications of two numbers each of size n	Multiplication of two numbers (one of size $2n$ and other of size n)
$G_0 = A_0B_0$	$D_0 = G_0C_0$
$G_1 = A_1B_1$	$D_4 = G_1C_1$
$G_2 = A_2B_2$	$D_8 = G_2C_2$
$G_{02} = (A_2 - A_0)(B_0 - B_2)$	$D_9 = G_{02}C_2$
	$D_{10} = G_{02}C_1$
	$D_{11} = G_{02}C_0$
$G_{01} = (A_2 - A_1)(B_1 - B_2)$	$D_{12} = G_{01}C_2$
	$D_{13} = G_{01}C_1$
	$D_{14} = G_{01}C_0$
$G_{10} = (A_1 - A_0)(B_0 - B_1)$	$D_{15} = G_{10}C_2$
	$D_{16} = G_{10}C_1$
	$D_{17} = G_{10}C_0$
	$E_{10} = (G_1 - G_0)(C_0 - C_1)$
	$E_{02} = (G_0 - G_2)(C_2 - C_0)$
	$E_{12} = (G_1 - G_2)(C_2 - C_1)$

7.5 Extended Karatsuba Algorithm: Uses and comparison with the School Book Algorithm

Fast multiplication of 3 integers can be used in the following circumstances:

MultiPrime RSA: MultiPrime RSA is a variant of conventional RSA, where the modulus can be the product of more than two primes. i.e., $N = p_1, p_2 \dots p_r$ and $\phi(N) = (p_1 - 1)(p_2 - 1)(p_3 - 1) \dots (p_r - 1)$
Encryption and Decryption are the same as in conventional RSA. In [69, Section 4], the author compares the security levels offered by symmetric cryptosystems such as DES or AES with RSA. The author recommends that the number of factors of the MultiPrime RSA Modulus can range from 2 to 6 [Table 1,2 and 3 in [69]] and that they should more or less be of the same size. The extended versions of 2-way KA or 3-way KA to multiply 3 integers could be utilized to multiply the RSA factors.

Modular Arithmetic: The Chinese Reminder Theorem is extensively used in current day cryptography and may involve multiplication of 3 numbers or more. For instance, the CRT is used in various variants of RSA such as Batch RSA, MultiPower RSA, MultiPrime RSA etc [14]. The extended versions of KA to multiply 3 integers could be utilized in this situation as well.
Below, we compare the Karatsuba algorithm with conventional School Book Algorithm under different situations.

1. 2-way KA v/s School Book to multiply 2 degree-1 polynomials:

	#MUL	#ADD
School Book	4	1
2-way KA	3	4

If T_{MUL} and T_{ADD} can be taken to denote the time for one multiplication and one addition respectively, then the cost of the School Book Algorithm is $COST_{SB} = 4T_{MUL} + 3T_{ADD}$ and that of the Karatsuba Algorithm is $COST_{KA} = 3T_{MUL} + 4T_{ADD}$. The Karatsuba Algorithm is cheaper if $COST_{SB}$ is greater than $COST_{KA}$ and this would occur if T_{MUL}/T_{ADD} is > 3.

2. 3-way KA v/s School Book to multiply 2 degree-2 polynomials:

	#MUL	#ADD
School Book	9	4
3-way KA	6	13

In this case $COST_{SB}$ is greater than $COST_{KA}$ when $T_{MUL}/T_{ADD} > 3$.

3. 2-way KA v/s School Book to multiply 3 degree-1 polynomials:

	#MUL	#ADD
School Book	8	4
2-way KA	6	8

In this case $COST_{SB}$ is greater than $COST_{KA}$ when T_{MUL}/T_{ADD} is greater than 2.

4. 3-way KA v/s School Book to multiply 3 degree-2 polynomials:

	#MUL	#ADD
School Book	27	20
3-way KA	18	29

In this case $COST_{SB}$ is greater than $COST_{KA}$ when T_{MUL}/T_{ADD} is greater than 1.

Chapter 8

Conclusion

In this thesis, we looked at point arithmetic formulae and some scalar multiplication algorithms. We also considered double scalar multiplication algorithms in the context of differential addition chains and algorithms to compute pairings. Whilst doing so, we have touched upon a very small fraction of the literature in this area. Recalling Lang's comment(see Sec 1.2), it appears that it is possible to write endlessly on elliptic curve arithmetic and opportunities to improve this arithmetic.

In Chapter 2, we provided an introduction to elliptic curve arithmetic formulae and provided improved formulae for point quintupling. A systematic analysis of the minimum number of arithmetic operations to compute bilinear forms [107] may yield further improvements in the algorithms to perform elliptic curve arithmetic.

In Chapter 3, we provided differential point tripling formulae for Montgomery curves which can be used in Montgomery's PRAC algorithm which is an Euclidean chain algorithm. A topic for future research is to derive similar formulae for other forms of Elliptic curves with differential addition formulae. We also provided improved faster differential arithmetic algorithms for Edwards curves.

In Chapter 4, we provided 3-dimensional extensions of 2-dimensional binary differential chains. While we focused on left-to-right algorithms, one can work toward constructing 3-dimensional extensions of right-to-left algorithms. A topic for future research is the construction of 3-dimensional extensions of 2-dimensional Euclidean differential chains. We also observed that differential tripling can be

useful in Euclidean differential chains.

The precomputation schemes in Chapter 5 focus on double scalar multiplication and we showed that the Co-Z precomputation scheme due to Lin and Zhang is incorrect. We provided a new algorithm for precomputation. Further research can focus on extending these precomputation schemes to triple and higher dimensional precomputation schemes.

In Chapter 6, we provided an improvement of Stange's elliptic net algorithm to compute the Tate pairing. We also improved the computation of the Tate pairing on Selmer curves. Stange's elliptic net algorithm was based on Shipsey's double-and-add algorithm to compute terms of an elliptic divisibility sequence. It would be interesting to research the benefits of using a Euclidean chain algorithm to compute terms of an elliptic divisibility sequence.

Finally, we provide a new family of formulae to multiply two quadratics in Chapter 7. We provide an extension of 2-way KA and 3-way KA to multiply three numbers.

Appendix

Algorithm A.1: L-R 3-Dimensional Montgomery Ladder

INPUT: Points P, Q, R on E_m and positive integers k, l, u;
$k = (k_t \cdots k_1, k_0)_2, l = (l_t \cdots l_1, l_0)_2, u = (u_t \cdots u_1, u_0)_2$;
(at least one of k_t or l_t or $u_t = 1$).
Precompute $A \leftarrow (P + Q)$, $B \leftarrow (P - Q)$, $C \leftarrow (P + R)$, $D \leftarrow (P - R)$,
$E \leftarrow (Q + R)$, $F \leftarrow (Q - R)$, $G \leftarrow (P + Q + R)$, $H \leftarrow (P + Q - R)$;
OUTPUT: x coordinate of $W = kP + lQ + uR$.

[Initialize]
if $(k_t, l_t, u_t) = (0, 0, 1)$
$T_0 \leftarrow \mathcal{O}$, $T_1 \leftarrow R$, $T_2 \leftarrow E$, $T_3 \leftarrow C$, $T_4 \leftarrow G$;
else if $(k_t, l_t, u_t) = (0, 1, 0)$
$T_0 \leftarrow \mathcal{O}$, $T_1 \leftarrow Q$, $T_2 \leftarrow E$, $T_3 \leftarrow A$, $T_4 \leftarrow G$;
else if $(k_t, l_t, u_t) = (0, 1, 1)$
$T_0 \leftarrow \mathcal{O}$, $T_1 \leftarrow R$, $T_2 \leftarrow Q$, $T_3 \leftarrow E$, $T_4 \leftarrow G$;
else if $(k_t, l_t, u_t) = (1, 0, 0)$
$T_0 \leftarrow \mathcal{O}$, $T_1 \leftarrow P$, $T_2 \leftarrow C$, $T_3 \leftarrow A$, $T_4 \leftarrow G$;
else if $(k_t, l_t, u_t) = (1, 0, 1)$
$T_0 \leftarrow \mathcal{O}$, $T_1 \leftarrow R$, $T_2 \leftarrow P$, $T_3 \leftarrow C$, $T_4 \leftarrow G$;
else if $(k_t, l_t, u_t) = (1, 1, 0)$
$T_0 \leftarrow \mathcal{O}$, $T_1 \leftarrow Q$, $T_2 \leftarrow P$, $T_3 \leftarrow A$, $T_4 \leftarrow G$;

else if $(k_t, l_t, u_t) = (1, 1, 1)$
$T_0 \leftarrow R$, $T_1 \leftarrow E$, $T_2 \leftarrow C$, $T_3 \leftarrow A$, $T_4 \leftarrow G$;

[Process the three scalar bits simultaneously]
for i from t down to 1
$T_0\text{Tmp} \leftarrow T_0$, $T_1\text{Tmp} \leftarrow T_1$, $T_2\text{Tmp} \leftarrow T_2$, $T_3\text{Tmp} \leftarrow T_3$, $T_4\text{Tmp} \leftarrow T_4$;
if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 0, 0, 0, 0, 0)$
$T_0 \leftarrow 2T_0\text{Tmp}$, $T_1 \leftarrow T_1\text{Tmp}+T_0\text{Tmp}(\text{R})$,
$T_2 \leftarrow T_2\text{Tmp}+T_0\text{Tmp}(\text{Q})$, $T_3 \leftarrow T_3\text{Tmp}+T_0\text{Tmp}(\text{P})$,
$T_4 \leftarrow T_4\text{Tmp}+T_0\text{Tmp}(\text{A})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 0, 0, 0, 0, 1)$
$T_0 \leftarrow 2T_0\text{Tmp}$, $T_1 \leftarrow T_1\text{Tmp}+T_0\text{Tmp}(\text{R})$,
$T_2 \leftarrow T_2\text{Tmp}+T_1\text{Tmp}(\text{F})$, $T_3 \leftarrow T_3\text{Tmp}+T_1\text{Tmp}(\text{D})$,
$T_4 \leftarrow T_4\text{Tmp}+T_1\text{Tmp}(\text{H})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 0, 0, 0, 1, 0)$
$T_0 \leftarrow 2T_0\text{Tmp}$, $T_1 \leftarrow T_2\text{Tmp}+T_0\text{Tmp}(\text{Q})$,
$T_2 \leftarrow T_2\text{Tmp}+T_1\text{Tmp}(\text{F})$, $T_3 \leftarrow T_4\text{Tmp}+T_0\text{Tmp}(\text{A})$,
$T_4 \leftarrow T_4\text{Tmp}+T_1\text{Tmp}(\text{H})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 0, 0, 0, 1, 1)$
$T_0 \leftarrow 2T_0\text{Tmp}$, $T_1 \leftarrow T_1\text{Tmp}+T_0\text{Tmp}(\text{R})$,
$T_2 \leftarrow T_2\text{Tmp}+T_0\text{Tmp}(\text{Q})$, $T_3 \leftarrow T_2\text{Tmp}+T_1\text{Tmp}(\text{F})$,
$T_4 \leftarrow T_4\text{Tmp}+T_1\text{Tmp}(\text{H})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 0, 0, 1, 0, 0)$
$T_0 \leftarrow 2T_0\text{Tmp}$, $T_1 \leftarrow T_3\text{Tmp}+T_0\text{Tmp}(\text{P})$,
$T_2 \leftarrow T_3\text{Tmp}+T_1\text{Tmp}(\text{D})$, $T_3 \leftarrow T_4\text{Tmp}+T_0\text{Tmp}(\text{A})$,
$T_4 \leftarrow T_4\text{Tmp}+T_1\text{Tmp}(\text{H})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 0, 0, 1, 0, 1)$
$T_0 \leftarrow 2T_0\text{Tmp}$, $T_1 \leftarrow T_1\text{Tmp}+T_0\text{Tmp}(\text{R})$,
$T_2 \leftarrow T_3\text{Tmp}+T_0\text{Tmp}(\text{P})$, $T_3 \leftarrow T_3\text{Tmp}+T_1\text{Tmp}(\text{D})$,
$T_4 \leftarrow T_4\text{Tmp}+T_1\text{Tmp}(\text{H})$;

else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,0,0,1,1,0)$
$T_0 \leftarrow 2T_0\mathrm{Tmp}$, $T_1 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{Q})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{P})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{H})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,0,0,1,1,1)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{R})$, $T_1 \leftarrow T_2\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{F})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{D})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{H})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,0,1,0,0,0)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{R})$, $T_1 \leftarrow 2T_1\mathrm{Tmp}$,
$T_2 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}$,(E), $T_3 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,0,1,0,0,1)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{R})$, $T_1 \leftarrow 2T_1\mathrm{Tmp}$,
$T_2 \leftarrow T_2\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{Q})$, $T_3 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$,
$T_4 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{B})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,0,1,0,1,0)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{R})$, $T_1 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$,
$T_2 \leftarrow T_2\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{Q})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_4 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{B})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,0,1,0,1,1)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{R})$, $T_1 \leftarrow 2T_1\mathrm{Tmp}$,
$T_2 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$, $T_3 \leftarrow T_2\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{Q})$,
$T_4 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{B})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,0,1,1,0,0)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{R})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$, $\quad T_3 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_4 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{B})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,0,1,1,0,1)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{R})$, $T_1 \leftarrow 2T_1\mathrm{Tmp}$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$, $T_3 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$,
$T_4 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{B})$;

else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,0,1,1,1,0)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{R})$, $T_1 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_4 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{B})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,0,1,1,1,1)$
$T_0 \leftarrow 2T_1\mathrm{Tmp}$, $T_1 \leftarrow T_2\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{Q})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_4 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{B})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,1,0,0,0,0)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{Q})$, $T_1 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$,
$T_2 \leftarrow 2T_1\mathrm{Tmp}$, $T_3 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$,
$T_4 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{C})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,1,0,0,0,1)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{Q})$, $T_1 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$,
$T_2 \leftarrow T_2\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{R})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{C})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,1,0,0,1,0)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{Q})$, $T_1 \leftarrow 2T_1\mathrm{Tmp}$,
$T_2 \leftarrow T_2\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{R})$, $T_3 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{C})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,1,0,0,1,1)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{Q})$, $T_1 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$,
$T_2 \leftarrow 2T_1\mathrm{Tmp}$, $T_3 \leftarrow T_2\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{R})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{C})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,1,0,1,0,0)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{Q})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$, $T_3 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{C})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0,1,0,1,0,1)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{Q})$, $T_1 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{C})$;

else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 1, 0, 1, 1, 0)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{Q})$, $T_1 \leftarrow 2T_1\mathrm{Tmp}$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$, $T_3 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{C})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 1, 0, 1, 1, 1)$
$T_0 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$, $T_1 \leftarrow T_2\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{R})$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$, $T_3 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{C})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 1, 1, 0, 0, 0)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{Q})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{R})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{C})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 1, 1, 0, 0, 1)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{Q})$,
$T_2 \leftarrow 2T_3\mathrm{Tmp}$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{A})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{P})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 1, 1, 0, 1, 0)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{R})$,
$T_2 \leftarrow 2T_3\mathrm{Tmp}$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{C})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{P})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 1, 1, 0, 1, 1)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{Q})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{R})$, $T_3 \leftarrow 2T_3\mathrm{Tmp}$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{P})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 1, 1, 1, 0, 0)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$, $T_1 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{A})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{C})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{P})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 1, 1, 1, 0, 1)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{Q})$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{A})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{P})$;

else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 1, 1, 1, 1, 0)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{E})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{R})$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{C})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{P})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (0, 1, 1, 1, 1, 1)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{Q})$, $T_1 \leftarrow 2T_3\mathrm{Tmp}$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{A})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{C})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{P})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 0, 0, 0, 0)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{P})$, $T_1 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$, $T_3 \leftarrow 2T_1\mathrm{Tmp}$,
$T_4 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{Q})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 0, 0, 0, 1)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{P})$, $T_1 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$, $T_3 \leftarrow T_2\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{R})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{E})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 0, 0, 1, 0)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{P})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$, $T_3 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{Q})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{E})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 0, 0, 1, 1)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{P})$, $T_1 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{E})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 0, 1, 0, 0)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{P})$, $T_1 \leftarrow 2T_1\mathrm{Tmp}$,
$T_2 \leftarrow T_2\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{R})$, $T_3 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{Q})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{E})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 0, 1, 0, 1)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{P})$, $T_1 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$,
$T_2 \leftarrow 2T_1\mathrm{Tmp}$, $T_3 \leftarrow T_2\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{R})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{E})$;

else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 0, 1, 1, 0)$
$T_0 \leftarrow T_1\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{P})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$,
$T_2 \leftarrow 2T_1\mathrm{Tmp}$, $T_3 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{Q})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{E})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 0, 1, 1, 1)$
$T_0 \leftarrow T_2\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$, $T_1 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_2 \leftarrow T_2\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{R})$, $T_3 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{Q})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{E})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 1, 0, 0, 0)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$, $T_3 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{R})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{E})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 1, 0, 0, 1)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{A})$, $T_3 \leftarrow 2T_3\mathrm{Tmp}$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{Q})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 1, 0, 1, 0)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$, $T_1 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{A})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{E})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{Q})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 1, 0, 1, 1)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{A})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{Q})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 1, 1, 0, 0)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{R})$,
$T_2 \leftarrow 2T_3\mathrm{Tmp}$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{E})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{Q})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 1, 1, 0, 1)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{R})$, $T_3 \leftarrow 2T_3\mathrm{Tmp}$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{Q})$;

else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 1, 1, 1, 0)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{C})$, $T_1 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{R})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{E})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{Q})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 0, 1, 1, 1, 1)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$, $T_1 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{A})$,
$T_2 \leftarrow 2T_3\mathrm{Tmp}$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{E})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{Q})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 1, 0, 0, 0, 0)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$, $T_1 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$, $T_3 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{Q})$,
$T_4 \leftarrow 2T_3\mathrm{Tmp}$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 1, 0, 0, 0, 1)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$, $T_1 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{C})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{E})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{R})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 1, 0, 0, 1, 0)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{C})$, $T_3 \leftarrow 2T_3\mathrm{Tmp}$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{R})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 1, 0, 0, 1, 1)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$, $T_1 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{P})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_1\mathrm{Tmp}(\mathrm{C})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{R})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 1, 0, 1, 0, 0)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$, $T_1 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{Q})$,
$T_2 \leftarrow T_4\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{E})$, $T_3 \leftarrow 2T_3\mathrm{Tmp}$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{R})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 1, 0, 1, 0, 1)$
$T_0 \leftarrow T_3\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{A})$, $T_1 \leftarrow T_4\mathrm{Tmp}{+}T_0\mathrm{Tmp}(\mathrm{G})$,
$T_2 \leftarrow T_3\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{Q})$, $T_3 \leftarrow T_4\mathrm{Tmp}{+}T_2\mathrm{Tmp}(\mathrm{E})$,
$T_4 \leftarrow T_4\mathrm{Tmp}{+}T_3\mathrm{Tmp}(\mathrm{R})$;

else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 1, 0, 1, 1, 0)$
$T_0 \leftarrow T_3\text{Tmp}+T_0\text{Tmp}(\text{A})$, $T_1 \leftarrow T_3\text{Tmp}+T_1\text{Tmp}(\text{P})$,
$T_2 \leftarrow T_3\text{Tmp}+T_2\text{Tmp}(\text{Q})$, $T_3 \leftarrow 2T_3\text{Tmp}$,
$T_4 \leftarrow T_4\text{Tmp}+T_3\text{Tmp}(\text{R})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 1, 0, 1, 1, 1)$
$T_0 \leftarrow T_4\text{Tmp}+T_0\text{Tmp}(\text{G})$, $T_1 \leftarrow T_4\text{Tmp}+T1\text{Tmp}(\text{C})$,
$T_2 \leftarrow T_4\text{Tmp}+T_2\text{Tmp}(\text{E})$, $T_3 \leftarrow 2T_3\text{Tmp}$,
$T_4 \leftarrow T_4\text{Tmp}+T_3\text{Tmp}(\text{R})$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 1, 1, 0, 0, 0)$
$T_0 \leftarrow T_3\text{Tmp}+T_0\text{Tmp}(\text{H})$, $T_1 \leftarrow T_4\text{Tmp}+T_0\text{Tmp}(\text{A})$,
$T_2 \leftarrow T_3\text{Tmp}+T_1\text{Tmp}(\text{D})$, $T_3 \leftarrow T_3\text{Tmp}+T_2\text{Tmp}(\text{F})$,
$T_4 \leftarrow T_4\text{Tmp}+T_3\text{Tmp}(\text{R})$
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 1, 1, 0, 0, 1)$
$T_0 \leftarrow T_3\text{Tmp}+T_0\text{Tmp}(\text{H})$, $T_1 \leftarrow T_4\text{Tmp}+T_0\text{Tmp}(\text{A})$,
$T_2 \leftarrow T_4\text{Tmp}+T_1\text{Tmp}(\text{P})$, $T_3 \leftarrow T_4\text{Tmp}+T_2\text{Tmp}(\text{Q})$,
$T_4 \leftarrow 2T_4\text{Tmp}$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 1, 1, 0, 1, 0)$
$T_0 \leftarrow T_3\text{Tmp}+T_0\text{Tmp}(\text{H})$, $T_1 \leftarrow T_3\text{Tmp}+T_1\text{Tmp}(\text{D})$,
$T_2 \leftarrow T_4\text{Tmp}+T_1\text{Tmp}(\text{P})$, $T_3 \leftarrow T_4\text{Tmp}+T_3\text{Tmp}(\text{R})$,
$T_4 \leftarrow 2T_4\text{Tmp}$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 1, 1, 0, 1, 1)$
$T_0 \leftarrow T_3\text{Tmp}+T_0\text{Tmp}(\text{H})$, $T_1 \leftarrow T_4\text{Tmp}+T_0\text{Tmp}(\text{A})$,
$T_2 \leftarrow T_3\text{Tmp}+T_1\text{Tmp}(\text{D})$, $T_3 \leftarrow T_4\text{Tmp}+T_1\text{Tmp}(\text{P})$,
$T_4 \leftarrow 2T_4\text{Tmp}$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 1, 1, 1, 0, 0)$
$T_0 \leftarrow T_3\text{Tmp}+T_0\text{Tmp}(\text{H})$, $T_1 \leftarrow T_3\text{Tmp}+T_2\text{Tmp}(\text{F})$,
$T_2 \leftarrow T_4\text{Tmp}+T_2\text{Tmp}(\text{Q})$, $T_3 \leftarrow T_4\text{Tmp}+T_3\text{Tmp}(\text{R})$,
$T_4 \leftarrow 2T_4\text{Tmp}$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1, 1, 1, 1, 0, 1)$
$T_0 \leftarrow T_3\text{Tmp}+T_0\text{Tmp}(\text{H})$, $T_1 \leftarrow T_4\text{Tmp}+T_0\text{Tmp}(\text{A})$,
$T_2 \leftarrow T_3\text{Tmp}+T_2\text{Tmp}(\text{F})$, $T_3 \leftarrow T_4\text{Tmp}+T_2\text{Tmp}(\text{Q})$,
$T_4 \leftarrow 2T_4\text{Tmp}$;

else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1,1,1,1,1,0)$
$T_0 \leftarrow T_3\mathrm{Tmp}+T_0\mathrm{Tmp}(\mathrm{H})$, $T_1 \leftarrow T_3\mathrm{Tmp}+T_1\mathrm{Tmp}(\mathrm{D})$,
$T_2 \leftarrow T_3\mathrm{Tmp}+T_2\mathrm{Tmp}(\mathrm{F})$, $T_3 \leftarrow T_4\mathrm{Tmp}+T_3\mathrm{Tmp}(\mathrm{R})$,
$T_4 \leftarrow 2T_4\mathrm{Tmp}$;
else if $(k_i, l_i, u_i, k_{i-1}, l_{i-1}, u_{i-1}) = (1,1,1,1,1,1)$
$T_0 \leftarrow T_4\mathrm{Tmp}+T_0\mathrm{Tmp}(\mathrm{A})$, $T_1 \leftarrow T_4\mathrm{Tmp}+T_1\mathrm{Tmp}(\mathrm{P})$,
$T_2 \leftarrow T_4\mathrm{Tmp}+T_2\mathrm{Tmp}(\mathrm{Q})$, $T_3 \leftarrow T_4\mathrm{Tmp}+T_3\mathrm{Tmp}(\mathrm{R})$,
$T_4 \leftarrow 2T_4\mathrm{Tmp}$;
end for

[Finalize]
if $(k_0, l_0, u_0) = (0,0,0)$
$W \leftarrow 2T_0$
else if $(k_0, l_0, u_0) = (0,0,1)$
$W \leftarrow T_1 + T_0$ (R)
else if $(k_0, l_0, u_0) = (0,1,0)$
$W \leftarrow T_1 + T_0$ (Q)
else if $(k_0, l_0, u_0) = (0,1,1)$
$W \leftarrow T_3 + T_0$ (E)
else if $(k_0, l_0, u_0) = (1,0,0)$
$W \leftarrow T_1 + T_0$ (P)
else if $(k_0, l_0, u_0) = (1,0,1)$
$W \leftarrow T_3 + T_0$ (C)
else if $(k_0, l_0, u_0) = (1,1,0)$
$W \leftarrow T_3 + T_0$ (A)
else if $(k_0, l_0, u_0) = (1,1,1)$
$W \leftarrow T_3 + T_0$ (H)
Return $x-$coordinate of W by computing $x = X/Z$

Bibliography

[1] R. Abarzua and N. Theriault, *Complete Atomic Blocks for Elliptic Curves in Jacobian Coordinates over Prime Fields*, Proceedings of LATINCRYPT 2012, LNCS volume 7533, pp. 37-55.

[2] T. Akishita, *Fast Simultaneous Scalar Multiplication on Elliptic Curve with Montgomery Form*, Selected Areas in Cryptology(SAC) 2001, LNCS volume 2259, pp. 255-267.

[3] A. Antipa, D. Brown, R. Gallant, R. Lambert, R. Struik and S. Vanstone, *Accelerated Verification of ECDSA Signatures*, `http://cacr.uwaterloo.ca/techreports/2005/cacr2005-28.pdf`, Accessed: 2nd Feb 2016.

[4] C. Arène, T. Lange, M. Naehrig and C. Ritzenthaler, *Faster Computation of the Tate Pairing*, `https://eprint.iacr.org/2009/155.pdf`, Accessed: 2nd Feb 2016.

[5] R. Azarderakhsh and K. Karabina, *A New Double Point Multiplication Method and its Implementation on Binary Elliptic Curves with Endomorphisms*, `http://cacr.uwaterloo.ca/techreports/2012/cacr2012-24.pdf`, Accessed: 2nd Feb 2016.

[6] R. Barbulescu, P. Gaudry, A. Joux, E. Thome, *A Heuristic Quasi-Polynomial Algorithm for Discrete Logarithm in Finite Fields of Small Characteristic*, Advances in Cryptology - EUROCRYPT 2014, LNCS 8441, pp 1-16.

[7] R. Bellman and E.G. Straus, *Addition Chains of Vectors (problem 5125)*, The American Mathematical Monthly 1964, 71, pp. 806-808.

[8] D.J. Bernstein, *Curve25519: New Diffie Hellman Speed Records* 2006, `https://cr.yp.to/ecdh/curve25519-20060209.pdf`, Accessed: 2nd Feb 2016.

[9] D.J. Bernstein, *Differential Addition Chains* 2006, `https://cr.yp.to/ecdh/diffchain-20060219.pdf`, Accessed: 2nd Feb 2016.

[10] D.J. Bernstein, P. Birkner, M. Joye, T. Lange, C. Peters, *Twisted Edwards Curves*, Progress in Cryptology, AFRICACRYPT 2008, LNCS 5023, pp. 389-405.

[11] D.J. Bernstein, P. Birkner, T. Lange and C. Peters, *Optimizing Double-base Elliptic Curve Single-Scalar Multiplication*, INDOCRYPT 2007, LNCS volume 4859, pp 167-182.

[12] D.J. Bernstein and T. Lange *Faster Addition and Doubling on Elliptic Curves*, ASIACRYPT 2007, LNCS volume 4833, pp 29-50.

[13] D.J. Bernstein, T. Lange and R.R. Farashahi, *Binary Edwards Curves*, Cryptographic Hardware and Embedded Systems (CHES) 2008, LNCS volume 5154, pp. 244-265, 2008.

[14] D. Boneh and H. Shacham, *Fast Variants of RSA*, CryptoBytes 2002, RSA Laboratories, 5, No 1 Winter/Spring 2002.

[15] R.P. Brent, *Some integer factorization algorithms using elliptic curves*, Australian Computer Science Communications 8 (1986), `http://maths-people.anu.edu.au/~brent/pub/pub102.html`

[16] R.P. Brent, *Factorization of the Tenth Fermat number*, Mathematics of Computation 1999, volume 68, pp. 429-451.

[17] R.P. Brent and P. Zimmermann, *Modern Computer Arithmetic*, Cambridge University Press 2011.

[18] E. Brier and M. Joye, *Weistrass Elliptic Curves and Side Channel Attacks*, Public Key Cryptography 2002, LNCS volume 2274, pp. 335-345.

[19] D.R.L. Brown, *Multi-dimensional Montgomery Ladders for Elliptic Curves 2006*, `http://eprint.iacr.org/2006/220`, Accessed: 25th Jan 2015.

[20] D.R.L. Brown, *Multi-dimensional Montgomery Ladders for Elliptic Curves 2006*, Patent No.US8750500 B2, `http://www.google.com/patents/US8750500`, Published 2014.

[21] E. Brown, *Three Fermat Trails to Elliptic Curves*, The College Mathematics Journal, Mathematical Association of America 2000, 3(31), pp. 162-172.

[22] E. Brown and B.T. Myers, *Elliptic Curves from Mordell to Diophantus and Back*, American Mathematical Monthly 2002, 109, pp. 639-649.

[23] W. Castryek, S. Galbraith and R. Farashahi, *Efficient Arithmetic on Elliptic Curves using a mixed Edwards-Montgomery representation*, 2008, `https://eprint.iacr.org/2008/218.pdf`.

[24] B. Chen, C. Hu and C. Zhao, *A Note on Scalar Multiplication Using Division Polynomials*, 2015, `https://eprint.iacr.org/2015/284.pdf`, Accessed: 29th Jun 2016.

[25] J.H. Cheon and J.H. Yi, *Fast Batch Verification of Multiple Signatures*, Public Key Cryptography(PKC) 2007, LNCS volume 4450, pp. 442-457.

[26] J. Chung and A. Hassan, *Asymmetric Squaring Formulae*, Technical Report, University of Waterloo, CACR, 2006.

[27] M. Ciet, M. Joye, K. Lauter and P.L. Montgomery, *Trading Inversions for Multiplications in Elliptic Curve Cryptography*, Designs Codes and Cryptography, volume 39, 2006, pp. 189-206.

[28] H. Cohen and G. Frey, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, 2006, CRC Press.

[29] H. Cohen, A. Miyaji and T. Ono, *Efficient Elliptic Curve Exponentiation Using Mixed Coordinates*, ASIACRYPT 1998, LNCS volume 1514, pp. 51-65.

[30] I. Connell, *Elliptic Curve Handbook*, 1999, `https://pendientedemigracion.ucm.es/BUCM/mat/doc8354.pdf`, Accessed: 4th Feb 2016.

[31] J. Coron, *Resistance Against Differential Power Analysis For Elliptic Curve Cryptosystems*, Cryptographic Hardware and Embedded Systems (CHES) 1999, LNCS volume 1717, pp. 292-302.

[32] C. Costello, *Faster Formulas for Computing Cryptographic Pairings*, 2012, PhD thesis, Queensland University of Technology.

[33] C. Costello, T. Lange and M. Naehrig, *Faster pairing computations on curves with high-degree twists*, Public Key Cryptography(PKC) 2010, LNCS volume 6056, pp. 224-242.

[34] C. Costello, P. Longa and M. Naehrig, *Efficient Algorithms for Supersingular Isogeny Diffie-Hellman*, Advances in Cryptology, CRYPTO 2016, LNCS 9814, pp. 572-601.

[35] R. Crandall and C. Pomerance, *Prime Numbers - A Computational Perspective*, 2005, Springer.

[36] J. Cremona, *Rational Points on Curves, Course Notes*, 2003, pp. 29-31. `https://cr.yp.to/bib/1992/montgomery-lucas.ps`, Accessed: 24th Aug 2017.

[37] E. Dahman, K. Okeya and D. Schepers, *Affine Precomputation with Sole Inversion in Elliptic Curve Cryptography*, ACISP 2007, Information Security and Privacy, LNCS volume 4586, pp. 245-258 .

[38] J. Devigne and M. Joye, *Binary Huff Curves*, CT-RSA 2011, LNCS Volume 6558, pp. 340-355.

[39] W. Diffie and M. Hellman, *New Directions in Cryptography*, 1976, IEEE Transactions on Information Theory, volume 22, pp. 644-654.

[40] V.S. Dimitrov and T. Cooklev, *Hybrid Algorithm for the Computation of the Matrix Polynomial* $I + A + \cdots + A^{n-1}$, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, volume 42, 1995, pp. 377-380.

[41] V.S. Dimitrov, L. Imbert and P.K. Mishra, *Efficient and Secure Elliptic Curve Point Multiplication Using Double-Base Chains*, 2005, `https://www.iacr.org/archive/asiacrypt2005/059/059.pdf`, Accessed: 2nd Feb 2016.

[42] C. Doche and D. Sutantyo, *Faster Repeated Doublings on Binary Elliptic Curves*, Selected Areas in Cryptography (SAC) 2013, LNCS volume 8282, pp. 456-470.

[43] B. Dodson and P.Zimmermann, *20 Years of ECM*, ANTS-VII 2006, Berlin.

[44] H.M. Edwards, *A normal form for Elliptic Curves*, Bulletin of the American Mathematical Society, volume 44, pp. 393-422, 2007.

[45] K. Eisentrager, K. Lauter and P.L. Montgomery, *Fast Elliptic Curve arithmetic and Improved Weil Pairing evaluation*, CT-RSA 2003, LNCS volume 2612, pp. 343-354.

[46] T. ElGamal, *A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms*, CRYPTO 1984, LNCS volume 196, pp. 10-18.

[47] R.R. Farashahi and M. Joye, *Efficient Arithmetic on Hessian Curves*, Public Key Cryptography(PKC) 2010, LNCS volume 6056, pp. 243-260.

[48] W. Fischer, C. Giraud, E.W. Knudsen and J.-P.Seifert, *Parallel Scalar Multiplication on general Elliptic Curves over F_p hedged against Non-Differential Side-Channel Attacks*, 2002, `http://eprint.iacr.org/2002/007.pdf`, Accessed: 2nd Feb 2016.

[49] P. Giorgi, L. Imbert and T. Izard, *Optimizing Elliptic Curve Scalar Multiplications for Small Scalars*, SPIE Proceedings, Mathematics for Signal and Information Processing 2009, volume 7444.

[50] H. Gu, D. Gu and W.L. Xie, *Efficient pairing computation on elliptic curves in Hessian form*, Information Security and Cryptology (ICISC) 2010, LNCS volume 6829, pp. 169-176.

[51] D. Hankerson, A. Menezes and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer-Verlag 2004.

[52] M. Hutter, M. Joye and Y. Sierra, *Memory-constrained implementations of Elliptic Curve Cryptography in co-Z coordinate representation*, AFRICACRYPT 2011, LNCS volume 6737, pp. 170-187.

[53] S. Ionica and A. Joux, *Another approach to pairing computation in Edwards Coordinates*, INDOCRYPT 2008, LNCS volume 5365, pp. 400-413.

[54] D. Jao and L.D. Feo, *Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies*, International Workshop on Post-Quantum Cryptography, PCCrypto 2011,LNCS Vol 7071, pp. 19-34.

[55] J.E. Janoski, *Elliptic Curves*, `http://www.math.clemson.edu/~janoski/reu/2012/REULec1KevinJ.tex`, Accessed: 2nd Feb 2016.

[56] A. Joux, *A One Round Protocol for Tripartite Diffie-Hellman*, ANTS-IV 2000, LNCS volume 1838, pp. 385-393.

[57] M. Joye, M. Tibouchi and D. Vergnaud, *Huff's Model for Elliptic Curves*, ANTS-IX 2010, LNCS volume 6197, pp. 234-250 .

[58] B. Justus and D. Loebenberger, *Differential Addition in Generalized Edwards Coordinates*, 5th International Workshop on Security (IWSEC) 2010, LNCS volume 6434, pp. 316-325.

[59] N. Kanayama, Y. Liu, E. Okamoto, K. Saito, T. Teruya and S. Uchiyama, *Implementation of an Elliptic Curve Scalar Multiplication Method using Division Polynomials*, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 2014, volume E97-A No.1, pp. 300-302.

[60] S. Karati, A. Das and D. Roychoudhury, *Randomized Batch Verification of Standard ECDSA Signatures*, Security, Privacy, and Applied Cryptography Engineering (SPACE) 2014, LNCS volume 8804, pp. 237-255.

[61] A. A. Karatsuba, *The Complexity of Computations*, Proceedings of the Steklov Institute of Mathematics 1995, volume 211, pp. 169-183.

[62] D.E. Knuth, *Seminumerical Algorithms, The Art of Computer Programming*, volume 2, Third Edition, Addison Wesley 1998.

[63] N. Koblitz and A.J. Menezes, *A Survey of Public-Key Cryptosystems*, 2004, `https://www.math.uwaterloo.ca/~ajmeneze/publications/publickey.pdf`, Accessed: 4th Feb 2016.

[64] N. Koblitz and A. Menezes, *Pairing-based Cryptography at High Security Levels*, Cryptography and Coding, 10th IMA International Conference, 2005, LNCS volume 3796, pp. 13-36.

[65] S. Lang, *Elliptic Curves: Diophantine Analysis*, Springer-Verlag, 1978.

[66] S. Lang, *A lively activity: To do Mathematics - Diophantine Equations*, The Beauty of Doing Mathematics: Three Public Dialogues, Springer-Verlag, 1985.

[67] D. Le and B. Nguyen, *Fast Point Quadrupling on Elliptic Curves*, SoICT 2012 (Hanoi, Vietnam), Proceedings of the Third Symposium on Information and Communication Technology, pp. 218-222.

[68] D. Le and C. Tan, *Improved Precomputation Scheme for Scalar Multiplication on Elliptic Curves*, Proceedings of Cryptography and Coding 2011, LNCS volume 7089, pp. 327-343.

[69] A. Lenstra, *Unbelievable Security - Matching AES Security using Public Key Systems*, ASIACRYPT 2001, (Gold Coast, Australia), LNCS Volume 2248, pp. 67-86 .

[70] H.W. Lenstra, Jr. *Factoring Integers with Elliptic Curves*, Annals of Mathematics - Second Series, volume 126, No. 3 (Nov., 1987), pp. 649-673.

[71] Q. Lin and F. Zhang, *Efficient Precomputation Schemes of* $kP + lQ$, Information Processing Letters 2012, volume 112, pp. 462-466, Elsevier North-Holland.

[72] P. Longa and C. Gebotys, *Novel Precomputation Schemes for Elliptic Curve Cryptosystems*, Applied Cryptography and Network Security (ACNS) 2009, LNCS volume 5536, pp. 71-88.

[73] P. Longa and A. Miri, *Fast and Flexible Elliptic Curves Point Arithmetic over Prime Fields*, IEEE Transactions on Information Theory 2008, volume 57, pp. 289-302.

[74] P.Longa and A. Miri, *New Multibase Non-Adjacent Form Scalar Multiplication and its applications to Elliptic Curve Cryptosystems*, `https://eprint.iacr.org/2008/052.pdf`, Accessed: 2nd Feb 2016.

[75] J. Lopez, and R. Dahab, *Fast Multiplication on Elliptic Curves over* $GF(2^m)$ *without Precomputation*, Cryptographic Hardware and Embedded Systems (CHES) 1999, LNCS volume 1717, pp. 316-327.

[76] N. Meloni, *New Point Addition formulae for ECC Applications*, Arithmetic of Finite Fields, WAIFI 2007, LNCS volume 4547, pp. 189-201.

[77] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, Taylor and Francis, CRC Press 1997.

[78] V.S. Miller, *The Weil Pairing, and its Efficient Calculation*, Journal of Cryptology 2004, volume 17, Issue 4, pp. 235-261.

[79] P.K. Mishra and V.S. Dimitrov, *Efficient Quintuple Formulas for Elliptic Curves and Efficient Scalar Multiplication Using Multibase Number Representation*, ISC 2007, LNCS volume 4779, pp. 390-406.

[80] P.L. Montgomery, *Speeding the Pollard and Elliptic Curve methods of Factorization*, Mathematics of Computation 1987, volume 48, pp. 243-264.

[81] P.L. Montgomery, *Evaluating Recurrences of Form $X_{m+n} = f(x_m, X_n, X_{m-n})$ via Lucas Chains(1992)*, `https://cr.yp.to/bib/1992/montgomery-lucas.ps`, Accessed: 2nd Feb 2016.

[82] P.L. Montgomery, *Five, Six and Seven term Karatsuba-like formulae*, IEEE Transaction on Computers 2005, volume 54, pp. 362-369.

[83] A.M. Odlyzko, *Discrete logarithms in finite fields and their cryptographic significance*, 1984, `www.dtc.umn.edu/~odlyzko/doc/arch/discrete.logs.pdf`, Accessed: 4th Feb 2016.

[84] K. Okeya and K. Sakurail, *Efficient Elliptic Curve Cryptosystems from a Scalar Multiplication Algorithm with Recovery of the y-coordinate on a Montgomery form Elliptic Curve*, Cryptographic Hardware and Embedded Systems (CHES) 2001, LNCS volume 2162, pp. 129-144.

[85] K. Okeya, T. Takagi and C. Vuillaume, *Efficient Representation on Koblitz curves with Resistance to Side Channel Attacks*, ACISP 2005, LNCS volume 3574, pp. 218-229.

[86] C. Pomerance, *A Tale of Two Sieves*, Notices of the AMS 43 (1996), pp.1473-1485.

[87] C. Pomerance, *Elementary Thoughts On Discrete Logarithms*, 2002, `https://math.dartmouth.edu/~carlp/PDF/dltalk4.pdf`, Accessed: 4th Feb 2016.

[88] B. Poonen, *Elliptic Curves*, 2001, `http://mathcircle.berkeley.edu/BMC4/Handouts/elliptic.pdf`, Accessed: 4th Feb 2016.

[89] R. Shipsey, *Elliptic Divisibility Sequences*, PhD thesis, Goldsmith's College (University of London), 2000.

[90] J.H. Silverman, *The Arithmetic of Elliptic Curves*, Springer-Verlag, 1992.

[91] J.H. Silverman, *Advanced Topics in The Arithmetic of Elliptic Curves*, Springer-Verlag, 1994.

[92] S. Singh, *The Code Book*, Fourth Estate 2000.

[93] J.A. Solinas, *Low-weight Binary representations for Pairs of Integers*, Combinatorics and Optimization Research Report CORR 2001-41, `http: cacr.uwaterloo.ca/techreports/2001/corr2001-41.ps`.

[94] S.R. Srinivasa, *A Note on Schoenmakers Algorithm for Multi Exponentiation*, 12th International Conference on Security and Cryptography(Colmer, France), Proceedings of SECRYPT 2015, pp. 284-391.

[95] S.R. Srinivasa, *Interesting Results Arising from Karatsuba Multiplication - Montgomery family of formulae*, ICCCT 2015, Proceedings of Sixth International Conference on Computer and Communication Technology 2015 (Allahabad, India), pp. 317-322.

[96] S.R. Srinivasa, *An improved EllipticNet Algorithm for Tate Pairing on Weierstrass' Curves, Faster Point Arithmetic and Pairing on Selmer Curves and a Note on Double Scalar Multiplication*, 7th International Conference on Applications and Technologies in Information Security, 2016 (Cairns, Australia), Proceedings of ATIS 2016, Communications in Computer and Information Science, volume 651, pp. 93-105.

[97] S.R. Srinivasa, *Differential Addition in Edwards Coordinates Revisited and a Short Note on Doubling in Twisted Edwards Form*, 13th International Conference on Security and Cryptography (Lisbon, Portugal), Proceedings of SECRYPT 2016, pp. 336-343.

[98] S.R. Srinivasa, *Three Dimensional Montgomery Ladder, Differential Point Tripling on Montgomery Curves and Point Quintupling on Weierstrass' and Edwards Curves*, Proceedings of AFRICACRYPT 2016, (Fes, Morocco), LNCS volume 9646, pp. 84-106.

[99] M. Stam, *Speeding up subgroup Cryptosystems*, 2003, PhD thesis, Technische Universiteit Eindhoven.

[100] K. Stange, *The Tate Pairing via Elliptic Nets*, Pairing-Based Cryptography - PAIRING 2007, LNCS volume 4575, pp. pp 329-348.

[101] K. Stange, *Elliptic Nets and Elliptic Curves*, 2008, PhD thesis, Brown University

[102] J. Stillwell, *The Evolution of Elliptic Curves*, American Mathematical Monthly 1995, volume 102 Issue 9, pp. 831-837.

[103] D. Stinson, *Cryptography: Theory and Practice, Third Edition*, CRC Press 2005.

[104] M. Ward, *Memoir on Elliptic Divisibility Sequences*, American Journal of Mathematics 1948, volume 70, pp. 31-74.

[105] A. Weimerskirch and C. Paar, *Generalization of the Karatsuba Algorithm for efficient Implementations*, 2003, `https://eprint.iacr.org/2006/224.pdf`.

[106] E. Wenger and M. Hutter, *Exploring the Design Space of Prime Field vs. Binary Field ECC-Hardware Implementations*, Information Security Technology for Applications-NordSec 2011, LNCS volume 7161, pp 256-271.

[107] S. Winograd, *Arithmetic Complexity of computations*, CBMS-NSF Regional Conference Series in Applied Mathematics, Society for Industrial and Applied Mathematics 1980, volume 33.

[108] H. Wu, C. Tang and R. Feng, *A new Model of Binary Elliptic Curves*, INDOCRYPT 2012, LNCS volume 7668, pp. 399-411.

[109] A.C. Yao, *On the Evaluation of Powers*, SIAM Journal on Computing 1976, volume 5, Issue 1, pp. 100-103.

[110] L. Zhang, K. Wang, H. Wang and D. Ye, *Another Elliptic Curve Model for Faster Pairing Computation*, ISPEC 2011, LNCS volume 6672, pp. 432-446.

[111] C.A. Zhao, F. Zhang and J. Huang, *A note on the Ate Pairing*, 2007, `https://eprint.iacr.org/2007/247.ps`, Accessed: 2nd Feb 2016.

Index