December 2018

Multi-Base Chains for Faster Elliptic Curve Cryptography

Saud Al Musa
University of Wisconsin-Milwaukee

MULTI-BASE CHAINS FOR FASTER ELLIPTIC CURVE CRYPTOGRAPHY

by

Saud Al Musa

A Dissertation Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

in Engineering

at

The University of Wisconsin–Milwaukee

December 2018

ABSTRACT

MULTI-BASE CHAINS FOR FASTER ELLIPTIC CURVE CRYPTOGRAPHY

by

Saud Al Musa

The University of Wisconsin–Milwaukee, 2018
Under the Supervision of Professor Guangwu Xu

This research addresses a multi-base number system (MBNS) for faster elliptic curve cryptography (ECC). The emphasis is on speeding up the main operation of ECC: scalar multiplication (tP). Mainly, it addresses the two issues of using the MBNS with ECC: deriving optimized formulas and choosing fast methods. To address the first issue, this research studies the optimized formulas (e.g., $3P$, $5P$) in different elliptic curve coordinate systems over prime and binary fields. For elliptic curves over prime fields, affine Weierstrass, Jacobian Weierstrass, and standard twisted Edwards coordinate systems are reviewed. For binary elliptic curves, affine, λ-projective, and twisted μ_4-normal coordinate systems are reviewed. Additionally, whenever possible, this research derives several optimized formulas for these coordinate systems.

To address the second issue, this research theoretically and experimentally studies the MBNS methods with respect to the average chain length, the average chain cost, and the average conversion cost. The reviewed MBNS methods are greedy, ternary/binary, multi-base NAF, tree-based, and rDAG-based. The emphasis is on these methods' techniques to convert integer t to multi-base chains. Additionally, this research develops bucket methods that advance the MBNS methods. The experimental results show that the MBNS methods with the optimized formulas, in general, have good improvements on the performance of scalar multiplication, compared to the single-base number system methods.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

ECC	elliptic curve cryptography
SBNS	single-base number system
DBNS	double-base number system
MBNS	multi-base number system
$\mathbb{F}$	field
$\mathbb{F}_q$	finite field
$\mathbb{F}_p$	prime field
$\mathbb{F}_{2^m}$	binary field
S	field squaring
M	field multiplication
I	field inversion
E	elliptic curve
P	point on an elliptic curve
Q	another point on an elliptic curve
t	positive integer
tP	scalar multiplication
ADD	point addition ($P + Q$)
DBL	point doubling ($2P$)
TPL	point tripling ($3P$)
QPL	point quintupling ($5P$)
NAF	non-adjacent form
DAG	directed acyclic graph
rDAG	rectangular directed acyclic graph

ACKNOWLEDGEMENTS

First, I thank my advisor Prof. Guangwu Xu for the time he spent on me to teach me what it means to become a good researcher. I will always appreciate his help and his support. I acknowledge that many ideas in this research were based on his suggestions during our weekly meetings. Second, my thanks go to Prof. Seyed Hosseini for being my initial advisor and being a part of the committee for this research. He supported me and opened the door for me to study at UWM.

Third, I thank the rest of the committee members: Prof. Adrian Dumitrescu, Prof. Jeb Willenbring, Prof. Lingfeng Wang, and Prof. Yi Hu. Prof. Dumitrescu supported me and suggested valuable comments to enhance the content of this research. Prof. Willenbring's comments were encouraging and his words are still motivating me to achieve my next goals. Finally, I thank my sponsor Taibah University in Saudi Arabia. They believe in me and granted me a scholarship to study at UWM. I also thank Erich Wegenke from the UWM writing center. He helped me to edit this research and suggested good comments to enhance the writing.

1 Introduction

Koblitz (1987) and Miller (1986) introduced Elliptic curve cryptography (ECC) as a new type of public-key cryptography. Another well-known type of public-key cryptography is Rivest, Shamir and Adleman (RSA) (Rivest, Shamir, & Adleman, 1978). The main advantage of ECC is that it is more efficient than RSA. For example, 283-bit integer in ECC is considered as secure as 3072-bit integer in RSA (Lenstra & Verheul, 2001). Public-key cryptography is also known as asymmetric-key cryptography, and it uses two different keys for data encryption and decryption. The first key is called a public key, and it is accessible to the public. The second key is called a private key because it is kept a secret by the owner. In contrast, symmetric-key cryptography uses one shared secret key for data encryption and decryption. As a result, public-key cryptography has the advantage of eliminating the need to share a secret key with others. Another advantage of public-key cryptography is that it provides the digital signature schemes. With this scheme, we guarantee that the message is sent by the owner and is not been modified by an attacker.

Besides the advantages, public-key cryptography has the disadvantage of slow performance in comparison to symmetric-key cryptography. As a result, public-key cryptography in practice is used to encrypt or decrypt data of a small size. For example, we use public-key cryptography to establish a shared secret key between two parties, which involves encrypting data of a small size (e.g., public key size bits). In digital signature schemes, we use public key cryptography to encrypt or decrypt a message digest (hash value) which is data of a small size (e.g., 512 bits).

The most expensive ECC operation is scalar multiplication (tP) where P is a point on an elliptic curve over finite fields and t is a positive integer. Scalar multiplication is an operation that adds a point to itself t times such that $tP = \underbrace{P + P + ... + P}_{t\ times}$. One of the primary techniques to speed up scalar multiplication is to use single-base number system (SBNS) forms. Integer t is represented in the SBNS in the form of $t = \sum_{i=1}^{l} s_i\ 2^{a_i}$ where $a_i \geqslant 0, s_i \in \{-1, +1\}$, and l is the form length. A single-base chain is a special form of SBNS that is utilized by ECC. The main methods

that convert integer t to a single-base chain are binary and NAF methods.

Dimitrov, Jullien, and Miller (1998) initially introduced a double-base number system (DBNS) for applications other than ECC, such as digital signal processing (Dimitrov, Eskritt, Imbert, Jullien, & Miller, 2001). Later, Dimitrov, Imbert, and Mishra (2005) proposed the DBNS to speed up ECC scalar multiplication operations. Integer t is represented in the DBNS in the form of $t = \sum_{i=1}^{l} s_i \; 2^{a_i} \; 3^{b_i}$ where $a_i, b_i \geqslant 0, s_i \in \{-1, +1\}$, and l is the form length. When we represent integer t in the DBNS, the form length, on average, is shorter than when we represent integer t in the SBNS. In other words, when we represent integer t in the DBNS, on average, the number of point additions is minimized and leads to accelerate scalar multiplication. This is because point addition moves slowly to reach the target tP, while other formulas such as point doubling and point tripling move quickly. A multi-base number system (MBNS) is an extended idea of the DBNS (Mishra & Dimitrov, 2007). When we represent integer t in the MBNS, the number point additions continues to reduce.

In this research, the first elliptic curves that we work on are Weierstrass curves E over finite fields $\mathbb{F}_q$, as represented by

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

where $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_q$ and the discriminant of E is not equal to 0. The reason for working on Weierstrass curves is that they are recommended by the National Institute of Standards and Technology (NIST). In fact, NIST recommends using three types of Weierstrass curves: the random curves over both binary and prime fields and Kobilitz curves over binary fields (Barker, 2013). Points on these curves not only can be represented in affine coordinate systems as x-coordinate and y-coordinate, but also in projective coordinate systems as X-coordinate, Y-coordinate, and Z-coordinate. Representing points using projective coordinate systems can achieve significant improvement for scalar multiplication operations, especially for a high inversion to multiplication ($\mathbf{I/M}$) ratio device. This improvement is obtained by replacing inversions with multiplication operations. Therefore, other studies proposed projective

coordinate systems for simplified Weierstrass curves over prime fields such as Chudnovsky and Jacobian (Chudnovsky & Chudnovsky, 1986; Hankerson, Menezes, & Vanstone, 2004). They also proposed projective coordinate systems for binary elliptic curves over binary fields such as Lopez-Dahab, Lambda, twisted μ_4-normal form (Lange, 2004; Oliveira, Lopez, Aranha, & Rodriguez-Henriquez, 2014; Kohel, 2017).

A special class of Weierstrass curves can have an efficiently computable endomorphism (Gallant, Lambert, & Vanstone, 2001; Koblitz, 1992). Other studies show that these special curves which have efficiently commutable endomorphisms can significantly speed up scalar multiplication (Xu, 2009; Ajeena & Kamarulhaili, 2014; Kim & Lim, 2002). However, this technique, unlike the MBNS, is applicable for a certain class of Weierstrass curves. The second type of elliptic curves that we study is twisted Edward curves E over prime fields $\mathbb{F}_p$, as represented by

$$E : ax^2 + y^2 = 1 + dx^2y^2$$

where $a, d \in \mathbb{F}_p$, $p > 3$ is a prime number, $a \neq d$ and $a, d \neq 0$. The reason for studying twisted Edwards curves is that they have the most efficient $2P$ formula in comparison to other elliptic curves over prime fields. In the context of the MBNS, $2P$ is the most important formula because it is the most used operation during scalar multiplication.

For efficiency reasons, the MBNS requires concise point tripling ($3P$) and point quintupling ($5P$) formulas. Therefore, other studies developed concise $3P$ and $5P$ formulas in different coordinates systems. The most efficient $3P$ and $5P$ formulas are described as follows. For Weierstrass curves, Yu, Kim, and Jo (2015) developed $3P$ formulas in affine coordinates over both prime and binary fields. Longa and Miri (2008a) developed $3P$ formula, and Giorgi, Imbert, and Izard (2009) developed $5P$ formula in Jacobian Weierstrass projective coordinates over prime fields. Al Musa and Xu (2017) developed $3P$ and $5P$ formulas in λ-projective coordinates over binary fields. For twisted Edwards curves over prime fields, Bernstein, Chuengsatiansup, and Lange (2017) developed $3P$ formula, and Li, Yu, and Wang (2016) developed $5P$ formula in standard projective coordinates.

For other efficiency reasons, when we represent integer t in the DBNS, we represent it as a double-base chain. The double-base chain is a special form of the DBNS where the sequence of the exponents a_i and b_j decreases. Therefore, other studies introduced methods that represent t as a double-base chain. Dimitrov et al. (2005) introduced the greedy method. Ciet, Joye, Lauter, and Montgomery (2006) introduced the ternary/binary method. Longa and Gebotys (2009) introduced the multi-base NAF method. Doche and Habsieger (2008) introduced the tree-based method. Bernstein et al. (2017) introduced the rDAG-based method. Similar to the DBNS, when we represent integer t in the MBNS, we represent it as a multi-base chain. The above methods can be extended to represent integer t to a multi-base chain.

1.1 Problem Statement

This dissertation studies the efficiency problem of scalar multiplication for elliptic curve cryptography. The approach of solving this problem is to use a multi-base number system (MBNS). The motivation to investigate this approach is that when integer t is represented by a multi-base chain, the chain length, on average, becomes shorter than a single-base chain. This property can speed up scalar multiplication. However, this approach needs further work in two parts: we need to derive optimized formulas (e.g., $3P$, $5P$) and find a method that utilizes these formulas efficiently during scalar multiplication.

Dedicated formulas for small bases (e.g., $3P$, $5P$) in many elliptic curve coordinate systems have not been derived, or they need further optimization. Additionally, other studies proposed MBNS methods such as greedy, multi-base NAF, ternary/binary, tree-based, and rDAG-based. Some of the aspects of theoretical analysis and experimental results of these MBNS methods have not been investigated. These aspects include the average chain length, the average chain cost, and the average conversion cost. They determine a method that utilizes formulas efficiently. Therefore, deriving optimized formulas and choosing a method that utilizes the formulas efficiently lead to faster scalar multiplication.

1.2 Our Contributions

The first part of our contribution is to derive optimized formulas for elliptic curves over prime and binary fields. We work on simplified Weierstrass curves over prime and binary elliptic curves. We also work on twisted Edwards curves over prime fields. We propose several formulas for these curves using various techniques in both affine and projective coordinates. We use the notations $\mathbf{I}$, $\mathbf{M}$, or $\mathbf{S}$ to represent field inversions, multiplications, or squarings respectively. We also use the notations $\mathbf{M}_a$, $\mathbf{M}_b$, or $\mathbf{M}_d$ to represent field multiplication with one of the multiplicands to be the curve coefficient a, b, or d respectively. We can summarize our efforts in this part as follows:

1. We propose $4P$ and $5P$ formulas in affine coordinates for simplified Weierstrass curves over prime fields. When the curve coefficient a is selected as a small constant, our proposed $4P$ formula saves $1\mathbf{M}$ in comparison to the state of the art $4P$ formula. Our proposed $5P$ formula trades $2\mathbf{I}$ with $6\mathbf{M} + 3\mathbf{S}$ in comparison to $2(2P) + P$. To the best of our knowledge, our proposed $5P$ formula is the first $5P$ formula in affine coordinates for simplified Weierstrass curves over prime fields.

2. We propose a $2P$ formula in extended Jacobian coordinates for simplified Weierstrass curves over prime fields. To the best of our knowledge, we are the first study that proposes this extended Jacobian coordinates for simplified Weierstrass curves over prime fields. When the curve coefficient a is selected as a large constant, this proposed $2P$ formula saves approximately $0.6\mathbf{M}$ in comparison to the state of the art $2P$ formula in non-extended Jacobian coordinates.

3. We propose $5P$ and $2Q + P$ formulas in standard projective coordinates for twisted Edwards curves over prime fields. Our proposed $5P$ formula saves approximately $4.2\mathbf{M}$ in comparison to the state of the art $5P$ formula. Our proposed $2Q + P$ formula saves $1\mathbf{M} + 1\mathbf{M}_d + 1\mathbf{M}_a + 1\mathbf{S}$ in comparison to a non-dedicated formula. To the best of our knowledge, our proposed $2Q + P$ formula is the first $2Q + P$ formula for twisted Edwards curves. Additionally, we are able to save $1\mathbf{M}$ for the $P + Q$ formula in these coordinate systems by using the pre-computation concept.

4. Al Musa and Xu (2017) proposed a $5P$ formula in affine coordinates for binary elliptic curves. Their proposed $5P$ formula saves $2\mathbf{M} - 1\mathbf{S}$ in comparison to the state of art $5P$ formula.

5. Al Musa and Xu (2017) proposed $3P$ and $5P$ formulas in λ-projective coordinates for binary elliptic curves. Their proposed $3P$ formula saves $2\mathbf{M} + 1\mathbf{M}_a + 2\mathbf{S}$ in comparison to the state of the art $3P$ formula in Lopez-Dahab projective coordinates. Their $5P$ formula saves $6\mathbf{M} + 1\mathbf{M}_a + 2\mathbf{S}$ in comparison to $2(2P) + P$. To the best of our knowledge, their $5P$ formula is the first $5P$ formula in projective coordinates for binary elliptic curves.

6. We generalize the $2P$ formula in twisted μ_4-normal coordinates for binary elliptic curves. When the curve coefficient b is selected as a small constant, this proposed $2P$ saves approximately $1\mathbf{M}$ in comparison to the state of the art $2P$ formula in λ-projective coordinates.

7. We propose a $\bar{\tau}P = \mu P - \tau P$ formula in twisted μ_4-normal coordinates for Koblitz curve over binary fields. When the curve coefficient $a = 0$, our proposed $\bar{\tau}P$ formula saves $1\mathbf{M}$ in comparison to the state of the art $\bar{\tau}P$ formula in twisted μ_4-normal coordinates. It has the same cost as the state of the art $\bar{\tau}P$ formula in Lopez-Dahab projective coordinates. However, this proposed $\bar{\tau}P$ formula is still preferred because when $a = 0$, twisted μ_4-normal coordinates have the most efficient $P + Q$ formula for Koblitz curve over binary fields.

In the second part of our contributions, we advance the MBNS methods in three ways. First advancement: we theoretically study the MBNS methods, including the greedy, the ternary/binary, the multi-base NAF, the tree-based, the rDAG-based methods. We emphasize three properties of the methods: the conversion cost, the chain cost, and the chain length. Second advancement: we develop bucket methods for the DAG-based and the tree-based abstract ideas. The bucket methods systematically balance the chain cost and the time to find the chain. As a consequence, they enable us to produce a near optimal chain in significantly less running time than an optimal

chain. The optimal chain is the one produced by the rDAG-based method. The bucket methods also enable us to show that the tree-based method does not produce a near optimal chain. We understand that we are the first study that suggests systematically finding a near optimal chain for the MBNS methods.

Third advancement: we experimentally study the MBNS methods with and without pre-computation. In these experimental results, we utilize the state of the art formulas presented in this dissertation. To the best of our knowledge, we are the first study that shows experimental comparison results between the MBNS methods over both binary and prime fields. We show the results with respect to the chain length, the chain cost, and the running time. The running time evaluates the two phases of the methods: the converting phase and the performing phase. The chain cost evaluates only the performing phase of the methods. Other experimental results were conducted either in binary fields or prime fields without evaluating the converting phase. The compared MBNS methods without pre-computation are the greedy, the ternary/binary, the multi-base NAF, and the tree-based. The compared MBNS methods with pre-computation are the rDAG-based and the bucket. These methods utilize the pre-computation concept in the converting phase to further lower the average chain cost.

Our experimental results show that MBNS methods without pre-computation had an approximately 6% to 11% lower average chain cost in comparison to the SBNS methods. Except for the greedy method, they had an approximately 6% to 14% faster running time in comparison to the SBNS methods. They show that the MBNS method with pre-computation could further lower the average chain cost and affect the average running time.

Additionally, we conducted experiments to compare the optimal and the novel pre-computation schemes of window τ-NAF for Koblitz curves. Our experimental results show that novel scheme improved the performance significantly in comparison to the optimal scheme. To the best of our knowledge, these are the first experimental results for the optimal and the novel pre-computation schemes in twisted μ_4-normal coordinates in both cases, $a = 0$ and $a = 1$.

1.3 ECC Applications

ECC is widely deployed in the restricted resources computing systems. Embedded systems and sensors are often manufactured with restricted resource capabilities such as lower battery capacity, smaller memory size, and slower processor speed. Therefore, ECC is suitable to be deployed in these systems because it uses the resources more efficiently, compared to other public-key cryptographic systems. For example, He and Zeadally (2014) proposed ECC authentication schemes for radio-frequency identification (RFID) chips. These small chips can be used as real-time monitoring systems that are attached to patients to gather and transfer health information to physicians. Santoso and Vun (2015) also suggested ECC authentication schemes for smart home systems. These systems rely on small sensors that send information related to a home's energy efficiency.

ECC is also used with blockchains. The blockchain can be seen as a type of a distributed system that proves transactions (Crosby, Nachiappan, Pattanayak, Verma, & Kalyanaraman, 2016). For example, the Bitcoin blockchain uses ECC digital signature schemes to prove digital currency exchange transactions between two users (Nakamoto, 2011). Ethereum, Litecoin, and Ripple blockchains also use ECC digital signature schemes to prove transactions (Watson, 2018).

ECC is supported by computer network protocols. Computer networks use ECC in key agreement and digital signature schemes. For example, the transport layer security (TLS) protocol, which provides secure communications through a web browser, supports ECC in key agreement and digital signature schemes (Rescorla, 2018). The Kerberos authentication protocol also supports ECC in the key agreement scheme (Zhu, Jaganathan, & Lauter, 2008). Additionally, Secure Shell (SSH) protocol, which provides remote access to computer systems, supports ECC in key agreement and digital signature schemes (Stebila & Green, 2009).

2 Basic Background

Definition 1. An elliptic curve E over a field $\mathbb{F}$ in the general Weierstrass equation is represented by

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

where $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}$ and the discriminant of E is not equal to 0.

The above equation (1) can be simplified through the use of the admissible changes of variables. When the characteristic of $\mathbb{F}$ is not equal to 2 or 3, we can transform the equation to what we call simplified Weierstrass curves over prime fields, as represented by

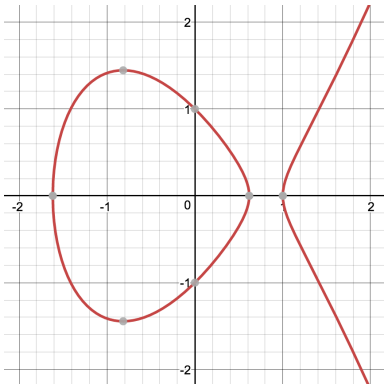
$$y^2 = x^3 + a_4x + a_6$$

where the discriminant $-16(4a_4^3 + 27a_6^2) \neq 0$. When the characteristic of $\mathbb{F}$ is equal to 2, we can transform the equation (1) to both supersingular and non-supersingular curves over binary fields. We say that E is supersingular curves if the characteristic of $\mathbb{F}$ divides the trace of E; otherwise, E is non-supersingular curves. Non-supersingular curves over binary fields are called binary elliptic curves, as represented by

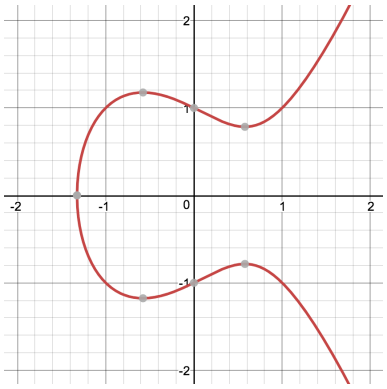
$$y^2 + xy = x^3 + a_2x^2 + a_6$$

where the discriminant $a_6 \neq 0$. Supersingular curves over binary fields are not used in practice because of security issues related to the discrete logarithm problem (Granger, Kleinjung, & Zumbrägel, 2014). See Figure 2.1 for examples of Weierstrass curves over real numbers ($\mathbb{R}$).

In elliptic curve cryptographic systems, we have to define an elliptic curve E over finite fields $\mathbb{F}_q$. The denotation $E(\mathbb{F}_q)$ is the set of all points (x, y) where $x, y \in \mathbb{F}_q$ that satisfies the above equation (1) together with the point at infinity $\mathcal{O}$. This set forms an abelian group under point addition. Precisely, this group is closed, associative, commutative, has an identity, and has the inverse property. The identity point of the group is the point at infinity $\mathcal{O}$. The point addition can be computed by the chord and



$E : y^2 = x^3 - 2x + 1$



$E : y^2 = x^3 - x + 1$

Figure 2.1. Examples of Weierstrass Curves over $\mathbb{R}$

tangent rule. See Section 2.2 for the chord and tangent rule.

Let r be the group order of $E(\mathbb{F}_q)$. For security reasons, it is important that r be selected as a large prime number or a product of a large prime number and a small number. We focus on two types of finite fields: prime fields $\mathbb{F}_p$ and binary fields $\mathbb{F}_{2^m}$ where $p > 3$ is a prime number and m is the degree of an irreducible polynomial. The finite field arithmetic (e.g., squaring, addition) gets affected by the finite field types. In Figure 2.2, we see the finite field arithmetic in the bottom layer of ECC operations, which means it affects all upper layers of ECC operations.

Protocols:	ECDH | ECDSA
Methods:	scalar multiplication (tP)
Formulas:	point doubling ($2P$) | point addition ($P + Q$)
Finite field arithmetic:	inversion (**I**) | multiplication (**M**) | squaring (**S**) | addition/subtraction

Figure 2.2. ECC Operations Hierarchy

2.1 Finite Field Arithmetic

The main operations for finite field arithmetic are inversion (**I**), multiplication (**M**), squaring (**S**), and addition/subtraction. Our focus in the context of finite field arithmetic is to reduce the number of inversion, multiplication, and squaring operations. We ignore addition/subtraction operations because they are cheap operations over both

binary and prime fields. It is not our intention to improve the finite field arithmetic algorithms. This is because there are tremendous ongoing efforts to improve the finite field arithmetic algorithms. For example, GMP and MIRACL are well-known available software libraries for finite field arithmetic (Granlund & et al., n.d.; Matula & Kornerup, n.d.). Therefore, we utilize GMP and MIRACL to perform the finite field arithmetic.

2.1.1 Squaring. We use the squaring to multiplication (**S/M**) ratio to measure the squaring cost. This is because the **S/M** ratio varies in different devices over different finite fields. One field squaring costs less than or equal to one field multiplication operation. For prime fields, the **S/M** ratio is a high ratio and it is closer to one. For binary fields, it is a low ratio and the squaring is closer to a free operation. For example, Bernstein et al. (2017) assumed $1\mathbf{S} = 0.8\mathbf{M}$ over prime fields. Doche, Kohel, and Sica (2009) assumed $1\mathbf{S} = 0.1\mathbf{M}$ over binary fields. To simplify the comparison over prime fields, we assume $1\mathbf{S} = 0.8\mathbf{M}$.

The main technique that we use to reduce the number of squaring operations is substitution. Substitution can be obtained from the curve equation or relations that are derived. For example, we use this relation over binary fields: $a^2 + b^2 = (a + b)^2$, which saves $1\mathbf{S}$. We use this relation over prime fields: $a^2 - b^2 = (a + b) \cdot (a - b)$, which trades $2\mathbf{S}$ with $1\mathbf{M}$.

2.1.2 Inversion. An inversion operation is the most expensive finite field arithmetic over both binary and prime fields. We use the inversion to multiplication (**I/M**) ratio to measure the inversion cost. This is also because the **I/M** ratio varies in different devices over different finite fields. We say that a low **I/M** ratio is 5, and a high I/M ratio is 8, as suggested by Hankerson et al. (2004). We use two techniques to reduce the number of inversion operations: Montgomery's trick and projective coordinate systems.

Montgomery's trick performs multiple inversions at once to reduce the number of inversions. However, it adds extra $3(n - 1)\mathbf{M}$ where n is the number of inversion operations. We see that Montgomery's trick only reduces the number of inversions by replacing an inversion with multiplication operations. However, we still use inversion

operations. For more examples on Montgomery's trick, see theorem proofs in Section 3.1 and Section 4.1.

Another technique for reducing the number of inversion operations is to use a projective coordinate system. A projective point is commonly represented as X-coordinate, Y-coordinate, and Z-coordinate, while an affine point represented as x-coordinate and y-coordinate. It is important to note that we capitalize projective point elements because we want to differentiate projective point elements from affine point elements.

When we represent a point in projective coordinates, inversion operations are not used during scalar multiplication operation. However, we need to do the following extra steps. First, before performing scalar multiplication, we need to convert an affine point to a projective point. Then, after scalar multiplication, we need to reverse a projective point to an affine point. The cost of these extra steps are minor in comparison to the cost of scalar multiplication. These extra steps minimize the number of operations and speed up scalar multiplication. This minimization increases in a high $\mathbf{I/M}$ ratio device.

Other studies used projective coordinate systems for different elliptic curves to speed up scalar multiplication. For simplified Weierstrass curves over prime fields, the most efficient projective coordinates are Jacobian and Chudnovsky coordinates (Hankerson et al., 2004; Chudnovsky & Chudnovsky, 1986). For binary elliptic curves, the most efficient projective coordinates are Lopez-Dahab (LD), Lambda (λ), and twisted μ_4-normal coordinates (Lange, 2004; Oliveira et al., 2014; Kohel, 2017).

For twisted Edwards curves over prime fields, the most efficient projective coordinates are standard and inverted coordinates (Bernstein, Birkner, Joye, Lange, & Peters, 2008). See Section 3.2 for Jacobian Weierstrass projective coordinates and Section 3.4 for standard twisted Edwards coordinates. Also, see Section 4.2 for λ-projective coordinates and Section 4.3 for twisted μ_4-normal coordinates.

2.1.3 Multiplication. We use multiplication as the main unit to measure the cost of a formula or a method. Therefore, we usually approximate squaring and inversion to multiplication by using the $\mathbf{I/M}$ and $\mathbf{S/M}$ ratio assumptions. We use the

notations $\mathbf{M}_a$, $\mathbf{M}_b$, or $\mathbf{M}_d$ to represent field multiplication with one of the multiplicands to be the curve coefficient a, d, or d respectively. To simplify the comparison over prime fields, we assume $\mathbf{M}_d = 1\mathbf{M}$, $\mathbf{M}_b = 1\mathbf{M}$, and $\mathbf{M}_a$ is a free operation. This is because the coefficient b or d are often selected as large constants and the coefficient a is often selected as a small constant.

The following techniques are used to reduce the number of multiplication operations: the distribution law, the cancellation law, substitution, and factoring polynomials. For example, this relation: $a \cdot b + c \cdot d = (a + d) \cdot (b + c) - a \cdot c - b \cdot d$ saves $1\mathbf{M}$ in certain cases. Also, this relation over prime fields: $2 \cdot a \cdot b = (a + b)^2 - a^2 - b^2$ replaces $1\mathbf{M}$ with $1\mathbf{S}$ in certain cases. For more examples, see the theorem proofs in Section 3 and Section 4.

2.2 Formulas

One of the main factors that impacts the efficiency of scalar multiplication is formulas, as Figure 2.2 shows. We measure a formula by the number of **S**, **M**, and **I** operations. An example formula is $P + Q$, $2P$, $3P$, and $5P$. We also called them point addition (**ADD**), point doubling (**DBL**), point tripling (**TPL**), and point quintupling (**QPL**) respectively. We use the notation mixed $P + Q$ to indicate that one point is represented in affine coordinates (e.g., $P = (x, y)$), and the other point is represented in projective coordinates (e.g., $Q = (X, Y, Z)$). When we state that a formula is an "efficient formula", it is to indicate that efforts were made to reduce the number of operations.

$P + Q$ is necessary for any proposed coordinate systems. In fact, it is sufficient to perform scalar multiplication using $P + Q$. However, we use $2P$, $3P$, and $5P$ formulas to speed up scalar multiplication. Other formulas that are used to speed up scalar multiplication are $2Q + P$ and $4P$. All formulas can be derived from a $P + Q$ formula. For example, $2P$ in twisted Edward coordinates is directly obtained from a $P + Q$ formula. A $3P$ formula can be derived from the fact that $3P = 2P + P$ and a $5P$ formula can be derived from the fact that $5P = 3P + 2P$.

2.2.1 P + Q derivation.

Assume we work on the following curve $E : y^2 = x^3 + ax + b$ over $\mathbb{R}$. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two different points on the curve E. We want to find $P + Q = (x_3, y_3)$. We can geometrically find $P + Q$, as Figure 2.3 shows. First, we draw the line $L : y = sx + m$ where s is slope and m is the y-intercept. This line L intersects with points P and Q and intersects with the curve E at the third point. This third point is the inverse of $P + Q$. Second, we draw a vertical line that intersects with the third point. This vertical line will intersect with the curve at the fourth point. This fourth point is $P + Q$.

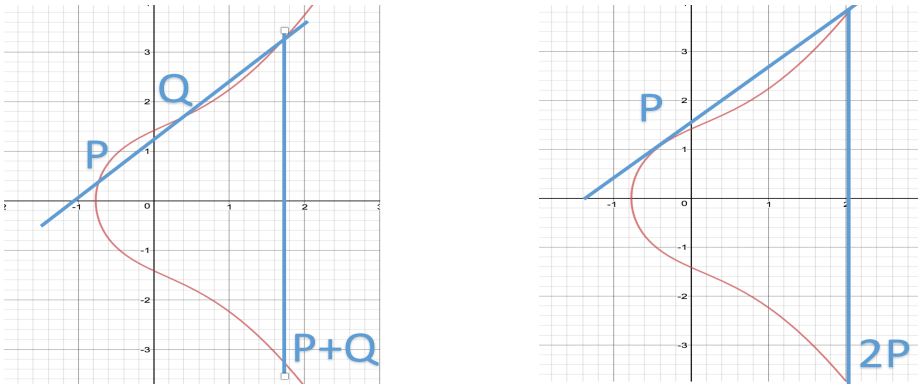


Figure 2.3. $P + Q$ and $2P$ Geometric Descriptions

To translate the above $P + Q$ geometric descriptions to a formula, we have

$$E : y^2 = x^3 + ax + b.$$
$$L : y = sx + m.$$

By substituting the y value, we have

$$(sx + m)^2 = x^3 + ax + b.$$

By solving the above cubic equation, we have three solutions, such that

$$x^3 + ax + b - (sx + m)^2 = (x - x_1)(x - x_2)(x - x_3).$$

The above equation can be expressed by

$$x^3 - s^2x^2 - (2ms - a)x + b - m^2 = x^3 - (x_1 + x_2 + x_3)x^2 + (x_1x_2 + x_1x_3 + x_2x_3)x - x_1x_2x_3.$$

It implies that

$$s^2 = x_1 + x_2 + x_3.$$

By rearranging the above equation, we have

$$x_3 = s^2 - x_1 - x_2.$$

Therefore, we obtained x_3. Next, we want to obtain y_3. From the above line equation, we have

$$y_3 = sx_3 + m.$$

By using this relation: $m = y_1 - sx_1$, we have

$$y_3 = s(x_3 - x_1) + y_1.$$

By taking the negative of y_3, we have

$$y_3 = s(x_1 - x_3) - y_1.$$

□

To derive a $2P$ formula for the above curve E, we can follow the same steps of $P + Q$ derivation. However, since $P = Q$, we have a tangent line $L : y = \lambda x + m$ where λ is the slope. Figure 2.2 shows this tangent line touches the curve E only at point P. To compute the slope λ of this tangent line, we compute the first derivative of the curve equation E. We use the power rule to compute the first derivate of E. Therefore, we have

$$\lambda = \frac{3x_1^2 + a}{2y_1}.$$

We use the above x_3 and y_3 equations to obtain a $2P$ formula. Therefore, we have

$$x_3 = \lambda^2 - 2x_1.$$
$$y_3 = \lambda(x_1 - x_3) - y_1.$$

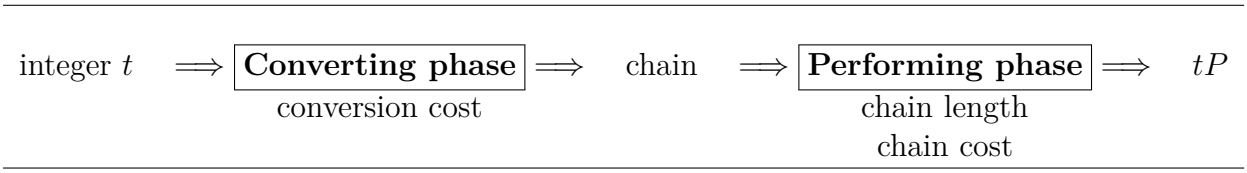


Figure 2.4. Description of a Scalar Multiplication Method

In the context of the MBNS, a scalar multiplication method consists of two phases: the converting phase and the performing phase. Figure 2.4 shows that the converting phase takes integer t and produces a chain. The performing phase takes a chain and produces tP. We focus on the converting phase of the methods. This is because the converting phase is sufficient to evaluate the methods. It determines the conversion cost, the chain cost, and the chain length of the methods.

2.3.1 The converting phase.

Definition 2. A positive integer t is represented in a single-base chain with $\{2\}$-integers in the form of

$$t = \sum_{i=1}^{l} s_i \; 2^{a_i}$$

where $s_i \in \{-1, +1\}$, l is the chain length, $a_1 \geqslant a_2 \geqslant \cdots \geqslant a_l \geqslant 0$.

Definition 3. A positive integer t is represented in a double-base chain with $\{2, 3\}$-integers in the form of

$$t = \sum_{i=1}^{l} s_i \; 2^{a_i} \; 3^{b_i}$$

where $s_i \in \{-1, +1\}$, l is the chain length, $a_1 \geqslant a_2 \geqslant \cdots \geqslant a_l \geqslant 0$, and $b_1 \geqslant b_2 \geqslant \cdots \geqslant b_l \geqslant 0$.

Definition 4. A positive integer t is represented in a multi-base chain with $\{2, 3, 5\}$-integers in the form of

$$t = \sum_{i=1}^{l} s_i \; 2^{a_i} \; 3^{b_i} \; 5^{c_i}$$

where $s_i \in \{-1, +1\}$, l is the chain length, $a_1 \geqslant a_2 \geqslant \cdots \geqslant a_l \geqslant 0$,
$b_1 \geqslant b_2 \geqslant \cdots \geqslant b_l \geqslant 0$, and $c_1 \geqslant c_2 \geqslant \cdots \geqslant c_l \geqslant 0$.

Other researchers introduced methods that convert integer t to single-base, double-base, or multi-base chains. The binary and the NAF methods convert integer t to a single-base chain with $\{2\}$-integers. These methods can be altered to convert integer t to a single-base chain with $\{3\}$-integers or $\{5\}$-integers (Takagi, Reis, Yen, & Wu, 2006). We focus on the single-base chain with $\{2\}$-integers since it is the most widely used type of the single-base chain.

The ternary/binary, the multi-base NAF, the tree-based, and the rDAG-based methods convert integer t to a double-base chain with $\{2, 3\}$-integers. These double-base methods can be altered to produce other types of chains, such as chains with $\{2, 5\}$-integers or $\{3, 5\}$-integers. We focus on the double-base chain with $\{2, 3\}$-integers since it is the simplest type of the double-base chain. These methods also can be extended to convert integer t to a multi-base chain with $\{2, 3, 5\}$-integers. We explain these methods in Section 5 and Section 6.

We use the conversion cost to evaluate the converting phase of a method. This is because these methods utilize different techniques to convert integer t to a chain. The conversion cost is the time complexity of the converting phase. For example, the conversion cost of the binary method is $\mathcal{O}(\log_2 t)$ and of the rDAG-based method is approximately $\mathcal{O}\left((\log_2 t)^2\right)$. It is important to note that the time complexity does not pay attention to the size of constants. These ignored constants can affect the running time of the methods. Therefore, the conversion cost might not be an accurate evaluation of the converting phase of methods. For example, the converting phase of the ternary/binary and the tree-based methods both take logarithmic time. However, they are different in the constant that is associated with the logarithmic time.

The conversion cost can be hard to be known for some methods, such as the greedy method. Therefore, we use our experimental results to estimate the conversion cost of these methods. The converting phase in the single-base methods is efficient. However, the converting phase in the multi-base methods can be inefficient. Some of

these multi-base methods generate a high quality chain, but they require extra time to find the chain, such as the rDAG-based method. See Section 7 for experimental comparison between these methods.

2.3.2 The performing phase. In this phase, we execute the chain that we received from the converting phase. The chain consists of number of **ADD**, **DBL**, **TPL**, and **QPL**. We use the chain length and the chain cost to evaluate this phase. The chain length is a rough estimate of the chain quality. This is because it considers only the number of **ADD** to evaluate a chain. The chain cost is an accurate estimate of the chain quality. This is because it considers **ADD**, **DBL**, **TPL**, and **QPL** to evaluate a chain.

For example, we want to know the chain length and the chain cost when we represent 935811 by a single-base chain with $\{2\}$-integers. We have

$$935811 = 2^{20} - 2^{17} + 2^{14} + 2^{11} - 2^7 + 2^2 - 1.$$

The chain length is 7, which indicates that we have $6\mathbf{ADD}$. The chain cost is $6\mathbf{ADD} + 20\mathbf{DBL}$. We obtain $20\mathbf{DBL}$ from the exponent of the leading factor. When we represent 935811 by a double-base chain with $\{2, 3\}$-integers, we have

$$935811 = 2^{12}3^5 - 2^8 3^5 + 2^5 3^4 + 2^5 3 + 3.$$

The chain length is 5 and costs $4\mathbf{ADD} + 12\mathbf{DBL} + 5\mathbf{TPL}$. When we represent 935811 by a multi-base chain with $\{2, 3, 5\}$-integers, we have

$$935811 = 2^8 3^6 5 + 2^3 3^4 5 - 2^2 3^3 5 - 3^2.$$

The chain length is 4 and costs $3\mathbf{ADD} + 8\mathbf{DBL} + 6\mathbf{TPL} + 1\mathbf{QPL}$. From the previous example, we see that the length of the double-base chain is less than the length of the single-base chain. However, the double-base chain has an extra number of **TPL**. We also see that the length of the triple-base chain is less than the length of the

double-base chai, but the triple-base chain has an extra number of **QPL**.

Remark 1. Our theoretical studies and experimental results show that when integer t is represented by a multi-base chain, on average, we have a shorter length and a lower cost than a single-base chain.

The question is, why does a multi-base chain, on average, have a lower cost? This is because the multi-base chain is based on the idea of replacing **ADD** with **DBL**, **TPL**, or **QPL**. This idea reduces the chain cost because **ADD** moves slowly to reach tP, while **DBL**, **TPL**, and **QPL** move quickly to reach tP. In other words, **DBL**, **TPL**, and **QPL** is the least costly way to do two, three, and five jumps respectively. See Table 3.12 and Table 4.6 for the cost of **ADD**, **DBL**, **TPL**, and **QPL** in twisted Edwards coordinates and λ-coordinates, as an example.

2.3.3 An optimal chain. The definition of an optimal chain for integer t varies among studies. Some studies defined an optimal chain as a high quality chain. Some studies defined an optimal chain as a chain with the shortest length. Other studies defined an optimal chain as a chain with the lowest cost. In this dissertation, we define an optimal chain as a chain with the lowest cost, as Definition 5 shows. The question is, is there a method can convert integer t to an optimal chain? Two studies introduced two methods that find an optimal chain: the enumeration method and the rDAG-based method.

Definition 5. An optimal chain for integer t is a chain that represents t, and it is the least costly in a particular coordinate system.

Definition 6. A near optimal chain is a chain that has a small cost difference from an optimal chain. This small difference is at most the cost of 1**DBL** in a particular coordinate system.

The first study to find an optimal chain is the enumeration method (Doche, 2014). The enumeration method focuses on an optimal chain with the shortest length. It takes exponential time to find an optimal chain for integer t. Moreover, this study defined an optimal chain for integer t as a chain that meets the following requirements: It has the shortest length that represents integer t and has a leading factor that divides a given

upper bound.

An optimal chain with respect to the length has the following disadvantages. First, we have many optimal chains for integer t that have similar length and different costs. For example, let $t = 935811$ and the upper bound $= 2^{20}3^{13}$. Then we can use the enumeration approach to represent t by the following optimal double-base chains:

$$t = 2^{12}\ 3^5 - 2^8\ 3^5 + 2^5\ 3^4 + 2^5\ 3 + 3.$$
$$t = 2^7\ 3^8 + 2^7\ 3^6 + 2^5\ 3^4 + 2^5\ 3 + 3.$$
$$t = 2^4\ 3^{10} - 2^2\ 3^7 - 3^5 + 3^3 - 3^2.$$

From the above chains, we have three optimal chains with length 5, and they have different costs in a coordinate system. The chain with the leading factor $2^{12}3^5$ is the least costly among the above chains in standard twisted Edwards coordinates.

Second, a shorter chain length generally indicates a lower chain cost. However, it is not true in all cases. For example, let $t = 1118848774838$. Then we can represent t by the following double-base chains:

$$t = 2^{16}3^{15} + 2^{15}3^{14} + 2^{14}3^{13} - 2^{13}3^{12} - 2^{10}3^9 + 2^9 3^8 - 2^8 3^7 + 2^7 3^4 - 2^3 3^3 + 2^2 3 + 2.$$
$$t = 2^{32}3^5 + 2^{30}3^4 - 2^{27}3^4 - 2^{25}3^3 - 2^{21}3^2 - 2^{19}3^2 - 2^{14}3^2 - 2^{11}3^2 - 2^9 3 + 2^6 3 - 2^3 - 2.$$

The first chain length is 11 and costs 368.2**M** in standard twisted Edwards coordinates. To explain the cost of the first chain, as Table 3.12 shows, we have

$$10\mathbf{ADD} + 16\mathbf{DBL} + 15\mathbf{TPL} = 10 \cdot 9.8 + 16 \cdot 6.2 + 15 \cdot 11.4 = 368.2\mathbf{M}.$$

The second chain length is 12 and costs 363.2**M** in standard twisted Edwards coordinates. To explain the cost of the second chain, we have

$$11\mathbf{ADD} + 32\mathbf{DBL} + 5\mathbf{TPL} = 11 \cdot 9.8 + 32 \cdot 6.2 + 5 \cdot 11.4 = 363.2\mathbf{M}.$$

We see that the first chain has a shorter chain length and a higher cost than the second

chain.

The second study to find an optimal chain is the rDAG-based method (Bernstein et al., 2017). The rDAG-based method focuses on an optimal chain with the lowest cost. The conversion cost in this method is approximately $\mathcal{O}\left((\log_2 t)^2\right)$. We see that the conversion cost of the rDAG-based method is significantly better than the enumeration method. However, the rDAG-based method takes impractical running time when the converting phase is performed on-the-fly, as Experiment II shows.

In this dissertation, we suggest finding a near optimal chain instead of finding an optimal chain. We define a near optimal chain as a chain that has a small difference from the optimal chain. We suggest the difference to be the cost of 1**DBL**, as Definition 6 shows. We propose the bucket methods to find a near optimal chain. The advantage of this method is that it significantly improves the running time of the rDAG-based method, as Experiment II shows.

2.4 Protocols

2.4.1 ECDH. One of main protocols that our work improves is Elliptic Curve Diffie Hellman (ECDH) (Diffie & Hellman, 1976). This protocol provides steps that help to establish a shared secret key between two parties over an insecure channel. This secret key is used by symmetric key cryptography (e.g., AES) to encrypt or decrypt data (Dworkin et al., 2011). ECDH protocol can be described as follows. First, assume ECC domain parameters are known for two parties (e.g., Alice, Bob). The ECC domain parameters are in the form of $< E, a, b, r, h, G, \mathbb{F}_q >$ and are explained in Table 2.1. Then, we can use Algorithm 2.1 to establish a shared secret key (K) between two parties.

We see that the secret key K in both parties has the same value because $t_A \cdot P_B = t_B \cdot P_A \implies t_A \cdot t_B \cdot G = t_B \cdot t_A \cdot G$. We also see that the main operation in ECDH algorithm is scalar multiplication. In fact, K is temporary and lives for a short period of time. Therefore, step 5 and step 6 in Algorithm 2.1 are repeated every time we have a new communication between two parties.

Table 2.1
ECC Domain Parameters

E	the curve equation (e.g., $y^2 = x^3 + ax + b$)
a, b	the curve coefficients (e.g., $a = 2, b = 2$)
r	the main subgroup order (e.g., $r = 19$)
h	cofactor (the group order $= h \cdot r$)
G	base point or generator (e.g., $(x, y) = (5, 1)$)
$\mathbb{F}_q$	finite fields (e.g., prime fields with $q = 17$)

Algorithm 2.1 ECDH

Input: domain parameters $< E, a, b, r, h, G, \mathbb{F}_q >$
Output: shared secret key K
1. Alice chooses random integer $t_A \in \{1, 2, ..., r - 1\}$ (private key)
2. Alice obtains point P_A such that $P_A = t_A \cdot G$ (public key)
4. Bob does similar steps to obtain t_B and P_B
3. Alice and Bob announce P_A and P_B
5. Alice obtains K such that $K = t_A \cdot P_B$
6. Bob obtains K such that $K = t_B \cdot P_A$

2.4.2 ECDSA. Another important protocol that our work improves is Elliptic Curve Digital Signature Algorithm (ECDSA) (Johnson, Menezes, & Vanstone, 2001). In digital signature schemes, we guarantee that the message is sent by the owner. In addition, we guarantee that the message is not modified by an attacker. We can describe ECDSA as follows. First, let us assume that the ECC domain parameters are known. Then, this scheme goes through two phases: signature generation and signature verification. In signature generation, Alice uses her private key (t_A) to sign the message, as Algorithm 2.2 shows. In signature verification, Bob uses Alice's public key (P_A) to ensure that the message was sent by Alice and is not modified by an attacker, as Algorithm 2.3 shows.

Algorithm 2.2 ECDSA Signature Generation

Input: domain parameters $< E, a, b, r, h, G, \mathbb{F}_q >$, message m, private key t_A
Output: signature (c, s)
1. Alice obtains e such that $e = hash(m)$
2. Alice chooses random integer $k \in \{1, 2, \ldots, r - 1\}$
3. Alice obtains point $Q = (x, y)$ such that $Q = k \cdot G$
4. Alice obtains c such that $c \equiv x \pmod{r}$. If $c = 0$, Alice repeats the steps
5. Alice obtains s such that $s \equiv k^{-1}(e + c \cdot t_A) \pmod{r}$. If $s = 0$, Alice repeats the steps
6. Alice sends message m and signature (c, s) to Bob

Algorithm 2.3 ECDSA Signature Verification

Input: domain parameters $< E, a, b, r, h, G, \mathbb{F}_q >$, signature (c, s), message m, public key P_A
Output: accept or reject the signature
1. Bob obtains e such that $e = hash(m)$
2. Bob obtains w such that $w \equiv s^{-1} \pmod{r}$
3. Bob obtains u_1 and u_2 such that $u_1 \equiv e \cdot w \pmod{r}$ and $u_2 \equiv c \cdot w \pmod{r}$
4. Bob obtains point $Q = (x, y)$ such that $Q = u_1 \cdot G + u_2 \cdot P_A$
5. Bob rejects the signature if $Q = \mathcal{O}$
6. Bob accepts the signature if $c \equiv x \pmod{r}$, otherwise, Bob rejects the signature

In these schemes, we actually sign the hash value of the message instead of the entire message for efficiency reasons. The hash value is a unique identifier for the message. We use a hash function to generate a hash value. This hash function is a one-way function that inputs a message of any bit size and outputs a small fixed-bit size (e.g., 512 bits). An example of a hash function that is recommended by NIST is SHA-3 (Dworkin, 2015). We see that we accept the signature in Algorithm 2.3 only if $c \equiv x \pmod{r}$. This is because the right side:

$$\begin{aligned} x &= u_1 \cdot G + u_2 \cdot P_A \\ &= u_1 \cdot G + u_2 \cdot t_A \cdot G \\ &= G \cdot (u_1 + u_2 \cdot t_A) \\ &= G \cdot (e \cdot w + c \cdot w \cdot t_A) \\ &= G \cdot w \cdot (e + c \cdot t_A) \\ &= G \cdot s^{-1} \cdot (e + c \cdot t_A) \end{aligned}$$

is equal to the left side:

$$\begin{aligned} c &= k \cdot G \\ &= s^{-1} \cdot (e + c \cdot t_A) \cdot G. \end{aligned}$$

ECC protocols are trusted for use because it is difficult to solve the discrete logarithm problem (DLP). The discrete logarithm problem in an elliptic curve cryptosystem (ECDLP) is defined as if two points, P and Q, on an elliptic curve over

finite fields are known. It is assumed to be hard to find integer t such that $tP = Q$ where t is a relatively large integer.

The fastest, general-purpose, known method for attacking ECDLP is Pollard's rho (Pollard, 1975). The Pollard's rho method takes exponential running time $\mathcal{O}(\sqrt{r})$ (Koblitz & Menezes, 2015). Therefore, NIST recommends using an elliptic curve over finite fields with a large group order (e.g., $r \geq 2^{163}$), since it requires an enormous amount of computation that make it impossible to solve ECDLP by today's technologies (Barker, 2013).

3 Elliptic Curves over Prime Fields

3.1 Simplified Weierstrass Curves

The simplified Weierstrass curves $E_{a,b}$ over prime fields $\mathbb{F}_p$ are represented by

$$E_{a,b} : y^2 = x^3 + ax + b$$

where $a, b \in \mathbb{F}_p$, $p > 3$ is a prime number, and the discriminant $-16(4a^3 + 27b^2) \neq 0$. The denotation $E_{a,b}(\mathbb{F}_p)$ is the set of all points (x, y) where $x, y \in \mathbb{F}_p$ that satisfies the above equation together with the point at infinity $\mathcal{O}$. This set forms an abelian group under the point addition operation. The identity point for the group is the point at infinity $\mathcal{O}$. The negative of point $P = (x, y) \in E_{a,b}(\mathbb{F}_p)$ is another point $-P = (x, -y) \in E_{a,b}(\mathbb{F}_p)$.

Let $P = (x_1, y_1) \in E_{a,b}(\mathbb{F}_p)$ and $Q = (x_2, y_2) \in E_{a,b}(\mathbb{F}_p)$ with $P \neq \pm Q$. Then $P + Q = (x_3, y_3) \in E_{a,b}(\mathbb{F}_p)$ can be computed by the formula shown in Table 3.1. It costs $1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$. Let $2P = (x_2, y_2) \in E_{a,b}(\mathbb{F}_p)$. Then, $2P$ can be computed by the formula shown in Table 3.2. It costs $1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$.

Table 3.1
$P + Q$ in Affine Weierstrass Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$s = \frac{y_2 - y_1}{x_2 - x_1}$	$1\mathbf{I} + 1\mathbf{M}$
$x_3 = s^2 - x_1 - x_2$	$1\mathbf{S}$
$y_3 = s \cdot (x_1 - x_3) - y_1$	$1\mathbf{M}$
	$1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$

Table 3.2
$2P$ in Affine Weierstrass Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$\lambda = \frac{3x_1^2 + a}{2y_1}$	$1\mathbf{I} + 1\mathbf{M} + 1\mathbf{S}$
$x_2 = \lambda^2 - 2x_1$	$1\mathbf{S}$
$y_2 = \lambda \cdot (x_1 - x_2) - y_1$	$1\mathbf{M}$
	$1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$

3.1.1 $3P$ formulas. $3P$ can be obtained without a dedicated formula through $2P + P$. The cost of $2P + P$ is $2\mathbf{I} + 4\mathbf{M} + 3\mathbf{S}$. However, many studies show that a

dedicated $3P$ formula in affine coordinates has a lower cost. Ciet et al. (2006) initially proposed a dedicated formula in affine Weierstrass coordinates over prime fields with cost $1\mathbf{I} + 7\mathbf{M} + 4\mathbf{S}$. Yu et al. (2015) proposed the most efficient $3P$ formula in affine Weierstrass coordinates over prime fields with cost $1\mathbf{I} + 7\mathbf{M} + 3\mathbf{S}$. It trades $1\mathbf{I}$ with $3\mathbf{M}$ in comparison to a non-dedicated formula. It saves $1\mathbf{S}$ in comparison to Ciet et al.'s formula. It seems hard to further reduce the cost of Yu et al.'s formula. See Table 3.3 for the cost of this $3P$ formula.

Table 3.3
$3P$ in Affine Weierstrass Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$t = 3x_1^2 + a$	$1\mathbf{S}$
$\alpha = t^2 - 12x_1 \cdot y_1^2$	$1\mathbf{M} + 2\mathbf{S}$
$\beta = -\frac{4y_1}{\alpha}$	$1\mathbf{I} + 1\mathbf{M}$
$x_3 = 2y_1 \cdot \beta \cdot (y_1^2 \cdot 2y_1\beta - t) + x_1$	$3\mathbf{M}$
$y_3 = \beta \cdot (2y_1^3\beta - t) \cdot (t - 4y_1^3\beta) - y_1$	$2\mathbf{M}$
	$1\mathbf{I} + 7\mathbf{M} + 3\mathbf{S}$

3.1.2 A proposal for a $4P$ formula. $4P$ can be obtained without a dedicated formula through $2(2P)$ or $3P + P$. The cost of $2(2P)$ is $2\mathbf{I} + 4\mathbf{M} + 4\mathbf{S}$ and the cost of $3P + P$ is $2\mathbf{I} + 9\mathbf{M} + 4\mathbf{S}$. As a result, $2(2P)$ is the most efficient way to obtain $4P$ without a dedicated formula because $2(2P)$ saves $5\mathbf{M}$ over $3P + P$. However, two studies proposed a dedicated $4P$ formula in affine coordinates to speed up scalar multiplication. Ciet et al. (2006) initially proposed a $4P$ formula in affine Weierstrass coordinates with cost $1\mathbf{I} + 8\mathbf{M} + 1\mathbf{M}_a + 9\mathbf{S}$. Le (2011) also proposed a $4P$ formula in affine Weierstrass coordinates with cost $1\mathbf{I} + 8\mathbf{M} + 8\mathbf{S}$. We see that Le's $4P$ formula saves $1\mathbf{M}_a + 1\mathbf{S}$ over Ciet et al.'s $4P$ formula.

We propose a $4P$ formula in affine Weierstrass coordinates with cost $1\mathbf{I} + 7\mathbf{M} + 1\mathbf{M}_a + 8\mathbf{S}$. When the curve coefficient a is selected as a small constant, our $4P$ formula has a lower cost than Le's $4P$ formula. For example, when $a = -3$, we can save $1\mathbf{M}$ by using our proposed $4P$ formula instead of Le's formula. This is because the cost of our $4P$ formula when $a = -3$ becomes $1\mathbf{I} + 7\mathbf{M} + 8\mathbf{S}$. Our proposed $4P$ formula approximately trades $1\mathbf{I}$ with $3\mathbf{M} + 4\mathbf{S}$ in comparison to $2(2P)$. See Theorem 1 for the $4P$ formula and Table 3.4 for the cost of this formula.

Table 3.4
4P in Affine Weierstrass Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$t = 3x_1^2 + a$	$1\mathbf{S}$
$12x_1y_1^2 = 6\Big((x_1 + y_1^2)^2 - x_1^2 - y_1^4\Big)$	$3\mathbf{S}$
$\alpha = t^2 - 12x_1y_1^2$	$1\mathbf{S}$
$\beta = -(2t \cdot \alpha + 16y_1^4)$	$1\mathbf{M}$
$\lambda = -\frac{t \cdot \beta}{2y_1 \cdot \beta}$	$1\mathbf{I} + 3\mathbf{M}$
$\lambda_2 = \frac{3(\alpha + 4x_1y_1^2)^2 + a \cdot 16y_1^4}{2y_1\beta}$	$1\mathbf{M} + 1\mathbf{M}_a + 1\mathbf{S}$
$x_4 = \lambda_2^2 - 2\lambda^2 + 4x_1$	$2\mathbf{S}$
$y_4 = \lambda_2 \cdot (\lambda^2 - 2x_1 - x_4) - \lambda \cdot (3x_1 - \lambda^2) + y_1$	$2\mathbf{M}$
	$1\mathbf{I} + 7\mathbf{M} + 1\mathbf{M}_a + 8\mathbf{S}$

Theorem 1. Let $P = (x_1, y_1) \in E_{a,b}(\mathbb{F}_p)$. Then $4P = (x_4, y_4)$ is represented by

$$
\begin{aligned}
t &= 3x_1^2 + a \\
\alpha &= t^2 - 12x_1y_1^2 \\
\beta &= -(2t\alpha + 16y_1^4) \\
\lambda &= \tfrac{t\beta}{2y_1\beta} \\
\lambda_2 &= \tfrac{3(\alpha + 4x_1y_1^2)^2 + a16y_1^4}{2y_1\beta} \\
x_4 &= \lambda_2^2 - 2\lambda^2 + 4x_1 \\
y_4 &= \lambda_2(\lambda^2 - 2x_1 - x_4) - \lambda(3x_1 - \lambda^2) + y_1.
\end{aligned}
$$

Proof. We shall prove Theorem 1 by the fact $4P = 2(2P)$. From the $2P$ formula shown in Table 3.2, we have

$$\lambda_2 = \frac{3x_2^2 + a}{2y_2}.$$

We substitute x_2, y_2 with their equivalent terms in Table 3.2. We have

$$
\begin{aligned}
\lambda &= \frac{3x_1^2 + a}{2y_1}. \\
\lambda_2 &= \frac{3(\lambda^2 - 2x_1)^2 + a}{2\lambda(3x_1 - \lambda^2) - 2y_1}.
\end{aligned}
$$

Let $t = 3x_1^2 + a$. We substitute λ with its equivalent term. We have

$$\lambda_2 = \frac{\frac{3(t^2 - 8x_1y_1^2)^2 + a16y_1^4}{16y_1^4}}{\frac{2t(12x_1y_1^2 - t^2) - 16y_1^4}{8y_1^3}}.$$

Let $\alpha = t^2 - 12x_1y_1^2$. We have

$$\lambda_2 = \frac{3(\alpha + 4x_1y_1^2)^2 + a16y_1^4}{2y_1(-2t\alpha - 16y_1^4)}.$$

Let $\beta = -(2t\alpha + 16y_1^4)$. We have

$$\lambda_2 = \frac{3\Big(\alpha + 4x_1y_1^2\Big)^2 + a16y_1^4}{2y_1\beta}.$$

From the $2P$ formula, we have

$$x_4 = \lambda_2^2 - 2x_2.$$
$$y_4 = \lambda_2(3x_2 - \lambda_2^2) - y_2.$$

We use the relations $x_2 = \lambda^2 - 2x_1$ and $y_2 = \lambda(3x_1 - \lambda^2) - y_1$. We have

$$\begin{aligned} x_4 &= \lambda_2^2 - 2\lambda^2 + 4x_1. \\ y_4 &= \lambda_2(3\lambda^2 - 6x_1 - \lambda_2^2) - \lambda(3x_1 - \lambda^2) + y_1 \\ &= \lambda_2(\lambda^2 - 2x_1 - x_4) - \lambda(3x_1 - \lambda^2) + y_1. \end{aligned}$$

We equalize the denominators of λ and λ_2 . We have

$$\lambda = \frac{t\beta}{2y_1\beta}.$$

□

3.1.3 A proposal for a $5P$ formula. $5P$ can be obtained without a dedicated formula through $2(2P) + P$ or $3P + 2P$. The cost of $2(2P) + P$ is $3\mathbf{I} + 6\mathbf{M} + 5\mathbf{S}$ and the cost of $3P + 2P$ is $3\mathbf{I} + 11\mathbf{M} + 6\mathbf{S}$. Therefore, $2(2P) + P$ is the most efficient way to obtain $5P$ without a dedicated formula. Also, $5P$ can be obtained using the dedicated $4P$ formulas with cost $2\mathbf{I} + 9\mathbf{M} + 1\mathbf{M}_a + 9\mathbf{S}$.

However, we can utilize a dedicated $5P$ formula to speed up the performance. We propose a $5P$ dedicated formula in affine Weierstrass coordinates with cost

$1\mathbf{I} + 12\mathbf{M} + 8\mathbf{S}$. This formula trades $2\mathbf{I}$ with $6\mathbf{M} + 3\mathbf{S}$ in comparison to $2(2P) + P$. We see that this proposed formula is useful for devices with $\mathbf{I}/\mathbf{M}$ ratio $\geq 5\mathbf{M}$. It also trades $1\mathbf{I}$ with $3\mathbf{M} - 1\mathbf{M}_a - 1\mathbf{S}$ in comparison to $4P + P$. To the best of our knowledge, this is the first dedicated $5P$ formula in affine Weierstrass coordinates over prime fields. See Theorem 2 for the formula and Table 3.5 for the cost of this formula.

Table 3.5
$5P$ in Affine Weierstrass Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$t = 3x_1^2 + a$	$1\mathbf{S}$
$12x_1y_1^2 = 6\Big((x_1 + y_1^2)^2 - x_1^2 - y_1^4\Big)$	$3\mathbf{S}$
$\alpha = t^2 - 12x_1y_1^2$	$1\mathbf{S}$
$2y_1\alpha = (y_1 + \alpha)^2 - y_1^2 - \alpha^2$	$2\mathbf{S}$
$\beta = 16y_1^4 \cdot (2t \cdot \alpha + 16y_1^4) - \alpha \cdot \alpha^2$	$3\mathbf{M}$
$\lambda = \frac{t\alpha \cdot \beta}{2y_1\alpha \cdot \beta}$	$1\mathbf{I} + 3\mathbf{M}$
$s = -\Big(\frac{16y_1^4 \cdot \beta}{2y_1\alpha\beta} + \lambda\Big)$	$1\mathbf{M}$
$s_2 = \frac{16y_1^4(2t\alpha + 16y_1^4) \cdot (2t\alpha + 16y_1^4)}{2y_1\alpha\beta} - \lambda$	$2\mathbf{M}$
$x_5 = (s_2 - s) \cdot (s_2 + s) + x_1$	$1\mathbf{M}$
$y_5 = s_2 \cdot (\lambda^2 - 2x_1 - x_5) - \lambda \cdot (3x_1 - \lambda^2) + y_1$	$2\mathbf{M} + 1\mathbf{S}$
	$1\mathbf{I} + 12\mathbf{M} + 8\mathbf{S}$

Theorem 2. Let $P = (x_1, y_1) \in E_{a,b}(\mathbb{F}_p)$. Then $5P = (x_5, y_5)$ is represented by

$$
\begin{aligned}
t &= 3x_1^2 + a \\
\alpha &= t^2 - 12x_1y_1^2 \\
\beta &= 16y_1^4(2t\alpha + 16y_1^4) - \alpha^3 \\
\lambda &= \tfrac{t\alpha\beta}{2y_1\alpha\beta} \\
s &= -\Big(\tfrac{16y_1^4\beta}{2y_1\alpha\beta} + \lambda\Big) \\
s_2 &= \tfrac{16y_1^4(2t\alpha + 16y_1^4)^2}{2y_1\alpha\beta} - \lambda \\
x_5 &= (s_2 - s)(s_2 + s) + x_1 \\
y_5 &= s_2(\lambda^2 - 2x_1 - x_5) - \lambda(3x_1 - \lambda^2) + y_1.
\end{aligned}
$$

Proof. We shall prove Theorem 2 by the fact $5P = 3P + 2P$. From the $P + Q$ formula shown in Table 3.1, we have

$$s_2 = \frac{y_3 - y_2}{x_3 - x_2}.$$

We use the following relations from the $2P$ and $3P$ formulas

$$
\begin{aligned}
t &= 3x_1^2 + a. \\
\lambda &= \frac{t}{2y_1}. \\
x_2 &= \lambda^2 - 2x_1. \\
y_2 &= \lambda(3x_1 - \lambda^2) - y_1. \\
\alpha &= t^2 - 12x_1y_1^2. \\
s &= -\Big(\frac{16y_1^4}{2y_1\alpha} + \lambda\Big). \\
x_3 &= (s-\lambda)(s+\lambda) + x_1. \\
y_3 &= -\Big(s(x_3 - x_1) + y_1\Big).
\end{aligned}
$$

Therefore, s_2 becomes

$$
\begin{aligned}
s_2 &= \frac{-s(s-\lambda)(s+\lambda) - \lambda(3x_1 - \lambda^2)}{(s-\lambda)(s+\lambda) + 3x_1 - \lambda^2} \\
&= \frac{\frac{(t\alpha+16y_1^4)(2t\alpha+16y_1^4)16y_1^4+t\alpha^4}{8y_1^3\alpha^3}}{\frac{(2t\alpha+16y_1^4)16y_1^4-\alpha^3}{4y_1^2\alpha^2}} \\
&= \frac{(t\alpha+16y_1^4)(2t\alpha+16y_1^4)16y_1^4+t\alpha^4}{2y_1\alpha\Big((2t\alpha+16y_1^4)16y_1^4-\alpha^3\Big)}.
\end{aligned}
$$

Let $\beta = (2t\alpha + 16y_1^4)16y_1^4 - \alpha^3$. We have

$$
s_2 = \frac{(t\alpha+16y_1^4)(2t\alpha+16y_1^4)16y_1^4+t\alpha^4}{2y_1\alpha\beta}.
$$

We note that $(t\alpha + 16y_1^4)(2t\alpha + 16y_1^4)16y_1^4 + t\alpha^4 = 16y_1^4(2t\alpha + 16y_1^4)^2 - t\alpha\beta$.

Therefore, we have

$$
\begin{aligned}
s_2 &= \frac{16y_1^4(2t\alpha + 16y_1^4)^2 - t\alpha\beta}{2y_1\alpha\beta} \\
&= \frac{16y_1^4(2t\alpha + 16y_1^4)^2}{2y_1\alpha\beta} - \lambda.
\end{aligned}
$$

From the $P + Q$ formula, we have

$$x_5 = s_2^2 - x_2 - x_3.$$
$$y_5 = s_2(x_2 - x_5) - y_2.$$

We substitute x_2, y_2, x_3 with their equivalent terms. We have

$$\begin{aligned} x_5 &= s_2^2 - \lambda^2 - (s - \lambda)(s + \lambda) + x_1 \\ &= s_2^2 - s^2 + x_1 \\ &= (s_2 - s)(s_2 + s) + x_1. \\ y_5 &= s_2(\lambda^2 - 2x_1 - x_5) - \lambda(3x_1 - \lambda^2) + y_1. \end{aligned}$$

We equalize the denominators of λ, s and s_2. We have

$$\lambda = \frac{t\alpha\beta}{2y_1\alpha\beta}.$$
$$s = -\Big(\frac{16y_1^4\beta}{2y_1\alpha\beta} + \lambda\Big).$$

□

3.2 Jacobian Coordinates

The well-known projective coordinates for simplified Weierstrass curves over prime fields are Chudnovsky and Jacobian coordinates. The main advantage of Jacobian coordinates is that they have the most efficient $2P$ formula for simplified Weierstrass curves over prime fields. In contrast, Chudnovsky coordinates have the most efficient $P + Q$ formula for simplified Weierstrass curves over prime fields. We focus on projective coordinates that have the most efficient $2P$ formula for the following reasons. $2P$ has a higher impact on the performance of multi-base scalar multiplication methods than $P + Q$. An efficient $2P$ formula can lead us to derive efficient $3P$ and $5P$ formulas. For these reasons, many researchers proposed efficient formulas for Jacobian coordinates. In this subsection, we show their efforts to derive efficient formulas in Jacobian coordinates.

In Jacobian coordinates, a point is represented as (X, Y, Z). An affine point can be converted to a projective point by using the relation $(X, Y, Z) = (x, y, 1)$. A projective point can be reversed to an affine point by using the relation $(x, y) = (\frac{X}{Z^2}, \frac{Y}{Z^3})$ where $Z \neq 0$. The simplified Weierstrass curves in Jacobian coordinates are represented by

$$Y^2 = X^3 + aXZ^4 + bZ^6.$$

3.2.1 $P + Q$ **formulas.** The cost of the original $P + Q$ formula in Jacobian coordinates is $12\mathbf{M} + 4\mathbf{S} \approx 15.2\mathbf{M}$. The mixed $P + Q$ formula can be deduced from the $P + Q$ when $Z_1 = 1$. As a result, the mixed $P + Q$ formula costs $8\mathbf{M} + 3\mathbf{S} \approx 10.4\mathbf{M}$. The original $P + Q$ formula in Jacobian coordinates costs extra $1\mathbf{M} + 1\mathbf{S}$ in comparison to the $P + Q$ formula in Chudnovsky coordinates. However, the mixed $P + Q$ formulas in both Jacobian and Chudnovsky coordinates have the same cost. See Hankerson et al. (2004) for more details about the cost of $P + Q$ formulas in Jacobian and Chudnovsky coordinates.

Longa and Miri (2008a) proposed a minor improvement for the original $P + Q$ formula. They used the facts $x = \frac{2^2X}{2^2Z^2}, y = \frac{2^3Y}{2^3Z^3}$ and applied the relation $2 \cdot a \cdot b = (a + b)^2 - a^2 - b^2$ to trade $1\mathbf{M}$ with $1\mathbf{S}$. Their formula costs $11\mathbf{M} + 5\mathbf{S} \approx 15\mathbf{M}$, as Table 3.6 shows. We see that their mixed $P + Q$ formula costs $7\mathbf{M} + 4\mathbf{S} \approx 10.2\mathbf{M}$. It is important to note that the improved $P + Q$ formula has extra multiplications with small constants in comparison to the original $P + Q$ formula.

Table 3.6
$P + Q$ in Jacobian Weierstrass Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$F = X_1 \cdot Z_2^2 - X_2 \cdot Z_1^2$	$2\mathbf{M} + 2\mathbf{S}$
$G = Y_1 \cdot Z_2^3 - Y_2 \cdot Z_1^3$	$4\mathbf{M}$
$X_3 = 4\Big(G^2 - F^3 - 2X_2Z_1^2 \cdot F^2\Big)$	$2\mathbf{M} + 2\mathbf{S}$
$Y_3 = 2\Big(G \cdot (4X_2Z_1^2F^2 - X_3) - 4Y_2Z_1^3 \cdot F^3\Big)$	$2\mathbf{M}$
$Z_3 = \Big((F + Z_2)^2 - F^2 - Z_2^2\Big) \cdot Z_1$	$1\mathbf{M} + 1\mathbf{S}$
	$11\mathbf{M} + 5\mathbf{S}$

3.2.2 $2P$ **formulas.** The cost of the original $2P$ formula in Jacobian coordinates is $3\mathbf{M} + 1\mathbf{M}_a + 6\mathbf{S} \approx 7.8\mathbf{M}$. When the curve coefficient $a = -3$, the cost of $2P$ becomes $4\mathbf{M} + 4\mathbf{S} \approx 7.2\mathbf{M}$. This is because the relation $3X^2 - 3Z^4 = 3(X + Z^2)(X - Z^2)$ reduces the cost from $1\mathbf{M}_a + 3\mathbf{S}$ to $1\mathbf{M} + 1\mathbf{S}$. The $2P$ formula in Jacobian coordinates saves $1\mathbf{M}$ in comparison to $2P$ in Chudnovsky coordinates. See Hankerson et al. (2004) for more details about the cost of the $2P$ formulas in Jacobian and Chudnovsky coordinates.

Similar to the $P + Q$ formula, Longa and Miri (2008a) suggested a minor improvement to the original $2P$ formula by trading $2\mathbf{M}$ with $2\mathbf{S}$. They used the same idea of improving $P + Q$ to improve the $2P$ formula. As a result, their $2P$ formula costs $1\mathbf{M} + 1\mathbf{M}_a + 8\mathbf{S} \approx 7.4\mathbf{M}$, as Table 3.7 shows. Their $2P$ formula can be reduced to $3\mathbf{M} + 5\mathbf{S} \approx 7\mathbf{M}$ when the coefficient $a = -3$.

Table 3.7
$2P$ in Jacobian Weierstrass Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts	$a = -3$
$T = 3X_1^2 + a \cdot Z_1^4$	$1\mathbf{M}_a + 3\mathbf{S}$	$1\mathbf{M} + 1\mathbf{S}$
$8X_1Y_1^2 = 4\Big((X_1 + Y_1^2)^2 - X_1^2 - Y_1^4\Big)$	$3\mathbf{S}$	
$X_2 = T^2 - 8X_1Y_1^2$	$1\mathbf{S}$	$1\mathbf{M} + 2\mathbf{S}$
$Y_2 = T \cdot (4X_1Y_1^2 - X_2) - 8Y_1^4$	$1\mathbf{M}$	$1\mathbf{M} + 1\mathbf{S}$
$Z_2 = (Y_1 + Z_1)^2 - Y_1^2 - Z_1^2$	$1\mathbf{S}$	$1\mathbf{S}$
	$1\mathbf{M} + 1\mathbf{M}_a + 8\mathbf{S}$	$3\mathbf{M} + 5\mathbf{S}$

3.2.3 $3P$ **formulas.** Recall that $3P$ can be obtained without a dedicated formula through $2P + P$. The cost of $2P + P$ in Jacobian coordinates is $8\mathbf{M} + 1\mathbf{M}_a + 12\mathbf{S} \approx 17.6\mathbf{M}$. However, the following studies show that a dedicated $3P$ formula has a lower cost. Dimitrov, Imbert, and Mishra (2008) initially proposed a dedicated $3P$ formula with cost $9\mathbf{M} + 1\mathbf{M}_a + 6\mathbf{S} \approx 13.8\mathbf{M}$.

Later, Longa and Miri (2008a) proposed the most efficient $3P$ formula in Jacobian coordinates with cost $6\mathbf{M} + 1\mathbf{M}_a + 9\mathbf{S} \approx 13.2\mathbf{M}$. They also further improved their $3P$ formula by using the same technique that was used to improve the $P + Q$ formula. As a result, their $3P$ formula costs $5\mathbf{M} + 1\mathbf{M}_a + 10\mathbf{S} \approx 13\mathbf{M}$, as Table 3.8 shows. Their $3P$ formula costs $7\mathbf{M} + 7\mathbf{S} \approx 12.6\mathbf{M}$ when the coefficient $a = -3$. We see that a dedicated

$3P$ formula can save $4.6\mathbf{M}$ in comparison to $2P + P$ in Jacobian coordinates.

Table 3.8
$3P$ in Jacobian Weierstrass Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts	$a = -3$
$T = 3X_1^2 + a \cdot Z_1^4$	$1\mathbf{M}_a + 3\mathbf{S}$	$1\mathbf{M} + 1\mathbf{S}$
$12X_1Y_1^2 = 6\Big((X_1 + Y_1^2)^2 - X_1^2 - Y_1^4\Big)$	$3\mathbf{S}$	
$A = T^2 - 12X_1Y_1^2$	$1\mathbf{S}$	$1\mathbf{M} + 2\mathbf{S}$
$B = (T + A)^2 - T^2 - A^2 + 16Y_1^4$	$2\mathbf{S}$	$3\mathbf{S}$
$X_3 = 4\Big(4Y_1^2 \cdot B + X_1 \cdot A^2\Big)$	$2\mathbf{M}$	$2\mathbf{M}$
$Y_3 = 8Y_1 \cdot \Big((B + 16Y_1^4) \cdot B - A^3\Big)$	$3\mathbf{M}$	$3\mathbf{M}$
$Z_3 = (A + Z_1)^2 - A^2 - Z_1^2$	$1\mathbf{S}$	$1\mathbf{S}$
	$5\mathbf{M} + 1\mathbf{M}_a + 10\mathbf{S}$	$7\mathbf{M} + 7\mathbf{S}$

3.2.4 $5P$ **formulas.** $5P$ can be obtained without a dedicated formula through $2(2P) + P$ or $3P + 2P$. The cost of $2(2P) + P$ is $13\mathbf{M} + 2\mathbf{M}_a + 21\mathbf{S} \approx 29.8\mathbf{M}$ and it is more efficient than $3P + 2P$. However, the following studies show that using a dedicated $5P$ formula can significantly reduce the cost of obtaining $5P$. Mishra and Dimitrov (2007) initially proposed a dedicated $5P$ formula with cost $14\mathbf{M} + 1\mathbf{M}_a + 10\mathbf{S} \approx 22\mathbf{M}$. Longa and Miri (2008c) also proposed a $5P$ formula. When the curve coefficient $a = -3$, their $5P$ formula costs $11\mathbf{M} + 11\mathbf{S} \approx 19.8\mathbf{M}$.

Later, Giorgi et al. (2009) proposed the most efficient $5P$ formula in Jacobian coordinates with cost $7\mathbf{M} + 1\mathbf{M}_a + 16\mathbf{S} \approx 19.8\mathbf{M}$. When the curve coefficient $a = -3$, their $5P$ formula costs $9\mathbf{M} + 13\mathbf{S} \approx 19.4\mathbf{M}$, as Table 3.9 shows. We see that a dedicated $5P$ formula saves approximately $10\mathbf{M}$ in comparison to $2(2P) + P$ in Jacobian coordinates.

3.3 Extended Jacobian Coordinates

In this subsection, we propose new projective coordinates for simplified Weierstrass curves over prime fields, which is called extended Jacobian coordinates. The idea of the proposed coordinates is to extend Jacobian coordinates with the value $\lambda = \frac{3x^2+a}{2y}$. An affine point in extended Jacobian coordinates is represented as (x, y, λ). A projective point in extended Jacobian coordinates is represented as (X, Y, L, Z). An affine point can be converted to a projective point by using the relation

Table 3.9
5P in Jacobian Weierstrass Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$T = 3X_1^2 + a \cdot Z_1^4$	$1\mathbf{M}_a + 3\mathbf{S}$
$12X_1Y_1^2 = 6\Big((X_1 + Y_1^2)^2 - X_1^2 - Y_1^4\Big)$	$3\mathbf{S}$
$A = T^2 - 12X_1Y_1^2$	$1\mathbf{S}$
$B = (T + A)^2 - T^2 - A^2 + 16Y_1^4$	$2\mathbf{S}$
$C = -16Y_1^4 \cdot B + A^3$	$2\mathbf{M}$
$2Y_1^2B = (B + Y_1^2)^2 - B^2 - Y_1^4$	$2\mathbf{S}$
$2A(C - B^2) = (A + C - B^2)^2 - A^2 - (C - B^2)^2$	$2\mathbf{S}$
$X_5 = 4\Big(X_1 \cdot C^2 - 2Y_1^2B \cdot 2A(C - B^2)\Big)$	$2\mathbf{M} + 1\mathbf{S}$
$Y_5 = 4Y_1 \cdot \Big(32Y_1^4B \cdot B^4 - A^3 \cdot \Big(C^2 + B^4 - 3(C - B^2)^2\Big)\Big)$	$3\mathbf{M} + 1\mathbf{S}$
$Z_5 = (C + Z_1)^2 - C^2 - Z_1^2$	$1\mathbf{S}$
	$7\mathbf{M} + 1\mathbf{M}_a + 16\mathbf{S}$

$(X, Y, L, Z) = (x, y, \lambda, 1)$. A projective point can be reversed to an affine point by using the relation $(x, y, \lambda) = (\frac{X}{Z^2}, \frac{Y}{Z^3}, \frac{L}{Z})$ where $Z \neq 0$. The curve equation in extended Jacobian coordinates can be represented by

$$3X^2 + aZ^4 = 2LY.$$

Table 3.10
2P in Extended Jacobian Weierstrass Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$A = L_1^2 - 2X_1$	$1\mathbf{S}$
$B = L_1 \cdot (X_1 - A) - Y_1$	$1\mathbf{M}$
$X_2 = 4A \cdot B^2$	$1\mathbf{M} + 1\mathbf{S}$
$Y_2 = 8B^4$	$1\mathbf{S}$
$L_2 = 3(A - X_1) \cdot (A + X) + 2L_1 \cdot Y_1$	$2\mathbf{M}$
$Z_2 = 2Z_1 \cdot B$	$1\mathbf{M}$
	$5\mathbf{M} + 3\mathbf{S}$

3.3.1 A proposal for a $2P$ **formula.** The cost of the proposed $2P$ formula is $5\mathbf{M} + 3\mathbf{S} \approx 7.4\mathbf{M}$, as Table 3.10 shows. We saw that the cost of $2P$ in Jacobian coordinates is $1\mathbf{M} + 1\mathbf{M}_a + 8\mathbf{S}$. The question is which one has a lower cost: $5\mathbf{M} + 3\mathbf{S}$ or $1\mathbf{M} + 1\mathbf{M}_a + 8\mathbf{S}$? The answer is that it depends on the selection of the curve coefficient a. When $a = -3$, the $2P$ in extended Jacobian coordinates costs approximately extra $0.4\mathbf{M}$ in comparison to $2P$ in Jacobian coordinates. This is because $2P$ n Jacobian

coordinates costs $3\mathbf{M} + 5\mathbf{S} \approx 7\mathbf{M}$, as Table 3.7 shows. However, when a is a large constant, $2P$ in extended coordinates saves $0.6\mathbf{M}$ in comparison to $2P$ in Jacobian coordinates. See Theorem 3 for our proposed $2P$ formula.

Theorem 3. Let $P = (X_1, Y_1, L_1, Z_1) \in E_{a,b}(\mathbb{F}_p)$. Then $2P = (X_2, Y_2, L_2, Z_2)$ in extended Jacobian coordinates is represented by

$$\begin{aligned}
A &= L_1^2 - 2X_1 \\
B &= L_1(X_1 - A) - Y_1 \\
X_2 &= 4AB^2 \\
Y_2 &= 8B^4 \\
L_2 &= 3(A - X_1)(A + X_1) + 2L_1Y_1 \\
Z_2 &= 2Z_1B.
\end{aligned}$$

Proof. From the $2P$ formula as shown in Table 3.2, we have

$$\begin{aligned}
x_2 &= \lambda^2 - 2x_1. \\
y_2 &= \lambda(x_1 - x_2) - y_1. \\
\lambda_2 &= \frac{3x_2^2 + a}{2y_2}.
\end{aligned}$$

We convert an affine point to a projective point by the relation $(x, y, \lambda) = (\frac{X}{Z^2}, \frac{Y}{Z^3}, \frac{L}{Z})$. We have

$$\begin{aligned}
x_2 &= \frac{L_1^2 - 2X_1}{Z_1^2} = \frac{A}{Z_1^2}. \\
y_2 &= \frac{L_1(X_1 - A) - Y_1}{Z_1^3} = \frac{B}{Z_1^3}. \\
\lambda_2 &= \frac{3A^2 + aZ_1^4}{2Z_1B}.
\end{aligned}$$

From the curve equation $aZ_1^4 = 2L_1Y_1 - 3X_1^2$, we have

$$3A^2 + aZ_1^4 = 3A^2 + 2L_1Y_1 - 3X_1^2 = 3(A - X_1)(A + X_1) + 2L_1Y_1.$$

We equalize the denominators. We have

$$x_2 = \frac{4AB^2}{(2Z_1B)^2} = \frac{X_2}{Z_2^2}.$$
$$y_2 = \frac{8B^4}{(2Z_1B)^3} = \frac{Y_2}{Z_2^3}.$$
$$\lambda_2 = \frac{3(A - X_1)(A + X_1) + 2L_1Y_1}{2Z_1B} = \frac{L_2}{Z_2}.$$

□

3.3.2 A proposal for a $P + Q$ formula. The $P + Q$ formula in Jacobian coordinates can be used in extended Jacobian coordinates. However, we need to add extra computation for λ. Let $P + Q = (X_3, Y_3, Z_3)$ be a point represented by Jacobian coordinates and $P + Q = (\mathbf{X_3}, \mathbf{Y_3}, \mathbf{L_3}, \mathbf{Z_3})$ be a point represented by extended Jacobian coordinates. Then $\mathbf{X_3} = X_3, \mathbf{Y_3} = Y_3, \mathbf{Z_3} = Z_3$, and $\mathbf{L_3}$ can be obtained by using the relation:

$$\lambda_3 = \frac{3X_3^2 + aZ_3^4}{2Z_3Y_3}.$$

Moreover, we need to equalize the denominators of x_3, y_3 and λ_3. Therefore, we have

$$\mathbf{X_3} = 4X_3Y_3^2.$$
$$\mathbf{Y_3} = 8Y_3^4.$$
$$\mathbf{L_3} = 3X_3^2 + aZ_3^4.$$
$$\mathbf{Z_3} = 2Z_3Y_3.$$

The cost of the extra step to obtain $\mathbf{L_3}$ is $1\mathbf{M}_a + 7\mathbf{S}$. Therefore, the cost of $P + Q$ is $12\mathbf{M} + 1\mathbf{M}_a + 11\mathbf{S}$ and the cost of mixed $P + Q$ is $8\mathbf{M} + 1\mathbf{M}_a + 10\mathbf{S}$. See Table 3.11 for the cost of $P + Q$ in extended Jacobian coordinates.

3.4 Twisted Edwards Curves

Bernstein and Lange (2007a) initially introduced Edwards curves over prime fields $\mathbb{F}_p$. Edwards curves E_d are represented by

$$E_d : x^2 + y^2 = 1 + dx^2y^2$$

Table 3.11
$P + Q$ in Extended Jacobian Weierstrass Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$F = X_1 \cdot Z_2^2 - X_2 \cdot Z_1^2$	$2\mathbf{M} + 2\mathbf{S}$
$G = Y_1 \cdot Z_2^3 - Y_2 \cdot Z_1^3$	$4\mathbf{M}$
$X_3 = G^2 - F^3 - 2X_2Z_1^2 \cdot F^2$	$2\mathbf{M} + 2\mathbf{S}$
$Y_3 = G \cdot (X_2Z_1^2F^2 - X_3) - Y_2Z_1^3 \cdot F^3$	$2\mathbf{M}$
$Z_3 = F \cdot Z_2 \cdot Z_1$	$2\mathbf{M}$
$\mathbf{X_3} = 2\Big((X_3 + Y_3^2)^2 - X_3^2 - Y_3^4\Big)$	$4\mathbf{S}$
$\mathbf{Y_3} = 8Y_3^4$	
$\mathbf{L_3} = 3X_3^2 + aZ_3^4$	$1\mathbf{M}_a + 2\mathbf{S}$
$\mathbf{Z_3} = (Z_3 + Y_3)^2 - Z_3^2 - Y_3^2$	$1\mathbf{S}$
	$12\mathbf{M} + 1\mathbf{M}_a + 11\mathbf{S}$

where $d \in \mathbb{F}_p$, $p > 3$ is a prime number, and $d \neq \{0, 1\}$. Later, Bernstein et al. (2008) generalized Edwards curves and called them twisted Edwards curves. Twisted Edwards curves $E_{a,d}$ over prime fields $\mathbb{F}_p$ are represented by

$$E_{a,d} : ax^2 + y^2 = 1 + dx^2y^2$$

where $a, d \in \mathbb{F}_p$, $p > 3$ is a prime number, $a \neq d$ and $a, d \neq 0$. We see that Edwards curves are twisted Edwards curves with the coefficient $a = 1$. We denote $E_{a,d}(\mathbb{F}_p)$ to be the set of all points (x, y) where $x, y \in \mathbb{F}_p$ that satisfies the above equation. This set forms an abelian group under the point addition operation. The identity point for the group is point $(0, 1)$. The negative of point $P = (x, y) \in E_{a,d}(\mathbb{F}_p)$ is another point $-P = (-x, y) \in E_{a,d}(\mathbb{F}_p)$. One advantage of $E_{a,d}$ is that it can resist side-channel attacks. This is because $E_{a,d}$ has a unified formula for both $P + Q$ and $2P$. Let $P = (x_1, y_1) \in E_{a,d}(\mathbb{F}_p)$ and $Q = (x_2, y_2) \in E_{a,d}(\mathbb{F}_p)$. Then $P + Q = (x_3, y_3) \in E_{a,d}(\mathbb{F}_p)$ can be computed by

$$x_3 = \frac{x_1y_2 + y_1x_2}{1 + dx_1y_1x_2y_2}$$
$$y_3 = \frac{y_1y_2 - ax_1x_2}{1 - dx_1y_1x_2y_2}.$$

3.4.1 Standard projective coordinates. Twisted Edwards curves can be represented mainly in two projective coordinate systems: standard and inverted

(Bernstein et al., 2008). We focus on standard projective coordinates because they are the most efficient system in the context of the MBNS. To explain, $2P$ in standard projective coordinates saves $1\mathbf{M}_d$ in comparison to $2P$ in inverted projective coordinates (Bernstein & Lange, 2007b). In contrast, $P+Q$ in standard projective coordinates costs extra $1\mathbf{M}$ in comparison to $P+Q$ in inverted projective coordinates. In the context of the MBNS, $2P$ is used more repeatedly than $P+Q$ during scalar multiplication operations. As a result, $2P$ has a greater impact on scalar multiplication than $P+Q$.

In standard projective coordinates, we represent a point as (X, Y, Z). An affine point can be mapped to a projective point by using the relation $(X, Y, Z) = (x, y, 1)$. A projective point can be mapped to an affine point by using the relation $(x, y) = (\frac{X}{Z}, \frac{Y}{Z})$ where $Z \neq 0$. The negative of point $P = (X, Y, Z)$ in standard projective coordinates is another point $-P = (-X, Y, Z)$. The curve equation in standard projective coordinates can be represented by

$$(aX^2 + Y^2)Z^2 = Z^4 + dX^2Y^2.$$

Table 3.12
The Cost of Efficient Formulas in Twisted Edwards and Jacobian Weierstrass Coordinates over $\mathbb{F}_p$

	Standard twisted Edwards ($a = 1$)	Jacobian Weierstrass ($a = -3$)
Mixed $P+Q$	$9\mathbf{M} + 1\mathbf{S} \approx 9.8\mathbf{M}$ (this work)	$7\mathbf{M} + 4\mathbf{S} \approx 10.2\mathbf{M}$
$P+Q$	$10\mathbf{M} + 1\mathbf{S} \approx 10.8\mathbf{M}$ (this work)	$11\mathbf{M} + 5\mathbf{S} \approx 15\mathbf{M}$
$2P$	$3\mathbf{M} + 4\mathbf{S} \approx 6.2\mathbf{M}$	$3\mathbf{M} + 5\mathbf{S} \approx 7\mathbf{M}$
$3P$	$9\mathbf{M} + 3\mathbf{S} \approx 11.4\mathbf{M}$	$7\mathbf{M} + 7\mathbf{S} \approx 12.6\mathbf{M}$
Mixed $2Q+P$	$11\mathbf{M} + 4\mathbf{S} \approx 14.2\mathbf{M}$ (this work)	$11\mathbf{M} + 7\mathbf{S} \approx 16.6\mathbf{M}$
$5P$	$15\mathbf{M} + 3\mathbf{S} \approx 17.4\mathbf{M}$ (this work)	$9\mathbf{M} + 13\mathbf{S} \approx 19.4\mathbf{M}$

$\approx$ means $1\mathbf{S} = 0.8\mathbf{M}$.

3.4.2 A proposal for a $P+Q$ formula. In standard projective coordinates, $P+Q$ costs $10\ \mathbf{M} + 1\mathbf{M}_d + 1\ \mathbf{M}_a + 1\mathbf{S} \approx 11.8\mathbf{M}$, as Table 3.13 shows. Mixed $P+Q$ costs $9\ \mathbf{M} + 1\mathbf{M}_d + 1\mathbf{M}_a + 1\mathbf{S} \approx 10.8\mathbf{M}$. In Table 3.13, we say that the conjugate of F is $\bar{F}$. The conjugate of a binomial can be obtained by changing the sign between the two terms. For example, let $T = Y + X$. Then $\bar{T} = Y - X$. As mentioned earlier, $P+Q$ costs extra $1\mathbf{M}$ in comparison to inverted projective coordinates. It also costs approximately extra $0.6\mathbf{M}$ in comparison to $P+Q$ in Jacobian Weierstrass coordinates

over $\mathbb{F}_p$, as Table 3.12 shows.

Table 3.13
$P + Q$ in Standard Twisted Edwards Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$F = (Z_1 \cdot Z_2)^2 - d \cdot X_1 \cdot X_2 \cdot Y_1 \cdot Y_2$	$4\mathbf{M} + 1\mathbf{M}_d + 1\mathbf{S}$
$X_3 = Z_1 Z_2 \cdot F \cdot \Big((X_1 + Y_1) \cdot (X_2 + Y_2) - X_1 X_2 - Y_1 Y_2\Big)$	$3\mathbf{M}$
$Y_3 = Z_1 Z_2 \cdot \bar{F} \cdot (Y_1 Y_2 - a \cdot X_1 X_2)$	$2\mathbf{M} + 1\mathbf{M}_a$
$Z_3 = F \cdot \bar{F}$	$1\mathbf{M}$
	$10\mathbf{M} + 1\mathbf{M}_d + 1\mathbf{M}_a + 1\mathbf{S}$

$\bar{F}$ is the conjugate of F.

Table 3.14
$P + Q$ with Pre-computation in Standard Twisted Edwards Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$F = (Z_1 \cdot Z_2)^2 - \underline{dX_1Y_1} \cdot X_2 \cdot Y_2$	$3\mathbf{M} + 1\mathbf{S}$
$X_3 = Z_1 Z_2 \cdot F \cdot \Big((X_1 + X_2) \cdot (Y_1 + Y_2) - \underline{X_1 Y_1} - X_2 Y_2\Big)$	$3\mathbf{M}$
$Y_3 = Z_1 Z_2 \cdot \bar{F} \cdot \Big((X_2 + Y_1) \cdot (Y_2 - \underline{aX_1}) - X_2 Y_2 + \underline{aX_1 Y_1}\Big)$	$3\mathbf{M}$
$Z_3 = F \cdot \bar{F}$	$1\mathbf{M}$
	$10\mathbf{M} + 1\mathbf{S}$

$\bar{F}$ is the conjugate of F.
Underlined terms are pre-computed.

Let $P = (X_1, Y_1, Z_1) \in E_{a,d}(\mathbb{F}_p)$ be the base point and $Q = (X_2, Y_2, Z_2) \in E_{a,d}(\mathbb{F}_p)$ be a temporary derived point. Then, we can improve the $P + Q$ formula by using the pre-computation concept. First, before performing scalar multiplication, we pre-compute the values $X_1 \cdot Y_1, d \cdot X_1 Y_1, a \cdot X_1 Y_1$, and $a \cdot X_1$. This pre-computation step costs $1\mathbf{M} + 1\mathbf{M}_d + 2\mathbf{M}_a \approx 2\mathbf{M}$. Second, during scalar multiplication, we use the pre-computed values with the proposed $P + Q$ formula, as Table 3.14 shows. We see that the $P + Q$ formula with pre-computation saves $1\mathbf{M}_d + 1\mathbf{M}_a \approx 1\mathbf{M}$ in comparison to the $P + Q$ formula without pre-computation. It saves $0.4\mathbf{M}$ in comparison to the mixed $P + Q$ formula in Jacobian Weierstrass coordinates over $\mathbb{F}_p$, as Table 3.12 shows.

3.4.3 $2P$ and $3P$ formulas. $2P$ in twisted Edwards coordinates is one of the most efficient $2P$ formulas for elliptic curves over prime fields. The cost of this formula is $3\mathbf{M} + 1\mathbf{M}_a + 4\mathbf{S} \approx 6.2\mathbf{M}$, as Table 3.15 shows. It saves approximately $0.8\mathbf{M}$ in comparison to $2P$ in Jacobian Weierstrass ($a = -3$) coordinates, as Table 3.12 shows.

Table 3.15
2P in Standard Twisted Edwards Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$T = Y_1^2 + a \cdot X_1^2$	$1\mathbf{M}_a + 2\mathbf{S}$
$X_2 = \left((X_1 + Y_1)^2 - X_1^2 - Y_1^2\right) \cdot (T - 2Z_1^2)$	$1\mathbf{M} + 2\mathbf{S}$
$Y_2 = -T \cdot \bar{T}$	$1\mathbf{M}$
$Z_2 = T \cdot (T - 2Z_1^2)$	$1\mathbf{M}$
	$3\mathbf{M} + 1\mathbf{M}_a + 4\mathbf{S}$

$\bar{T}$ is the conjugate of T.

Recall that $3P$ can be obtained without a dedicated formula through $2P + P$. The cost of $2P + P$ in twisted Edwards coordinates is $13\mathbf{M} + 2\mathbf{M}_d + 1\mathbf{M}_a + 5\mathbf{S} \approx 19\mathbf{M}$. However, many studies show that a dedicated $3P$ formula has a lower cost. Bernstein, Birkner, Lange, and Peters (2007) initially proposed dedicated $3P$ formulas in Edwards coordinates. They proposed two $3P$ formulas in standard Edwards coordinates with cost $9\mathbf{M} + 4\mathbf{S} \approx 12.2\mathbf{M}$ and $7\mathbf{M} + 7\mathbf{S} \approx 12.6\mathbf{M}$. They also proposed dedicated $3P$ formulas in inverted Edwards coordinates with extra $1\mathbf{M}_d$ in comparison to $3P$ standard Edwards coordinates. Later, Li et al. (2016) proposed a minor improvement to the $3P$ formula in standard Edwards coordinates by trading $1\mathbf{M}$ with $1\mathbf{S}$.

Recently, Bernstein et al. (2017) proposed the most efficient $3P$ formula in twisted Edwards coordinates with cost $9\mathbf{M} + 1\mathbf{M}_a + 3\mathbf{S} \approx 11.4\mathbf{M}$, as Table 3.16 shows. It saves approximately $1.2\mathbf{M}$ in comparison to the $3P$ in Jacobian Weierstrass ($a = -3$) coordinates, as Table 3.12 shows. The $3P$ formula in standard twisted Edwards coordinates needs 2 temporary variables to perform the formula terms. See Appendix for the steps to perform this $3P$ formula with the fewest temporary variables.

Table 3.16
3P in Standard Twisted Edwards Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$T = Y_1^2 + a \cdot X_1^2$	$1\mathbf{M}_a + 2\mathbf{S}$
$A = T \cdot \bar{T} + 2Y_1^2 \cdot (T - 2Z_1^2)$	$2\mathbf{M} + 1\mathbf{S}$
$B = T\bar{T} - 2aX_1^2 \cdot (T - 2Z_1^2)$	$1\mathbf{M}$
$X_3 = X_1 \cdot A \cdot \bar{A}$	$2\mathbf{M}$
$Y_3 = -Y_1 \cdot B \cdot \bar{B}$	$2\mathbf{M}$
$Z_3 = Z_1 \cdot A \cdot B$	$2\mathbf{M}$
	$9\mathbf{M} + 1\mathbf{M}_a + 3\mathbf{S}$

$\bar{T}, \bar{A}$, and $\bar{B}$ are the conjugates of T, A, and B respectively.

3.4.4 A proposal for a $5P$ formula. It is possible to obtain $5P$ through $2(2P) + P$ or $3P + 2P$. In standard projective coordinates, $2(2P) + P$ costs $16\mathbf{M} + 1\mathbf{M}_d + 3\mathbf{M}_a + 9\mathbf{S} \approx 26.2\mathbf{M}$ and $3P + 2P$ costs $22\mathbf{M} + 1\mathbf{M}_d + 3\mathbf{M}_a + 8\mathbf{S} \approx 29.4\mathbf{M}$. We see that $2(2P) + P$ saves $3.2\mathbf{M}$ in comparison to $3P + 2P$. However, a $5P$ dedicated formula has a lower cost than $2(2P) + P$, as we will show. Bernstein et al. (2007) was the first to propose two different $5P$ formulas in standard projective coordinates. The first formula costs $17\mathbf{M} + 7\mathbf{S} \approx 22.6\mathbf{M}$, and the second formula costs $14\mathbf{M} + 11\mathbf{S} \approx 22.8\mathbf{M}$. Later, Subramanya Rao (2016) and Li et al. (2016) proposed $5P$ formulas in standard projective coordinates with lower costs. Rao's formula costs $15\mathbf{M} + 9\mathbf{S} \approx 22.2\mathbf{M}$ and Li et al.'s formula costs $12\mathbf{M} + 12\mathbf{S} \approx 21.6\mathbf{M}$.

We propose a $5P$ formula with cost $15\mathbf{M} + 1\mathbf{M}_a + 3\mathbf{S} \approx 17.4\mathbf{M}$, as Table 3.17 shows. This proposed $5P$ formula saves approximately $5.2\mathbf{M}$ and $5.4\mathbf{M}$ over Bernstein et al.'s formulas. It saves approximately $4.8\mathbf{M}$ and $4.2\mathbf{M}$ over Rao's formula and Li et al.'s formula respectively. Also, it saves approximately $2\mathbf{M}$ in comparison to the $5P$ formula in Jacobian Weierstrass ($a = -3$) coordinates, as Table 3.12 shows.

Table 3.17
$5P$ in Standard Twisted Edwards Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$T = Y_1^2 + a \cdot X_1^2$	$1\mathbf{M}_a + 2\mathbf{S}$
$A = T \cdot \bar{T} + 2Y_1^2 \cdot (T - 2Z_1^2)$	$2\mathbf{M} + 1\mathbf{S}$
$B = T\bar{T} - 2aX_1^2 \cdot (T - 2Z_1^2)$	$1\mathbf{M}$
$C = -T\bar{T} \cdot A \cdot \bar{A} + 2Y_1^2(T - 2Z_1^2) \cdot B \cdot \bar{B}$	$4\mathbf{M}$
$D = T\bar{T} \cdot B\bar{B} + 2aX_1^2(T - 2Z_1^2) \cdot A\bar{A}$	$2\mathbf{M}$
$X_5 = X_1 \cdot C \cdot \bar{C}$	$2\mathbf{M}$
$Y_5 = Y_1 \cdot D \cdot \bar{D}$	$2\mathbf{M}$
$Z_5 = Z_1 \cdot C \cdot D$	$2\mathbf{M}$
	$15\mathbf{M} + 1\mathbf{M}_a + 3\mathbf{S}$

$\bar{T}, \bar{A}, \bar{B}, \bar{C}$, and $\bar{D}$ are the conjugates of T, A, B, C, and D respectively.

Moreover, this proposed $5P$ formula requires at least two temporary variables. It is similar to the number of temporary variable requirement of the $3P$ formula. This implies that it is not necessary for a higher-base formula to require more temporary variables than a lower-base formula. See Appendix for the steps to perform this $5P$ formula with the fewest temporary variables. See Theorem 4 for our $5P$ formula.

Theorem 4. Let $P = (X_1, Y_1, Z_1) \in E_{a,d}(\mathbb{F}_p)$. Then $5P = (X_5, Y_5, Z_5)$ in standard twisted Edwards coordinates is represented by

$$\begin{aligned}
T &= Y_1^2 + aX_1^2 \\
A &= T\bar{T} + 2Y_1^2(T - 2Z_1^2) \\
B &= T\bar{T} - 2aX_1^2(T - 2Z_1^2) \\
C &= -T\bar{T}A\bar{A} + 2Y_1^2(T - 2Z_1^2)B\bar{B} \\
D &= T\bar{T}B\bar{B} + 2aX_1^2(T - 2Z_1^2)A\bar{A} \\
X_5 &= X_1C\bar{C} \\
Y_5 &= Y_1D\bar{D} \\
Z_5 &= Z_1CD
\end{aligned}$$

where $\bar{T}, \bar{A}, \bar{B}, \bar{C}$, and $\bar{D}$ are the conjugates of T, A, B, C, and D respectively.

Proof. We shall prove Theorem 4 by the fact $5P = 2P + 3P$. From the $P + Q$ formula shown in Table 3.13, we have

$$F = (Z_2 \cdot Z_3)^2 - dX_2X_3Y_2Y_3.$$

We substitute $X_2, Y_2, Z_2, X_3, Y_3, Z_3$ with their equivalent terms in Table 3.15 and Table 3.16. We have

$$F = \Big(T(T - 2Z_1^2)Z_1AB\Big)^2 - 2dX_1^2Y_1^2(T - 2Z_1^2)A\bar{A}T\bar{T}B\bar{B}.$$

From the curve equation $dX_1^2Y_1^2 = Z_1^2(T - Z_1^2)$, we have

$$\begin{aligned}
F &= \Big(T(T - 2Z_1^2)Z_1AB\Big)^2 - 2Z_1^2(T - Z_1^2)(T - 2Z_1^2)A\bar{A}T\bar{T}B\bar{B} \\
&= T(T - 2Z_1^2)Z_1^2AB\Big(T(T - 2Z_1^2)AB - 2(T - Z_1^2)\bar{A}\bar{T}\bar{B}\Big).
\end{aligned}$$

From the P+Q formula, we have

$$\begin{aligned}
X_5 &= Z_2Z_3F(X_2Y_3 + Y_2X_3). \\
Y_5 &= Z_2Z_3\bar{F}(Y_2Y_3 - aX_2X_3).
\end{aligned}$$

$$Z_5 = F\bar{F}.$$

We cancel Z_2Z_3 by the facts $x_5 = \frac{X_5}{Z_5}$ and $y_5 = \frac{Y_5}{Z_5}$. We have

$$\begin{aligned} F &= T(T - 2Z_1^2)AB - 2(T - Z_1^2)\bar{A}\bar{T}\bar{B}. \\ X_5 &= F(X_2Y_3 + Y_2X_3). \\ Y_5 &= \bar{F}(Y_2Y_3 - aX_3X_2). \\ Z_5 &= Z_1F\bar{F}. \end{aligned}$$

For X_5, we substitute X_2, X_3, Y_2, X_3 with their equivalent terms. We have

$$\begin{aligned} X_5 &= F\Big(- 2X_1Y_1(T - 2Z_1^2)Y_1B\bar{B} - T\bar{T}X_1A\bar{A}\Big) \\ &= X_1F\Big(- T\bar{T}A\bar{A} - 2Y_1^2(T - 2Z_1^2)B\bar{B}\Big). \end{aligned}$$

Let $\bar{C} = -T\bar{T}X_1A\bar{A} - 2X_1Y_1^2(T - 2Z_1^2)B\bar{B}$. Then $C = F$ where $\bar{C}$ is the conjugate of C. Thus, we have

$$X_5 = X_1C\bar{C}.$$

For Y_5, we also substitute X_2, X_3, Y_2, Y_3 with their equivalent terms. We have

$$\begin{aligned} Y_5 &= \bar{F}\Big(T\bar{T}Y_1B\bar{B} - aX_1A\bar{A}2X_1Y_1(T - 2Z_1^2)\Big) \\ &= Y_1\bar{F}\Big(T\bar{T}B\bar{B} - 2aX_1^2A\bar{A}(T - 2Z_1^2)\Big). \end{aligned}$$

Let $\bar{D} = T\bar{T}B\bar{B} - 2aX_1^2A\bar{A}(T - 2Z_1^2)$. Then $D = \bar{F}$ where $\bar{D}$ is the conjugate of D. Thus, we have

$$Y_5 = Y_1D\bar{D}.$$

For Z_5, we use the relations $F = C$ and $\bar{F} = D$. Thus, we have

$$Z_5 = Z_1CD.$$

□

3.4.5 A proposal for a $2Q + P$ formula. It is possible to obtain $2Q + P$ without a dedicated formula through $2P$ and $P + Q$. In standard projective coordinates, $2Q + P$ without a dedicated formula costs $12\mathbf{M} + 1\mathbf{M}_d + 2\mathbf{M}_a + 5\mathbf{S}$. However, we will show that a dedicated $2Q + P$ formula has a lower cost than a non-dedicated formula. Our proposed $2Q + P$ formula can be used with or without pre-computation.

Table 3.18
$2Q + P$ with Pre-computation in Standard Twisted Edwards Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$T = Y_2^2 + a \cdot X_2^2$	$1\mathbf{M}_a + 2\mathbf{S}$
$2X_2Y_2 = (X_2 + Y_2)^2 - X_2^2 - Y_2^2$	$1\mathbf{S}$
$F = \underline{Z_1^2} \cdot T \cdot (T - 2Z_2^2) + \underline{dX_1Y_1} \cdot 2X_2Y_2 \cdot \bar{T}$	$4\mathbf{M} + 1\mathbf{S}$
$G = \underline{2X_2Y_2} \cdot (T - 2Z_2^2)$	$1\mathbf{M}$
$X_3 = F \cdot \Big((G + \underline{X_1Z_1}) \cdot (\underline{Y_1Z_1} - T \cdot \bar{T}) + G \cdot T\bar{T} - \underline{X_1Z_1\ Y_1Z_1}\Big)$	$4\mathbf{M}$
$Y_3 = \bar{F} \cdot \Big((G + \underline{Y_1Z_1}) \cdot (\underline{-aX_1Z_1} - T\bar{T}) + GT\bar{T} + \underline{aX_1Z_1\ Y_1Z_1}\Big)$	$2\mathbf{M}$
$Z_3 = F \cdot \bar{F}$	$1\mathbf{M}$
	$12\mathbf{M} + 1\mathbf{M}_a + 4\mathbf{S}$

$\bar{T}$ and $\bar{F}$ are the conjugates of T and F respectively.
Underlined terms are pre-computed.

The cost of the proposed $2Q + P$ formula with pre-computation is $12\mathbf{M} + 1\mathbf{M}_a + 4\mathbf{S} \approx 15.2\mathbf{M}$, as Table 3.18 shows. The pre-computation is for the values: $Z_1^2, X_1 \cdot Y_1, d \cdot X_1Y_1, X_1 \cdot Z_1, Y_1 \cdot Z_1, X_1Z_1 \cdot Y_1Z_1, a \cdot X_1Z_1$, and $a \cdot X_1Z_1Y_1Z_1$. The pre-computation costs $4\mathbf{M} + 1\mathbf{M}_d + 2\mathbf{M}_a + 1\mathbf{S} \approx 5.8\mathbf{M}$. We see that the $2Q + P$ formula with pre-computation saves $1\mathbf{M} + 1\mathbf{M}_d + 1\mathbf{M}_a + 1\mathbf{S} \approx 2.8\mathbf{M}$ in comparison to a non-dedicated $2Q + P$ formula.

The cost of the proposed mixed $2Q + P$ formula with pre-computation is $11\mathbf{M} + 1\mathbf{M}_a + 4\mathbf{S} \approx 14.2\mathbf{M}$. It saves $1\mathbf{M} + 1\mathbf{M}_d + 1\mathbf{M}_a + 1\mathbf{S} \approx 2.8\mathbf{M}$ in comparison to a non-dedicated mixed $2Q + P$ formula. The cost of the proposed mixed $2Q + P$ formula without pre-computation is $12\mathbf{M} + 1\mathbf{M}_d + 3\mathbf{M}_a + 4\mathbf{S} \approx 16.2\mathbf{M}$. It trades $1\mathbf{S}$ with $2\mathbf{M}_a$ in comparison to a non-dedicated mixed $2Q + P$ formula. Also, it saves $3\mathbf{S}$ in comparison to the $2Q + P$ formula in Jacobian Weierstrass ($a = -3$) coordinates, as Table 3.12 shows (Longa & Miri, 2008b). We understand that it is the first proposed $2Q + P$ formula for twisted Edwards curves. See Theorem 5 for our proposed $2Q + P$ formula.

Theorem 5. Let $P = (X_1, Y_1, Z_1) \in E_{a,d}(\mathbb{F}_p)$ and $Q = (X_2, Y_2, Z_2) \in E_{a,d}(\mathbb{F}_p)$. Then $2Q + P = (X_3, Y_3, Z_3)$ in standard twisted Edwards coordinates is represented by

$$\begin{aligned}
T &= Y_2^2 + aX_2^2 \\
F &= Z_1^2 T(T - 2Z_2^2) + dX_1Y_1\ 2X_2Y_2\bar{T} \\
G &= 2X_2Y_2(T - 2Z_2^2) \\
X_3 &= F\Big(- T\bar{T}X_1Z_1 + Y_1Z_1G\Big) \\
Y_3 &= \bar{F}\Big(- T\bar{T}Y_1Z_1 - aX_1Z_1G\Big) \\
Z_3 &= F\bar{F}
\end{aligned}$$

where $\bar{T}$ and $\bar{F}$ are the conjugates of T and F respectively.

Proof. We shall prove Theorem 5 by the fact

$$(X_{2Q+P}, Y_{2Q+P}, Z_{2Q+P}) = (X_{2Q}, Y_{2Q}, Z_{2Q}) + (X_P, Y_P, Z_P).$$

From the $Q + P$ formula shown in Table 3.13, We have

$$F = (Z_PZ_{2Q})^2 - dX_PY_PX_{2Q}Y_{2Q}.$$

Let $T = Y_Q^2 + aX_Q^2$. We substitute X_{2Q}, Y_{2Q} with their equivalent terms in Table 3.15. We have

$$\begin{aligned}
F &= \Big(Z_PT(T - 2Z_Q^2)\Big)^2 + dX_PY_P2X_QY_Q(T - 2Z_Q^2)T\bar{T} \\
&= T(T - 2Z_Q^2)\Big(Z_P^2T(T - 2Z_Q^2) + dX_PY_P2X_QY_Q\bar{T}\Big).
\end{aligned}$$

From the $P + Q$ formula, we have

$$\begin{aligned}
X_{2Q+P} &= Z_PZ_{2Q}F(X_PY_{2Q} + Y_PX_{2Q}). \\
Y_{2Q+P} &= Z_PZ_{2Q}\bar{F}(Y_PY_{2Q} - aX_PX_{2Q}). \\
Z_{2Q+P} &= F\bar{F}.
\end{aligned}$$

We cancel Z_{2Q} by the facts $x_{2Q+P} = X_{2Q+P}/Z_{2Q+P}$ and $y_{2Q+P} = Y_{2Q+P}/Z_{2Q+P}$. We have

$$\begin{aligned} F &= Z_P^2 T(T - 2Z_Q^2) + dX_P Y_P 2X_Q Y_Q \bar{T}. \\ X_{2Q+P} &= Z_P F(X_P Y_{2Q} + Y_P X_{2Q}). \\ Y_{2Q+P} &= Z_P \bar{F}(Y_P Y_{2Q} - aX_P X_{2Q}). \\ Z_{2Q+P} &= F\bar{F}. \end{aligned}$$

Let $G = 2X_Q Y_Q(T - 2Z_Q^2)$. Then X_{2Q+P} can be obtained by substituting X_{2Q}, Y_{2Q} with their equivalent terms. We have

$$\begin{aligned} X_{2Q+P} &= Z_P F\Big(X_P(-T\bar{T}) + Y_P G\Big). \\ &= F\Big(-T\bar{T}X_P Z_P + Y_P Z_P G\Big). \\ &= F\Big((G + X_P Z_P)(Y_P Z_P - T\bar{T}) + GT\bar{T} - X_P Y_P Z_P^2\Big). \end{aligned}$$

For Y_{2Q+P}, we substitute X_{2Q}, Y_{2Q} with their equivalent terms. We have

$$\begin{aligned} Y_{2Q+P} &= Z_P \bar{F}\Big(Y_P(-T\bar{T}) - aX_P G\Big). \\ &= \bar{F}\Big(-T\bar{T}Y_P Z_P - aX_P Z_P G\Big). \\ &= F\Big((G + Y_P Z_P)(-aX_P Z_P - T\bar{T}) + GT\bar{T} + aX_P Y_P Z_P^2\Big). \end{aligned}$$

□

4 Elliptic Curves over Binary Fields

4.1 Binary Elliptic Curves

Non-supersingular elliptic curves $E_{a,b}$ over binary fields $\mathbb{F}_{2^m}$ are represented by the Weierstrass equation

$$E_{a,b} : y^2 + xy = x^3 + ax^2 + b$$

where $a, b \in \mathbb{F}_{2^m}$ and $b \neq 0$. The denotation $E_{a,b}(\mathbb{F}_{2^m})$ is the set of all points (x, y) where $x, y \in \mathbb{F}_{2^m}$ that satisfies the above equation together with the point at infinity $\mathcal{O}$. This set forms an abelian group under the point addition operation. The identity point for the group is the point at infinity $\mathcal{O}$. The negative of point $P = (x, y) \in E_{a,b}(\mathbb{F}_{2^m})$ is another point $-P = (x, x + y) \in E_{a,b}(\mathbb{F}_{2^m})$.

Let $P = (x_1, y_1) \in E_{a,b}(\mathbb{F}_{2^m})$ and $Q = (x_2, y_2) \in E_{a,b}(\mathbb{F}_{2^m})$ with $P \neq \pm Q$. Then $P + Q = (x_3, y_3) \in E_{a,b}(\mathbb{F}_{2^m})$ can be computed by the formula shown in Table 4.1. It costs $1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$. Let $P = (x_1, y_1) \in E_{a,b}(\mathbb{F}_{2^m})$ with $P \neq -P$. Then $2P = (x_2, y_2) \in E_{a,b}(\mathbb{F}_{2^m})$ can be computed by the formula shown in Table 4.2. It costs $1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$.

Table 4.1
$P + Q$ for Binary Elliptic Curves in Affine Coordinates

Formula terms	Operation counts
$\lambda = \frac{y_1+y_2}{x_1+x_2}$	$1\mathbf{I} + 1\mathbf{M}$
$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$	$1\mathbf{S}$
$y_3 = \lambda \cdot (x_1 + x_3) + x_3 + y_1$	$1\mathbf{M}$
	$1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$

Table 4.2
$2P$ for Binary Elliptic Curves in Affine Coordinates

Formula terms	Operation counts
$\lambda = x_1 + \frac{y_1}{x_1}$	$1\mathbf{I} + 1\mathbf{M}$
$x_2 = \lambda^2 + \lambda + a$	$1\mathbf{S}$
$y_2 = x_1^2 + \lambda \cdot x_2 + x_2$	$1\mathbf{M} + 1\mathbf{S}$
	$1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$

4.1.1 $3P$ formulas. Recall that $3P$ can be obtained without a dedicated formula through $2P + P$. The cost of $2P + P$ is $2\mathbf{I} + 4\mathbf{M} + 3\mathbf{S}$. However, studies show

that a dedicated $3P$ formula in affine coordinates has a faster performance. Therefore, efforts were made to derive a dedicated $3P$ formula for binary elliptic curves in affine coordinates. Ciet et al. (2006) proposed one of the earliest dedicated $3P$ formulas with cost $1\mathbf{I} + 7\mathbf{M} + 4\mathbf{S}$. Dimitrov et al. (2008) derived a dedicated $3P$ formula with cost $1\mathbf{I} + 6\mathbf{M} + 3\mathbf{S}$. Recently, Yu et al. (2015) derived the most efficient $3P$ formula for binary elliptic curves in affine coordinates. Their $3P$ formula costs $1\mathbf{I} + 5\mathbf{M} + 2\mathbf{S}$. We can see that a dedicated a $3P$ formula trades approximately $1\mathbf{I}$ with $1\mathbf{M}$ over $2P + P$. See Table 4.3 for the cost of the $3P$ formula.

Table 4.3
$3P$ for Binary Elliptic Curves in Affine Coordinates

Formula terms	Operation counts
$\beta = \frac{x_1^2}{x_1^2 \cdot (x_1^2 + x_1) + b}$	$1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$
$x_3 = (x_1 \cdot \beta)^2 + x_1\beta + x_1$	$1\mathbf{M} + 1\mathbf{S}$
$y_y = (1 + x_1\beta) \cdot \Big((x_1\beta)^2 + \beta \cdot (x_1^2 + y_1)\Big) + x_1 + y_1$	$2\mathbf{M}$
	$1\mathbf{I} + 5\mathbf{M} + 2\mathbf{S}$

4.1.2 $4P$ **formulas.** We saw that $4P$ can be obtained without a dedicated formula through $2(2P)$ or $3P + P$. The cost of $2(2P)$ for binary elliptic curves in affine coordinates is $2\mathbf{I} + 4\mathbf{M} + 4\mathbf{S}$ and the cost of $3P + P$ is $2\mathbf{I} + 7\mathbf{M} + 3\mathbf{S}$. We see that $2(2P)$ saves approximately $3\mathbf{M}$ over $3P + P$. However, the following studies show that a dedicated $4P$ formula has a lower cost.

Lopez and Dahab (1999) initially proposed a $4P$ formula for binary elliptic curves in affine coordinates with cost $1\mathbf{I} + 5\mathbf{M} + 1\mathbf{M}_b + 5\mathbf{S}$. Ciet et al. (2006) later proposed a $4P$ formula for binary elliptic curves in affine coordinates with cost $1\mathbf{I} + 8\mathbf{M} + 5\mathbf{S}$. Their formula costs extra $3\mathbf{M} + 1\mathbf{M}_b$ in comparison to Lopez and Dahab's $4P$ formula. See Table 4.4 for the cost of Lopez and Dahab's formula.

4.1.3 $5P$ **formulas.** We know that $5P$ can be obtained without a dedicated formula through $4P + P$, $3P + 2P$, or $2(2P) + P$. The cost of $4P + P$ for binary elliptic curves in affine coordinates is approximately $2\mathbf{I} + 8\mathbf{M} + 6\mathbf{S}$. The cost of $3P + 2P$ is $3\mathbf{I} + 9\mathbf{M} + 5\mathbf{S}$ and the cost of $2(2P) + P$ is $3\mathbf{I} + 6\mathbf{M} + 5\mathbf{S}$. We see that $4P + P$ saves approximately $1\mathbf{I} + 1\mathbf{M}$ in comparison to $3P + 2P$ and trades $1\mathbf{I}$ with $2\mathbf{M} + 1\mathbf{S}$ in

Table 4.4
4P for Binary Elliptic Curves in Affine Coordinates

Formula terms	Operation counts
$\beta = \frac{1}{x_1 \cdot (x_1^4 + b)}$	$1\mathbf{I} + 1\mathbf{M} + 2\mathbf{S}$
$\lambda = x_1 + \beta \cdot (x_1^4 + b) \cdot y_1$	$2\mathbf{M}$
$x_2 = \lambda^2 + \lambda + a$	$1\mathbf{S}$
$\lambda_2 = \lambda^2 + \beta \cdot x_1 \cdot b + a$	$1\mathbf{M} + 1\mathbf{M}_b$
$x_4 = \lambda_2^2 + \lambda_2 + a$	$1\mathbf{S}$
$y_4 = x_2^2 + \lambda_2 \cdot x_4 + x_4$	$1\mathbf{M} + 1\mathbf{S}$
	$1\mathbf{I} + 5\mathbf{M} + 1\mathbf{M}_b + 5\mathbf{S}$

comparison to $2(2P) + P$. However, studies show that a dedicated $5P$ formula has a lower cost than $4P + P$.

Mishra and Dimitrov (2007) initially derived a dedicated $5P$ formula for binary elliptic curves in affine coordinates with cost $1\mathbf{I} + 13\mathbf{M} + 5\mathbf{S}$. Recently, Al Musa and Xu (2017) derived the most efficient $5P$ formula for binary elliptic curves in affine coordinates. Their $5P$ formula costs $1\mathbf{I} + 11\mathbf{M} + 6\mathbf{S}$. It saves $2\mathbf{M} - 1\mathbf{S}$ in comparison to Mishra and Dimitrovs' $5P$ formula. It trades $1\mathbf{I}$ with $3\mathbf{M}$ in comparison to $4P + P$. See Theorem 6 for the formula and Table 4.5 for the cost of this formula.

Table 4.5
5P for Binary Elliptic Curves in Affine Coordinates

Formula terms	Operation counts
$\alpha = x_1^4 + x_1^3 + b$	$1\mathbf{M} + 2\mathbf{S}$
$\beta = \alpha^2 + x_1^2 \cdot (x_1^4 + b)$	$1\mathbf{M} + 1\mathbf{S}$
$\gamma = \alpha^2 \cdot (x_1^4 + b) + x_1^3 \cdot \beta$	$2\mathbf{M}$
$x_5 = x_1 + \frac{x_1^3 \cdot \beta}{\gamma} + \left(\frac{x_1^3 \beta}{\gamma}\right)^2$	$1\mathbf{I} + 1\mathbf{M} + 1\mathbf{S}$
$y_5 = y_1 + x_1 + (x_5 + x_1) \cdot \left(\frac{x_1^3 \beta}{\gamma} + x_1^2 + a\right) + \frac{x_1 \cdot \beta \cdot \alpha^2 \cdot (\beta + (x_1^4 + b) \cdot (x_1^4 + b + y_1^2 + x_1^2))}{\gamma^2}$	$6\mathbf{M} + 2\mathbf{S}$
	$1\mathbf{I} + 11\mathbf{M} + 6\mathbf{S}$

Theorem 6. Let $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$. Then $5P = (x_5, y_5)$ is represented by

$$
\begin{aligned}
\alpha &= x_1^4 + x_1^3 + b \\
\beta &= \alpha^2 + x_1^2(x_1^4 + b) \\
\gamma &= \alpha^2(x_1^4 + b) + x_1^3 \beta \\
x_5 &= x_1 + \frac{x_1^3 \beta}{\gamma} + \left(\frac{x_1^3 \beta}{\gamma}\right)^2 \\
y_5 &= y_1 + x_1 + (x_5 + x_1)\left(\frac{x_1^3 \beta}{\gamma} + x_1^2 + a\right) + \frac{x_1 \beta \alpha^2 (\beta + (x_1^4 + b)(x_1^4 + b + y_1^2 + x_1^2))}{\gamma^2}.
\end{aligned}
$$

Proof. We shall prove Theorem 6 by the fact

$$(x_5, \lambda_5) = (x_2, \lambda_2) + (x_3, \lambda_3).$$

By using the $P + Q$ λ-affine formula given in Oliveira et al. (2014), we have

$$x_5 = \frac{x_3 x_2}{(x_3 + x_2)^2}(\lambda_3 + \lambda_2). \tag{2}$$

$$\lambda_5 = \frac{x_3(x_5 + x_2)^2}{x_5 x_2} + \lambda_2 + 1. \tag{3}$$

We apply $x_3 = x_1 + \frac{x_1^3}{\alpha} + \left(\frac{x_1^3}{\alpha}\right)^2$ and $x_2 = \frac{x_1^4+b}{x_1^2}$ in equation (2). We have

$$x_5 = \frac{x_1^3(\alpha^2 + x_1^2(x_1^4 + b))\alpha^2(x_1^4 + b)}{\left(\alpha^2(x_1^4 + b) + x_1^3(\alpha^2 + x_1^2(x_1^4 + b))\right)^2}(\lambda_3 + \lambda_2) \tag{4}$$

$$= \frac{x_1^3 \beta \alpha^2(x_1^4 + b)}{\gamma^2}(\lambda_3 + \lambda_2). \tag{5}$$

We note that

$$\lambda_3 + \lambda_2 = \frac{x_1 \gamma^2}{x_1^3 \beta \alpha^2(x_1^4 + b)} + 1. \tag{6}$$

By applying equation (6) in equation (5), we have

$$x_5 = x_1 + \frac{x_1^3 \beta \alpha^2(x_1^4 + b)}{\gamma^2} \tag{7}$$

$$= x_1 + \frac{x_1^3 \beta}{\gamma} + \left(\frac{x_1^3 \beta}{\gamma}\right)^2. \tag{8}$$

We have derived x_5. Next, we want to derive y_5. From equation (3), we have

$$\lambda_5 = \frac{x_3}{x_2} x_5 + \frac{x_3 x_2}{x_5} + \lambda_2 + 1. \tag{9}$$

We apply equation (9) to the fact $y_5 = x_5(\lambda_5 + x_5)$. We have

$$y_5 = x_5\left(\frac{x_3}{x_2} x_5 + \lambda_2 + 1 + x_5\right) + x_3 x_2. \tag{10}$$

We apply x_3, x_2 , and $\lambda_2 = \frac{x_1^4}{x_1^4+b} + \lambda_1^2 + a + 1$ in equation (10). We have

$$\begin{aligned} y_5 &= x_5\Big(\frac{x_1^3\beta}{\alpha^2(x_1^4+b)}x_5 + \frac{x_1^4}{x_1^4+b} + \frac{y_1^2}{x_1^2} + x_1^2 + a + x_5\Big) + \frac{(x_1^4+b)\beta}{x_1\alpha^2} \\ &= x_5\Big(\Big(\frac{x_1^3\beta}{\gamma}\Big)^2 + x_1^2 + a + x_5\Big) + \frac{(x_1^4+b)\beta}{x_1\alpha^2} + \frac{x_1^4\beta}{\alpha^2(x_1^4+b)}x_5 + \frac{x_1^6 + y_1^2(x_1^4+b)}{x_1^2(x_1^4+b)}x_5 \\ &= y_1 + x_1 + (x_5+x_1)\Big(\Big(\frac{x_1^3\beta}{\gamma}\Big)^2 + x_1^2 + a + x_5 + x_1\Big) + \frac{x_1\beta\alpha^2(x_1^6 + y_1^2(x_1^4+b))}{\gamma^2}. \end{aligned}$$

We note that $x_1^6 = \beta + (x_1^4+b)^2 + x_1^2(x_1^4+b)$ and $\Big(\frac{x_1^3\beta}{\gamma}\Big)^2 = \frac{x_1^3\beta}{\gamma} + x_5 + x_1$. We have

$$y_5 = y_1 + x_1 + (x_5+x_1)\Big(\frac{x_1^3\beta}{\gamma} + x_1^2 + a\Big) + \frac{x_1\beta\alpha^2(\beta + (x_1^4+b)(x_1^4+b+y_1^2+x_1^2))}{\gamma^2}.$$

□

4.2 Lambda Coordinates

Oliveira et al. (2014) proposed lambda coordinates (λ-coordinates) for binary elliptic curves. The main advantage of λ-coordinates is that they have the most efficient $2P$ formula, as Table 4.6, Table 4.7, and Table 4.12 show, in projective coordinates over binary fields. λ-coordinates, unlike Lopez-Dahab (LD) and Jacobian coordinates, have their own affine and projective coordinates. Lambda affine coordinates are called λ-affine coordinates. λ-projective coordinates are derived from λ-affine coordinates and most of the time are called λ-coordinates.

Table 4.6
The Cost for Efficient Formulas in Different Projective Coordinates over $\mathbb{F}_{2^m}$

	λ-projective	LD projective	Jacobian projective
$2P$	$4\mathbf{M} + 1\mathbf{M}_a + 4\mathbf{S}$	$4\mathbf{M} + 1\mathbf{M}_a + 4\mathbf{S}$	$4\mathbf{M} + 1\mathbf{M}_b + 5\mathbf{S}$
$P+Q$	$11\mathbf{M} + 2\mathbf{S}$	$13\mathbf{M} + 4\mathbf{S}$	$15\mathbf{M} + 1\mathbf{M}_a + 3\mathbf{S}$
$3P$	$8\mathbf{M} + 1\mathbf{M}_a + 5\mathbf{S}$ (this work)	$10\mathbf{M} + 2\mathbf{M}_a + 7\mathbf{S}$	$13\mathbf{M} + 2\mathbf{M}_{a,b} + 7\mathbf{S}$
$5P$	$13\mathbf{M} + 1\mathbf{M}_a + 8\mathbf{S}$ (this work)	N/A	N/A

Affine point (x, y) can be converted to λ-affine point (x, λ) by using the relation $(x, \lambda) = (x, x + \frac{y}{x})$. The reverse from a λ-affine point to an affine point can be done by $(x, y) = (x, x(x + \lambda))$. In addition, a λ-affine point can be converted to λ-projective point (X, L, Z) by using the relation $(X, L, Z) = (x, \lambda, 1)$. The reverse from a

λ-projective point to a λ-affine point can be done by $(x, \lambda) = (\frac{X}{Z}, \frac{L}{Z})$ where $Z \neq 0$. Therefore, the Weierstrass equation in λ-projective coordinates can be represented by

$$(L^2 + LZ + aZ^2)X^2 = X^4 + bZ^4.$$

The conversion from affine coordinates to λ-affine coordinates costs $1\mathbf{I} + 1\mathbf{M}$ and the reverse costs $1\mathbf{M}$. The conversion from λ-affine coordinates to λ-projective coordinates has no cost, and the reverse costs $1\mathbf{I} + 2\mathbf{M}$. Therefore, the total cost of conversion and reverse from affine coordinates to λ-projective coordinates is $2\mathbf{I} + 4\mathbf{M}$. It has higher cost than other coordinates. However, the cost of conversion and reverse is performed only one time and it has no impact on scalar multiplication operations.

Table 4.7
The Cost of Efficient Formulas in Affine and λ-coordinates for Binary Elliptic Curves

	Affine Coordinates	λ-projective Coordinates
$2P$	$1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$	$4\mathbf{M} + 1\mathbf{M}_a + 4\mathbf{S}$
Mixed $P + Q$	$1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$	$8\mathbf{M} + 2\mathbf{S}$
$3P$	$1\mathbf{I} + 5\mathbf{M} + 2\mathbf{S}$	$8\mathbf{M} + 1\mathbf{M}_a + 5\mathbf{S}$ (this work)
$5P$	$1\mathbf{I} + 11\mathbf{M} + 6\mathbf{S}$ (this work)	$13\mathbf{M} + 1\mathbf{M}_a + 8\mathbf{S}$ (this work)

4.2.1 $P + Q$ **formulas.** The cost of $P + Q$ formula in λ-projective coordinates, as Table 4.8 shows, is $11\mathbf{M} + 2\mathbf{S}$. It saves $2\mathbf{M} + 2\mathbf{S}$ in comparison to $P + Q$ in LD projective coordinates (Lange, 2004). It saves $4\mathbf{M} + 1\mathbf{M}_a + 1\mathbf{S}$ in comparison to $P + Q$ in Jacobian projective coordinates over binary fields (Avanzi et al., 2005). The mixed $P + Q$ formula can be deduced from the $P + Q$ formula when $Z_1 = 1$. As a result, the mixed $P + Q$ formula in λ-coordinates costs $8\mathbf{M} + 2\mathbf{S}$. It saves $1\mathbf{M}_a + 3\mathbf{S}$ in comparison to the mixed $P + Q$ formula in LD coordinates (Lange, 2004). It trades $1\mathbf{I}$ with $6\mathbf{M} + 1\mathbf{S}$ in comparison to $P + Q$ in affine coordinates, as Table 4.7 shows.

4.2.2 $2P$ **formulas.** The cost of $2P$ formula in λ-projective coordinates, as Table 4.8 shows, is $4\mathbf{M} + 1\mathbf{M}_a + 4\mathbf{S}$. It has the same cost as $2P$ in LD projective coordinates (Lange, 2004). It saves $1\mathbf{S}$ in comparison to $2P$ in Jacobian projective coordinates over binary fields (Avanzi et al., 2005). It trades $1\mathbf{I}$ with $2\mathbf{M} + 1\mathbf{M}_a + 2\mathbf{S}$ in comparison to $2P$ in affine coordinates, as Table 4.7 shows. When the curve

Table 4.8
$P + Q$ in λ-coordinates over $\mathbb{F}_{2^m}$

Formula terms	Operation counts
$A = L_1 \cdot Z_2 + L_2 \cdot Z_1$	$2\mathbf{M}$
$B = (X_1 \cdot Z_2 + X_2 \cdot Z_1)^2$	$2\mathbf{M} + 1\mathbf{S}$
$X_3 = A \cdot X_1 Z_2 \cdot X_2 Z_1 \cdot A$	$3\mathbf{M}$
$L_3 = (AX_2Z_1 + B)^2 + A \cdot B \cdot Z_2 \cdot (L_1 + Z_1)$	$3\mathbf{M} + 1\mathbf{S}$
$Z_3 = ABZ_2 \cdot Z_1$	$1\mathbf{M}$
	$11\mathbf{M} + 2\mathbf{S}$

coefficients are selected as small constants, the $2P$ formula in λ-projective coordinates can be reduced to $3\mathbf{M} + 1\mathbf{M}_{a^2+b} + 2\mathbf{M}_a + 5\mathbf{S}$. This is because L_2 can be represented by

$$L_2 = (L_1 + X_1)^2 \cdot \left((L_1 + X_1)^2 + T + Z_1^2\right) + (a^2 + b) \cdot Z_1^4 + T^2 + (a + 1) \cdot Z_2.$$

Table 4.9
$2P$ in λ-coordinates over $\mathbb{F}_{2^m}$

Formula terms	Operation counts
$T = L_1^2 + L_1 \cdot Z_1 + a \cdot Z_1^2$	$1\mathbf{M} + 1\mathbf{M}_a + 2\mathbf{S}$
$X_2 = T^2$	$1\mathbf{S}$
$Z_2 = T \cdot Z_1^2$	$1\mathbf{M}$
$L_2 = (T + X_1 \cdot Z_1)^2 + T \cdot (L_1 Z_1 + Z_1^2)$	$2\mathbf{M} + 1\mathbf{S}$
	$4\mathbf{M} + 1\mathbf{M}_a + 4\mathbf{S}$

4.2.3 $3P$ **formulas.** We saw that $3P$ can be obtained without a dedicated formula by $2P + P$. The cost of $2P + P$ in λ-coordinates is $15\mathbf{M} + 1\mathbf{M}_a + 6\mathbf{S}$. However, we can achieve more savings by using a dedicated $3P$ formula. Dimitrov et al. (2008) initially derived a $3P$ formula in Jacobian coordinates for binary elliptic curves. Their formula costs $13\mathbf{M} + 2\mathbf{M}_{a,b} + 7\mathbf{S}$. Yasin and Muda (2015) also derived a dedicated $3P$ formula in LD projective coordinates with cost $10\mathbf{M} + 2\mathbf{M}_a + 7\mathbf{S}$. We see that Yasin and Muda's $3P$ formula saves approximately $3\mathbf{M}$ over Dimitrov et al.'s $3P$ formula.

Recently, Al Musa and Xu (2017) derived the most efficient $3P$ formula in λ-projective coordinates with cost $8\mathbf{M} + 1\mathbf{M}_a + 5\mathbf{S}$. Their $3P$ formula saves $7\mathbf{M} + 1\mathbf{S}$ in comparison to $2P + P$. It saves $2\mathbf{M} + 1\mathbf{M}_a + 2\mathbf{S}$ in comparison to Yasin and Muda's $3P$ formula. It trades $1\mathbf{I}$ with $3\mathbf{M} + 1\mathbf{M}_a + 3\mathbf{S}$ in comparison to $3P$ in affine coordinates, as

Table 4.7 shows. Moreover, it requires at least three temporary variables for implementation. See Appendix for the steps to perform the $3P$ formula with the fewest temporary variables. See Theorem 7 for the $3P$ formula and Table 4.10 for the cost of this formula.

Table 4.10
$3P$ in λ-coordinates over $\mathbb{F}_{2^m}$

Formula terms	Operation counts
$T = L_1^2 + L_1 \cdot Z_1 + a \cdot Z_1^2$	$1\mathbf{M} + 1\mathbf{M}_a + 2\mathbf{S}$
$A = (T + X_1 \cdot Z_1)^2$	$1\mathbf{M} + 1\mathbf{S}$
$B = T \cdot Z_1^2 + A$	$1\mathbf{M}$
$X_3 = X_1 Z_1 \cdot B^2$	$1\mathbf{M} + 1\mathbf{S}$
$Z_3 = Z_1^2 \cdot A \cdot B$	$2\mathbf{M}$
$L_3 = T \cdot (A + B)^2 + (L_1 Z_1 + Z_1^2) \cdot AB$	$2\mathbf{M} + 1\mathbf{S}$
	$8\mathbf{M} + 1\mathbf{M}_a + 5\mathbf{S}$

Theorem 7. Let $P = (X_1, L_1, Z_1) \in E(\mathbb{F}_{2^m})$. Then $3P = (X_3, L_3, Z_3)$ in λ-projective coordinates is represented by

$$
\begin{aligned}
T &= L_1^2 + L_1 Z_1 + a Z_1^2 \\
A &= (T + X_1 Z_1)^2 \\
B &= T Z_1^2 + A \\
X_3 &= X_1 Z_1 B^2 \\
Z_3 &= Z_1^2 AB \\
L_3 &= T(A + B)^2 + (L_1 Z_1 + Z_1^2) AB.
\end{aligned}
$$

Proof. We shall prove Theorem 7 by the fact

$$(x_3, \lambda_3) = (x_1, \lambda_1) + (x_2, \lambda_2). \tag{11}$$

By using the $P + Q$ λ-affine formula given in Oliveira et al. (2014), we have

$$x_3 = \frac{x_1 x_2}{(x_1 + x_2)^2}(\lambda_1 + \lambda_2). \tag{12}$$

$$\lambda_3 = \frac{x_2(x_3 + x_1)^2}{x_3 x_1} + \lambda_1 + 1. \tag{13}$$

We apply the relation $\lambda_1 + \lambda_2 = \frac{(x_1+x_2)^2}{x_2} + 1$ in equation (12). We have

$$x_3 = x_1 + \frac{x_1 x_2}{(x_1 + x_2)^2} \tag{14}$$

$$= \frac{x_1\Big(x_2 + (x_2 + x_1)^2\Big)}{(x_1 + x_2)^2}. \tag{15}$$

We convert λ-affine point (x_1, λ_1) to λ-projective point (X_1, L_1, Z_1) by using the relation $(x_1, \lambda_1) = (\frac{X_1}{Z_1}, \frac{L_1}{Z_1})$. Thus, the equations above become

$$\begin{aligned}
x_2 &= \frac{L_1^2 + L_1 Z_1 + aZ^2}{Z_1^2} = \frac{T}{Z_1^2}. \\
x_3 &= \frac{\frac{X_1}{Z_1}\Big(\frac{T}{Z_1^2} + \frac{(T+X_1Z_1)^2}{Z_1^4}\Big)}{\frac{(T+X_1Z_1)^2}{Z_1^4}} \\
&= \frac{X_1\Big(TZ_1^2 + (T + X_1Z_1)^2\Big)}{Z_1(T + X_1Z_1)^2} = \frac{X_1 B}{Z_1 A}. \\
\lambda_3 &= \frac{\frac{T}{Z_1^2}\Big(\frac{X_1B}{Z_1A} + \frac{X_1}{Z_1}\Big)^2}{\frac{X_1^2B}{Z_1^2A}} + \frac{L_1Z_1 + Z_1^2}{Z_1^2} \\
&= \frac{T(A+B)^2}{Z_1^2AB} + \frac{L_1Z_1 + Z_1^2}{Z_1^2} \\
&= \frac{T(A+B)^2 + (L_1Z_1 + Z_1^2)AB}{Z_1^2AB}.
\end{aligned}$$

□

4.2.4 $5P$ **formulas.** We recall that $5P$ can be obtained in projective coordinates without a dedicated formula through $2(2P) + P$ or $3P + 2P$. The cost of $2(2P) + P$ in λ-coordinates is $19\mathbf{M} + 2\mathbf{M}_a + 10\mathbf{S}$ and the cost of $3P + 2P$ is $23\mathbf{M} + 2\mathbf{M}_a + 11\mathbf{S}$. We see that $2(2P) + P$ is more efficient than $3P + 2P$ because $2(2P) + P$ saves $4\mathbf{M} + 1\mathbf{S}$ over $3P + 2P$.

Recently, Al Musa and Xu (2017) derived a dedicated $5P$ formula with cost $13\mathbf{M} + 1\mathbf{M}_a + 8\mathbf{S}$. Their $5P$ formula saves $6\mathbf{M} + 1\mathbf{M}_a + 2\mathbf{S}$ in comparison to $2(2P) + P$. It trades $1\mathbf{I}$ with $2\mathbf{M} + 1\mathbf{M}_a + 2\mathbf{S}$ in comparison to $5P$ in affine coordinates, as Table 4.7 shows. It seems that it is the only $5P$ formula for projective coordinates over binary fields. Moreover, their $5P$ formula requires at least five temporary variables for

implementation. See Appendix for the steps to perform the $5P$ formula with the fewest temporary variables. See Theorem 8 for the $5P$ formula and see Table 4.11 for the cost of this formula.

Table 4.11
$5P$ in λ-coordinates over $\mathbb{F}_{2^m}$

Formula terms	Operation counts
$T = L_1^2 + L_1 \cdot Z_P + a \cdot Z_1^2$	$1\mathbf{M} + 1\mathbf{M}_a + 2\mathbf{S}$
$A = (T + X_1 \cdot Z_1)^2$	$1\mathbf{M} + 1\mathbf{S}$
$B = T \cdot Z_1^2 + A$	$1\mathbf{M}$
$C = (T \cdot (A + B))^2 + A \cdot B^2$	$2\mathbf{M} + 2\mathbf{S}$
$D = AB^2 + A^2 \cdot B + C$	$1\mathbf{M} + 1\mathbf{S}$
$X_5 = X_1 Z_1 \cdot D^2$	$1\mathbf{M} + 1\mathbf{S}$
$Z_5 = Z_1^2 \cdot C \cdot D$	$2\mathbf{M}$
$L_5 = T \cdot (C + D)^2 + (L_1 Z_1 + Z_1^2) \cdot CD + Z_1^2 \cdot AB^2 \cdot A^2 B$	$4\mathbf{M} + 1\mathbf{S}$
	$13\mathbf{M} + 1\mathbf{M}_a + 8\mathbf{S}$

Theorem 8. Let $P = (X_1, L_1, Z_1) \in E(\mathbb{F}_{2^m})$. Then $5P = (X_5, L_5, Z_5)$ in λ-projective coordinates is represented by

$$
\begin{aligned}
T &= L_1^2 + L_1 Z_1 + a Z_1^2 \\
A &= (T + X_1 Z_1)^2 \\
B &= T Z_1^2 + A \\
C &= (T(A + B))^2 + AB^2 \\
D &= A^2 B + AB^2 + C \\
X_5 &= X_1 Z_1 D^2 \\
Z_5 &= Z_1^2 CD \\
L_5 &= T(C + D)^2 + (L_1 Z_1 + Z_1^2) CD + Z_1^2 (AB)^3.
\end{aligned}
$$

Proof. We shall prove the first part of Theorem 8 by the fact

$$(x_5, \lambda_5) = (x_2, \lambda_2) + (x_3, \lambda_3).$$

From the $P + Q$ λ-affine formula given in Oliveira et al. (2014), we have

$$x_5 = \frac{x_2 x_3}{(x_2 + x_3)^2}(\lambda_2 + \lambda_3). \tag{16}$$

We apply the relation $\lambda_2 + \lambda_3 = \frac{x_1(x_2+x_3)^2}{x_2x_3} + 1$ to equation (16). We have

$$\begin{aligned} x_5 &= x_1 + \frac{x_2x_3}{(x_2+x_3)^2} && (17)\\ &= \frac{x_1(x_2+x_3)^2 + x_2x_3}{(x_2+x_3)^2}. && (18) \end{aligned}$$

Next, we shall prove the second part of Theorem 8 by the fact

$$(x_5, \lambda_5) = (x_1, \lambda_1) + (x_4, \lambda_4).$$

From the $P + Q$ λ-affine formula, we have

$$\lambda_5 = \frac{x_4(x_5+x_1)^2}{x_5x_1} + \lambda_1 + 1. \quad (19)$$

We convert λ-affine point (x_1, λ_1) to λ-projective point (X_1, L_1, Z_1) by using the relation $(x_1, \lambda_1) = (\frac{X_1}{Z_1}, \frac{L_1}{Z_1})$. Thus, the equations above become

$$\begin{aligned} x_2 &= \frac{L_1^2 + L_1Z_1 + aZ^2}{Z_1^2} = \frac{T}{Z_1^2}.\\ x_3 &= \frac{X_1\Big(TZ_1^2 + (T + X_1Z_1)^2\Big)}{Z_1(T+X_1Z_1)^2} = \frac{X_1B}{Z_1A}.\\ x_4 &= \frac{L_2^2 + L_2TZ_1^2 + a(TZ_1^2)^2}{(TZ_1^2)^2} = \frac{T_2}{(TZ_1^2)^2}.\\ x_5 &= \frac{\frac{X_1}{Z_1}\Big(\frac{T}{Z_1^2} + \frac{X_1B}{Z_1A}\Big)^2 + \frac{TX_1B}{Z_1^3A}}{(\frac{T}{Z_1^2} + \frac{X_1B}{Z_1A})^2}\\ &= \frac{X_1\Big((TA + X_1Z_1B)^2 + TZ_1^2AB\Big)}{Z_1(TA+X_1Z_1B)^2}\\ &= \frac{X_1D}{Z_1C}.\\ \lambda_5 &= \frac{\frac{T_2}{(TZ_1^2)^2}\Big(\frac{X_1D}{Z_1C} + \frac{X_1}{Z_1}\Big)^2}{\frac{X_1^2D}{Z_1^2C}} + \frac{L_1Z_1 + Z_1^2}{Z_1^2}\\ &= \frac{T_2(C+D)^2}{(TZ_1^2)^2CD} + \frac{L_1Z_1 + Z_1^2}{Z_1^2}\\ &= \frac{Z_1^2T_2(AB)^2 + (L_1Z_1 + Z_1^2)CD}{Z_1^2CD}. \end{aligned}$$

We note the following relations

$$\begin{aligned} Z_1^2 T_2 &= T(A+B)^2 + Z_1^2 AB. \\ C &= (TA + X_1 Z_1 B)^2 = (T(A+B))^2 + AB^2. \\ D &= TZ_1^2 AB + C = A^2 B + AB^2 + C. \end{aligned}$$

Thus, we have

$$L_5 = T(C+D)^2 + (L_1 Z_1 + Z_1^2)CD + Z_1^2 (AB)^3.$$

□

4.3 Twisted μ_4-normal Coordinates

Kohel (2017) proposed twisted μ_4-normal coordinates for binary elliptic curves. The main advantage of twisted μ_4-normal coordinates is that they have the most efficient $P+Q$ formula over binary fields when the curve coefficient $a = 0$. A point in twisted μ_4-normal coordinates is represented as (X, Y, Z, T). An affine point (x, y) of binary elliptic curves can be mapped to a corresponding point in twisted μ_4-normal coordinates by using the relation $(X, Y, Z, T) = (x^2, x^2 + y, 1, x^2 + y + x)$. A point in twisted μ_4-normal coordinates can be reversed to a corresponding affine point by using the relation $(x, y) = (\frac{Y+T}{Z}, \frac{Y+X}{Z})$ where $Z \neq 0$. The negative of point $P = (X, Y, Z, T)$ in twisted μ_4-normal coordinates is another point $-P = (X, T, Z, Y)$. The curve equations in twisted μ_4-normal coordinates can be represented by

$$\begin{aligned} YT &= X^2 + bZ^2 + a(Y+T)^2, \\ XZ &= (Y+T)^2. \end{aligned}$$

4.3.1 $P+Q$ **formulas.** The cost of $P+Q$ in twisted μ_4-normal coordinates is $9\mathbf{M} + 1\mathbf{M}_a + 2\mathbf{S}$, as Table 4.13 shows. It saves approximately $2\mathbf{M}$ in comparison to $P+Q$ in λ-coordinates. Mixed $P+Q$ can be deduced from $P+Q$ by assuming $Z_1 = 1$. As a result, the cost of mixed $P+Q$ in twisted μ_4-normal coordinates is $8\mathbf{M} + 1\mathbf{M}_a + 2\mathbf{S}$. We see that mixed $P+Q$ in twisted μ_4-normal coordinates has

Table 4.12
The Cost of Efficient Formulas in Twisted μ_4-normal Coordinates and λ-coordinates over $\mathbb{F}_{2^m}$

	twisted μ_4-normal coordinates	λ-coordinates
Mixed $P+Q$	$8\mathbf{M} + 1\mathbf{M}_a + 2\mathbf{S}$	$8\mathbf{M} + 2\mathbf{S}$
$P+Q$	$9\mathbf{M} + 1\mathbf{M}_a + 2\mathbf{S}$	$11\mathbf{M} + 2\mathbf{S}$
$2P$	$2\mathbf{M} + 1\mathbf{M}_{a^2+b} + 1\mathbf{M}_b + 1\mathbf{M}_a + 5\mathbf{S}$ (this work)	$3\mathbf{M} + 1\mathbf{M}_{a^2+b} + 2\mathbf{M}_a + 5\mathbf{S}$ or $4\mathbf{M} + 1\mathbf{M}_a + 4\mathbf{S}$

approximately the same cost as mixed $P+Q$ in λ-coordinates since in practice the curve coefficient a is selected as a small constant. Moreover, the cost of $P+Q$ in twisted μ_4-normal coordinates can be significantly reduced when the curves coefficient $a=0$. This is because the cost of $P+Q$ becomes $7\mathbf{M}+2\mathbf{S}$ and the cost of mixed $P+Q$ becomes $6\mathbf{M}+2\mathbf{S}$.

Table 4.13
$P+Q$ in Twisted μ_4-normal Coordinates over $\mathbb{F}_{2^m}$

Formula terms	Operation counts	$a=0$
$A = X_1 \cdot Z_2 + X_2 \cdot Z_1$	$2\mathbf{M}$	$2\mathbf{M}$
$B = Y_1 \cdot T_2 + Y_2 \cdot T_1$	$2\mathbf{M}$	$2\mathbf{M}$
$X_3 = B^2$	$1\mathbf{S}$	$1\mathbf{S}$
$Z_3 = A^2$	$1\mathbf{S}$	$1\mathbf{S}$
$F = (Y_1 + T_1) \cdot (Y_2 + T_2) \cdot A$	$2\mathbf{M}$	
$T_3 = X_1Z_2 \cdot Y_1T_2 + X_2Z_1 \cdot Y_2T_1 + a \cdot F$	$2\mathbf{M} + 1\mathbf{M}_a$	$2\mathbf{M}$
$Y_3 = T_3 + A \cdot B$	$1\mathbf{M}$	$1\mathbf{M}$
	$9\mathbf{M} + 1\mathbf{M}_a + 2\mathbf{S}$	$7\mathbf{M} + 2\mathbf{S}$

4.3.2 A proposal for a $2P$ formula. Kohel (2017) proposed a $2P$ formula in twisted μ_4-normal coordinates with the assumption that the curve coefficient $b=1$. We propose a $2P$ formula in twisted μ_4-normal coordinates for any value of b. Also, we propose an improvement to this newly proposed $2P$ formula by precomputing the value of $\sqrt{b}$. This extra step saves $1\mathbf{S}$ in comparison to $2P$ without precomputing $\sqrt{b}$. Therefore, the cost of this proposed $2P$ formula is $2\mathbf{M} + 1\mathbf{M}_{a^2+b} + 1\mathbf{M}_b + 1\mathbf{M}_a + 5\mathbf{S}$, as Table 4.14 shows. We see that this proposed $2P$ formula saves approximately $1\mathbf{M}$ in comparison to $2P$ in λ-coordinates when the curve coefficients are small constants. However, when the curve coefficients are large constants, it costs extra $1\mathbf{S}$ in comparison to $2P$ in λ-coordinates.

Table 4.14
2P in Twisted μ_4-normal Coordinates over $\mathbb{F}_{2^m}$

Formula terms	Operation counts
$A = (X_1 + \sqrt{b} \cdot Z_1)^2$	$1\mathbf{M}_b + 1\mathbf{S}$
$B = (Y_1 + T_1)^2$	$1\mathbf{S}$
$X_2 = A^2$	$1\mathbf{S}$
$Z_2 = B^2$	$1\mathbf{S}$
$T_2 = X_2 + (a^2 + b) \cdot Z_2 + \left((X_1 + Y_1) \cdot (T_1 + \sqrt{b}\ Z_1)\right)^2 + a \cdot A \cdot B$	$2\mathbf{M} + 1\mathbf{M}_{a^2+b} + 1\mathbf{M}_a + 1\mathbf{S}$
$Y_2 = T_2 + AB$	
	$2\mathbf{M} + 1\mathbf{M}_{a^2+b} + 1\mathbf{M}_b + 1\mathbf{M}_a + 5\mathbf{S}$

4.4 Koblitz Curves

Koblitz (1987) introduced an efficiently computable endomorphism with a special class of elliptic curves over binary fields. Koblitz Curves E_a can be represented by

$$E_a : y^2 + xy = x^3 + ax^2 + 1$$

where $a \in \{0, 1\}$. The denotation $E_a(\mathbb{F}_{2^m})$ is the set of all points (x, y) that satisfies the equation above, together with the point at infinity $\mathcal{O}$. The main advantage of Koblitz curves is that their properties allow scalar multiplication methods to use the *Frobenius* map instead of point doubling. The *Frobenius* map $\tau : E_a(\mathbb{F}_{2^m}) \to E_a(\mathbb{F}_{2^m})$ is defined by

$$\tau(x, y) = (x^2, y^2),$$
$$\tau(\mathcal{O}) = \mathcal{O}.$$

One important property of E_a is that $\tau^2(P) + 2P = \mu\tau(P)$ for all $P \in E_a(\mathbb{F}_{2^m})$ where $\mu = (-1)^{1-a}$. This means τ can be considered to be a complex number that satisfies $\tau^2 + 2 = \mu\tau$. By solving $\tau^2 - \mu\tau + 2 = 0$, we have $\tau = \frac{\mu + \sqrt{-7}}{2}$.

Let $\mathbb{Z}[\tau]$ be a ring of polynomials in τ with integer coefficients. New scalar multiplication methods that take advantage of *Frobenius* map τ were proposed by the following studies. Koblitz (1992) initially proposed a method that converts a positive integer t to a unique unsigned base-τ form. The base-τ form can be represented by $t = \sum_{i=0}^{l-1} u_i \tau^i$ where $u_i \in \{0, 1\}$ and $u_{l-1} \neq 0$.

Later, Solinas (2000) showed an improved method that converts a positive integer t to a unique signed digits representation called τ-NAF. When we represent t in τ-NAF, the average form length becomes less than its base-τ form. As a result, the number of point additions are minimized, and that significantly improves the performance of a scalar multiplication operation. Moreover, Avanzi, Dimitrov, Doche, and Sica (2006) proposed a double-base number system (DBNS) method that take advantage of the *Frobenius* map τ for Koblitz Curves. A positive integer t is represented by τ-DBNS in the form of $t = \sum_{i=0}^{l-1} \pm \tau\ \zeta$ where ζ can be $\bar{\tau}$ or 3. The $\bar{\tau}$ is called the conjugate of τ and it can be defined by

$$\bar{\tau}P = \mu P - \tau P.$$

Table 4.15
The Cost of Efficient Formulas in Different Coordinates for Koblitz Curves

	twisted μ_4-normal	λ-projective	LD projective
τP	4**S**	3**S**	3**S**
$\bar{\tau}P(a=0)$	2**M** + 2**S** (this work)	5**M** + 3**S**	2**M** + 2**S**
$\bar{\tau}P(a=1)$	6**M** + 5**S** (this work)	5**M** + 3**S**	2**M** + 1**S**
mixed $P+Q(a=0)$	6**M** + 2**S**	8**M** + 2**S**	8**M** + 5**S**
mixed $P+Q(a=1)$	8**M** + 2**S**	8**M** + 2**S**	8**M** + 5**S**

4.4.1 The window τ-NAF. Solinas (2000) suggested using the reduced window τ-NAF method with Koblitz curves for speeding up scalar multiplication. The window τ-NAF method initially needs to pre-compute $Q_j = c_j P$ for each $j = 1, 3, \cdots, 2^{w-1} - 1$ where w is the window width. Then, the performing phase of the method utilizes the pre-computed Q_j for faster performances. See Hankerson et al. (2004) for the converting and the performing phases of the window τ-NAF method. More details of the window τ-NAF can be found in Blake, Murty, and Xu (2005) and Blake, Murty, and Xu (2008).

Trost and Xu (2016) suggested an optimal arrangement scheme for the pre-computed points of window τ-NAF, as Table 4.16 shows. The optimal pre-computation of window τ-NAF costs 1**ADD** + 2**S** at most for each pre-computed point. Moreover, Trost and Xu (2016) implied that a $\bar{\tau}P$ formula can be used with the

optimal pre-computation of window τ-NAF. Recently, Yu, Al Musa, Xu, and Li (2018) introduced a novel arrangement scheme for the window τ-NAF. This pre-computation scheme takes advantage of the efficiency of $\bar{\tau}P$, as Table 4.17 shows.

Table 4.16
The Optimal Pre-computation of Window τ-NAF when $a = 0$

w	$c_j \equiv j \pmod{\tau^w}$		pre-computed points	
4	$c_3 = -3 - \tau$	$c_5 = -1 - \tau$	$Q_3 = -P + \tau^2 P$	$Q_5 = -P - \tau P$
	$c_7 = 1 - \tau$		$Q_7 = P - \tau P$	
5	$c_3 = -3 - \tau$	$c_5 = -1 - \tau$	$Q_3 = -P + \tau^2 P$	$Q_5 = -P - \tau P$
	$c_7 = 1 - \tau$	$c_9 = -3 - 2\tau$	$Q_7 = P - \tau P$	$Q_9 = Q_3 - \tau P$
	$c_{11} = -1 - 2\tau$	$c_{13} = 1 - 2\tau$	$Q_{11} = Q_5 - \tau P$	$Q_{13} = Q_7 - \tau P$
	$c_{15} = 1 + 3\tau$		$Q_{15} = -Q_{11} + \tau P$	
6	$c_{29} = 3 + \tau$	$c_3 = 3$	$Q_{29} = P - \tau^2 P$	$Q_3 = Q_{29} - \tau P$
	$c_{31} = 5 + \tau$	$c_5 = 5$	$Q_{31} = Q_3 - \tau^2 P$	$Q_5 = Q_{31} - \tau P$
	$c_7 = -5 - 2\tau$	$c_9 = -3 - 2\tau$	$Q_7 = -Q_{31} - \tau P$	$Q_9 = -Q_{29} - \tau P$
	$c_{27} = 1 + \tau$	$c_{11} = -1 - 2\tau$	$Q_{27} = P + \tau P$	$Q_{11} = -Q_{27} - \tau P$
	$c_{25} = -1 + \tau$	$c_{13} = 1 - 2\tau$	$Q_{25} = -P + \tau P$	$Q_{13} = -Q_{25} - \tau P$
	$c_{15} = 1 + 3\tau$	$c_{17} = 3 + 3\tau$	$Q_{15} = -Q_{11} + \tau P$	$Q_{17} = -Q_9 + \tau P$
	$c_{19} = 5 + 3\tau$	$c_{21} = -3 - 4\tau$	$Q_{19} = -Q_7 + \tau P$	$Q_{21} = -Q_{17} - \tau P$
	$c_{23} = -3 + \tau$		$Q_{23} = -Q_3 + \tau P$	

When $a = 1$, Q_j and c_j can be obtained by changing only the sign of τ.

Table 4.17
The Novel Pre-computation of Window τ-NAF when $a = 0$

w	$c_j \equiv j \pmod{\tau^w}$		pre-computed points	
4	$c_5 = -1 - \tau$	$c_7 = 1 - \tau$	$Q_5 = \bar{\tau}P$	$Q_7 = -\bar{\tau}Q_5$
	$c_3 = -3 - \tau$		$Q_3 = \bar{\tau}Q_7$	
5	$c_5 = -1 - \tau$	$c_7 = 1 - \tau$	$Q_5 = \bar{\tau}P$	$Q_7 = -\bar{\tau}Q_5$
	$c_3 = -3 - \tau$	$c_{15} = 1 + 3\tau$	$Q_3 = \bar{\tau}Q_7$	$Q_{15} = \bar{\tau}Q_3$
	$c_{11} = -1 - 2\tau$	$c_9 = 3 - \tau$	$Q_{11} = Q_5 - \tau P$	$Q_9 = -\bar{\tau}Q_{11}$
	$c_{13} = -5 - 3\tau$		$Q_{13} = \bar{\tau}Q_9$	
6	$c_{27} = 1 + \tau$	$c_{25} = -1 + \tau$	$Q_{27} = -\bar{\tau}P$	$Q_{25} = -\bar{\tau}Q_{27}$
	$c_{29} = 3 + \tau$	$c_{15} = 1 + 3\tau$	$Q_{29} = \bar{\tau}Q_{25}$	$Q_{15} = -\bar{\tau}Q_{29}$
	$c_{21} = -5 + \tau$	$c_9 = -3 - 2\tau$	$Q_{21} = -\bar{\tau}Q_{15}$	$Q_9 = -Q_{29} - \tau P$
	$c_3 = 3$	$c_{13} = -1 + 3\tau$	$Q_3 = Q_{29} - \tau P$	$Q_{13} = \bar{\tau}Q_9$
	$c_{31} = -7 - \tau$	$c_{17} = 3 + 3\tau$	$Q_{31} = -\bar{\tau}Q_{13}$	$Q_{17} = -\bar{\tau}Q_3$
	$c_{11} = -3 + 3\tau$	$c_{23} = -1 - 4\tau$	$Q_{11} = -\bar{\tau}Q_{17}$	$Q_{23} = -Q_{15} - \tau P$
	$c_5 = -7 - 2\tau$	$c_7 = 7$	$Q_5 = Q_{31} - \tau P$	$Q_7 = -Q_{31} - \tau P$
	$c_{19} = -7 + \tau$		$Q_{19} = \bar{\tau}Q_{23}$	

When $a = 1$, Q_j and c_j can be obtained by changing the sign of τ and $\bar{\tau}$.

Table 4.18
$\bar{\tau}P$ in λ-Coordinates when $a = 1$

Formula terms	Operation counts
$A = X_1 \cdot (X_1 + Z_1)^2$	$1\mathbf{M} + 1\mathbf{S}$
$X_3 = (X_1 + Z_1)^4$	$1\mathbf{S}$
$L_3 = L_1 \cdot A + X_1^3 \cdot Z_1$	$3\mathbf{M} + 1\mathbf{S}$
$Z_3 = Z_1 \cdot A$	$1\mathbf{M}$
	$5\mathbf{M} + 3\mathbf{S}$

When $a = 0$, $\bar{\tau}P$ can be obtained by taking the negative of this formula.

4.4.2 A proposal for a $\bar{\tau}P$ formula when $a = 0$. Let the curve coefficient $a = 0$. Then $\bar{\tau}P$ can be obtained without a dedicated formula in twisted μ_4-normal coordinates with cost $7\mathbf{M} + 6\mathbf{S}$. However, a $\bar{\tau}P$ dedicated formula has a lower cost than a non-dedicated formula as the following studies show. Doche et al. (2009) proposed a dedicated $\bar{\tau}P$ formula in LD coordinates. When $a = 0$, their formula costs $2\mathbf{M} + 2\mathbf{S}$.

Recently, Trost and Xu (2016) proposed a dedicated $P - \mu\tau P$ formula in λ-coordinates. When $a = 0$, we can use the negative of the $P - \mu\tau P$ formula to obtain $\bar{\tau}P$, as Table 4.18 shows. As a result, their formula costs $5\mathbf{M} + 3\mathbf{S}$. Yu et al. (2018) proposed a $\bar{\tau}P$ formula only for the case $a = 0$. Their formula costs $3\mathbf{M} + 2\mathbf{S}$. We propose a dedicated $\bar{\tau}P$ formula for Koblitz curves when $a = 0$. The cost of this newly proposed formula in twisted μ_4-normal coordinates is $2\mathbf{M} + 2\mathbf{S}$, as Table 4.19 shows.

Table 4.19
$\bar{\tau}P$ in Twisted μ_4-normal Coordinates when $a = 0$

Formula terms	Operation counts
$X_3 = (X_1 + Z_1)^2$	$1\mathbf{S}$
$Z_3 = (Y_1 + T_1)^2$	$1\mathbf{S}$
$T_3 = X_3 + Z_3 + (X_1 + Y_1) \cdot (Z_1 + T_1)$	$1\mathbf{M}$
$Y_3 = T_3 + (X_1 + Z_1) \cdot (Y_1 + T_1)$	$1\mathbf{M}$
	$2\mathbf{M} + 2\mathbf{S}$

This proposed formula saves $5\mathbf{M} + 4\mathbf{S}$ in comparison to a non-dedicated formula. It saves $3\mathbf{M} + 1\mathbf{S}$ in comparison to $\bar{\tau}P$ in λ-coordinates. It saves $1\mathbf{M}$ in comparison to Yu et al.'s formula. It has the same cost as $\bar{\tau}P$ in LD coordinates. However, this proposed $\bar{\tau}P$ formula may be preferred in comparison to $\bar{\tau}P$ in LD coordinates because the cost of $P + Q$ in twisted μ_4-normal coordinates is less than the cost of $P + Q$ in LD

coordinates, as Table 4.15 shows. See Theorem 9 for our proposed $\bar{\tau}P$ formula.

Theorem 9. Let $P = (X_1, Y_1, Z_1, T_1) \in E_a(\mathbb{F}_{2^m})$ where $a = 0$. Then $\bar{\tau}P = (X_3, Y_3, Z_3, T_3)$ in twisted μ_4-normal coordinates is represented by

$$\begin{aligned} X_3 &= (X_1 + Z_1)^2 \\ Z_3 &= (Y_1 + T_1)^2 \\ T_3 &= X_3 + Z_3 + (X_1 + Y_1)(Z_1 + T_1) \\ Y_3 &= T_3 + (X_1 + Z_1)(Y_1 + T_1). \end{aligned}$$

Proof. Let $a = 0$. We shall prove Theorem 9 by the negative of the fact

$$(X_{P+\tau P}, Y_{P+\tau P}, Z_{P+\tau P}, T_{P+\tau P}) = (X_{\tau P}, Y_{\tau P}, Z_{\tau P}, T_{\tau P}) + (X_P, Y_P, Z_P, T_P).$$

From the $Q + P$ formula shown in Table 4.13, We have

$$\begin{aligned} X_{P+\tau P} &= (Y_{\tau P}T_P + Y_P T_{\tau P})^2 \\ Z_{P+\tau P} &= (X_{\tau P}Z_P + X_P Z_{\tau P})^2 \\ T_{P+\tau P} &= X_{\tau P}Z_P \; Y_{\tau P}T_P + X_P Z_{\tau P} \; Y_P T_{\tau P} \\ Y_{P+\tau P} &= T_{P+\tau P} + (Y_{\tau P}T_P + Y_P T_{\tau P})(X_{\tau P}Z_P + X_P Z_{\tau P}). \end{aligned}$$

We substitute $X_{\tau P}, Y_{\tau P}, Z_{\tau P}, T_{\tau P}$ with their τ evaluations. We have

$$\begin{aligned} X_{P+\tau P} &= (Y_P^2 T_P + Y_P T_P^2)^2 \\ &= (Y_P T_P)^2 (Y_P + T_P)^2 = (Y_P T_P)^2 X_P Z_P \\ Z_{P+\tau P} &= (X_P^2 Z_P + X_P Z_P^2)^2 \\ &= (X_P Z_P)^2 (X_P + Z_P)^2 = (X_P Z_P)^2 Y_P T_P \\ T_{P+\tau P} &= X_P^2 Z_P \; Y_P^2 T_P + X_P Z_P^2 \; Y_P T_P^2 \\ &= X_P Z_P \; Y_P T_P (X_P Y_P + T_P Z_P) \\ Y_{P+\tau P} &= T_{P+\tau P} + Y_P T_P (Y_P + T_P) \; X_P Z_P (X_P + Z_P). \end{aligned}$$

We cancel $X_P Z_P \; Y_P T_P$ by the facts $x_{P+\tau P} = \frac{Y_{P+\tau P} + T_{P+\tau P}}{Z_{P+\tau P}}$ and $y_{P+\tau P} = \frac{Y_{P+\tau P} + X_{P+\tau P}}{Z_{P+\tau P}}$.

We have

$$\begin{aligned} X_{P+\tau P} &= Y_P T_P = (X_P + Z_P)^2 \\ Z_{P+\tau P} &= X_P Z_P = (Y_P + T_P)^2 \\ T_{P+\tau P} &= X_P Y_P + T_P Z_P = X_{P+\tau P} + Z_{P+\tau P} + (X_P + T_P)(Y_P + Z_P) \\ Y_{P+\tau P} &= T_{P+\tau P} + (Y_P + T_P)(X_P + Z_P). \end{aligned}$$

We take the negative of $(X_{P+\tau P}, Y_{P+\tau P}, Z_{P+\tau P}, T_{P+\tau P})$. We have

$$\begin{aligned} X_{\bar{\tau} P} &= Y_P T_P = (X_P + Z_P)^2 \\ Z_{\bar{\tau} P} &= X_P Z_P = (Y_P + T_P)^2 \\ T_{\bar{\tau} P} &= X_{\bar{\tau} P} + Z_{\bar{\tau} P} + (X_P + Y_P)(Z_P + T_P) \\ Y_{\bar{\tau} P} &= T_{\bar{\tau} P} + (Y_P + T_P)(X_P + Z_P). \end{aligned}$$

□

4.4.3 A proposal for a $\bar{\tau}P$ formula when $a = 1$. Let the curve coefficient $a = 1$. The cost of $\bar{\tau}P$ without a dedicated formula in twisted μ_4-normal coordinates is $9\mathbf{M} + 6\mathbf{S}$. The cost of a dedicated $\bar{\tau}P$ formula in LD coordinates is $2\mathbf{M} + 1\mathbf{S}$ (Doche et al., 2009). The cost of a dedicated $\bar{\tau}P$ formula in λ-coordinates is $5\mathbf{M} + 3\mathbf{S}$ (Trost & Xu, 2016). We propose a dedicated $\bar{\tau}P$ formula for Koblitz curves when $a = 1$. The cost of this newly proposed formula in twisted μ_4-normal coordinates is $6\mathbf{M} + 5\mathbf{S}$, as Table 4.20 shows.

We see that this proposed formula saves $3\mathbf{M} + 1\mathbf{S}$ in comparison to a non-dedicated formula. It costs extra $1\mathbf{M} + 2\mathbf{S}$ in comparison to $\bar{\tau}P$ in λ-coordinates and extra $4\mathbf{M} + 4\mathbf{S}$ in comparison to $\bar{\tau}P$ in LD coordinates. As Table 4.15 shows, we conclude that when $a = 1$, twisted μ_4-normal coordinates is not the best choice for the window τ-NAF method. However, it is still one of the best choices for the window τ-NAF method when $a = 0$. See Theorem 10 for our proposed $\bar{\tau}P$ formula.

Table 4.20
$\bar{\tau}P$ in Twisted μ_4-normal Coordinates when $a = 1$

Formula terms	Operation counts
$A = (Y_1 + T_1) \cdot (X_1 + Z_1)$	1**M**
$X_3 = (X_1 + Z_1)^4$	2**S**
$Z_3 = A^2$	1**S**
$F = (Y_1^2 + T_1^2) \cdot A$	1**M** + 2**S**
$T_3 = X_3 + Z_3 + (T_1^2 + Y_1 \cdot Z_1) \cdot (Y_1^2 + X_1 \cdot T_1) + F$	3**M**
$Y_3 = T_3 + (X_1 + Z_1)^2 \cdot A$	1**M**
	6**M** + 5**S**

Theorem 10. Let $P = (X_1, Y_1, Z_1, T_1) \in E_a(\mathbb{F}_{2^m})$ where $a = 1$. Then $\bar{\tau}P = (X_3, Y_3, Z_3, T_3)$ in twisted μ_4-normal coordinates is represented by

$$
\begin{aligned}
&A = (Y_1 + T_1)(X_1 + Z_1)\\
&X_3 = (X_1 + Z_1)^4\\
&Z_3 = A^2\\
&F = (Y_1^2 + T_1^2)A\\
&T_3 = X_3 + Z_3 + (T_1^2 + Y_1Z_1)(Y_1^2 + X_1T_1) + F\\
&Y_3 = T_3 + (X_1 + Z_1)^2A.
\end{aligned}
$$

Proof. Let $a = 1$. We shall prove Theorem 10 by the fact

$$
\begin{aligned}
(X_{P-\tau P}, Y_{P-\tau P}, Z_{P-\tau P}, T_{P-\tau P}) &= -(X_{\tau P}, Y_{\tau P}, Z_{\tau P}, T_{\tau P}) + (X_P, Y_P, Z_P, T_P)\\
&= (X_{\tau P}, T_{\tau P}, Z_{\tau P}, Y_{\tau P}) + (X_P, Y_P, Z_P, T_P).
\end{aligned}
$$

From the $Q + P$ formula shown in Table 4.13, we have

$$
\begin{aligned}
X_{P-\tau P} &= (T_{\tau P}T_P + Y_PY_{\tau P})^2\\
Z_{P-\tau P} &= (X_{\tau P}Z_P + X_PZ_{\tau P})^2\\
F &= (Y_{\tau P} + T_{\tau P})(Y_P + T_P)(X_{\tau P}Z_P + X_PZ_{\tau P})\\
T_{P-\tau P} &= X_{\tau P}Z_P\ T_{\tau P}T_P + X_PZ_{\tau P}\ Y_PY_{\tau P} + F\\
Y_{P-\tau P} &= T_{P-\tau P} + (T_{\tau P}T_P + Y_PY_{\tau P})(X_{\tau P}Z_P + X_PZ_{\tau P}).
\end{aligned}
$$

We substitute $X_{\tau P}, Y_{\tau P}, Z_{\tau P}, T_{\tau P}$ with their τ evaluations. We have

$$\begin{aligned}
X_{P-\tau P} &= (T_P^2 T_P + Y_P Y_P^2)^2 = (Y_P^3 + T_P^3)^2 \\
&= (Y_P + T_P)^2 (X_P + Z_P)^4 = X_P Z_P (X_P + Z_P)^4 \\
Z_{P-\tau P} &= (X_P^2 Z_P + X_P Z_P^2)^2 = (X_P Z_P)^2 (X_P + Z_P)^2 \\
F &= (Y_P^2 + T_P^2)(Y_P + T_P) X_P Z_P (X_P + Z_P) \\
T_{P-\tau P} &= X_P^2 Z_P \; T_P^2 T_P + X_P Z_P^2 \; Y_P Y_P^2 + F \\
&= X_P Z_P (X_P T_P^3 + Z_P Y_P^3) + F \\
Y_{P-\tau P} &= T_{P-\tau P} + (Y_P + T_P)(X_P + Z_P)^2 \; X_P Z_P (X_P + Z_P).
\end{aligned}$$

We cancel $X_P Z_P$ by the facts $x_{P-\tau P} = \frac{Y_{P-\tau P} + T_{P-\tau P}}{Z_{P-\tau P}}$ and $y_{P-\tau P} = \frac{Y_{P-\tau P} + X_{P-\tau P}}{Z_{P-\tau P}}$. We have

$$\begin{aligned}
X_{P-\tau P} &= (X_P + Z_P)^4 \\
Z_{P-\tau P} &= (X_P Z_P)(X_P + Z_P)^2 \\
&= (Y_P + T_P)^2 (X_P + Z_P)^2 \\
F &= (Y_P^2 + T_P^2)(Y_P + T_P)(X_P + Z_P) \\
T_{P-\tau P} &= (X_P T_P^3 + Z_P Y_P^3) + F \\
Y_{P-\tau P} &= T_{P-\tau P} + (Y_P + T_P)(X_P + Z_P)(X_P + Z_P)^2.
\end{aligned}$$

Let $A = (Y_P + T_P)(X_P + Z_P)$. We have

$$\begin{aligned}
X_{\bar{\tau} P} &= (X_P + Z_P)^4 \\
Z_{\bar{\tau} P} &= A^2 \\
F &= (Y_P^2 + T_P^2) A \\
T_{\bar{\tau} P} &= X_{\bar{\tau} P} + Z_{\bar{\tau} P} + (T_P^2 + Y_P Z_P)(Y_P^2 + X_P T_P) + F \\
Y_{\bar{\tau} P} &= T_{\bar{\tau} P} + (X_P + Z_P)^2 A.
\end{aligned}$$

□

5 MBNS Methods without Pre-computation

5.1 Binary Method

The binary method is one of the primary methods that converts integer t to an unsigned single-base chain with $\{2\}$-integers. Even though the binary method cannot convert t to a multi-base chain, it is important to understand the steps of this method, because the MBNS methods are extended steps of the binary method. We can describe the binary method in the following three steps. Step one: the reduction step is that t is repeatedly divided by 2. We can perform this step by utilizing 2-adic valuation of t. Step two: the problem step is that t cannot be divided by 2. In other words, t is coprime to 2. Step three: the solution step is $t - 1$. This method repeats the steps until $t = 1$.

Algorithm 5.1 shows the three explained steps of the binary method. It shows the first phase of the method, which converts integer t to an unsigned single-base chain. The second phase of the method is to perform a scalar multiplication on a given chain, as Algorithm 5.2 shows. The average chain length of the binary method is approximately $\mathcal{O}\left(\frac{\log_2 n}{2}\right)$. The average chain cost of the binary method is approximately $\log_2 n \left(\frac{1}{2}\mathbf{ADD} + \mathbf{DBL}\right)$.

Algorithm 5.1 Binary Method

Input: positive integer t
Output: $(s_1, a_1), ..., (s_l, a_l)$ where $s_i \in \{0, 1\}, a_i \geq 0$, l is chain length
$t \leftarrow t/2^a$ where a is the 2-adic valuation of t
$l \leftarrow 1$
$(s_l, a_l) \leftarrow (0, a)$
while $(t > 1)$ **do**
  $l \leftarrow l + 1$
  $s \leftarrow 1$
  $t \leftarrow (t - s)/2^a$ where a is the 2-adic valuation of $(t - s)$
  $(s_l, a_l) \leftarrow (s, a)$
return $(s_1, a_1), ..., (s_l, a_l)$

5.2 NAF Method

The NAF method converts integer t to a signed single-base chain with $\{2\}$-integers. We can describe this method in the following four steps: the reduction,

Algorithm 5.2 Performing Scalar Multiplication on Binary Chains

Input: $(s_1, a_1), ..., (s_l, a_l)$ and $P \in E(\mathbb{F}_q)$
Output: tP
$Q \leftarrow P$
for $i \leftarrow l$ **to** 1 **do**
  for $j \leftarrow 1$ **to** a_i **do** $Q \leftarrow 2Q$
  if $(s_i = 1)$ **then** $Q \leftarrow Q + P$
  else if $(s_i = -1)$ **then** $Q \leftarrow Q - P$
return Q

problem, solution, and selection steps. The first two steps are similar to the binary method. However, it is different in the solution step. It selects either $t - 1$ or $t + 1$. We see that $t \mod 4$ decides the selection of either $t - 1$ or $t + 1$ because the result of $t \mod 4$ is either 1 or -1.

Algorithm 5.3 shows the four explained steps of the NAF method. It shows the first phase of the NAF method. The second phase of the method is similar to the binary method, as Algorithm 5.2 shows. The average chain length of the NAF method is approximately $\mathcal{O}\left(\frac{\log_2 n}{3}\right)$. The average chain cost of the NAF method is approximately $\log_2 n \left(\frac{1}{3}\mathbf{ADD} + \mathbf{DBL}\right)$. We see that the NAF method has a shorter average chain length than the binary method because the NAF method utilizes the sign technique in the selection step. It is important to note that the sign technique is used in the MBNS methods also.

Algorithm 5.3 NAF Method

Input: positive integer t
Output: $(s_1, a_1), ..., (s_l, a_l)$ where $s_i \in \{-1, 0, +1\}, a_i \geq 0$, l is chain length
$t \leftarrow t/2^a$ where a is the 2-adic valuation of t
$l \leftarrow 1$
$(s_l, a_l) \leftarrow (0, a)$
while $(t > 1)$ **do**
  $l \leftarrow l + 1$
  if $(t \pmod 4 \equiv 1)$ **then** $s \leftarrow 1$ **else** $s \leftarrow -1$
  $t \leftarrow (t - s)/2^a$ where a is the 2-adic valuation of $(t - s)$
  $(s_l, a_l) \leftarrow (s, a)$
return $(s_1, a_1), ..., (s_l, a_l)$

5.3 Greedy Method

Dimitrov et al. (1998) initially proposed the greedy method to generate a double-base number system. Later, Dimitrov et al. (2005) modified the greedy method to convert integer t to a double-base chain. The modification required adding the upper bound (a_{max}, b_{max}) to find the best approximation for t. The modified version is called the greedy method with restricted exponents which this dissertation focuses on. The steps of the greedy method with restricted exponents are shown in Algorithm 5.4.

Algorithm 5.4 Greedy Method

Input: positive integer t and the upper bound (a_{max}, b_{max})
Output: $(s_0, a_0, b_0), ..., (s_l, a_l, b_l)$ where $s_i \in \{-1, 0, +1\}$, $a_i, b_i \geq 0$, l is chain length
$l \leftarrow 0,\ s \leftarrow 1$
while $(t \geq 1)$ **do**
  find the best approximation of t in the form of $z = 2^a 3^b$ where $0 \leq a \leq a_{max}$ and $0 \leq b \leq b_{max}$
  $(s_l, a_l, b_l) \leftarrow (s, a, b)$
  $(a_{max}, b_{max}) \leftarrow (a, b)$
  if $(t < z)$ **then** $s \leftarrow -s$
  $t \leftarrow |t - z|,\ l \leftarrow l + 1$
$(s_l, a_l, b_l) \leftarrow (0, 0, 0)$
return $(s_0, a_0, b_0), ..., (s_l, a_l, b_l)$

The greedy method has two factors that impact the quality of a chain and the time to find the chain. The first factor is to determine the upper bound (a_{max}, b_{max}). This upper bound varies in each coordinate system. This is because each coordinate system has a different cost for **DBL**, **TPL**, and **ADD**. Therefore, it is recommended to test different values of (a_{max}, b_{max}) in particular coordinate system such that

$$a_{max} + b_{max} \log_2 3 = \log_2 t.$$

Then, the value of (a_{max}, b_{max}) that results in the lowest chain cost is selected. For example, Table 5.1 shows that the value of $(a_{max} = 160, b_{max} = 78)$ is selected for 283-bit integers in λ-coordinates over binary fields. The value of $(a_{max} = 140, b_{max} = 73)$ is selected for 254-bit integers in standard twisted Edwards coordinates over prime fields.

The second factor that impacts the greedy method is finding the best approximation of t in the form of $z = 2^a 3^b$. This dissertation explains two approaches to

Table 5.1
Greedy Method with Different Upper Bounds (a_{max}, b_{max}) *in Different Coordinates*

λ-coordinates				standard twisted Edwards coordinates			
283-bit				254-bit			
a_{max}	b_{max}	l	m	a_{max}	b_{max}	l	m
140	91	63.73	2229.81	120	85	58.78	2322.45
150	84	62.47	2211.32	130	79	56.52	2288.31
160	78	62.67	2206.11	140	73	55.49	2272.66
170	72	64.41	2214.64	150	66	57.34	2276.93
180	65	67.07	2230.56	160	60	59.64	2292.25
200	52	72.50	2263.33	180	47	65.03	2330.07
220	40	77.57	2294.31	200	34	70.41	2368.2
240	27	82.91	2327.21	220	22	75.46	2403.47

l: The average chain length.
m: The average chain cost.

find the best approximation of t: the look-up table and the line equation.

5.3.1 Look-up table. Doche and Imbert (2006) proposed a look-up table to find the best approximation of t. The table contains the binary expansion for elements $3^0, \ldots, 3^{b_{max}}$. These elements are sorted in lexicographic order. Then, a binary search is performed to find the element 3^b that best matches t. The difference between the binary expansion lengths for t and 3^b is used to find element 2^a. For example, assume we have a table that contains the binary expansion for the element $3^0, \ldots, 3^{10}$. We want to find the best approximation for $t = 935811$ in the form of $2^a 3^b$.

Table 5.2
An Example of the Difference between the Binary Expansion Lengths for t *and* 3^b

Decimal	Binary
3^{10}	11100<u>11</u>010101001
935811	11100<u>11</u>00011110000<u>011</u>

A binary search in the table returns element 3^{10} because element 3^{10} is the closest match to t. The difference of the binary expansion lengths of 3^{10} and 935811 is 4. This implies that element $2^a = 2^4$, as shown in Table 5.2. Thus, the best approximation for 935811 is $3^{10}2^4$.

5.3.2 Line equation. The line equation can be used to find the best approximation for integer t in the form of $z = 2^a 3^b$ (Berthe & Imbert, 2004). The line equation can be obtained from the form $z = 2^a 3^b$ by multiplying both sides by $\log_3$. We

have $\log_3 z = a \log_3 2 + b$. This implies

$$b = -a \log_3 2 + \log_3 z.$$

The main advantage of the line equation solution is that, unlike the look-up table, it does not require space memory consumption to find the best approximation of t. Yu, Wang, Li, and Tian (2013) proposed the line search method that scans the values (a, b) that are near the line equation. Algorithm 5.5 shows the steps of scanning the values that are near the line equation. Then, the algorithm returns the value of (a, b) that has the smallest difference from z.

Algorithm 5.5 Line Search

Input: positive integer t and the upper bound (a_{max}, b_{max})
Output:(z, a, b) such that $t \approx (z = 2^a 3^b)$
$n \leftarrow 2^{a_{max}}$
$(z, i, j) \leftarrow (t, a_{max}, 0)$
while $(i \geq 0)$ **do**
  if $(z > |t - n|)$ **then** $(z, a, b) \leftarrow (|t - n|, i, j)$
  if $(t > n)$ **then**
    $n \leftarrow 3 \cdot n,\ j \leftarrow j + 1$
    if $(j > b_{max})$ **then return** (z, a, b)
  else $n \leftarrow n/2,\ i \leftarrow i - 1$
return (z, a, b)

For example, assume the upper bound $(a_{max} = 10, b_{max} = 10)$ is given. We want to find the best approximation for $z = 935811$ in the form of $2^a 3^b$ using line search algorithm. We have the line equation is $b = -a \log_3 2 + \log_3 93581$. It can be drawn, as Figure 5.1 shows. The algorithm scans values that are near the line and selects the value of $(a = 4, b = 10)$ because it has the smallest difference from z.

The average expansion length of the greedy method with $\{2, 3\}$-integers is approximately $\mathcal{O}\left(\frac{\log t}{\log \log t}\right)$ (Dimitrov et al., 2008). However, the average chain length of the greedy method with restricted exponents is still unknown (Doche & Habsieger, 2008). Mishra and Dimitrov (2007) extended the greedy method to convert integer t to a multi-base chain with $\{2, 3, 5\}$-integers. The extension can be achieved by finding the best approximation of t in the form of $z = 2^a 3^b 5^c$. They also extended the line search

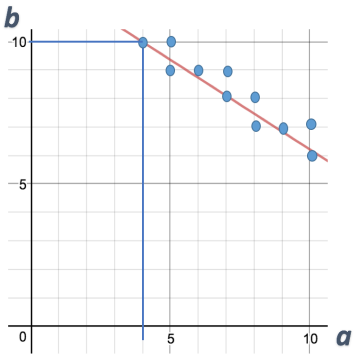


Figure 5.1. An Example of Line Search: $b = -a \log_3 2 + \log_3 935811$

algorithm to find the best approximation of t in the form of $z = 2^a 3^b 5^c$. Algorithm 5.6 shows how to perform scalar multiplication for the greedy chain.

Algorithm 5.6 Performing Scalar Multiplication on Greedy Chains

Input: $(s_0, a_0, b_0), ..., (s_l, a_l, b_l)$ and $P \in E(\mathbb{F}_q)$
Output: tP
$Q \leftarrow P$
for $i \leftarrow 0$ **to** $l - 1$ **do**
  $u \leftarrow a_i - a_{i+1}$, $v \leftarrow b_i - b_{i+1}$
  for $j \leftarrow 1$ **to** u **do** $Q \leftarrow 2Q$
  for $j \leftarrow 1$ **to** v **do** $Q \leftarrow 3Q$
  if $(s_{i+1} = 1)$ **then** $Q \leftarrow Q + P$
  else if $(s_{i+1} = -1)$ **then** $Q \leftarrow Q - P$
return Q

5.4 Ternary/binary Method

Ciet et al. (2006) proposed the ternary/binary method that can be used for a double-base number system. The ternary/binary method, as Algorithm 5.7 shows, converts integer t to a double-base chain with $\{2, 3\}$-integers. It can be described in the following four step process. Step one: the reduction step is that t is repeatedly divided by 2 or 3 and is performed by utilizing 2-adic and 3-adic valuations of t. Step two: the problem step is that t cannot be divided by 2 or 3. In other words, t is coprime to 6. Step three: the solution step is to select either $t + 1$ or $t - 1$. Step four: the selection step is directed by $t \mod 6$ and determines the selection of either $t + 1$ or $t - 1$. This is because the result of $t \mod 6$ is either 1 or 5. This method repeats the steps until $t = 1$.

Algorithm 5.7 Ternary/binary Method

Input: positive integer t
Output: $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$ where $s_i \in \{-1, 0, +1\}, a_i, b_i \geq 0$, l is chain length
$t \leftarrow t/(2^a 3^b)$ where a, b are the 2-adic and 3-adic valuations of t
$l \leftarrow 1$
$(s_l, a_l, b_l) \leftarrow (0, a, b)$
while $(t > 1)$ **do**
  $l \leftarrow l + 1$
  if $(t \pmod 6 \equiv 1)$ **then** $s \leftarrow 1$ **else** $s \leftarrow -1$
  $t \leftarrow (t - s)/(2^a 3^b)$ where a, b are the 2-adic and 3-adic valuations of $(t - s)$
  $(s_l, a_l, b_l) \leftarrow (s, a, b)$
return $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$

For example, we want to find a chain for $t = 4627$ using the ternary/binary method. Then, the output of Algorithm 5.7 is the following:

$(t, (s_1, a_1, b_1))$	$(t, (s_2, a_2, b_2))$	$(t, (s_3, a_3, b_3))$	$(t, (s_4, a_4, b_4))$	$(t, (s_5, a_5, b_5))$
$(4627, (0, 0, 0))$	$(257, (1, 1, 2))$	$(43, (-1, 1, 1))$	$(7, (1, 1, 1))$	$(1, (1, 1, 1))$

To demonstrate the steps, we add integer t to the output. Getting the chain can be accomplished by Algorithm 5.8 in the following Horner's rule manner:

$(1, (1, 1, 1)) \implies \text{chain} = 2\ 3 + 1$

$(7, (1, 1, 1)) \implies \text{chain} = (2\ 3 + 1)2\ 3 + 1$

$(43, (-1, 1, 1)) \implies \text{chain} = ((2\ 3 + 1)2\ 3 + 1)2\ 3 - 1$

$(257, (1, 1, 2)) \implies \text{chain} = (((2\ 3 + 1)2\ 3 + 1)2\ 3 - 1)2\ 3^2 + 1.$

Thus, $\text{chain} = 2^4 3^5 + 2^3 3^4 + 2^2 3^3 - 2\ 3^2 + 1 = 4627$.

The average chain length of the ternary/binary method is approximately $\mathcal{O}\left(\frac{\log_2 t}{4.3774}\right)$. The average chain cost of the ternary/binary method is approximately

$$\log_2 t \left(\frac{1}{4.3774}\mathbf{ADD} + 0.4569\mathbf{DBL} + 0.3427\mathbf{TPL}\right)$$

(Doche & Habsieger, 2008). The ternary/binary method can be extended to convert integer t to a multi-base chain with $\{2, 3, 5\}$-integers. The extension is achieved by modifying the reduction step. The reduction step change to t is repeatedly divided by 2, 3, or 5. As a result, t becomes coprime to 15 in the problem step.

Algorithm 5.8 Performing Scalar Multiplication on Ternary/binary Chains

Input: $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$ and $P \in E(\mathbb{F}_q)$
Output: tP
$Q \leftarrow P$
for $i \leftarrow l$ **to** 1 **do**
  for $j \leftarrow 1$ **to** a_i **do** $Q \leftarrow 2Q$
  for $j \leftarrow 1$ **to** b_i **do** $Q \leftarrow 3Q$
  if $(s_i = 1)$ **then** $Q \leftarrow Q + P$
  else if $(s_i = -1)$ **then** $Q \leftarrow Q - P$
return Q

5.5 Multi-base NAF Method

Longa and Gebotys (2009) proposed the multi-base NAF method for a multi-base number system. One form of this method is to convert t to a double-base chain with $\{2, 3\}$-integers. Algorithm 5.9 shows the steps of this method, and it can be described in the following four steps: the reduction, problem, solution, and selection steps. The first three steps are similar to the ternary/binary method. The selection step is different from the ternary/binary method in that it is directed by $t \mod 4$. It is important to note that $t \mod 4$ decides the selection of either $t - 1$ or $t + 1$ because the result of $t \mod 4$ is either 1 or 3.

Algorithm 5.9 Multi-base NAF Method

Input: positive integer t
Output: $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$ where $s_i \in \{-1, 0, +1\}, a_i, b_i \geq 0$, l is chain length
$t \leftarrow t/(2^a 3^b)$ where a, b are the 2-adic and 3-adic valuations of t
$l \leftarrow 1$
$(s_l, a_l, b_l) \leftarrow (0, a, b)$
while $(t > 1)$ **do**
  $l \leftarrow l + 1$
  if $(t \pmod 4 \equiv 1)$ **then** $s \leftarrow 1$ **else** $s \leftarrow -1$
  $t \leftarrow (t - s)/(2^a 3^b)$ where a, b are the 2-adic and 3-adic valuations of $(t - s)$
  $(s_l, a_l, b_l) \leftarrow (s, a, b)$
return $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$

For example, we want to find a chain for $t = 4627$ using the multi-base NAF method with $\{2, 3\}$-integers. Then, the output of Algorithm 5.9 is the following:

$(t, (s_1, a_1, b_1))$	$(t, (s_2, a_2, b_2))$	$(t, (s_3, a_3, b_3))$	$(t, (s_4, a_4, b_4))$
$(4627, (0, 0, 0))$	$(1157, (-1, 2, 0))$	$(289, (1, 2, 0))$	$(1, (1, 5, 2))$

Getting the chain steps can be accomplished in the same way of the ternary/binary method.

$(1,(1,5,2)) \implies \text{chain} = 2^5 3^2 + 1$

$(289,(1,2,0)) \implies \text{chain} = (2^5 3^2 + 1)2^2 + 1$

$(1157,(-1,2,0)) \implies \text{chain} = ((2^5 3^2 + 1)2^2 + 1)2^2 - 1.$

Thus, chain $= 2^9 3^2 + 2^4 + 2^2 - 1 = 4627$.

The average chain length of the multi-base NAF method with $\{2,3\}$-integers is approximately $\mathcal{O}\left(\frac{\log_2 t}{4.1887}\right)$. The average chain cost of the multi-base NAF method with $\{2,3\}$-integers is approximately

$$\log_2 t \left(\frac{1}{4.1887}\mathbf{ADD} + 0.7162\mathbf{DBL} + 0.179\mathbf{TPL}\right).$$

The multi-base NAF method also can be extended to convert t to a multi-base chain with $\{2,3,5\}$-integers. The extension can be achieved in the same manner as the ternary/binary method. The reduction step change to t is repeatedly divided by 2, 3, or 5. Thus, the problem step becomes t is coprime to 15. The average chain length of the multi-base NAF method with $\{2,3,5\}$-integers is approximately $\mathcal{O}\left(\frac{\log_2 t}{4.9143}\right)$. The average chain cost of the multi-base NAF method with $\{2,3,5\}$-integers is approximately

$$\log_2 t \left(\frac{1}{4.9143}\mathbf{ADD} + 0.6104\mathbf{DBL} + 0.1526\mathbf{TPL} + 0.0635\mathbf{QPL}\right).$$

5.6 Tree-based Method

Doche and Habsieger (2008) proposed the tree-based method for a multi-base number system. The simplest form of the tree-based method is to convert t to a double-base with $\{2,3\}$-integers. This section emphasizes the primary version of the tree-based method as Algorithm 5.10 shows. The next section emphasizes on the tree-based method with bound-size for a higher quality chain. The tree-based method uses the same four steps as the ternary/binary method. However, the approach is different in the selection step because it is dependent on whether $t-1$ or $t+1$ results in

the smallest t after the reduction step. For an example of the selection step, assume $t = 115$. Then the method will select $t - 1$ because the result 19 is less than 29. To explain, the result of $t - 1$ after the reduction step is $(115 - 1)/(2\ 3) = 19$ and the result of $t + 1$ after the step is $(115 + 1)/2^2 = 29$.

Algorithm 5.10 Tree-based Method

Input: positive integer t
Output: $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$ where $s_i \in \{-1, 0, +1\}, a_i, b_i \geq 0$, l is chain length.
$t \leftarrow t/(2^a 3^b)$ where a, b are the 2-adic and 3-adic valuations of t
$l \leftarrow 1$
$(s_l, a_l, b_l) \leftarrow (0, a, b)$
while $(t > 1)$ **do**
  $l \leftarrow l + 1,\ s \leftarrow 1$
  $t \leftarrow (t - 1)/(2^a 3^b)$ where a, b are the 2-adic and 3-adic valuations of $(t - 1)$
  $n \leftarrow (t + 1)/(2^c 3^d)$ where c, d are the 2-adic and 3-adic valuations of $(t + 1)$
  if $(t > n)$ **then** $t \leftarrow n,\ (s, a, b) \leftarrow (-1, c, d)$
  $(s_l, a_l, b_l) \leftarrow (s, a, b)$
return $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$

For an example of this method, let us find a chain for $t = 4627$. Then the output of Algorithm 5.10 is the following:

$(t, (s_1, a_1, b_1))$	$(t, (s_2, a_2, b_2))$	$(t, (s_3, a_3, b_3))$
$(4627, (0, 0, 0))$	$(257, (1, 1, 2))$	$(1, (1, 8, 0))$

The steps of getting the chain follows the same manner of the ternary/binary method.

$(1, (1, 8, 0)) \implies$ chain $= 2^8 + 1$

$(257, (1, 1, 2)) \implies$ chain $= (2^8 + 1)2\ 3^2 + 1$.

Therefore, chain $= 2^9 3^2 + 2\ 3^2 + 1 = 4627$.

The average chain length of the tree-based method with $\{2, 3\}$-integers is approximately $\mathcal{O}\left(\frac{\log_2 t}{4.6419}\right)$. The average chain cost of the tree-based chain with $\{2, 3\}$-integers is approximately

$$\log_2 t \left(\frac{1}{4.6419}\mathbf{ADD} + 0.5569\mathbf{DBL} + 0.2795\mathbf{TPL}\right).$$

The tree-based method can be extended to convert t to multi-base chain with

$\{2, 3, 5\}$-integers. The extension also can be achieved as the same manner of the ternary/binary method. The reduction step change is to repeatedly divide t by 2, 3 or 5. Thus, the problem step becomes t is coprime to 15. Yu et al. (2013) proved the average chain length of the tree-based method with $\{2, 3, 5\}$-integers is approximately $\mathcal{O}\left(\frac{\log_2 t}{5.6142}\right)$. The average chain cost of the tree-based method with $\{2, 3, 5\}$-integers is approximately

$$\log_2 t \left(\frac{1}{5.6142}\mathbf{ADD} + 0.454\mathbf{DBL} + 0.216\mathbf{TPL} + 0.0876\mathbf{QPL}\right).$$

6 MBNS Methods with Pre-computation

The pre-computation concept can be used with the MBNS methods to speed up the performance. It is used in both the converting phase and the performing phase of the methods. For example, the window τ-NAF method uses the pre-computation concept in the performing phase, as explained in Section 4.4.1. In this section, we focus on the pre-computation concept in the converting phase of the methods. The main advantage of using the pre-computation concept in the converting phase, compared to the performing phase, is that we do not have to do this pre-computation on-the-fly. This might be useful for certain applications such as the MBNS in controlled environments (Doche, 2014).

6.1 rDAG-based Method

Bernstein et al. (2017) recently proposed the rDAG-based method for a double-base number system. The main objective for the rDAG-based method is to find an optimal chain with respect to its cost. The idea of the rDAG-based method is to create a table with b_{max} columns and a_{max} rows. The value of (a_{max}, b_{max}) can be estimated by $a_{max} = \lceil \log_2 t \rceil$ and $b_{max} = \lceil \log_3 t \rceil$. Each table cell contains 4 nodes at most, according to the theorem in Bernstein et al. (2017). A node is represented in the form of $(t, (s, a, b), cost, (seq, pre))$, as Table 6.1 shows.

Table 6.1
Node Attributes for the rDAG-based Method

t	positive integer
(s, a, b)	$s\ 2^a 3^b$ where $s \in \{+1, 0, -1\}$ and $a, b \in \{0, 1\}$
cost	chain cost
(seq, pre)	seq is a node sequence number in Table$[i][j]$, and $seq \in \{0, 1, 2, 3\}$ pre is an index of the previous node in Table$[i-1][j]$ or Table$[i][j-1]$

Algorithm 6.1 explains the steps of the rDAG-based method. First, the method starts to investigate nodes in the table cell with row = 0 and column = 0. Next, the method moves to investigate nodes in the table cell with row = 0 and column = 1. Once all nodes in table cells with row = 0 are investigated, the method moves to investigate

nodes in the next table cell row. The method repeats these steps for all rows. It doesn't stop when it finds a node with $t = 1$. It stops when all the table cells are investigated. Finally, the method returns an optimal chain using the steps in Algorithm 6.2.

Definition 7. The DAG-based abstract idea states that if t is odd, three options are investigated: $(t-1)/2$, $(t+1)/2$, and $(t-s)/3$. If t is even, two options are investigated: $t/2$ and $(t-s)/3$ where $s \in \{-1, 0, +1\}$.

An investigated node creates two or three nodes that are inserted into the next right and the next below table cells. This is because this method follows the DAG-based abstract idea. A node is inserted into a bucket according to its value (a, b). The value (a, b) of nodes monotonically increases. This property guarantees that the method investigates all nodes and no node is skipped. The time complexity of the rDAG-based method is approximately $\mathcal{O}\left((\log_2 t)^2\right)$.

Algorithm 6.1 rDAG-based Method

Input: positive integer t and the upper bound (a_{max}, b_{max})
Output: $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$ where l is expansion length
initialize Table$[a_{max}][b_{max}]$ and $optimal(0, 0, 0, \infty)$
insert new $node(t, (0, 0, 0), 0, (0, 0))$ in Table[0][0]
for $i \leftarrow 0$ **to** a_{max} **do**
  for $j \leftarrow 0$ **to** b_{max} **do**
    if (Table$[i][j]$ is empty) **then continue**
    for each *node* in Table$[i][j]$
      if $(node.t = 1)$ **then**
        if $(optimal.cost > node.cost)$ **then** $optimal(i, j, node.seq, node.cost)$
        continue
      for each $s \in \{+1, 0, -1\}$
        if $((node.t - s) \pmod 2 \equiv 0)$ **then**
          $t \leftarrow (node.t - s)/2$
          $cost \leftarrow node.cost$ + **DBL** + $|s|$ **ADD**
          insert new $node(t, (s, 1, 0), cost, (seq, node.seq))$ in Table$[i+1][j]$
          If there are nodes with similar t in Table$[i+1][j]$ **then**
            keep only the one with the lowest cost
        if $((node.t - s) \pmod 3 \equiv 0)$ **then**
          $t \leftarrow (node.t - s)/3$
          $cost \leftarrow node.cost$ + **TPL** + $|s|$ **ADD**
          insert new $node(t, (s, 0, 1), cost, (seq, node.seq))$ in Table$[i][j+1]$
          If there are nodes with similar t in Table$[i][j+1]$ **then**
            keep only the one with the lowest cost
return get-chain(Table, *optimal*)

For example, we want to find a chain for $t = 13$ using the rDAG-based method with $\{2, 3\}$-integers. Assume the cost of **ADD** $= 2$, **DBL** $= 1$, **TPL** $= 2$, and the upper bound ($a_{max} = 4, b_{max} = 2$) is given. Then, Algorithm 6.1 creates a table, as shown in Table 6.2.

Table 6.2
Example of the rDAG-based Method for $t = 13$

	0	1	2
0	$(13,(0,0,0),0,(0,0))$	$(4,(1,0,1),4,(0,0))$	$(1,(1,0,1),8,(0,0))$
1	$(6,(1,1,0),3,(0,0))$ $(7,(-1,1,0),3,(1,0))$	$(2,(0,1,0),5,(0,0))$	$(1,(-1,0,1),9,(0,0))$
2	$(3,(0,1,0),4,(0,0))$ $(4,(-1,1,0),6,(1,1))$	$(1,(0,1,0),6,(0,0))$	
3	$(1,(1,1,0),7,(0,0))$ $(2,(-1,1,0),7,(1,0))$	$(1,(-1,0,1),11,(0,1))$	
4	$(1,(0,1,0),8,(0,1)$		

Getting the chain steps can be accomplished by Algorithm 6.2 in the following Horner's rule manner:

$(1,(0,1,0),6,(0,0)) \implies \text{chain} = 2$

$(2,(0,1,0),5,(0,0)) \implies \text{chain} = (2)2$

$(4,(1,0,1),4,(0,0)) \implies \text{chain} = ((2)2)3 + 1.$

Thus, chain $= 2^2\ 3 + 1 = 13$.

Algorithm 6.2 get-chain for the rDAG-based Method

Input: Table, i, j, k
Output: $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$
$l \leftarrow 0$
while ($i > 0$ or $j > 0$) **do**
  $l \leftarrow l + 1$
  find *node* in Table$[i][j]$ such that $node.seq = k$
  $(s_l, a_l, b_l) \leftarrow (node.s, node.a, node.b)$
  if $(node.a = 1)$ **then** $i \leftarrow i - 1$ **else** $j \leftarrow j - 1$
  $k \leftarrow node.pre$
return $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$

6.2 Proposed Bucket Methods

6.2.1 DAG/bucket method. The idea of the DAG/bucket method is that we have buckets that are indexed by chain cost. A bucket contains nodes. A node can

be represented by $(t, (s, a, b), (cost, seq), (pre_i, pre_j))$, as Table 6.3 describes. Nodes in a bucket have similar chain cost and different t, and they are ordered by their t. This is because the method deletes any repeated nodes with similar t in a bucket. The number of nodes in a bucket is determined by bucket-size.

Table 6.3
Node Attributes for the DAG/bucket Method

t	positive integer
(s, a, b)	$s\ 2^a\ 3^b$ where $s \in \{+1, 0, -1\}$ and $a, b \in \{0, 1\}$
$(cost, seq)$	$cost$ is chain cost seq is a node sequence number in Bucket$[i]$, and $seq \in \{0, ..., bucket\text{-}size - 1\}$
(pre_i, pre_j)	pre_i is an index of the previous bucket pre_j is an index of the previous node in Bucket$[pre_i]$

Algorithm 6.3 shows the steps of the DAG/bucket method. First step: the method investigates a bucket that has nodes with chain cost = 0. In this case, we have only one node that needs to be investigated. An investigated node generates two or three new nodes. This is because this method follows the DAG-based abstract idea, as Definition 7 shows. These news nodes are inserted into the next buckets. These next buckets have not been investigated yet. In other words, an investigated node cannot generate new nodes that are inserted into the same bucket. This is because the chain cost of nodes monotonically increases. This is an important property because it guarantees that the method investigates all nodes and does not skip any nodes.

Next step: the method moves to the next bucket that has nodes with the next lowest chain cost. In this bucket, the method starts to investigate the node with the smallest t. The method continues to investigate all nodes in this bucket and generates new nodes that are inserted into the next buckets. Then, the method moves to the next bucket and repeats the steps of investigating nodes and generating new nodes. Finally, when the method finds a node with $t = 1$, it stops and returns the chain. Algorithm 6.4 shows the steps of returning the chain.

For example, we want to find a chain for $t = 13$ using the DAG/bucket method with $\{2, 3\}$-integers. Assume the cost of **ADD**=2, **DBL**=1, **TPL**=2. Algorithm 6.3 creates the buckets shown in Table 6.4.

Algorithm 6.3 DAG/bucket Method

Input: positive integer t
Output: $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$ where l is expansion length
initialize Bucket[*bucket-max*]
insert new $node(t, (0,0,0), (0,0), (0,0))$ in Bucket[0]
for $i \leftarrow 0$ **to** *bucket-max* **do**
  if (Bucket[i] is empty) **then continue**
  for each *node* in Bucket[i]
    if ($node.t = 1$) **then return** get-chain(Bucket, $i, node.seq$)
    for each $s \in \{+1, 0, -1\}$
      if $((node.t - s) \pmod 2 \equiv 0)$ **then**
        $t \leftarrow (node.t - s)/2$
        $cost \leftarrow node.cost$ + **DBL** + $|s|$ **ADD**
        insert new $node(t, (s, 1, 0), (cost, seq), (i, node.seq))$ in Bucket[**round**($cost$)]
        remove any redundant nodes with respect to t
        keep the smallest nodes with respect to t within *bucket-size*
      if $((node.t - s) \pmod 3 \equiv 0)$ **then**
        $t \leftarrow (node.t - s)/3$
        $cost \leftarrow node.cost$ + **TPL** + $|s|$ **ADD**
        insert new $node(t, (s, 0, 1), (cost, seq), (i, node.seq))$ in Bucket[**round**($cost$)]
        remove any redundant nodes with respect to t
        keep the smallest nodes with respect to t within *bucket-size*

Table 6.4
Example of the DAG/bucket Method for $t = 13$

cost	**nodes**
0	$(13, (0,0,0), (0,0), (0,0))$
3	$(6, (1,1,0), (3,0), (0,0))$, $(7, (-1,1,0), (3,1), (0,0))$
4	$(3, (0,1,0), (4,0), (3,0))$, $(4, (1,0,1), (4,1), (0,0))$
5	$(2, (0,0,1), (5,0), (3,0))$
6	$(1, (0,0,1), (6,0), (4,0))$, $(3, (1,1,0), (6,1), (3,1))$, $(4, (-1,1,0), (6,2), (3,1))$
7	$(1, (1,1,0), (7,0), (4,0))$, $(2, (1,0,1), (7,1), (3,1))$
8	$(1, (1,0,1), (8,0), (4,1))$
9	$(1, (-1,0,1), (9,0), (5,0))$

Getting the chain steps can be accomplished by Algorithm 6.4 in the following manner:

$(1, (0,0,1), (6,0), (4,0)) \implies$ chain=3

$(3, (0,1,0), (4,0), (3,0)) \implies$ chain=(3)2

$(6, (1,1,0), (3,0), (0,0)) \implies$ chain=((3)2)2 + 1.

Thus, chain=2^2 3 + 1 = 13.

The similarity between the DAG/bucket and the rDAG-based methods is in the way of generating new nodes. This is because they both follow the DAG-based abstract

Algorithm 6.4 get-chain for the DAG/bucket Method

Input: Bucket, i, j
Output: $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$
$l \leftarrow 0$
while $(i > 0)$ **do**
  $l \leftarrow l + 1$
  find *node* in Bucket[i] such that $node.seq = j$
  $(s_l, a_l, b_l) \leftarrow (node.s, node.a, node.b)$
  $i \leftarrow node.pre\_i,\ j \leftarrow node.pre\_j$
return $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$

idea, as Definition 7 shows. However, the main difference between the DAG/bucket and the rDAG-based method is the following. The main objective of the rDAG-based method is to find an optimal chain for integer t. It emphasizes finding an optimal chain without paying much attention to the time to find the chain. In contrast, the DAG/bucket method provides a systematic way to control the chain quality and the time to find the chain. This can be achieved by using the bucket-size idea. The bucket-size is a value that allow us to balance the chain quality and the time to find the chain.

6.2.2 Tree/bucket method. The idea of the tree/bucket method, in general, is similar to the DAG/bucket method because they both use the bucket-size to control the chain quality and the time to find the chain. In fact, this is the key idea of the bucket methods. However, they are different in the following ways. The tree/bucket method follows the tree-based abstract idea to generate new nodes, as Definition 8 shows. As a result, the tree/bucket generates a chain in faster running time than the DAG/bucket method. In contrast, the DAG/bucket follows the DAG-based abstract idea to generate new nodes. As a result, the DAG/bucket generates a higher quality chain than tree/bucket, as Experiment II shows.

Definition 8. The tree-based abstract idea states that if t is coprime with 6, then two options are investigated: $(t-1)/(2^a 3^b)$ and $(t+1)/(2^a 3^b)$ where a, b are 2-adic and 3-adic valuations of $(t-1)$ or $(t+1)$.

Besides the first difference, structurally, buckets in the tree/bucket method are indexed by chain length. A node can be represented by $(t, (s, a, b), (seq, pre))$, as Table

6.5 describes. Nodes in a bucket have similar chain length and different t, and they are ordered by their t.

Table 6.5
Node Attributes for the Tree/bucket Method

t	positive integer
(s, a, b)	$s\ 2^a\ 3^b$ where $s \in \{+1, 0, -1\}$ and $a, b \geq 0$
(seq, pre)	seq is a node sequence number in Bucket$[i]$, and $seq \in \{0, ..., bucket\text{-}size - 1\}$ pre is an index of the previous node in Bucket$[i-1]$

Algorithm 6.5 shows the steps of the tree/bucket method. It follows the same general idea of the DAG/bucket algorithm. However, we will explain the steps because we need to point out some differences. First step: the method initially investigates a bucket that has nodes with chain length = 1. In this bucket, we have only one node that needs to be investigated.

An investigated node generates two nodes. This is because this method follows the tree-based abstract idea, as Definition 8 shows. These new nodes are inserted into the next following bucket. An investigated node cannot generate new nodes that inserted into the same bucket. This is because the chain length of nodes monotonically increases. This guarantees that the method investigates all nodes and does not skip any nodes.

Next step: the method moves to the next bucket that has nodes with chain length = 2. In this bucket, the method starts to investigate the node with the smallest t. The method continues to investigates all nodes and generates new nodes that are inserted into the next following bucket. Then, the method moves to the next bucket and repeats the steps of investigating nodes and generating new nodes. The method stops and returns the chain when it finds a node with $t = 1$. The method must find nodes with $t = 1$ because t of nodes monotonically decreases. Algorithm 6.6 shows the step of returning the chain.

For example, we want to find a chain for $t = 29$ using the tree/bucket method with $\{2, 3\}$-integers. Algorithm 6.5 creates the following buckets:

Algorithm 6.5 Tree/bucket Method

Input: positive integer t
Output: $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$ where l is chain length
initialize Bucket[*bucket-max*]
$t \leftarrow t/(2^a\ 3^b)$ where a, b are 2-adic and 3-adic valuations of t
insert new $node(t, (0, a, b), (0, 0))$ in Bucket[1]
for $i \leftarrow 1$ **to** *bucket-max* **do**
  if (Bucket[i] is empty) **then continue**
  for each *node* in Bucket[i]
    if $(node.t = 1)$ **then return** get-chain(Bucket, $i, node.seq$)
    for each $s \in \{+1, -1\}$
      $t \leftarrow (node.t - s)/(2^a\ 3^b)$ where a, b are 2-adic and 3-adic valuations of $(node.t - s)$
      insert new $node(t, (s, a, b), (seq, node.seq))$ in Bucket[$i + 1$]
      remove any redundant nodes with respect to t
      keep the smallest nodes with respect to t within *bucket-size*

length	**nodes**
1	$(29, (0, 0, 0), (0, 0))$
2	$(5, (-1, 1, 1), (0, 1))$, $(7, (1, 2, 0), (1, 1))$
3	$(1, (1, 2, 0), (0, 2))$

Getting the chain steps can be accomplished by Algorithm 6.6 in the following manner:

$(1, (1, 2, 0), (0, 2)) \implies \text{chain} = 2^2 + 1$

$(5, (-1, 1, 1), (0, 1)) \implies \text{chain} = (2^2 + 1)2\ 3 - 1$

$(29, (0, 0, 0), (0, 0)) \implies \text{chain} = (2^2 + 1)2\ 3 - 1.$

Thus, chain $= 2^3\ 3 + 2\ 3 - 1 = 29$.

Algorithm 6.6 get-chain for the Tree/bucket Method

Input: Bucket, i, j
Output: $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$
$l \leftarrow 0$
while $(i > 0)$ **do**
  $l \leftarrow l + 1$
  find *node* in Bucket[i] such that $node.seq = j$
  $(s_l, a_l, b_l) \leftarrow (node.s, node.a, node.b)$
  $i \leftarrow i - 1,\ j \leftarrow node.pre$
return $(s_1, a_1, b_1), ..., (s_l, a_l, b_l)$

6.2.3 Bucket-size and bucket-max. The performance of the bucket methods, including the tree/bucket and the DAG/bucket, is affected by two main

factors: bucket-size and bucket-max. The bucket-size is the number of nodes at maximum in a bucket. It also can be described as a value that controls the chain quality and the time to find the chain. A large bucket-size gives a higher quality chain and a higher conversion cost. A smaller bucket-size gives a lower chain quality and a lower conversion cost, as Experiment II shows. The question is what is the best value for the bucket-size? In applications where the converting phase is performed offline, a higher bucket-size is more practical. In applications where the converting phase is performed on-the-fly, we select a bucket-size that gives a faster running time. This is with the assumption that the running time considers the converting phase and the performing phase as one operation.

Bucket-max is the second factor that affects the performance of the bucket methods. The bucket-max is the number of buckets that are needed at maximum to convert t to a chain. The question is how can we determine the bucket-max of the bucket methods? We can use the NAF method properties to estimate the bucket-max of the bucket methods. To explain, in the tree/bucket method, buckets are indexed by chain length. Therefore, we can use the average chain length of NAF method to estimate the bucket-max of the tree/bucket method. Recall that the average chain length of the NAF method is approximately $(\log_2 t)/3$. In the DAG/bucket method, buckets are indexed by chain cost. Therefore, we can use the average chain cost of the NAF method to estimate the bucket-max of the DAG/bucket method. In this case, we round the chain cost of nodes to the nearest integer. Recall that the average chain cost of the NAF method is approximately $\log_2 t(1/3\ \mathbf{ADD} + \mathbf{DBL})$.

7 Experimental Results

In this section, we show three different experimental results. The purpose of these experiments was to verify some of the theoretical results in this dissertation. In the first experiment, we compared the performance of the multi-base methods without pre-computation. In the second experiment, we compared the performance of the multi-base methods with pre-computation. In the third experiment, we compared the pre-computation schemes of window τ-NAF for Koblitz curves. In these experiments, we utilized the most efficient formulas presented in Section 3 and Section 4. We used three properties to compare the performance of the methods: the average chain length, the average chain cost, and the average running time.

The main advantage of the chain length and the chain cost is that they are not affected by the device specifications. In contrast, the running time is affected by the device specifications. To explain, the chain length is affected by the integer bit size. Besides the integer bit size, the chain cost is affected by various factors, such as the method, the cost of formulas in particular coordinate systems, and the $\mathbf{S/M}$ and the $\mathbf{I/M}$ ratio assumptions. We assumed $1\mathbf{S} = 0.8\mathbf{M}$ over prime fields as suggested in Bernstein et al. (2017). We assumed $1\mathbf{S} = 0.4\mathbf{M}$ over binary fields because it agrees with the $\mathbf{S/M}$ ratio of the experiments' device. We did not have an assumption about the $\mathbf{I/M}$ ratio because we worked on projective coordinates over both prime and binary fields. We see that chain cost is more accurate than chain length because it considers various factors to evaluate a chain.

The main advantage of the running time is that it evaluates all implementation details of the methods. It evaluates the time to convert integer t to a chain. This is an important evaluation because some methods spend more time in the converting phase to enhance the chain cost. The running time is affected by the experiment computing environment, such as the CPU, the OS, the program language, the number of temporary variables, and the finite field arithmetic library. We used an Intel Core i5-8250U processor with the speed 1.6 GHz. We used C programming language and GCC compiler on 64-bit Ubuntu Linux OS.

For prime fields, we used GMP library for field arithmetic operations and worked in standard twisted Edwards coordinates (Granlund & et al., n.d.; Bernstein et al., 2008). We selected twisted Edwards curves that are suggested in Josefsson and Liusvaara (2017) and Aranha, Barreto, Pereira, and Ricardini (2013) because standard twisted Edwards coordinates are the most efficient option for curves with coefficient a as a small constant (e.g., 1, -1) and d as a large constant. More twisted Edwards curves were suggested by Hamburg (2015) and Bos, Costello, Longa, and Naehrig (2014).

For binary fields, we used MIRACL library for field arithmetic operations (Matula & Kornerup, n.d.). We worked in λ-coordinates with the NIST random elliptic curves because λ-coordinates are the most efficient option for these curves when their coefficient $a = 1$ and b is a large constant. We also worked in twisted μ_4-normal coordinates with the NIST Koblitz curves because μ_4-normal coordinates are the most efficient option for these curves when their coefficient $a = 0$ and $b = 1$ (Barker, 2013). We utilized the formulas presented in Section 4.

7.1 Experiment I

Our objective in this experiment was to compare the performance of the multi-base methods without pre-computation. We wanted to know whether the multi-base methods improved the performance over the single-base methods. What was the percentage of improvement if we used the multi-base methods over the single-base methods? Which one of the multi-base methods gave the fastest performance? We present the results of this experiment in tables: Table 7.1 and Table 7.2 show performance comparison between the methods with respect to the average chain length (l) and the average chain cost (m) over both prime and binary fields. Table 7.3 and Table 7.4 show performance comparison between the methods with respect to the average running time ($time$) over both prime and binary fields.

We selected the binary and the NAF for the single-base methods, as shown in Algorithm 5.1 and Algorithm 5.3 respectively. We selected the greedy, the ternary/binary, the multi-base NAF, and the tree-based for the multi-base methods, as

shown in Algorithm 5.4, Algorithm 5.7, Algorithm 5.9, and Algorithm 5.10 respectively. For standard twisted Edwards coordinates, we utilized the mixed $P + Q$, $2P$, $3P$, and $5P$ formulas, as shown in Table 3.13, Table 3.15, 3.16, and Table 3.17 respectively. For λ-coordinates, we utilized the mixed $P + Q$, $2P$, $3P$, and $5P$ formulas, as shown in Table 4.8, Table 4.9, 4.10, and Table 4.11 respectively.

7.1.1 Results of Experiment I. We obtained four main results from Experiment I. First result: the multi-base methods, in general, had a better performance than the single-base methods over both prime and binary fields. To explain, the results show that the multi-base methods gave an approximately 6% to 9% lower m than the single-base methods over both prime and binary methods. They, except for the greedy, gave an approximately 6% to 14% less *time* than the single-base methods over both prime and binary fields. The tree-based method had the highest percentage of improvement among the tested methods.

Second result: the methods over binary fields had lower m than the methods over prime fields. In contrast, the methods over prime fields had less *time* than the methods over binary fields. To explain, the reason for having lower m over binary fields is that the **S/M** ratio over binary fields is lower than the **S/M** ratio over prime fields. The reason for having lesser *time* over prime fields is that GMP library was faster than MIRACL library. GMP library can only support prime field arithmetic, while MIRACL library can support both binary and prime field arithmetic. We decided to use GMP library for prime field arithmetic because GMP library gave us a faster performance than MIRACL library.

Third result: the performance improvement decreases with a higher-base method. To explain, the results show that the improvement from a single-base method to a double-base method was higher than the improvement from a double-base method to a triple-base method. For example, consider m of the multi-base NAF method over prime fields. The improvement from the single-base method to the double-base method is 7.65%. The improvement from the double-base method to the triple-base method is 2.37%. This drop implies that a higher-base formula (e.g., $7P$, $11P$) is less important

Table 7.1
Theoretical Comparison between Single-base and Multi-base Methods in Standard Twisted Edwards Coordinates over $\mathbb{F}_p$

	254-bit			382-bit			521-bit		
	l	m	%	l	m	%	l	m	%
Binary	126.97	2922.86		191.01	4408.98		260.52	6020.79	
(2)NAF	85.13	2475.16		127.78	3729.39		174.17	5092.31	
(2, 3)greedy	55.94	2272.66	8.18	83.42	3419.97	8.20	113.64	4666.11	8.36
ternary/binary	58.48	2322.94	6.14	87.67	3500.82	6.12	119.51	4781.17	6.10
(2, 3)NAF	61.07	2285.58	7.65	91.59	3444.74	7.63	124.87	4705.08	7.60
(2, 3)tree	55.11	2260.44	8.67	82.62	3407.26	8.63	112.63	4654.09	8.60
(2, 3, 5)NAF	52.11	2226.92	10.02	78.21	3357.49	9.97	106.60	4585.60	9.95
(2, 3, 5)tree	45.65	2202.94	10.99	68.40	3320.77	10.95	93.15	4535.11	10.94

l: The average chain length.
m: The average chain cost.
%: The improvement percentage with respect to m in comparison to (2)NAF.

Table 7.2
Theoretical Comparison between Single-base and Multi-base Methods in λ-coordinates over $\mathbb{F}_{2^m}$

	283-bit			409-bit			571-bit		
	l	m	%	l	m	%	l	m	%
Binary	141.49	2809.84		204.53	4070.26		285.43	5689.35	
(2)NAF	94.77	2402.42		136.78	3477.85		190.83	4860.64	
(2, 3)greedy	62.67	2206.11	8.17	88.98	3190.37	8.26	123.71	4453.86	8.36
ternary/binary	65.13	2249.43	6.35	93.89	3256.74	6.35	130.94	4552.48	6.33
(2, 3)NAF	68.01	2222.78	7.47	98.05	3218.30	7.46	136.72	4498.39	7.45
(2, 3)tree	61.45	2196.23	8.5	88.52	3179.72	8.57	123.34	4444.23	8.56
(2, 3, 5)NAF	57.98	2183.51	9.11	83.71	3162.88	9.05	116.69	4420.97	9.04
(2, 3, 5)tree	50.87	2164.95	9.88	73.23	3134.84	9.86	101.89	4380.68	9.87

l: The average chain length.
m: The average chain cost.
%: The improvement percentage with respect to m in comparison to (2)NAF.

Table 7.3
Running Time Comparison between Single-base and Multi-base Methods in Standard Twisted Edwards Coordinates over $\mathbb{F}_p$

	254-bit		382-bit		521-bit	
	time	%	*time*	%	*time*	%
Binary	618.24		1121.56		2132.3	
(2)NAF	544.52		1025.42		1864.24	
(2, 3)greedy	557.59	-2.40	1064.81	-3.84	1923.91	-3.20
ternary/binary	491.15	9.80	915.92	10.63	1646.30	11.69
(2, 3)NAF	493.37	9.39	918.69	10.41	1652.19	11.37
(2, 3)tree	488.06	10.36	913.26	10.93	1644.94	11.76
(2, 3, 5)NAF	476.59	12.47	886.84	13.51	1604.57	13.92
(2, 3, 5)tree	470.93	13.51	881.19	14.06	1593.85	14.50

time: The average running time in μs.
%: The improvement percentage in comparison to (2)NAF.

Table 7.4
Running Time Comparison between Single-base and Multi-base Methods in λ-coordinates over $\mathbb{F}_{2^m}$

	283-bit		409-bit		571-bit	
	time	%	*time*	%	*time*	%
Binary	889.85		1924.70		4016.82	
(2)NAF	772.19		1651.49		3437.60	
(2, 3)greedy	788.48	-2.10	1696.77	-2.74	3534.77	-2.82
ternary/binary	712.15	7.77	1529.90	7.36	3218.32	6.37
(2, 3)NAF	710.70	7.96	1516.71	8.16	3185.95	7.32
(2, 3)tree	709.51	8.11	1515.82	8.21	3177.18	7.57
(2, 3, 5)NAF	695.16	9.97	1502.76	9.01	3117.46	9.31
(2, 3, 5)tree	690.52	10.57	1490.13	9.77	3112.44	9.45

time: The average running time in μs.
%: The improvement percentage in comparison to (2)NAF.

than a lower-base formula (e.g. $2P$, $3P$). This is because a higher-base method gives less percentage of improvement than a lower-base method.

Fourth result: the greedy method is impractical for implementation for two reasons. The first reason is that the greedy method requires finding the best upper bound (a_{max}, b_{max}). We describe how to find the best upper bound in Section 5.3. In standard twisted Edwards coordinates, we found that the best upper bound for 254-bit, 382-bit, and 521-bit integers is $(140, 73)$, $(210, 109)$, and $(290, 146)$ respectively. In λ-projective coordinates, the best upper bound for 283-bit, 409-bit, and 571-bit integers is $(160, 78)$, $(220, 120)$, and $(310, 165)$ respectively. We see that the greedy method requires extra step to find the best upper bound, while the other multi-base methods do not require this step.

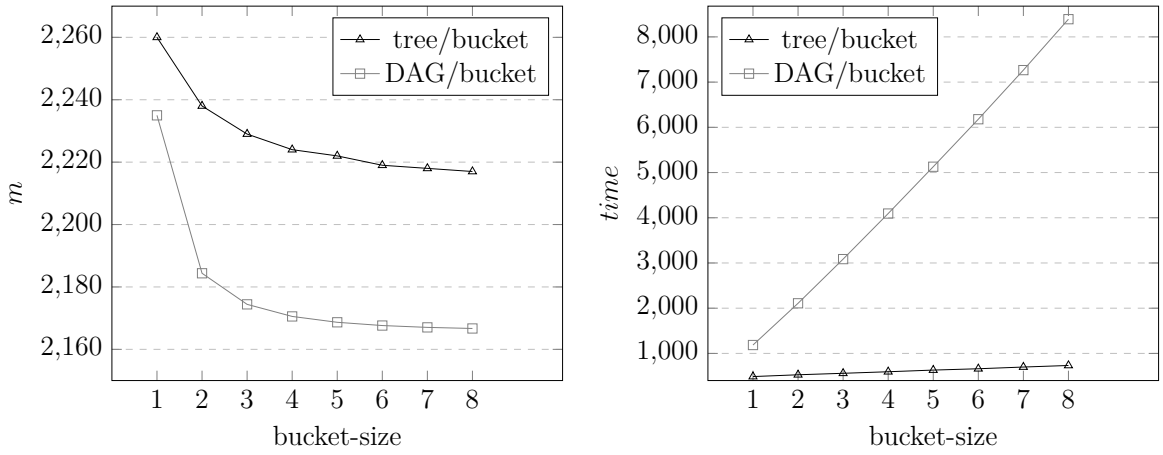
The second reason is that it requires finding the best approximation for integer t in terms of a $\{2, 3\}$-integer. We describe two ways to find the best approximation in Section 5.3. We decided to use the line search to find the best approximation because it does not require any pre-computation. However, the converting phase in the greedy method with the line search is still not as efficient as the other multi-base methods. This is because the other methods use a dynamic approach to convert integer t to a chain.

7.2 Experiment II

Our objective in this experiment was to compare the performance of the multi-base methods with pre-computation. We considered, unlike Experiment I, the

pre-computation concept in the converting phase of a method, as explained in Section 6. We compared the performance of the tree/bucket and the DAG/bucket methods with respect to the average chain cost (m) and the average running time ($time$) in μs. We wanted to know the impact of the bucket-size on the bucket methods. We focused on 254-bit integers in standard twisted Edwards coordinates over prime fields. We present the results of this experiment in Figure 7.1. It shows the impact of the bucket-size on the performance of the bucket methods.

Figure 7.1. Experimental Comparison between Tree/bucket and DAG/bucket Methods



7.2.1 Results of Experiment II. We obtained three main results from Experiment II. First result: the bucket-size of the bucket methods is a value to balance the chain cost and the time to find the chain. To explain, a higher bucket-size gave lower m and more $time$. In contrast, a lower bucket-size gave higher m and less $time$. The highest percentage increase of m was when bucket-size $= 2$. The percentage increase of m became minor when bucket-size > 4. Therefore, it is more practical to select the bucket-size from 2 to 4 in standard twisted Edwards coordinates. It is important to note that this conclusion might be slightly different when we use different coordinate systems over different finite fields.

Second result: the tree-based method with unbound-size does not produce an optimal chain. We define an optimal chain as a chain that represents integer t with the lowest cost, as Definition 5 shows. We know the optimal chain is approximately 2165.58**M**, as Table 7.5 shows. The average chain cost of the tree/bucket with

Table 7.5
Experimental Comparison between Optimal and Near Optimal Chains

	254-bit		
	l	m	*time*
(2, 3)tree/bucket$_{size=\infty}$	51.01	2210.97	
(2, 3)DAG/bucket$_{size=4}$	49.71	2170.55	4093.47
(2, 3)rDAG-based	49.43	2165.58	14818.80

l: The average chain length.
m: The average chain cost.
time: The average running time in μs.

bucket-size $= \infty$ is approximately 2210.97**M**. The difference between them is 45.39**M**. This implies that the tree-based method with unbound-size is far from generating an optimal chain. We also saw that the tree/bucket method with bucket-size $= \infty$ does not produce lower m than the DAG/bucket with bucket-size > 1. Nevertheless, the tree/bucket had less *time* than DAG/bucket when it comes to generating a good quality chain.

Third result: generating a near optimal chain is more practical than generating an optimal chain. We define that a near optimal chain is a chain that is different from an optimal chain by 1**DBL** cost at most, as Definition 6 shows. The results show that the DAG/bucket method generated a near optimal chain in significantly less *time* than the rDAG-based method. To explain, Table 7.5 shows that when bucket-size $= 4$, the DAG/bucket method generated a near optimal chain. This is because the m difference between the rDAG-based method and the DAG/bucket method with bucket-size $= 4$ is $2170.55\mathbf{M} - 2165.58\mathbf{M} = 4.97\mathbf{M}$. This implies it is a near optimal chain because 4.97**M** is less than 1**DBL** cost in standard twisted Edwards coordinates. We noted that *time* $= 4093.47\mu$s in the DAG/bucket method with bucket-size $= 4$ and *time* $= 14818.8\mu$s in the rDAG-based method. This implies that the percentage increase of *time* is 72.37%. Therefore, it is more practical to generate an near optimal chain because it takes significantly less *time* than an optimal chain.

7.3 Experiment III

Our objective in this experiment was to compare the optimal and the novel pre-computation schemes of window τ-NAF for Koblitz curves. We implemented the

optimal and the novel pre-computation schemes, as Table 4.16 and Table 4.17 show. We utilized the $\bar{\tau}P$ formula in λ-projective and twisted μ_4-normal coordinates, as Table 4.18, Table 4.19, and Table 4.20 show. We present the results of Experiment III in Table 7.6, Table 7.7, Table 7.8, and Table 7.9.

Table 7.6 and Table 7.8 show the theoretical comparison between the optimal and the novel pre-computation schemes in both λ-projective and twisted μ_4-normal coordinates. In these tables, we counted the number of multiplications (m) for the window width 4, 5, and 6. For twisted μ_4-normal coordinates, when $a = 0$, we assumed the cost of mixed $P + Q = 6\mathbf{M} + 2\mathbf{S}$, $\bar{\tau}P = 2\mathbf{M} + 2\mathbf{S}$ and $\tau P = 3\mathbf{S}$. When $a = 1$, the cost of mixed $P + Q = 8\mathbf{M} + 2\mathbf{S}$, $\bar{\tau}P = 6\mathbf{M} + 5\mathbf{S}$ and $\tau P = 3\mathbf{S}$. For λ-coordinates, we assumed the cost of mixed $P + Q = 8\mathbf{M} + 2\mathbf{S}$, $\bar{\tau}P = 5\mathbf{M} + 3\mathbf{S}$ and $\tau P = 2\mathbf{S}$.

Table 7.7 and Table 7.9 show the average running time (*time*) comparison between the optimal and the novel pre-computation schemes in λ-projective and twisted μ_4-normal coordinates. We focused on 283-bit integers when Koblitz curve coefficient $a = 0$ and 163-bit integers when Koblitz curve coefficient $a = 1$. The domain parameters for the used Koblitz curves can be found in Barker (2013).

7.3.1 Results of Experiment III. We obtained two main results from Experiment III. First result: with respect to m and *time*, the novel scheme had a better performance than the optimal scheme. To explain, for λ-coordinates, the novel scheme had an approximately 20% to 33% lower m than the optimal scheme. For twisted μ_4-normal coordinates, when $a = 0$, the novel scheme had an approximately 39% to 63% lower m than the optimal scheme. When $a = 1$, the novel scheme had an approximately 6% to 16% lower m. We see that when $a = 0$, the novel scheme in twisted μ_4-normal coordinates showed significant improvement over the optimal scheme. We also see that the novel scheme in λ-coordinates had the same percentage of improvement in both cases, $a = 1$ and $a = 0$.

Second result: consider only the novel scheme. With respect to m and *time*, when $a = 0$, the novel scheme in μ_4-normal coordinates had a better performance than λ-coordinates. In contrast, when $a = 1$, the novel scheme in λ-coordinates had a better

performance than μ_4-normal coordinates. To explain, when $a = 0$, the novel scheme in μ_4-normal coordinates had an approximately 40% to 54% lower m than λ-coordinates. When $a = 1$, the novel scheme in λ-coordinates had an approximately 14% to 22% lower m than μ_4-normal coordinates.

Table 7.6
Theoretical Comparison between the Optimal and the Novel Pre-computation Schemes when $a = 0$

	λ-projective			μ_4-normal		
	Optimal	**Novel**	%	**Optimal**	**Novel**	%
$w = 4$	$24\mathbf{M} + 10\mathbf{S} \approx 28.0\mathbf{M}$	$15\mathbf{M} + 9\mathbf{S} \approx 18.6\mathbf{M}$	33.57	$18\mathbf{M} + 12\mathbf{S} \approx 22.8\mathbf{M}$	$6\mathbf{M} + 6\mathbf{S} \approx 8.4\mathbf{M}$	63.15
$w = 5$	$56\mathbf{M} + 18\mathbf{S} \approx 63.2\mathbf{M}$	$38\mathbf{M} + 22\mathbf{S} \approx 46.8\mathbf{M}$	25.94	$42\mathbf{M} + 20\mathbf{S} \approx 50.0\mathbf{M}$	$18\mathbf{M} + 17\mathbf{S} \approx 24.8\mathbf{M}$	50.40
$w = 6$	$120\mathbf{M} + 34\mathbf{S} \approx 133.6\mathbf{M}$	$90\mathbf{M} + 42\mathbf{S} \approx 106.8\mathbf{M}$	20.05	$90\mathbf{M} + 36\mathbf{S} \approx 104.4\mathbf{M}$	$50\mathbf{M} + 33\mathbf{S} \approx 63.2\mathbf{M}$	39.46

%: The improvement percentage.

Table 7.7
Running Time Comparison between the Optimal and the Novel Pre-computation Schemes when $a = 0$

	λ-projective			μ_4-normal		
	Optimal*	**Novel***	%	**Optimal***	**Novel***	%
$w = 4$	9.17	5.90	35.65	7.75	3.11	59.87
$w = 5$	21.67	15.25	29.62	16.94	9.11	46.22
$w = 6$	47.07	36.08	23.34	34.96	22.25	36.35

%: The improvement percentage.
*: The average running time in μs.

Table 7.8
Theoretical Comparison between the Optimal and the Novel Pre-computation Schemes when $a = 1$

	λ-projective*			μ_4-normal		
	Optimal	**Novel**	%	**Optimal**	**Novel**	%
$w = 4$	$24\mathbf{M} + 10\mathbf{S} \approx 28.0\mathbf{M}$	$15\mathbf{M} + 9\mathbf{S} \approx 18.6\mathbf{M}$	33.57	$24\mathbf{M} + 12\mathbf{S} \approx 28.8\mathbf{M}$	$18\mathbf{M} + 15\mathbf{S} \approx 24.0\mathbf{M}$	16.66
$w = 5$	$56\mathbf{M} + 18\mathbf{S} \approx 63.2\mathbf{M}$	$38\mathbf{M} + 22\mathbf{S} \approx 46.8\mathbf{M}$	25.94	$56\mathbf{M} + 20\mathbf{S} \approx 64.0\mathbf{M}$	$44\mathbf{M} + 35\mathbf{S} \approx 58.0\mathbf{M}$	9.37
$w = 6$	$120\mathbf{M} + 34\mathbf{S} \approx 133.6\mathbf{M}$	$90\mathbf{M} + 42\mathbf{S} \approx 106.8\mathbf{M}$	20.05	$120\mathbf{M} + 36\mathbf{S} \approx 134.4\mathbf{M}$	$100\mathbf{M} + 63\mathbf{S} \approx 125.2\mathbf{M}$	6.84

%: The improvement percentage.

Table 7.9
Running Time Comparison between the Optimal and the Novel Pre-computation Schemes when $a = 1$

	λ-projective			μ_4-normal		
	Optimal*	**Novel***	%	**Optimal***	**Novel***	%
$w = 4$	5.79	3.75	35.23	5.84	4.79	17.97
$w = 5$	13.28	9.37	29.44	12.73	11.52	9.51
$w = 6$	28.04	21.32	23.96	27.09	28.08	7.41

%: The improvement percentage.
*: The average running time in μs.

8 Conclusion & Future Directions

8.1 Conclusion

Elliptic curve cryptography (ECC) is one of the most efficient types of public-key cryptography. It is appropriate for application when computing resources are restricted. Scalar multiplication (tP), an operation that adds point P on an elliptic curve to itself t times, is the main operation that is extensively used by ECC protocols. Therefore, this research studied the efficiency problem of scalar multiplication of ECC. One of the primary methods of speeding up scalar multiplication is to represent integer t by a single-base number system (SBNS). In the SBNS, point additions ($P + Q$) are replaced with point doublings ($2P$) for a faster performance. Minimizing the number of point additions is an essential technique to speed up scalar multiplication because a $P + Q$ formula has a higher cost than other formulas, such as $2P$.

A multi-base number system (MBNS) is a natural extension of the SBNS. The theoretical analysis and experimental results showed that when integer t was represented in the MBNS, the form length, on average, became shorter than when it is represented in the SBNS. This property leads to continued minimization of the number of point additions. However, it raises two new problems that need to be solved: the first problem is that the MBNS requires new optimized formulas such as $3P$ and $5P$. The second problem is that the MBNS needs a method that generates a high quality chain in an efficient manner.

To address the first problem, whenever possible, this research derived several optimized formulas in different elliptic curve coordinate systems. Weierstrass curves were the main focus because they are recommended by the National Institute of Standards and Technology (NIST). In the context of the MBNS, Jacobian projective coordinates are the most efficient coordinates for simplified Weierstrass curves over prime fields. λ-projective and twisted μ_4-normal coordinates are the most efficient coordinates for binary elliptic curves and Koblitz curves. Additionally, this research studied twisted Edwards curves because they have the most efficient $2P$ formulas over prime fields. In the context of the MBNS, $2P$ is the most important formula because it

is the most frequently used operation during scalar multiplication.

For Weierstrass curves, this research reviewed the previous studies of formula derivations in affine and projective coordinates over prime and binary fields. Then, new $4P$ and $5P$ formulas in affine coordinates and a new $2P$ formula in projective coordinates over prime fields were proposed. Also, new $2P$ and $\bar{\tau}P$ formulas in twisted μ_4-normal coordinates over binary fields were proposed. For Koblitz curves, the $\bar{\tau}P = \mu P - \tau P$ formula was used to improve the performance of the pre-computation schemes of the window τ-NAF. For twisted Edwards curves, this research reviewed previous studies of formula derivations and proposed $2Q + P$ and $5P$ formulas in standard projective coordinates.

To address the second problem, this research theoretically and experimentally advanced the MBNS methods. First, this research studied the existing MBNS methods: greedy, ternary/binary, multi-base NAF, tree-based, and rDAG-based. The emphasis was on the average chain length, the average chain cost, and the average conversion cost of the methods. The conversion cost can be expressed by the time complexity of a method to convert integer t to a chain. It is critical for a method to balance the conversion cost and the chain cost for a faster performance. This is because this research assumed a method goes through two phases. The first phase is to convert integer t to a chain and the second phase is to perform scalar multiplication on a given chain. Second, this research developed bucket methods for the tree-based and DAG-based abstract ideas. These bucket methods systematically balance the chain cost and the time to find the chain. They can generate a near optimal chain in efficient manner.

Last, this research conducted experiments to compare the SBNS methods and the MBNS methods utilizing state of the art formulas in both prime and binary fields. The experimental results showed that the MBNS methods without pre-computation had an approximately 6% to 11% lower average chain cost than the SBNS methods. Except for the greedy method, the experimental results showed that the MBNS methods sped up the running time by approximately 6% to 14% in comparison to the SBNS methods. They showed that the average chain cost of the MBNS methods could be further

improved when the pre-computation concept was considered. However, this pre-computation could affect the average running time of a method. Additionally, this research conducted experiments to compare the pre-computation schemes of the window τ-NAF. The experimental results showed that the pre-computation scheme with the newly proposed $\bar{\tau}P$ formula improved the performance significantly in comparison to other schemes.

8.2 Future Directions

The next step in this research is to continue the work of deriving optimized formulas for different elliptic curves over prime fields. This research studied only Weierstrass curves and twisted Edwards curves and learned many techniques that can be applied to other elliptic curves. First, we need to review the previous studies of formula derivations represented by other elliptic curves. Second, we need to see in which situations these other elliptic curves are useful. Third, we need to apply this research's techniques of deriving optimized formulas to the other elliptic curves. Finally, we need to conduct experiments to see whether the MBNS methods represented by the other elliptic curves speed up scalar multiplication.

Other studies proposed optimized formulas for different elliptic curves over prime fields. Doche, Icart, and Kohel (2006) proposed an elliptic curve with the most efficient $3P$ formula over prime fields. Their $3P$ costs $6\mathbf{M} + 6\mathbf{S}$ when the curve coefficient is selected as a small constant. It saves approximately $0.6\mathbf{M}$ in comparison to $3P$ in twisted Edwards curves. Additionally, Jacobi intersection and Jacobi quartic curves are suggested against side-channel attacks (Liardet & Smart, 2001; Billet & Joye, 2003). Later, Hisil, Carter, and Dawson (2007) and Li et al. (2016) derived formulas represented by these curves. Smart (2001) and Joye and Quisquater (2001) suggested Hessian curves for multiprocessing systems and side-channel attack resistance. Later, Hisil, Wong, Carter, and Dawson (2007) derived formulas represented by Hessian curves. We see that other studies suggested different elliptic curves for different purposes such as for efficiency, side-channel resistance, and multiprocessing systems.

Further research needs to be done on deriving optimized formulas for these curves.

The second future direction of this research is to study the MBNS in controlled environments. Doche (2014) introduced the idea that states that we do not need to convert integer t to its MBNS form, but instead we should generate direct uniform random integer t in its MBNS form. In this research, we assumed that we should convert integer t to its MBNS form as a part of the MBNS methods. Nevertheless, this idea has the potential to significantly speed up scalar multiplication. However, it raises many questions that need answers. How can we generate direct uniform random integer t in its MBNS form? How can we guarantee the optimality of the generated random number in its MBNS form? It is important that the uniform random number in its MBNS form be optimal so that the MBNS methods can achieve the highest possible performance. This will lead us to investigate the question of how we can know if the MBNS form is an optimal. In Section 2.3, we assumed that the rDAG-based method generates an optimal MBNS form because we do not know a method that is better than the rDAG-based method. Therefore, generating a uniform random number in its optimal MBNS form will be one of our future studies.

The third future direction of this research is to investigate how resistant the MBNS is against side-channel attacks. In a side-channel attack, an attacker listens to a channel (e.g., power consumption, noise level, length of time) to predict internal operations of ECC. We know that the SBNS methods without countermeasures are not resistant to side-channel attacks because it is possible for an attacker to predict the secret key by listening to $P + Q$ and $2P$ operations during scalar multiplication (Kocher, Jaffe, & Jun, 1999; Kocher, 1996).

The situation is different in the MBNS methods, so we need to answer the following questions. First, how hard is it for an attacker to predict the secret key by listening to the internal operations (e.g., $P + Q$, $2P$, $3P$, $5P$) of the MBNS methods? Second, which techniques should we use in the MBNS methods to resist side-channel attacks? In SBNS, we use Montgomery ladder, double-and-add always, or unified $P + Q$ formulas to resist simple side-channel attacks (Montgomery, 1987; Coron, 1999; Brier &

Joye, 2002). Third, which elliptic curve is more efficient for the MBNS methods to resist side-channel attacks? Twisted Edwards, Weierstrass, Doche/Icart/Kohel, Jacobi intersection, Jacobi quartic, and Hessian curves have potential for future studies. Fourth, do the MBNS methods with side-channel countermeasures give faster performance than SNBS methods? To answer this question, we need to conduct experiments to see if the MBNS methods that resist side-channel attacks can speed up scalar multiplication over SNBS methods. We see many questions about the MBNS methods regarding their resistance to side-channel attacks that need to be answered. Therefore, we plan to study the MBNS methods that resist side-channel attacks.

The fourth direction of this research is to propose formulas for multiprocessing systems. In this research, we proposed our formulas with the goal of minimizing the number of inversion, multiplication, and squaring operations. We relied on operating systems to utilize the multiprocessing capability. In future studies, we plan to propose algorithms for $P + Q$, $2P$, $3P$, and $5P$ formulas that are capable of working in parallel on multiple processors. Our goal in this new approach is to distribute the operations evenly among processors such that we minimize the number of steps to perform the formulas.

In this research, we stated that Twisted Edwards curves are the most efficient option for serial processors but that they might not be the most efficient option for multiprocessing systems. In fact, Smart (2001) suggested that Hessian curves are efficient options for multiprocessing systems. For example, by using three processors working in parallel, Smart (2001) proposed $P + Q$ and $2P$ formulas represented by Hessian curves with cost $4\mathbf{M}$ and $2\mathbf{M} + 1\mathbf{S}$ respectively. We also need to conduct experiments to verify if these formulas are capable of working on multiprocessing systems to speed up scalar multiplication. Therefore, we will derive optimized formulas for multiprocessing systems in our future studies.

9 References

Ajeena, R. K. K., & Kamarulhaili, H. (2014). Point multiplication using integer sub-decomposition for elliptic curve cryptography. *Applied Math. Info. Sciences*, *8*(2).

Al Musa, S., & Xu, G. (2017). Fast scalar multiplication for elliptic curves over binary fields by efficiently computable formulas. In A. Patra & N. Smart (Eds.), *Proc. INDOCRYPT 2017* (p. 206-226). Cham: Springer.

Aranha, D. F., Barreto, P. S. L. M., Pereira, G. C. C. F., & Ricardini, J. E. (2013). *A note on high-security general-purpose elliptic curves.* Cryptology ePrint Archive, Report 2013/647. (`https://eprint.iacr.org/2013/647`)

Avanzi, R., Cohen, H., Doche, C., Frey, G., Nguyen, K., Lange, T., & Vercauteren, F. (2005). *Handbook of elliptic and hyperelliptic curve cryptography.* Chapman & Hall/CRC.

Avanzi, R., Dimitrov, V., Doche, C., & Sica, F. (2006). Extending scalar multiplication using double bases. In K. Chen (Ed.), *Proc. ASIACRYPT 2006* (p. 130-144). Heidelberg: Springer.

Barker, E. (2013). *Digital signature standard (DSS). FIPS PUB.* Retrieved from `https://www.nist.gov/publications/digital-signature-standard-dss-2`

Bernstein, D., Birkner, P., Joye, M., Lange, T., & Peters, C. (2008). Twisted Edwards curves. In S. Vaudenay (Ed.), *Proc. AFRICACRYPT 2008* (p. 389-405). Heidelberg: Springer.

Bernstein, D., Birkner, P., Lange, T., & Peters, C. (2007). Optimizing double-base elliptic curve single-scalar multiplication. In K. Srinathan, C. Rangan, & M. Yung (Eds.), *Proc. INDOCRYPT 2007* (p. 167-182). Heidelberg: Springer.

Bernstein, D., Chuengsatiansup, C., & Lange, T. (2017). *Double-base scalar multiplication revisited.* Cryptology ePrint Archive, Report 2017/037. (`https://eprint.iacr.org/2017/037`)

Bernstein, D., & Lange, T. (2007a). *Faster addition and doubling on elliptic curves.* Cryptology ePrint Archive, Report 2007/286.

(https://eprint.iacr.org/2007/286)

Bernstein, D., & Lange, T. (2007b). Inverted Edwards coordinates. In S. Boztas & H. Lu (Eds.), *Proc. AAECC 2007* (p. 20-27). Heidelberg: Springer.

Berthe, V., & Imbert, L. (2004). On converting numbers to the double-base number system. *Adv. Signal Pro. Algo. Archit. Implement. XIV.*, *5559*, 70–78.

Billet, O., & Joye, M. (2003). The Jacobi model of an elliptic curve and side-channel analysis. In M. Fossorier, T. Høholdt, & A. Poli (Eds.), *Proc. AAECC 2003* (p. 34-42). Heidelberg: Springer.

Blake, I. F., Murty, V. K., & Xu, G. (2005). A note on window τ-NAF algorithm. *Info. Processing Letters.*, *95*(5), 496-502.

Blake, I. F., Murty, V. K., & Xu, G. (2008). Nonadjacent radix-τ expansions of integers in euclidean imaginary quadratic number fields. *Canadian J. of Math.*, *60*, 1267-1282.

Bos, J. W., Costello, C., Longa, P., & Naehrig, M. (2014). *Selecting elliptic curves for cryptography: An efficiency and security analysis.* Cryptology ePrint Archive, Report 2014/130. (https://eprint.iacr.org/2014/130)

Brier, E., & Joye, M. (2002). Weierstrass elliptic curves and side-channel attacks. In D. Naccache & P. Paillier (Eds.), *Proc. PKC 2002* (p. 335-345). Heidelberg: Springer.

Chudnovsky, D. V., & Chudnovsky, G. V. (1986). Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Advances in Applied Math.*, *7*, 385–434.

Ciet, M., Joye, M., Lauter, K., & Montgomery, P. (2006). Trading inversions for multiplications in elliptic curve cryptography. *Designs, Codes and Cryptography*, *39*(2), 189-206.

Coron, J.-S. (1999). Resistance against differential power analysis for elliptic curve cryptosystems. In C. Koc & C. Paar (Eds.), *Proc. CHES 1999* (p. 292-302). Heidelberg: Springer.

Crosby, M., Nachiappan, Pattanayak, P., Verma, S., & Kalyanaraman, V. (2016).

Blockchain technology: Beyond bitcoin. Retrieved from `http://scet.berkeley.edu/wp-content/uploads/AIR-2016-Blockchain.pdf`

Diffie, W., & Hellman, M. (1976). New directions in cryptography. *IEEE Information Theory Society*, *22*, 644-654.

Dimitrov, V., Eskritt, J., Imbert, L., Jullien, G., & Miller, W. (2001). The use of the multi-dimensional logarithmic number system in DSP applications. In *Proc. 15th IEEE Symposium on Computer Arithmetic.* USA: IEEE.

Dimitrov, V., Imbert, L., & Mishra, P. (2005). Efficient and secure elliptic curve point multiplication using double-base chains. In B. Roy (Ed.), *Proc. ASIACRYPT 2005* (p. 59-78). Heidelberg: Springer.

Dimitrov, V., Imbert, L., & Mishra, P. (2008). The double-base number system and its application to elliptic curve cryptography. *Mathematics of Computation*, *77*(262), 1075-1104.

Dimitrov, V., Jullien, G., & Miller, W. (1998). An algorithm for modular exponentiation. *Information Processing Letters*, *66*, 155-159.

Doche, C. (2014). On the enumeration of double-base chains with applications to elliptic curve cryptography. In P. Sarkar & T. Iwata (Eds.), *Proc. ASIACRYPT 2014* (p. 297-316). Heidelberg: Springer.

Doche, C., & Habsieger, L. (2008). A tree-based approach for computing double-base chains. In Y. Mu, W. Susilo, & J. Seberry (Eds.), *Proc. ACISP 2008* (p. 433-446). Heidelberg: Springer.

Doche, C., Icart, T., & Kohel, D. R. (2006). Efficient scalar multiplication by isogeny decompositions. In M. Yung, Y. Dodis, A. Kiayias, & T. Malkin (Eds.), *Proc. PKC 2006* (p. 191-206). Heidelberg: Springer.

Doche, C., & Imbert, L. (2006). Extended double-base number system with applications to elliptic curve cryptography. In R. Barua & T. Lange (Eds.), *Proc. INDOCRYPT 2006* (p. 335-348). Heidelberg: Springer.

Doche, C., Kohel, D., & Sica, F. (2009). Double-base number system for multi-scalar multiplications. In A. Joux (Ed.), *Proc. EUROCRYPT 2009* (p. 502-517).

Heidelberg: Springer.

Dworkin, M. J. (2015). *SHA-3 standard: Permutation-based hash and extendable-output functions. FIPS PUB.* Retrieved from `https://www.nist.gov/publications/`

Dworkin, M. J., Barker, E. B., Nechvatal, J. R., Foti, J., Bassham, L. E., Roback, E., & Jr., J. F. D. (2011). *Advanced encryption standard (AES). FIPS PUB.* Retrieved from `https://www.nist.gov/publications/advanced-encryption-standard-aes`

Gallant, R., Lambert, R., & Vanstone, S. (2001). Faster point multiplication on elliptic curves with efficient endomorphisms. In J. Kilian (Ed.), *Proc. CRYPTO 2001* (p. 190-200). Heidelberg: Springer.

Giorgi, P., Imbert, L., & Izard, T. (2009). Optimizing elliptic curve scalar multiplication for small scalars. *Mathematics for Signal and Information Processing*, *7444*.

Granger, R., Kleinjung, T., & Zumbrägel, J. (2014). *Breaking '128-bit secure' supersingular binary curves.* Cryptology ePrint Archive, Report 2014/119. (`https://eprint.iacr.org/2014/119`)

Granlund, T., & et al. (n.d.). *GNU multiple precision arithmetic library.* Retrieved from `http://www.gmplib.org`

Hamburg, M. (2015). *Ed448-Goldilocks, a new elliptic curve.* Cryptology ePrint Archive, Report 2015/625. (`https://eprint.iacr.org/2015/625`)

Hankerson, D., Menezes, A., & Vanstone, S. (2004). *Guide to elliptic curve cryptography.* Springer-Verlag.

He, D., & Zeadally, S. (2014). An analysis of RFID authentication schemes for internet of things in healthcare environment using elliptic curve cryptography. *IEEE Internet of Things Journal*, *2*, 72-83.

Hisil, H., Carter, G., & Dawson, E. (2007). New formulae for efficient elliptic curve arithmetic. In S. K., R. C.P., & Y. M. (Eds.), *Proc. INDOCRYPT 2007* (p. 138-151). Heidelberg: Springer.

Hisil, H., Wong, K. K.-H., Carter, G., & Dawson, E. (2007). *Faster group operations on elliptic curves.* Cryptology ePrint Archive, Report 2007/441.

(https://eprint.iacr.org/2007/441)

Johnson, D., Menezes, A., & Vanstone, S. (2001). The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, *1*, 36-63.

Josefsson, S., & Liusvaara, I. (2017). *RFC8032: Edwards-curve digital signature algorithm (EdDSA).* Retrieved from https://tools.ietf.org/html/rfc8032

Joye, M., & Quisquater, J.-J. (2001). Hessian elliptic curves and side-channel attacks. In C. Koc, D. Naccache, & C. Paar (Eds.), *Proc. CHES 2001* (p. 402-410). Heidelberg: Springer.

Kim, D., & Lim, S. (2002). Integer decomposition for fast scalar multiplication on elliptic curves. In K. Nyberg & H. Heys (Eds.), *Proc. SAC 2002* (p. 13-20). Heidelberg: Springer.

Koblitz, N. (1987). Elliptic curve cryptosystems. *Math. of Computation*, *48*(177), 203-209.

Koblitz, N. (1992). CM-curves with good cryptographic properties. In J. Feigenbaum (Ed.), *Proc. CRYPTO 1991* (p. 279-287). Heidelberg: Springer.

Koblitz, N., & Menezes, A. (2015). *A riddle wrapped in an enigma.* Cryptology ePrint Archive, Report 2015/1018. (https://eprint.iacr.org/2015/1018)

Kocher, P. (1996). Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz (Ed.), *Proc. CRYPTO 96* (p. 104-113). Heidelberg: Springer.

Kocher, P., Jaffe, J., & Jun, B. (1999). Differential power analysis. In M. Wiener (Ed.), *Proc. CRYPTO 99* (p. 388–397). Heidelberg: Springer.

Kohel, D. (2017). *Twisted μ_4-normal form for elliptic curve.* Cryptology ePrint Archive, Report 2017/121. (https://eprint.iacr.org/2017/121)

Lange, T. (2004). *A note on Lopez-Dahab coordinates.* Cryptology ePrint Archive, Report 2004/323. (https://eprint.iacr.org/2004/323)

Le, D. (2011). *Fast quadrupling of a point in elliptic curve cryptography.* Cryptology ePrint Archive, Report 2011/039. (https://eprint.iacr.org/2011/039)

Lenstra, A., & Verheul, E. (2001). Selecting cryptographic key sizes. *Journal of*

Cryptology, *14*, 255-293.

Li, W., Yu, W., & Wang, K. (2016). Improved tripling on elliptic curves. In D. Lin, X. Wang, & M. Yung (Eds.), *Proc. Inscrypt 2015* (p. 193-205). Cham: Springer.

Liardet, P., & Smart, N. (2001). Preventing SPA/DPA in ECC systems using the Jacobi form. In C. Koc, D. Naccache, & C. Paar (Eds.), *Proc. CHES 2001* (p. 391-401). Heidelberg: Springer.

Longa, P., & Gebotys, C. (2009). Fast multibase methods and other several optimizations for elliptic curve scalar multiplication. In S. Jarecki & G. Tsudik (Eds.), *Proc. PKC 2009* (p. 443-462). Heidelberg: Springer.

Longa, P., & Miri, A. (2008a). Fast and flexible elliptic curve point arithmetic over prime fields. *IEEE Transactions on Computers*, *57*, 289-302.

Longa, P., & Miri, A. (2008b). New composite operations and precomputation scheme for elliptic curve cryptosystems over prime fields. In R. Cramer (Ed.), *Proc. PKC 2008* (p. 229-247). Heidelberg: Springer.

Longa, P., & Miri, A. (2008c). *New multibase non-adjacent form scalar multiplication and its application to elliptic curve cryptosystems.* Cryptology ePrint Archive, Report 2008/052. (`https://eprint.iacr.org/2008/052`)

Lopez, J., & Dahab, R. (1999). Improved algorithms for elliptic curve arithmetic in GF(2^n). In S. Tavares & H. Meijer (Eds.), *Proc. SAC 1998* (p. 201–212). Heidelberg: Springer.

Matula, D., & Kornerup, P. (n.d.). *Multiprecision integer and rational arithmetic c library.* Retrieved from `http://www.miracl.com`

Miller, V. (1986). Use of elliptic curves in cryptography. In H. Williams (Ed.), *Proc. CRYPTO 1985* (p. 417-426). Heidelberg: Springer.

Mishra, P., & Dimitrov, V. (2007). Efficient quintuple formulas for elliptic curves and efficient scalar multiplication using multibase number representation. In J. Garay, A. Lenstra, M. Mambo, & R. Peralta (Eds.), *Proc. ISC 2007* (p. 390-406). Heidelberg: Springer.

Montgomery, P. L. (1987). Speeding the Pollard and elliptic curve methods of

factorization. *Mathematics of Computation*, *48*, 243–264.

Nakamoto, S. (2011). *Bitcoin: A peer-to-peer electronic cash system.* Retrieved from `https://bitcoin.org/bitcoin.pdf`

Oliveira, T., Lopez, J., Aranha, D., & Rodriguez-Henriquez, F. (2014). Two is the fastest prime: lambda coordinates for binary elliptic curves. *Journal of Cryptographic Engineering*, *4*(1), 3-17.

Pollard, J. M. (1975). A monte carlo method for factorization. *BIT Numerical Mathematics*, *15*, 331–334.

Rescorla, E. (2018). *RFC8446: The transport layer security (TLS) protocol version 1.3.*

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21*(2).

Santoso, F., & Vun, N. (2015). Securing iot for smart home system. In *Proc. ISCE 2015* (p. 1-2). Spain: IEEE.

Smart, N. (2001). The Hessian form of an elliptic curve. In C. Koc, D. Naccache, & C. Paar (Eds.), *Proc. CHES 2001* (p. 118-125). Heidelberg: Springer.

Solinas, J. (2000). Efficient arithmetic on Koblitz curves. *Designs, Codes and Cryptography*, *19*(23), 195-249.

Stebila, D., & Green, J. (2009). *RFC5656: Elliptic curve algorithm integration in the secure shell transport layer.* Retrieved from `https://tools.ietf.org/html/rfc5656`

Subramanya Rao, S. (2016). Three dimensional Montgomery ladder, differential point tripling on Montgomery curves and point quintupling on Weierstrass' and Edwards curves. In D. Pointcheval, A. Nitaj, & T. Rachidi (Eds.), *Proc. AFRICACRYPT 2016* (p. 84-106). Cham: Springer.

Takagi, T., Reis, D., Yen, S.-M., & Wu, B.-C. (2006). Radix-r nonadjacent form and its application to pairing-based cryptosystem. *IEICE Transactions*, *E89-A*, 115-123.

Trost, W., & Xu, G. (2016). On the optimal pre-computation of window tNAF for Koblitz curves. *IEEE Transactions on Computers*, *65*, 29195-249.

Watson, L. (2018). *Blockchain: The ultimate guide to understanding the technology*

behind bitcoin and cryptocurrency. CreateSpace Independent Publishing Platform.

Xu, G. (2009, 01). Short vectors, the GLV method and discrete logarithms. , *45*.

Yasin, S., & Muda, Z. (2015). Tripling formulae of elliptic curve over binary field in Lopez-Dahab model. *Journal of Theoretical and Applied Information Technology*, *75*(2).

Yu, W., Al Musa, S., Xu, G., & Li, B. (2018). *A novel pre-computation scheme of window τNAF for Koblitz curves.* Cryptology ePrint Archive, Report 2017/1020. (`https://eprint.iacr.org/2017/1020`)

Yu, W., Kim, K., & Jo, M. (2015). New fast algorithms for elliptic curve arithmetic in affine coordinates. In K. Tanaka & Y. Suga (Eds.), *Proc. IWSEC 2015* (p. 56-64). Cham: Springer.

Yu, W., Wang, K., Li, B., & Tian, S. (2013). Triple-base number system for scalar multiplication. In A. Youssef, A. Nitaj, & A. Hassanien (Eds.), *Proc. AFRICACRYPT 2013* (p. 433-451). Heidelberg: Springer.

Zhu, L., Jaganathan, K., & Lauter, K. (2008). *RFC5349: Elliptic curve cryptography (ECC) support for public key cryptography for initial authentication in Kerberos (PKINIT).* Retrieved from `https://www.ietf.org/rfc/rfc5349.txt`

10 Appendix: Algorithms

Algorithm 10.1 shows that point tripling (**TPL**) in standard twisted Edwards coordinates needs 2 temporary variables. This $3P$ formula is shown in Table 3.16.

Algorithm 10.1 TPL in Standard Twisted Edwards Coordinates

Input: $P = (X, Y, Z)$
Output: $3P = (X_3, Y_3, Z_3)$

$X_3 = Y^2$	
$Y_3 = X^2$	
$Y_3 = a \cdot X_3$	
$Z_3 = X_3 + Y_3$	$\{Z_3 = Y^2 + aX_3 = T\}$
$T_1 = X_3 - Y_3$	$\{T_1 = \bar{T}\}$
$T_1 = T_1 \cdot Z_3$	$\{T_1 = T\,\bar{T}\}$
$T_2 = Z^2$	
$T_2 = T_2 + T_2$	
$T_2 = Z_3 - T_2$	$\{T_2 = T - 2Z^2\}$
$X_3 = X_3 + X_3$	
$X_3 = X_3 \cdot T_2$	$\{X_3 = 2Y^2(T - 2Z^2)\}$
$Y_3 = Y_3 + Y_3$	
$Y_3 = Y_3 \cdot T_2$	$\{Y_3 = 2aX^2(T - 2Z^2)\}$
$Z_3 = T_1 + X_3$	$\{Z_3 = T\bar{T} + 2Y^2(T - 2Z^2) = A\}$
$X_3 = T_1 - X_3$	$\{X_3 = \bar{A}\}$
$X_3 = X_3 \cdot Z_3$	$\{X_3 = A\bar{A}\}$
$T_2 = T_1 - Y_3$	$\{T_2 = T\bar{T} - 2aX^2(T - 2Z^2) = B\}$
$Y_3 = T_1 + Y_3$	$\{Y_3 = \bar{B}\}$
$Y_3 = Y_3 \cdot T_2$	$\{Y_3 = B\bar{B}\}$
$Z_3 = Z_3 \cdot T_2$	$\{Z_3 = AB\}$
$Z_3 = Z \cdot Z_3$	$\{Z_3 = ZAB\}$
$X_3 = X \cdot X_3$	$\{X_3 = XA\bar{A}\}$
$Y_3 = Y \cdot Y_3$	
$Y_3 = -Y_3$	$\{Y_3 = -YB\bar{B}\}$

return: (X_3, Y_3, Z_3)

Algorithm 10.2 shows that point quintupling (**QPL**) in standard twisted Edwards coordinates needs 2 temporary variables. This $5P$ formula is shown in Table 3.17.

Algorithm 10.2 QPL in Standard Twisted Edwards Coordinates

Input: $P = (X, Y, Z)$
Output: $5P = (X_5, Y_5, Z_5)$

$Z_5 = Y^2$	
$Y_5 = X^2$	
$Y_5 = a \cdot Y_5$	
$X_5 = Z_5 + Y_5$	$\{X_5 = Y^2 + aX^2 = T\}$
$T_1 = Z^2$	
$T_1 = T_1 + T_1$	
$T_1 = X_5 - T_1$	$\{T_1 = T - 2Z^2\}$
$Z_5 = Z_5 + Z_5$	$\{Z_5 = 2Y^2\}$
$Y_5 = Y_5 + Y_5$	$\{Y_5 = 2aX^2\}$
$Z_5 = Z_5 \cdot T_1$	$\{Z_5 = 2Y^2(T - 2Z^2)\}$
$T_1 = Y_5 \cdot T_1$	$\{T_1 = 2aX^2(T - 2Z^2)\}$
$Y_5 = X_5 - Y_5$	$\{Y_5 = T - aX^2 = \bar{T}\}$
$Y_5 = Y_5 \cdot X_5$	$\{Y_5 = T\ \bar{T}\}$
$X_5 = Y_5 + Z_5$	$\{X_5 = T\bar{T} + 2Y^2(T - 2Z^2) = A\}$
$T_2 = Y_5 - Z_5$	$\{T_2 = \bar{A}\}$
$X_5 = X_5 \cdot T_2$	$\{X_5 = A\bar{A}\}$
$T_2 = Y_5 - T_1$	$\{T_2 = T\bar{T} - 2aX^2(T - 2Z^2) = B\}$
$Y_5 = Y_5 + T_1$	$\{Y_5 = \bar{B}\}$
$Y_5 = T_2 \cdot Y_5$	$\{Y_5 = B\bar{B}\}$
$T_2 = T_2 + T_1$	$\{T_2 = T\bar{T}\}$
$T_1 = T_1 \cdot X_5$	$\{T_1 = 2aX^2(T - 2Z^2)A\bar{A}\}$
$Z_5 = Z_5 \cdot Y_5$	$\{Z_5 = 2Y^2(T - 2Z^2)B\bar{B}\}$
$X_5 = T_2 \cdot X_5$	
$X_5 = -X_5$	$\{X_5 = -T\bar{T}\ A\bar{A}\}$
$Y_5 = T_2 \cdot Y_5$	$\{Y_5 = T\bar{T}\ B\bar{B}\}$
$T_2 = X_5 + Z_5$	$\{T_2 = -T\bar{T}\ A\bar{A} + 2Y^2(T - 2Z^2)B\bar{B} = C\}$
$X_5 = X_5 - Z_5$	
$X_5 = T_2 \cdot X_5$	
$X_5 = X \cdot X_5$	$\{X_5 = X\ C\bar{C}\}$
$Z_5 = Y_5 + T_1$	$\{Z_5 = T\bar{T}\ B\bar{B} + 2aX^2(T - 2Z^2)A\bar{A} = D\}$
$Y_5 = Y_5 - T_1$	
$Y_5 = Z_5 \cdot Y_5$	
$Y_5 = Y \cdot Y_5$	$\{Y_5 = Y\ D\bar{D}\}$
$Z_5 = T_2 \cdot Z_5$	
$Z_5 = Z \cdot Z_5$	$\{Z_5 = ZCD\}$

return: (X_5, Y_5, Z_5)

Algorithm 10.3 shows that **TPL** in λ-coordinates needs 3 temporary variables. This $3P$ formula is shown in Table 4.10.

Algorithm 10.3 TPL in λ-coordinates over $\mathbb{F}_{2^m}$

Input: $P = (X, L, Z)$
Output: $3P = (X_3, L_3, Z_3)$

$L_3 = L \cdot Z$	
$T_1 = Z^2$	
$X_3 = a \cdot T_1$	$\{X_3 = aZ^2\}$
$Z_3 = L^2$	
$Z_3 = Z_3 + L_3$	
$Z_3 = Z_3 + X_3$	$\{Z_3 = L^2 + LZ + aZ^2 = T\}$
$X_3 = Z_3 \cdot T_1$	$\{X_3 = TZ^2\}$
$T_2 = X_3^2$	
$T_2 = Z_3 \cdot T_2$	$\{T(A+B)^2\}$
$T_3 = X \cdot Z$	
$Z_3 = Z_3 + T_3$	
$Z_3 = Z_3^2$	$\{Z_3 = (T + XZ)^2 = A\}$
$X_3 = X_3 + Z_3$	$\{X_3 = TZ^2 + A = B\}$
$Z_3 = Z_3 \cdot X_3$	$\{Z_3 = AB\}$
$X_3 = X_3^2$	
$X_3 = T_3 \cdot X_3$	$\{X_3 = XZB^2\}$
$L_3 = L_3 + T_1$	$\{L_3 = LZ + Z^2\}$
$L_3 = L_3 \cdot Z_3$	$\{L_3 = (LZ + Z^2)AB\}$
$L_3 = T_2 + L_3$	$\{L_3 = T(A+B)^2 + (LZ + Z^2)AB\}$
$Z_3 = T_1 \cdot Z_3$	$\{Z_3 = Z^2AB\}$

return: (X_3, L_3, Z_3)

Algorithm 10.4 shows that **QPL** in λ-coordinates needs 5 temporary variables. This $5P$ formula is shown in Table 4.11.

Algorithm 10.4 QPL in λ-coordinates over $\mathbb{F}_{2^m}$

Input: $P = (X, Y, Z)$
Output: $5P = (X_5, L_5, Z_5)$

$X_5 = L^2$	
$L_5 = L \cdot Z$	
$Z_5 = Z^2$	
$T_1 = a \cdot Z_5$	$\{T_1 = aZ^2\}$
$T_1 = X_5 + T_1$	$\{T_1 = L^2 + aZ^2\}$
$T_1 = L_5 + T_1$	$\{T_1 = LZ + L^2 + aZ^2 = T\}$
$L_5 = L_5 + Z_5$	$\{L_5 = LZ + Z^2\}$
$X_5 = X \cdot Z$	$\{X_5 = XZ\}$
$T_2 = T_1 + X_5$	
$T_2 = T_2^2$	$\{T_2 = (T + XZ)^2 = A\}$
$T_3 = T_1 \cdot Z_5$	$\{T_3 = TZ^2\}$
$T_4 = T_3 \cdot T_1$	
$T_4 = T_4^2$	$\{T_4 = (T(A + B))^2\}$
$T_3 = T_3 + T_2$	$\{T_3 = TZ^2 + A = B\}$
$T_5 = T_3^2$	
$T_5 = T_2 \cdot T_5$	$\{T_5 = AB^2\}$
$T_4 = T_4 + T_5$	$\{T_4 = (T(A + B))^2 + AB^2 = C\}$
$T_2 = T_2^2$	
$T_2 = T_2 \cdot T_3$	$\{T_2 = A^2B\}$
$T_3 = T_2 + T_5$	$\{T_3 = A^2B + AB^2\}$
$T_2 = T_2 \cdot T_5$	$\{T_2 = A^2BAB^2\}$
$T_5 = T_3 + T_4$	$\{T_5 = A^2B + AB^2 + C = D\}$
$T_3 = T_3^2$	
$T_3 = T_3 \cdot T_1$	$\{T_3 = T(C + D)^2\}$
$T_4 = T_4 \cdot T_5$	$\{T_4 = CD\}$
$L_5 = L_5 \cdot T_4$	
$L_5 = T_3 + L_5$	$\{L_5 = T(C + D)^2 + (LZ + Z^2)CD\}$
$T_2 = Z_5 \cdot T_2$	$\{T_2 = Z^2A^2BAB^2\}$
$L_5 = L_5 + T_2$	$\{L_5 = T(C + D)^2 + (LZ + Z^2)CD + Z^2A^2BAB^2\}$
$Z_5 = Z_5 \cdot T_4$	$\{Z_5 = Z^2CD\}$
$T_5 = T_5^2$	
$X_5 = X_5 \cdot T_5$	$\{XZD^2\}$

return: (X_5, L_5, Z_5)

11 Curriculum Vitae

Personal Information

Name:	Saud Al Musa
Place of birth:	Saudi Arabia

Education

Dec 2018	University of Wisconsin Milwaukee (UWM). U.S.A.
	Graduated with doctoral's degree in computer science.
Aug 2011	Middle Tennessee State University (MTSU). U.S.A
	Graduated with master's degree in computer science.
Dec 2005	King Saud University (KSU). Saudi Arabia.
	Graduated with bachelor's degree in computer science.

Work Experience

Sept 2012 to Present	Taibah University, Saudi Arabia.
	Working as a Teaching Assistant.
Jan 2006 to Mar 2008	CERT-SA, Saudi Arabia.
	Worked as an Information Security Engineer.

Certification

Jan 2012	Cisco Certified Network Associate (CCNA).
Dec 2011	CompTIA Security+.
Mar 2008	Certified Information Systems Security Professional (CISSP).
Aug 2007	Certified Ethical Hacker.

Publication

- Al Musa, S., & Xu, G. (2018). *Fast scalar multiplication for elliptic curves over prime fields by efficiently computable formulas.* Cryptology ePrint Archive, Report 2018/964. (https://eprint.iacr.org/2018/964) [In submission].
- Yu, W., Al Musa, S., Xu, G., & Li, B. (2018). *A novel pre-computation scheme of window τNAF for Koblitz curves.* Cryptology ePrint Archive, Report 2017/1020. (https://eprint.iacr.org/2017/1020) [In submission].
- Al Musa, S., & Xu, G. (2017). *Fast scalar multiplication for elliptic curves over binary fields by efficiently computable formulas.* In A. Patra & N. Smart (Eds.), *Proc. INDOCRYPT 2017* (p. 206-226). Cham: Springer.