Fast Scalar Multiplication for Elliptic Curves over Prime Fields by Efficiently Computable Formulas

Saud Al Musa and Guangwu Xu *

Department of EE & CS,
University of Wisconsin-Milwaukee, USA,
{salmusa,gxu4uwm}@uwm.edu

Abstract. This paper addresses fast scalar multiplication for elliptic curves over finite fields. In the first part of the paper, we obtain several efficiently computable formulas for basic elliptic curves arithmetic in the family of twisted Edwards curves over prime fields. Our $2Q + P$ formula saves about 2.8 field multiplications, and our $5P$ formula saves about 4.2 field multiplications in standard projective coordinate systems, compared to the latest existing results. In the second part of the paper, we formulate bucket methods for the DAG-based and the tree-based abstract ideas. We propose systematically finding a near optimal chain for multi-base number systems (MBNS). These proposed bucket methods take significantly less time to find a near optimal chain, compared to an optimal chain. We conducted extensive experiments to compare the performance of the MBNS methods (e.g., greedy, ternary/binary, multi-base NAF, tree-based, rDAG-based, and bucket). Our proposed formulas were integrated in these methods. Our results show our work had an important role in advancing the efficiency of scalar multiplication.

Keywords: twisted Edwards curves, Edwards curves, scalar multiplication, efficient formulas, DBNS, MBNS

1 Introduction

Elliptic curve cryptography (ECC) is a type of public key cryptography that was initially introduced by Koblitz and Miller [22, 30]. ECC has an efficiency advantage over other public key cryptographies. For instance, a key length of 283-bit in ECC is regarded as secure as 3072-bit in RSA public key cryptography [25]. Scalar multiplication (tP) is the most expensive ECC operation that is extensively used in cryptographic protocols. It is an operation that adds P to itself t times such that $tP = \underbrace{P + P + ... + P}_{t\ times}$ where P is a point on an elliptic

* Research supported in part by the National Key Research and Development Program of China (No. 2017YFA0303903).

curve over finite fields and t is a large positive integer. Scalar multiplication efficiency in our framework relies on the method (e.g., binary, NAF) and the cost of formulas (e.g., point doubling ($2P$), point addition ($P + Q$)). The cost of $2P$ and $P + Q$ formulas varies in different coordinate systems over different finite fields.

We express the cost of formulas by the number of field inversions ($\mathbf{I}$), multiplications ($\mathbf{M}$), and squarings ($\mathbf{S}$). We discuss a class of specific curves over prime fields called twisted Edwards curves $E_{a,d}$, as represented by Equation (1) and Equation (3). We express the notations $\mathbf{M}_a$ or $\mathbf{M}_d$ to represent field multiplication with one of the multiplicands to be twisted Edwards curves coefficient a or d. Addition/subtraction operations are ignored since they are cheap operations over both prime and binary fields. We express the conjugate of T by $\bar{T}$. To obtain the conjugate of a binomial, we need to change the sign between the two terms. For example, let $T = Y + X$. Then $\bar{T} = Y - X$.

We use the squaring to multiplication ($\mathbf{S/M}$) ratio to measure the squaring cost. This is because the $\mathbf{S/M}$ ratio varies in different devices over different finite fields. The cost of one squaring operation is less than or equal to one multiplication operation. For prime fields, the $\mathbf{S/M}$ 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 [19, 8]. To simplify the comparison over prime fields, we assume $1\mathbf{S} = 0.8\mathbf{M}$, $\mathbf{M}_d = 1\mathbf{M}$, and $\mathbf{M}_a$ is a free operation as will be seen later, a is usually a small number. An inversion operation is more expensive than a multiplication operation over both binary and prime fields. We use the inversion to multiplication ($\mathbf{I/M}$) ratio to measure the inversion cost. The $\mathbf{I/M}$ ratio also varies in different devices over different finite fields [19, 8].

One technique to speed up scalar multiplication is to use projective coordinate systems. When we use projective coordinates, we avoid inversion operations completely. First, we convert an affine point to a projective point. Then, we perform scalar multiplication without any inversion operations. Lastly, we reverse a projective point to an affine point. The cost of converting an affine point to a projective point or vice versa is very minor in comparison to the cost of a scalar multiplication operation. As a result, the number of operations can be significantly reduced, especially for a high $\mathbf{I/M}$ ratio device. For this reason, numerous studies proposed projective coordinate systems for different elliptic curves [2, 5, 33, 24, 23, 20]. For Weierstrass curves over prime fields, Jacobian coordinates in our framework are the most efficient projective coordinates [20]. For twisted Edwards curves over prime fields, standard projective coordinates in our framework are the most efficient projective coordinates [2].

Another technique to speed up scalar multiplication is to use double-base number system (DBNS) forms [13]. 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 integer t is represented in the DBNS, the form length on average becomes shorter than when it is represented in the single-base number system. This minimizes the number of point additions and speeds up scalar

multiplication since point addition is an expensive operation in comparison to other formulas. Multi-base number system (MBNS) is a natural extension of the DBNS [31]. When integer t is represented in the MBNS, the number of point additions continues to minimize.

The MBNS requires optimized point tripling ($3P$) and point quintupling ($5P$) formulas to be fast and practical. Therefore, numerous studies proposed optimized $3P$ and $5P$ formulas in different elliptic curves over different finite fields. The state of the art $3P$ and $5P$ formulas are described as follows. For Weierstrass curves, Yu, Kim, and Jo derived $3P$ formulas in affine coordinates over both binary and prime fields [36]. Longa and Miri derived a $3P$ formula and Giorgi, Imbert, and Izard derived a $5P$ formula in Jacobian projective coordinates over prime fields [28, 17]. Al Musa and Xu derived $3P$ and $5P$ formulas in λ-projective coordinates over binary fields [1]. For twisted Edwards curves over prime fields, Bernstein, Chuengsatiansup, and Lange derived a $3P$ formula and Li, Yu, and Wang derived a $5P$ formula in standard projective coordinates [4, 26].

For efficiency reasons, when integer t is represented in the DBNS, it has to be represented as a double-base chain. In a double-base chain, the sequence of the exponents a_i and b_i decreases. Therefore, other studies proposed methods that convert integer t to a double-base chain. Dimitrov, Imbert, and Mishra proposed the greedy method [11]. Ciet, Joye, Lauter, and Montgomery proposed the ternary/binary method [9]. Longa and Gebotys proposed the multi-base NAF method [27]. Doche and Habsieger proposed the tree-based method [15]. Bernstein et al. proposed the rDAG-based method [4]. These methods can be extended to convert integer t to a multi-base chain [31, 37].

1.1 Our Contribution

In this paper, we consider the efficiency problem of scalar multiplication for elliptic curve cryptography and present three main contributions. The first part of our contribution is to derive formulas for twisted Edwards curves. We derive $2Q + P$ and 5P formulas in standard twisted Edwards coordinate systems over prime fields. To the best of our knowledge, our $2Q + P$ is the first proposed dedicated formula for twisted Edwards curves. 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 formula. Our $5P$ formula costs $15\mathbf{M} + 1\mathbf{M}_a + 3\mathbf{S} \approx 17.4\mathbf{M}$. It saves $4.2\mathbf{M}$ in comparison to the state of the art $5P$ formula [26]. It saves approximately $2\mathbf{M}$ in comparison to $5P$ in Jacobian Weierstrass ($a = -3$) coordinates. We also improve the $P + Q$ formula by using the pre-computation concept. This proposed $P + Q$ formula with pre-computation saves $1\mathbf{M}_d + 1\mathbf{M}_a \approx 1\mathbf{M}$ over the $P + Q$ formula without pre-computation.

The second part of our contribution is to advance the MBNS methods. We formulate bucket methods for the DAG-based and the tree-based abstract ideas. The bucket methods provide a systematic way to balance the chain quality and the time to find the chain. They find a near optimal chain for integer t in significantly less time than finding an optimal chain. To the best of our knowledge, we are the first study that proposes systematically finding a near optimal chain for

the MBNS methods. Moreover, these proposed methods show that the tree-based method is far from producing a near optimal chain.

The third part of our contribution is to conduct experimental comparison between the MBNS and the NAF methods. We utilize our formulas in these MBNS methods. The investigated MBNS methods are greedy, ternary/binary, multi-base NAF, tree-based, rDAG-based, and bucket methods. We use three types of measurements to compare the methods: the chain length, the chain cost, and the running time. The running time takes into consideration the conversion cost. Other experimental results in [4, 9, 11, 15, 27] were conducted without considering the running converting time. Our experimental results show that the MBNS methods without pre-computation had an approximately 12% to 16% lower average chain cost in comparison to the NAF method. Except for the greedy method, they gave an approximately 12% to 18% faster average running time. They also show that the chain cost could be further improved if the MBNS methods with pre-computation were considered. However, this pre-computation could lead to extra time to find the chain.

1.2 ECC Applications

ECC is appropriate for an application that requires higher security and efficiency. One application for ECC is the Internet of Things (IoT). IoT can be seen as many embedded systems and sensors connected through an insecure channel. These embedded systems and sensors have restricted resource capabilities. For example, He and Zeadally suggested ECC authentication schemes for radio-frequency identification (RFID) chips [21]. This is because ECC is more efficient for the computing resources than other public-key cryptographic systems. Another practical application for ECC is within a blockchain. The blockchain is formed from distributed systems that work together to verify transactions [10]. For example, Bitcoin blockchain uses ECC digital signature schemes to verify digital currency transactions [32]. In addtion, computer network protocols use ECC in key agreement and digital signature schemes. For example, the latest version of the Transport Layer Security (TLS 1.3) protocol includes ECC in the supported cipher suite list [34].

The rest of the paper is organized into five sections. The proposed $2Q+P$ and $5P$ formulas are presented in Section 2. In Section 3, we introduce the MBNS and proposed the bucket methods. In Section 4, we conducted experiments to compare the performance of the MBNS methods with and without pre-computation. Finally, we conclude our work in Section 5.

2 Twisted Edwards Curves

Bernstein, Birkner, Joye, Lange, and Peters introduced twisted Edwards curves over prime fields $\mathbb{F}_p$ [2]. Twisted Edwards curves $E_{a,d}$ are represented by

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

where $a, d \in \mathbb{F}_p$, $p > 3$ is a prime number, $a \neq d$ and $a, d \neq 0$. The denotation $E_{a,d}(\mathbb{F}_p)$ is the set of all points (x, y) where $x, y \in \mathbb{F}_p$ that satisfies the above equation. $E_{a,d}(\mathbb{F}_p)$ is an abelian group under the point addition operation. The identity point for the group is the point $(0, 1)$. The negative of point $P = (x, y) \in E_{a,d}(\mathbb{F}_p)$ is another point $-P = (-x, y) \in E_{a,d}(\mathbb{F}_p)$. $E_{a,d}$ 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

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

Edwards curves are a special class of twisted Edwards curves. Edwards curves E_d are represented by

$$E_d : x^2 + y^2 = 1 + dx^2y^2 \tag{2}$$

where $d \in \mathbb{F}_p$ and $d \neq \{0, 1\}$. In other words, Edwards curves are twisted Edwards curves with the coefficient $a = 1$ [6, 5].

2.1 Standard Projective Coordinates

Bernstein et al. also introduced two main representations for twisted Edwards curves in projective coordinates: standard twisted Edwards coordinates and inverted twisted Edwards coordinates [2]. A projective point in standard twisted Edwards coordinates is represented as (X, Y, Z). An affine point can be converted to projective point by using the relation $(X, Y, Z) = (x, y, 1)$. A projective point can be converted to an affine point by using the relation $(x, y) = (\frac{X}{Z}, \frac{Y}{Z})$ where $Z \neq 0$. The curve equation in standard twisted Edwards coordinates is represented by

$$(aX^2 + Y^2)Z^2 = Z^4 + dX^2Y^2. \tag{3}$$

$2P$ in standard twisted Edwards coordinates saves $1\mathbf{M}_d$ in comparison to $2P$ in inverted twisted Edwards coordinates. It saves approximately $0.8\mathbf{M}$ in comparison to $2P$ in Jacobian Weierstrass coordinates ($a = -3$), as Table 1 shows. In our framework of scalar multiplication methods, $2P$ is used more frequently than other formulas (e.g., $P + Q$, $3P$). Therefore, $2P$ has a greater impact on the performance than other formulas. As a result, more efforts were made to derive efficient formulas in standard twisted Edwards coordinates and this is the reason we derive efficient formulas in standard twisted Edwards coordinates.

2.2 A Proposed $P + Q$ Formula

In standard twisted Edwards coordinates, the cost of $P + Q$ is $10\mathbf{M} + 1\mathbf{M}_d + 1\mathbf{M}_a + 1\mathbf{S} \approx 11.8\mathbf{M}$ [2]. It costs an extra $1\mathbf{M}$ in comparison to $P + Q$ in inverted twisted Edwards coordinates. It costs approximately an extra $0.6\mathbf{M}$ in

Table 1. 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)	$7\mathbf{M} + 4\mathbf{S} \approx 10.2\mathbf{M}$ [28]
$P + Q$	$10\mathbf{M} + 1\mathbf{S} \approx 10.8\mathbf{M}$ (this work)	$11\mathbf{M} + 5\mathbf{S} \approx 15\mathbf{M}$ [28]
$2P$	$3\mathbf{M} + 4\mathbf{S} \approx 6.2\mathbf{M}$ [2]	$3\mathbf{M} + 5\mathbf{S} \approx 7\mathbf{M}$ [28]
$3P$	$9\mathbf{M} + 3\mathbf{S} \approx 11.4\mathbf{M}$ [4]	$7\mathbf{M} + 7\mathbf{S} \approx 12.6\mathbf{M}$ [28] [12]
Mixed $2Q + P$	$11\mathbf{M} + 4\mathbf{S} \approx 14.2\mathbf{M}$ (this work)	N/A
$5P$	$15\mathbf{M} + 3\mathbf{S} \approx 17.4\mathbf{M}$ (this work)	$9\mathbf{M} + 13\mathbf{S} \approx 19.4\mathbf{M}$ [17] [31]

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

comparison to $P + Q$ in Jacobian Weierstrass coordinates [28]. See Appendix A for the $P + Q$ fromula in standard twisted Edwards coordinates.

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, $P + Q$ can be improved by using our proposed steps. First, pre-compute the values $X_1 \cdot Y_1, d \cdot X_1Y_1, a \cdot X_1$, and $a \cdot X_1Y_1$. This pre-computation step costs $1\mathbf{M} + 1\mathbf{M}_d + 2\mathbf{M}_a \approx 2\mathbf{M}$. Then, the pre-computed values can be used with the proposed $P+Q$ formula during scalar multiplication, as Table 2 shows. As a result, $P+Q$ with pre-computation saves $1\mathbf{M}_d + 1\mathbf{M}_a \approx 1\mathbf{M}$ in comparison to $P + Q$ without pre-computation. We say that $P+Q$ is mixed $P+Q$ when $Z_1 = 1$. The cost of mixed $P+Q$ is $9\mathbf{M} + 1\mathbf{M}_d + 1\mathbf{M}_a + 1\mathbf{S} \approx 10.8\mathbf{M}$. It saves approximately $0.4\mathbf{M}$ in comparison to mixed $P + Q$ in Jacobian Weierstrass coordinates [28].

Table 2. Operation Counts for $P + Q$ with pre-computation in Standard Twisted Edwards Coordinates over $\mathbb{F}_p$

Formula terms	Operation counts
$F = (Z_1 \cdot Z_2)^2 - \underline{dX_1Y_1} \cdot X_2 \cdot Y_2$	$3\mathbf{M} + 1\mathbf{S}$
$X_3 = Z_1Z_2 \cdot F \cdot \big((X_1 + X_2) \cdot (Y_1 + Y_2) - \underline{X_1Y_1} - X_2Y_2\big)$	$3\mathbf{M}$
$Y_3 = Z_1Z_2 \cdot \bar{F} \cdot \big((X_2 + Y_1) \cdot (Y_2 - \underline{aX_1}) - X_2Y_2 + \underline{aX_1Y_1}\big)$	$3\mathbf{M}$
$Z_3 = F \cdot \bar{F}$	$1\mathbf{M}$
	$10\mathbf{M} + 1\mathbf{S}$

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

Underlined terms are pre-computed.

2.3 A Proposed $2Q + P$ Formula

In standard twisted Edwards coordinates, the cost of a non-dedicated $2Q + P$ formula is $13\mathbf{M} + 1\mathbf{M}_d + 1\mathbf{M}_a + 5\mathbf{S} \approx 18\mathbf{M}$. However, our work shows that a dedicated $2Q + P$ formula has a lower cost. We propose a $2Q + P$ formula with or without pre-computation. See Theorem 1 for the proposed $2Q + P$ formula and Appendix B for the proof.

Theorem 1 *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.

Remark 1.

1. 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 shows. It saves $1\mathbf{M} + 1\mathbf{M}_d + 1\mathbf{S} \approx 2.8\mathbf{M}$ in comparison to a non-dedicated $2Q + P$ formula. The pre-computed values are $Z_1^2, X_1 \cdot Y_1, d \cdot X_1Y_1, X_1 \cdot Z_1, Y_1 \cdot Z_1, X_1Z_1 \cdot Y_1Z_1, a \cdot X_1Z_1$, and $a \cdot X_1Z_1Y_1Z_1$. This pre-computation step costs $4\mathbf{M} + 1\mathbf{M}_d + 2\mathbf{M}_a + 1\mathbf{S} \approx 5.8\mathbf{M}$.
2. 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 $1\mathbf{M} + 1\mathbf{M}_d + 1\mathbf{S} \approx 2.8\mathbf{M}$ 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. To the best of our knowledge, we are the first to propose a $2Q + P$ formula in twisted Edwards coordinates.

Table 3. Operation Counts for $2Q + P$ 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 conjugate of T and F respectively.

Underlined terms are pre-computed.

2.4 A Proposed $5P$ Formula

$5P$ can be obtained without a dedicated formula through $4P + P$ or $2P + 3P$. In standard twisted Edwards coordinates, the cost of $4P + P$ is $15\mathbf{M} + 1\mathbf{M}_d$

+ $3\mathbf{M}_a$ + $9\mathbf{S} \approx 25.2\mathbf{M}$ and the cost of $2P + 3P$ is $22\mathbf{M}$ + $1\mathbf{M}_d$ + $3\mathbf{M}_a$ + $8\mathbf{S} \approx 29.4\mathbf{M}$. However, many studies show a dedicated $5P$ formula has a lower cost. Bernstein, Birkner, Lange, and Peters initially proposed two dedicated $5P$ formulas in standard Edwards coordinates with cost $17\mathbf{M} + 7\mathbf{S} \approx 22.6\mathbf{M}$ and $14\mathbf{M} + 11\mathbf{S} \approx 22.8\mathbf{M}$ [3]. Recently, Rao proposed a dedicated $5P$ formula in standard Edwards coordinates with cost $15\mathbf{M} + 9\mathbf{S} \approx 22.2\mathbf{M}$ [35]. Li, Yu, and Wang proposed a $5P$ formula in standard Edwards coordinates with cost $12\mathbf{M}$ + $12\mathbf{S} \approx 21.6\mathbf{M}$ [26]. We propose the most efficient $5P$ formula. See Theorem 2 for the proposed $5P$ formula and Appendix B for the proof.

Theorem 2 *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.

Remark 2.
1. The cost of the proposed $5P$ formula is $15\mathbf{M}$ + $1\mathbf{M}_a$ + $3\mathbf{S} \approx 17.4\mathbf{M}$, as Table 4 shows. It saves $4.2\mathbf{M}$ over Li et al.'s formula. Also, it saves approximately $2\mathbf{M}$ in comparison to $5P$ in Jacobian Weierstrass coordinates ($a = -3$), as Table 1 shows [17] [31].
2. The proposed $5P$ formula needs only 2 temporary variables. This means that it can be performed using the same number of temporary variables as the $3P$ formula. See Appendix A for the cost of $3P$ in standard twisted Edwards coordinates and Appendix C for our proposed steps to perform the $3P$ and the $5P$ formulas with the fewest temporary variables.

3 MBNS Methods

3.1 Introduction

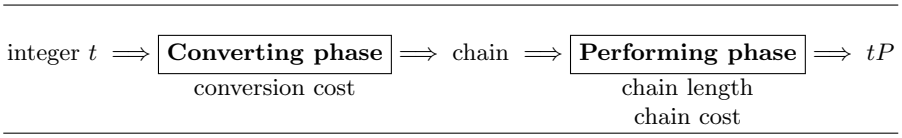

Fig. 1. Description of a Scalar Multiplication Method

Table 4. Operation Counts for $5P$ 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.

A scalar multiplication method in our framework, as figure 1 shows, goes through two phases: the converting phase and the performing phase. The first phase is to convert integer t to a chain and the second phase is to perform scalar multiplication on a given chain. We emphasize the conversion phase of a method since it is sufficient to determine the conversion cost, the average chain length, and the average chain cost of a method.

The converting phase The first phase of a method is to convert integer t to a chain. A chain can be a single-base chain or a multi-base chain. We focus on three types of chains: a single-base chain with $\{2\}$-integers, a double-base chain with $\{2, 3\}$-integers, and a multi-base chain with $\{2, 3, 5\}$-integers.

Definition 1 *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 studies proposed many methods that convert integer t to single-base chains or multi-base chains. The binary and the NAF methods convert integer t to a single-base chain with $\{2\}$-integers [20]. The greedy, the ternary/binary, the multi-base NAF, the tree-based, and the rDAG-based methods convert integer t to a multi-base chain [11, 31, 9, 27, 15, 37, 4]. These methods use different techniques to convert integer t to a chain. Therefore, we use the converting cost to measure the converting phase of a method. We express the converting cost by the time complexity of converting integer t to a chain. 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 that the time complexity of the converting phase might not be an accurate evaluation. This is because the

time complexity ignores the constant. The constant can affect the running time of the methods. It is hard to know the conversion cost of some methods, such as the greedy method. Therefore, we use our experimental results to determine the conversion cost of these methods.

The performing phase The second phase of a method is to perform a chain by executing a number of point addition (**ADD**), doubling (**DBL**), tripling (**TPL**), and quintupling (**QPL**) operations. The performing phase highly depends on a chain that is provided by the converting phase. We use the chain length and the chain cost to measure the performing phase. The chain length shows a broad indication of how good a chain is. It considers only the number of **ADD** to evaluate a chain. The chain cost shows an accurate indication of how good a chain is. It considers the number of **ADD**, **DBL**, **TPL**, and **QPL** to evaluate a chain.

Remark 3. 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 do the multi-base chains have a lower average cost than the single-base chain? In other words, why, when we replace **ADD** with **DBL**, **TPL**, or **QPL**, do we have lower average chain cost? We can explain that by the following reason. The cost of **ADD** is higher than the cost of **DBL** in many coordinate systems. Even though the cost of **ADD** is lower than the cost of **TPL** and **QPL** in many coordinate systems, we still prefer **TPL** and **QPL** over **ADD** because they are the least costly way to do three and five jumps. See Table 1 for the cost of **ADD**, **DBL**, **TPL**, and **QPL** in twisted Edwards and Jacobian coordinates.

An optimal chain There are no agreements among researchers on the definition of an optimal chain that represents integer t. However, we define an optimal chain as a chain that represents t with the lowest cost, as Definition 2 shows. The question is, do we have a method that generates an optimal chain? We know, as Experiment I shows, the greedy, ternary/binary, multi-base NAF, and tree-based methods do not produce an optimal chain. However, two methods were proposed by other studies to find an optimal chain: the enumeration approach and the rDAG-based method.

Definition 2 *An optimal chain for integer t is a chain that represents t, and it is the least costly in a particular coordinate system.*

Definition 3 *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.*

In the first study, Doche proposed the enumeration approach to find an optimal chain with respect to the length [14]. This is a type of brute force approach that takes exponential time to find an optimal chain. In the second study, Bernstein, Chuengsatiansup, and Lange proposed the rDAG-based method to find an optimal chain with respect to the cost [4]. The conversion cost in the rDAG-based method is approximately $\mathcal{O}\left((\log_2 t)^2\right)$. Even though the conversion cost of the rDAG-based method is much better than the conversion cost of the enumeration approach, the rDAG-based method seems impractical for applications that require the converting phase to be on-the-fly, as Experiment II shows. We propose a solution to improve the rDAG-based method. We develop bucket methods to find a near optimal chain with the advantage of taking significantly less time than the rDAG-based method for an optimal chain. See Definition 3 for a near optimal chain definition and see the third result of Experiment II for a near optimal chain example.

3.2 Proposed Bucket Methods

DAG/bucket method The idea for this newly proposed method is to create buckets that are indexed by chain cost. A bucket contains nodes that are ordered by t. A node is represented in the form of $(t, (s, a, b), (cost, seq), (pre_i, pre_j))$ where :

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 1 shows the steps of the DAG/bucket method. First, this method investigates nodes in a bucket with the lowest chain cost. Then, the method moves to the next bucket with the next lowest chain cost and investigates nodes. The method repeats the steps until it finds a node with $t = 1$. Finally, the method stops and returns the chain. Returning the chain is accomplished by *get-chain*, which traces (pre_i, pre_j) of nodes that are necessary for the chain. See Appendix D for an example of the DAG/bucket method.

An investigated node creates two or three nodes that are inserted into new buckets with a higher cost. This is because this method follows the DAG-based abstract idea, as Definition 4 shows. A node is inserted into a bucket according to its cost. The chain cost of nodes monotonically increases. This property guarantees that the method investigates all nodes and no node is skipped.

Definition 4 *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 $s \in \{-1, 0, +1\}$.*

The DAG/bucket method is similar to the rDAG-based method in that they both follow the DAG-based abstract idea. However, they are different in the

following way. The objective of the rDAG-based method is to find an optimal chain without paying much attention to the time it takes to reach it [4]. The objective of the DAG/bucket method is to provide flexibility to control the chain quality and the time to find the chain. The flexibility is obtained by bucket-size, a value which balances the chain quality and the time to find the chain.

Algorithm 1 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*

Tree/bucket method The idea of the tree/bucket method is to create buckets that are indexed by chain length. A bucket contains nodes that are ordered by t. A node is represented in the form of $(t, (s, a, b), (seq, pre))$ where:

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 2 shows the steps of the tree/bucket method. First, the method starts to investigate a node at bucket length = 1. Then, the method moves to investigate nodes at bucket length = 2. The method continues investigating nodes in each next bucket until it finds a node with $t = 1$. Finally, the method stops and returns the chain. Returning the chain is accomplished by get-chain,

which traces *pre* of nodes that are necessary for the chain. See Appendix D for an example of the tree/bucket method.

An investigated node always creates two new nodes that are inserted into the next bucket. This is because the tree/bucket method follows the tree-based abstract idea, as Definition 5 shows. A node is inserted into a bucket according to its length. The chain length of nodes monotonically increases. This property guarantees that the method investigates all nodes and no node is skipped.

Definition 5 *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 $(t-1)$.*

The similarity between the DAG/bucket and tree/bucket methods is that they both use bucket-size to control the chain quality and the time to find a chain. The main difference between them is that the tree/bucket method utilizes the tree-based abstract idea, and the DAG/bucket method utilizes the DAG-based abstract idea. As a result, the DAG/bucket method generates a higher chain quality than the tree/bucket method, and the tree/bucket method generates a chain faster than the DAG/bucket method, as Experiment II shows.

Algorithm 2 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*

Bucket-size and bucket-max Two important factors that determine the chain quality in the bucket methods are bucket-size and bucket-max. Bucket-size determines the number of nodes to be kept in a bucket. When bucket-size increases, it enhances the chain quality. However, it also increases the time to find the chain. The improvement percentage of the chain quality gradually decreases by increasing bucket-size. Eventually, the chain quality cannot be enhanced even though bucket-size is increased. Experimentally, we found that a practical choice for bucket-size is from 2 to 4.

The second factor that impacts the chain quality in the bucket methods is bucket-max. Bucket-max determines the number of buckets that are needed at maximum to find a chain. Experimentally, we found that when buckets are indexed by chain cost, bucket-max does not exceed the average chain cost of the NAF method. When buckets are indexed by chain length, bucket-max does not exceed the average chain length of the NAF method. This is with the assumption that the DAG/bucket method rounds the chain cost to the nearest integer. As a result, the average chain length and the average chain cost of the NAF method can estimate the value of bucket-max in the bucket methods. The average chain length of the NAF method is approximately $(log_2 t)/3$. The average chain cost of the NAF method is approximately $log_2 t(1/3\ \mathbf{ADD} + \mathbf{DBL})$.

4 Experimental Results

In this section, we show the results of two experiments. Experiment I compares the multi-base methods with the single-base methods. Experiment II compares the DAG/bucket method with the tree/bucket method. We incorporated our efficient formulas $P+Q, 2Q+P$ and $5P$ in these MBNS methods. We conducted these experiments in standard twisted Edwards coordinates over prime fields. We used three types of measurements to compare the methods: the average chain length, the average chain cost, and the average running time.

The advantage of the chain length and the chain cost measurements is that they give the same results if we test the methods in different devices. This is because the chain length is affected by the integer bit size and the method. The chain cost is effected by the integer bit size, the method, the coordinate system, and the $\mathbf{S}/\mathbf{M}$ ratio assumption. For the chain cost, we ignored addition/subtraction operations since they are cheap operations. We assumed $1\mathbf{S} = 0.8\mathbf{M}$ over prime fields. We did not use inversion operations since we worked in projective coordinates.

The advantage of the running time measurement is that it evaluates all the implementation aspects of the methods. It takes into consideration, unlike other measurements, the time to convert integer t to a chain. The disadvantage of the running time measurement is that it gives different results if we test the methods in different devices. This is because the running time is affected directly by various factors such as the CPU specification, the number of temporary variables, and the finite field arithmetic library (e.g., GMP, MIRCAL) [18] [29].

The devices specifications for these experiments are described as follows. We used C programming language and GCC compiler on 64-bit Ubuntu Linux OS. We used an Intel Core i5-8250U processor with the speed 1.6 GHz. We used GMP library for field arithmetic operations [18]. We generated 10,000 random integers within bit size [1, 254]. We repeated this step with 382-bit and 521-bit integers. Then we took the average for each reading results for accuracy reasons.

4.1 Experiment I

The goal for Experiment I was to compare the single-base methods with the multi-base methods. We wanted to measure the obtained improvement if the multi-base methods were used over the single-base methods. For the single-base methods, we implemented the binary and the NAF methods [20]. For the multi-base methods, we implemented the greedy, the ternary/binary, the multi-base NAF, and the tree-based methods [11] [9] [27] [15]. We assumed bound-size = 1 in the tree-based method because we did not consider the pre-computation option in Experiment I. However, we considered the pre-computation option in the converting phase in Experiment II. We present the results of Experiment I in Table 5 and Table 6. It shows the comparison results of the single-base and the multi-base methods with respect to the average chain length (l), the average chain cost (m), and the average running time (t).

Table 5. Theoretical Comparison Between Single-base and Multi-base Methods

	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	2135.48	13.72	83.42	3213.16	13.84	113.64	4381.26	13.96
ternary/binary	58.48	2161.99	12.65	87.67	3258.12	12.64	119.51	4449.32	12.63
(2, 3)NAF	61.07	2117.36	14.46	91.59	3191.09	14.43	124.87	4358.21	14.42
(2, 3)tree	55.11	2108.92	14.80	82.62	3178.69	14.77	112.63	4341.50	14.73
(2, 3, 5)NAF	52.11	2083.81	15.81	78.21	3141.32	15.77	106.60	4289.92	15.76
(2, 3, 5)tree	45.65	2077.91	16.05	68.40	3132.04	16.02	93.15	4277.15	16.01

l: The average chain length.

m: The average chain cost.

%: The improvement percentage with respect to m in comparison to (2)NAF.

Results of Experiment I We obtained three main results from Experiment I. First, with respect to m, all the investigated multi-base methods performed better than the single-base methods. We saw that the multi-base methods gave approximately 12% to 16% improvement in comparison to the single-base NAF method. With respect to t, they gave approximately 12% to 18% improvement, except for the greedy method (addressed below). The tree-based method with $\{2, 3, 5\}$-integers succeeded to give the highest percentage of improvement in comparison to other methods.

Second, the triple-base method with $\{2, 3, 5\}$-integers had a better performance than the double-base method with $\{2, 3\}$-integers. This implies that quadruple-base and quintuple-base methods may lead to a better performance than the triple-base method. However, we should also note that the improvement decreases with a higher-base method. This is because the improvement from single-base

Table 6. Running Time Comparison Between Single-base and Multi-base Methods

	254-bit		382-bit		521-bit	
	t	%	t	%	t	%
Binary	618.24		1121.56		2132.3	
(2)NAF	544.52		1025.42		1864.24	
(2, 3)greedy	545.71	−0.22	1034.57	−0.89	1867.16	−0.16
ternary/binary	475.45	12.68	874.45	14.72	1562.54	16.18
(2, 3)NAF	479.01	12.03	878.38	14.34	1566.15	15.99
(2, 3)tree	477.20	12.36	877.53	14.42	1563.37	16.14
(2, 3, 5)NAF	469.86	13.71	855.31	16.60	1527.33	18.07
(2, 3, 5)tree	465.18	14.57	849.76	17.13	1522.31	18.34

t: The average running time in μs.

%: The improvement percentage in comparison to (2)NAF.

to double-base methods was higher than the improvement from double-base to triple-base methods. As a result, we conclude that $7P$ and $11P$ dedicated formulas seem impractical to use with the multi-base methods since they only achieve a minor improvement.

Lastly, we found that the greedy method with $\{2,3\}$-integers is impractical for implementation for two reasons. The first reason is that it requires extra set-up time to find the best upper bound (a_{max}, b_{max}). We found that the upper bounds $(140, 73)$, $(210, 109)$, and $(290, 146)$ are approximately the best value for 254-bit, 382-bit, and 521-bit integers respectively. In contrast, other multi-base methods do not require extra set-up time to find the best upper bound.

The second reason it is impractical is the converting time. We did not have an efficient way to find the best approximation for integer t in terms of a $\{2,3\}$-integer. However, we tried the best available solutions. We used the line search algorithm [37] [7]. Another option is to use a look-up table [16]. However, a look-up table requires an offline pre-computation and extra memory space. In contrast, the converting times in other multi-base methods are more efficient and do not require offline pre-computations. This is because they use a dynamic approach to convert integer t to a multi-base chain.

4.2 Experiment II

The goal of Experiment II was to compare the tree/bucket and the DAG/bucket methods. We wanted to measure the bucket-size impact of the bucket methods on the chain cost and the time to find the chain. We implemented the DAG/bucket and tree/bucket methods with $\{2,3\}$-integers by using Algorithm 1 and Algorithm 2 respectively. We focused on 254-bit integers. We present the results of Experiment II in Figure 2. It shows the bucket-size impact of bucket methods on the chain cost and the time to find the chain.

Results of Experiment II We obtained three main results from Experiment II. First, the bucket methods use the bucket-size to balance the chain cost and

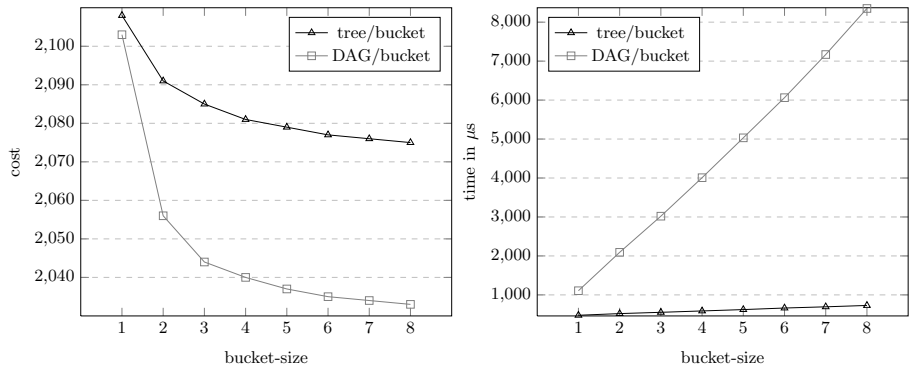

Fig. 2. Experimental Comparison Between Tree/bucket and DAG/bucket Methods

the time to find the chain. We noted that when the bucket-size increases, the chain cost gradually decreases and the time to find a chain gradually increases. The largest improvement of the chain cost in the bucket methods was when bucket-size = 2. The improvement of the chain cost became minor when bucket-size > 4. As a result, we concluded that the bucket methods seem more practical when the bucket-size is selected from 2 to 4.

Second, the DAG/bucket method produces a lower chain cost than the tree/bucket method. However, the DAG/bucket method also requires more time to find a chain. This is due to the ways these methods generate new nodes, as explained in Section 3. We also saw that the tree/bucket method with bucket-size = ∞ cannot produce a lower chain cost than the DAG/bucket method with bucket-size > 1. This implies that the tree-based method with unbound-size does not produce an optimal chain nor a near optimal chain, as Table 7 shows.

Table 7. Experimental Comparison Between Optimal and Near Optimal Chains

	254-bit		
	l	m	t
(2, 3)tree/bucket$_{size=\infty}$	51.01	2070.73	
(2, 3)DAG/bucket$_{size=4}$	51.54	2040.01	4008.17
(2, 3)rDAG-based	49.61	2035.56	14941.53

l: The average chain length.

m: The average chain cost.

t: The average running time in μs.

Third, a near optimal chain in the DAG/bucket method can be reached in significantly less time than an optimal chain. We noted that the near optimal chain for the DAG/bucket method could be reached when bucket-size = 4. This is because the average chain cost of the rDAG-based method, which produces an optimal chain, is 2035.56**M** and the average chain cost of the DAG/bucket

method when bucket-size = 4 is 2040.01**M**. Therefore, the difference between them is less than **DBL** cost. We also noted that the near optimal chain of the DAG/bucket method sped up the performance 73.17% in comparison to the optimal chain. This is because the time to reach an optimal chain in the rDAG-based method is 14941.53μs and the time to reach a near optimal chain in the DAG/bucket method is 4008.17μs, as Table 7 shows.

5 Conclusion

In this paper, we consider the efficiency problem of scalar multiplication for elliptic curves over finite fields with a twofold focus. Our first focus was to derive several efficiently computable formulas for the class of twisted Edwards curves over prime fields. We proposed $2Q+P$ and $5P$ formulas in standard projective coordinate systems. To the best of our knowledge, the $2Q+P$ formula given in this paper is the first dedicated formula for twisted Edwards curves. It saves about 2.8 field multiplications, compared to a non-dedicated formula. Our $5P$ formula improves on the best one in the literature [26] by about 4.2 field multiplications. It saves about 2 field multiplications, compared to $5P$ in Jacobian Weierstrass ($a = -3$) coordinates. In addition, using the pre-computation concept, we were able to compute $P + Q$ with one less field multiplication.

Our second focus dealt with the efficiency of generating MBNS chains. We formulated bucket methods for the DAG-based and the tree-based abstract ideas. These proposed bucket methods systematically balance the chain quality and the time to find the chain. We proposed finding a near optimal chain that relaxes the requirement of a chain being optimal. This is because finding a near optimal chain is more efficient than finding an optimal chain. We also demonstrated that the tree-based method does not produce an optimal chain nor a near optimal chain.

We conducted extensive experiments to compare the performance of the MBNS methods with respect to the average chain cost and the average running time. We used the running time to evaluate the two phases of the methods: the converting phase and the performing phase. We used the chain cost to evaluate only the performing phase of the methods. The investigated MBNS methods were the greedy, the ternary/binary, the multi-base NAF, the tree-based, the rDAG-based, and the bucket. Our proposed $P + Q$, $2Q + P$, and $5P$ formulas were utilized in these MBNS methods. Our results showed that the MBNS methods without pre-computation had an approximately 12% to 16% lower average chain cost than the NAF method. Except for the greedy method, they showed that the MBNS methods had an approximately 12% to 18% faster average running time. They showed that the MBNS methods with pre-computation could further lower the average chain cost and affect the average running time.

References

1. Al Musa, S., Xu, G.: Fast scalar multiplication for elliptic curves over binary fields by efficiently computable formulas. In: Patra, A., Smart, N. (eds.) Proc. IN-

DOCRYPT 20017. pp. 206–226. LNCS, Springer, Cham (2017)
2. Bernstein, D., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted edwards curves. In: Vaudenay, S. (ed.) Proc. AFRICACRYPT 2008. pp. 389–405. LNCS, Springer, Heidelberg (2008)
3. Bernstein, D., Birkner, P., Lange, T., Peters, C.: Optimizing double-base elliptic curve single-scalar multiplication. In: Srinathan, K., Rangan, C., Yung, M. (eds.) Proc. INDOCRYPT 2007 . pp. 167–182. LNCS, Springer, Heidelberg (2007)
4. Bernstein, D., Chuengsatiansup, C., Lange, T.: Double-base scalar multiplication revisited. Cryptology ePrint Archive, Report 2017/037 (2017), `https://eprint.iacr.org/2017/037`
5. Bernstein, D., Lange, T.: Inverted edwards coordinates. In: Boztas, S., Lu, H. (eds.) Proc. AAECC 2007. pp. 20–27. LNCS, Springer, Heidelberg (2007)
6. Bernstein, D., Lange, T.: Faster addition and doubling on elliptic curves. Cryptology ePrint Archive, Report 2007/286 (2007), `https://eprint.iacr.org/2007/286`
7. Berthe, V., Imbert, L.: On converting numbers to the double-base number system. Adv. Signal Pro. Algo. Archit. Implement. XIV. **5559**, 7078 (2004)
8. Brown, M., Hankerson, D., Lopez, J., Menezes, A.: Software implementation of the nist elliptic curves over prime fields. In: Naccache, D. (ed.) Proc. CT-RSA 2001. pp. 250–265. LNCS, Springer, Heidelberg (2001)
9. Ciet, M., Joye, M., Lauter, K., Montgomery, P.: Trading inversions for multiplications in elliptic curve cryptography. Designs, Codes and Cryptography **39**(2), 189–206 (2006)
10. Crosby, M., Nachiappan, Pattanayak, P., Verma, S., Kalyanaraman, V.: Blockchain technology: Beyond bitcoin (2016), `http://scet.berkeley.edu/wp-content/uploads/AIR-2016-Blockchain.pdf`
11. Dimitrov, V., Imbert, L., Mishra, P.: Efficient and secure elliptic curve point multiplication using double-base chains. In: Roy, B. (ed.) Proc. ASIACRYPT 2005. pp. 59–78. LNCS, Springer, Heidelberg (2005)
12. Dimitrov, V., Imbert, L., Mishra, P.: The double-base number system and its application to elliptic curve cryptography. Mathematics of Computation **77**(262), 1075–1104 (2008)
13. Dimitrov, V., Jullien, G., Miller, W.: An algorithm for modular exponentiation. Information Processing Letters **66**, 155–159 (1998)
14. Doche, C.: On the enumeration of double-base chains with applications to elliptic curve cryptography. In: Sarkar, P., Iwata, T. (eds.) Proc. ASIACRYPT 2014. pp. 297–316. LNCS, Springer, Heidelberg (2014)
15. Doche, C., Habsieger, L.: A tree-based approach for computing double-base chains. In: Mu, Y., Susilo, W., Seberry, J. (eds.) Proc. ACISP 2008. pp. 433–446. LNCS, Springer, Heidelberg (2008)
16. Doche, C., Imbert, L.: Extended double-base number system with applications to elliptic curve cryptography. In: Barua, R., Lange, T. (eds.) Proc. INDOCRYPT 2006. pp. 335–348. LNCS, Springer, Heidelberg (2006)
17. Giorgi, P., Imbert, L., Izard, T.: Optimizing elliptic curve scalar multiplication for small scalars. Mathematics for Signal and Information Processing **7444** (2009)
18. Granlund, T., et al.: Gnu multiple precision arithmetic library, `http://www.gmplib.org`
19. Hankerson, D., Lopez, J., Menezes, A.: Software implementation of elliptic curve cryptography over binary fields. In: Koc, C., Paar, C. (eds.) Proc. CHES 2000. pp. 1–24. LNCS, Springer, Heidelberg (2000)
20. Hankerson, D., Menezes, A., Vanstone, S.: Guide to elliptic curve cryptography. Springer-Verlag (2004)

21. He, D., Zeadally, S.: 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 (2014)
22. Koblitz, N.: Elliptic curve cryptosystems. Math. of Computation **48**(177), 203–209 (1987)
23. Kohel, D.: Twisted u4-normal form for elliptic curve. Cryptology ePrint Archive, Report 2017/121 (2017), `https://eprint.iacr.org/2017/121`
24. Lange, T.: A note on lopez-dahab coordinates. Cryptology ePrint Archive, Report 2004/323 (2004), `https://eprint.iacr.org/2004/323`
25. Lenstra, A., Verheul, E.: Iselecting cryptographic key sizes. Journal of Cryptology **14**, 255–293 (2001)
26. Li, W., Yu, W., Wang, K.: Improved tripling on elliptic curves. In: Lin, D., Wang, X., Yung, M. (eds.) Proc. Inscrypt 2015. pp. 193–205. LNCS, Springer, Cham (2016)
27. Longa, P., Gebotys, C.: Fast multibase methods and other several optimizations for elliptic curve scalar multiplication. In: Jarecki, S., Tsudik, G. (eds.) Proc. PKC 2009. pp. 443–462. LNCS, Springer, Heidelberg (2009)
28. Longa, P., Miri, A.: Fast and flexible elliptic curve point arithmetic over prime fields. IEEE Transactions on Computers **57**, 289–302 (2008)
29. Matula, D., Kornerup, P.: Multiprecision integer and rational arithmetic c library, `http://www.miracl.com`
30. Miller, V.: Use of elliptic curves in cryptography. In: Williams, H. (ed.) Proc. CRYPTO 1985. pp. 417–426. LNCS, Springer, Heidelberg (1986)
31. Mishra, P., Dimitrov, V.: Efficient quintuple formulas for elliptic curves and efficient scalar multiplication using multibase number representation. In: Garay, J., Lenstra, A., Mambo, M., Peralta, R. (eds.) Proc. ISC 2007 . pp. 390–406. LNCS, Springer, Heidelberg (2007)
32. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2011), `https://bitcoin.org/bitcoin.pdf`
33. Oliveira, T., Lopez, J., Aranha, D., Rodriguez-Henriquez, F.: Two is the fastest prime: lambda coordinates for binary elliptic curves. Journal of Cryptographic Engineering **4**(1), 3–17 (2014)
34. Rescorla, E.: Rfc8446: The Transport Layer Security (TLS) Protocol Version 1.3 (2018)
35. Subramanya Rao, S.: Three dimensional montgomery ladder, differential point tripling on montgomery curves and point quintupling on weierstrass and edwards curves. In: Pointcheval, D., Nitaj, A., Rachidi, T. (eds.) Proc. AFRICACRYPT 2016. pp. 84–106. LNCS, Springer, Cham (2016)
36. Yu, W., Kim, K., Jo, M.: New fast algorithms for elliptic curve arithmetic in affine coordinates. In: Tanaka, K., Suga, Y. (eds.) Proc. IWSEC 2015. pp. 56–64. Springer, Cham (2015)
37. Yu, W., Wang, K., Li, B., Tian, S.: Triple-base number system for scalar multiplication. In: Youssef, A., Nitaj, A., Hassanien, A. (eds.) Proc. AFRICACRYPT 2013. pp. 433–451. LNCS, Springer, Heidelberg (2013)

6 Appendix A: Formulas

This appendix shows the operation counts for $P + Q$, $2P$, and $3P$ formulas represented in standard twisted Edwards coordinates. These formulas can be found in [2, 4]. However, we use different notations to present these formulas because we want to unify the notations for the complete set $P+Q$, $2P$, $3P$, and $5P$. Also, it is necessary to present these formulas here because the proofs of the new $5P$, and $2Q + P$ formulas depend on these formulas.

Table 8. Operation Counts for $P + Q$ in Standard Twisted Edwards Coordinates

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 9. Operation Counts for $2P$ in Standard Twisted Edwards Coordinates

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

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

Table 10. Operation Counts for $3P$ in Standard Twisted Edwards Coordinates

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.

7 Appendix B: Proofs

7.1 Proof of Theorem 1

Proof. We shall prove Theorem 1 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 $P+Q$ formula in Table 8, We have

$$F = (Z_P Z_{2Q})^2 - dX_P Y_P X_{2Q} Y_{2Q}.$$

We substitute X_{2Q}, Y_{2Q}, Z_{2Q} with their equivalent terms in Table 9. We have

$$\begin{aligned} T &= Y_Q^2 + aX_Q^2. \\ F &= \big(Z_P T(T - 2Z_Q^2)\big)^2 + dX_P Y_P 2X_Q Y_Q (T - 2Z_Q^2) T\bar{T} \\ &= T(T - 2Z_Q^2)\big(Z_P^2 T(T - 2Z_Q^2) + dX_P Y_P 2X_Q Y_Q \bar{T}\big). \end{aligned}$$

From the $P+Q$ formula, we have

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

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

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

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

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

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

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

7.2 Proof of Theorem 2

Proof. We shall prove Theorem 2 by the fact $5P = 2P + 3P$. From the $P + Q$ formula shown in Table 8, 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 9 and Table 10. 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

$$\begin{aligned} X_5 &= Z_2Z_3F(X_2Y_3 + Y_2X_3). \\ Y_5 &= Z_2Z_3\bar{F}(Y_2Y_3 - aX_2X_3). \\ Z_5 &= F\bar{F}. \end{aligned}$$

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

$$Z_5 = Z_1CD.\square$$

8 Appendix C: Algorithms

Algorithm 3 and Algorithm 4 show that point tripling (**TPL**), and point quintupling (**QPL**) in standard twisted Edwards coordinates need only 2 temporary variables. See Table 10 and Table 4 for the $3P$ and the $5P$ formulas.

Algorithm 3 Point Tripling
Input: $P = (X, Y, Z)$
Output: $3P = (X_3, Y_3, Z_3)$
$X_3 = Y^2$
$Y_3 = X^2$
$Y_3 = a \cdot X_3$
$Z_3 = X_3 + Y_3$
$T_1 = X_3 - Y_3$
$T_1 = T_1 \cdot Z_3$
$T_2 = Z^2$
$T_2 = T_2 + T_2$
$T_2 = Z_3 - T_2$
$X_3 = X_3 + X_3$
$X_3 = X_3 \cdot T_2$
$Y_3 = Y_3 + Y_3$
$Y_3 = Y_3 \cdot T_2$
$Z_3 = T_1 + X_3$
$X_3 = T_1 - X_3$
$X_3 = X_3 \cdot Z_3$
$T_2 = T_1 - Y_3$
$Y_3 = T_1 + Y_3$
$Y_3 = Y_3 \cdot T_2$
$Z_3 = Z_3 \cdot T_2$
$Z_3 = Z \cdot Z_3$
$X_3 = X \cdot X_3$
$Y_3 = Y \cdot Y_3$
$Y_3 = -Y_3$
return: (X_3, Y_3, Z_3)

Algorithm 4 Point Quintupling
Input: $P = (X, Y, Z)$
Output: $5P = (X_5, Y_5, Z_5)$
$Z_5 = Y^2$
$Y_5 = X^2$
$Y_5 = a \cdot Y_5$
$X_5 = Z_5 + Y_5$
$T_1 = Z^2$
$T_1 = T_1 + T_1$
$T_1 = X_5 - T_1$
$Z_5 = Z_5 + Z_5$
$Y_5 = Y_5 + Y_5$
$Z_5 = Z_5 \cdot T_1$
$T_1 = Y_5 \cdot T_1$
$Y_5 = X_5 - Y_5$
$Y_5 = Y_5 \cdot X_5$
$X_5 = Y_5 + Z_5$
$T_2 = Y_5 - Z_5$
$X_5 = X_5 \cdot T_2$
$T_2 = Y_5 - T_1$
$Y_5 = Y_5 + T_1$
$Y_5 = T_2 \cdot Y_5$
$T_2 = T_2 + T_1$
$T_1 = T_1 \cdot X_5$
$Z_5 = Z_5 \cdot Y_5$
$X_5 = T_2 \cdot X_5$
$X_5 = -X_5$
$Y_5 = T_2 \cdot Y_5$
$T_2 = X_5 + Z_5$
$X_5 = X_5 - Z_5$
$X_5 = T_2 \cdot X_5$
$X_5 = X \cdot X_5$
$Z_5 = Y_5 + T_1$
$Y_5 = Y_5 - T_1$
$Y_5 = Z_5 \cdot Y_5$
$Y_5 = Y \cdot Y_5$
$Z_5 = T_2 \cdot Z_5$
$Z_5 = Z \cdot Z_5$
return: (X_5, Y_5, Z_5)

9 Appendix D: Examples

9.1 DAG/bucket method

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 1 creates the following buckets:

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	$(1, (-1, 0, 1), (9, 0), (5, 0))$

Then, getting the chain steps can be accomplished by the following Horner's rule manner:
$(1, (0, 0, 1), (6, 0), (4, 0)) \implies \text{chain} = 3$
$(3, (0, 1, 0), (4, 0), (3, 0)) \implies \text{chain} = (3)2$
$(6, (1, 1, 0), (3, 0), (0, 0)) \implies \text{chain} = ((3)2)2 + 1.$
Thus, chain $= 2^2\ 3 + 1 = 13$.

9.2 Tree/bucket method

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

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 the following Horner's rule 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 $= 2^3\ 3 + 2\ 3 - 1 = 29$.