IMPROVED ELLIPTIC CURVE ARITHMETIC BY REORDERING OPERATION SEQUENCES

演算順序変更による楕円曲線算術の改善

by

Chitchanok Chuengsatiansup
チュンサティアンサップ　チッチャノック

A Master Thesis
修士論文

Submitted to
the Graduate School of the University of Tokyo
on February 5, 2013
in Partial Fulfillment of the Requirements
for the Degree of Master of Information Science and Technology
in Computer Science

Thesis Supervisor: Hiroshi Imai　今井　浩
Professor of Computer Science

ABSTRACT

Elliptic curve-based is another field of cryptography which has recently gained a lot of attention due to its advantage of shorter key length while still preserving the same level of security compared to the famous RSA cryptosystem. Another reason why elliptic curve-based cryptography has become popular is that it allows some features which cannot be provided by RSA cryptosystem. Scalar multiplication, one of the main operations in elliptic curve cryptography, is very time-consuming but has to be executed frequently. Speeding up scalar multiplication has long been studied, and a number of algorithms have already been proposed. Nevertheless, faster and more efficient algorithms are still greatly desired.

To implement elliptic curve scalar multiplication, there are many choices of curve shapes, coordinate systems, and point arithmetic formulas. Each of those selections has different advantages. Bernstein, Kohel, and Lange proposed point doubling formulas for twisted Hessian curves in projective coordinates which allow faster computation than the general Weierstrass curves. Dimitrov, Imbert, and Mishra tried a different approach and proposed a faster scalar multiplication through an introduction of double-base chain representation. Equally important to speeding up the computation, security over side-channel analysis is another issue that has increasingly been taken into consideration. Chevallier-Mames, Ciet, and Joye proposed a method to prevent simple side-channel analysis by introducing a concept of atomicity. All of the above mentioned works considered cases where scalar multiplication is executed in a specific order of point operations. However, different order of point operations may allow faster computation through a reduction of unnecessary operations resulted from efficiently reuse the previously computed intermediate values.

We derived the improved point doubling formulas for twisted Hessian curves in projective coordinates. These new formulas are obtained by differently arranging field operations and efficiently reused intermediate values. In contrast to previous work which achieved faster formulas through an introduction of i with an assumption $i^2 = -1$ and the fact that some fields allow cheap multiplication by i, our method optimizes the operation cost by using the equality of square of the sum of two numbers. That is, an expensive field multiplication is calculated at a cheaper cost of field squaring. We also removed the necessity of defining i to match the required assumption. In brief, we successfully replaced field multiplication by field squaring and traded multiplication by i with multiplication by constant. We would like to emphasize that we improved point doubling operation which is the operation that frequently executed when computing scalar multiplication.

Moreover, we proposed a faster method for computing scalar multiplication represented in (2,3) double-base chain. This method is achieved by reordering point operation sequences together with utilizing the already computed intermediate values to avoid redundant calculation. Unlike previous work which computed point tripling prior to point doubling, our study revealed that computing point doubling prior to point tripling can reuse more intermediate values which leads to faster scalar multiplication since the number of operations required is reduced. Our method is also secure over simple side-channel analysis because the previously proposed atomic block can be applied directly. In addition to curves defined over prime field, we also examined curves defined over binary field. In this case, we found that computing point tripling prior to point doubling allows intermediate values to be reused. We believe we are the first who showed that reusing intermediate values is also possible for curves defined over binary field.

Additionally, we introduced multi-pattern atomic block which allows faster scalar multiplication than the conventional single-pattern atomic block. Because it is rather difficult to fit the entire original operation sequences into a single pattern, we coped with this problem by defining more than one pattern. By rearranging operation sequences to fit into our multi-pattern, it turned out to be that many dummy operations can be decreased which resulted in more efficient computational time. Our multi-pattern atomic block preserves the essential characteristic of atomicity by arranging those different patterns in a systematic way so that they appear equivalently observed by side-channel. We proposed three different atomic patterns for different usages: left-to-right binary scalar multiplication, right-to-left binary scalar multiplication for resource constrained system, and binary-ternary scalar multiplication involving point triple.

論文要旨

本論文では、楕円曲線スカラー倍算における高速アルゴリズムについて論ずる。楕円曲線に基づく暗号は、同程度の安全性を持ち、広範にわたって知られている RSA 暗号と比べると鍵の長さが短い。また、RSA 暗号では構成できず、楕円曲線に基づく暗号を用いることで初めて適用できる手法の存在が知られている。このような特長から、楕円曲線に基づく暗号が注目されている。楕円曲線暗号の重要な演算の一つとしてはスカラー倍算が挙げられる。この演算は非常に計算時間がかかるが、頻繁に実行される。したがって、スカラー倍算の高速化が広く研究されており、様々なアルゴリズムが提案されてきた。しかし依然として、より高速かつ効率的なアルゴリズムが求められている。

楕円曲線スカラー倍算を実行する際、曲線の形状、座標系およびポイント算術方式において様々な選択肢があり、それぞれが異なる長所を持っている。Bernstein, Kohel, Lange らは射影座標上で twisted Hessian curves を利用し、一般 Weierstrass curves より高速なスカラー倍算のポイント二倍方式を提案した。Dimitrov, Imbert, Mishra らは異なる方法で、二重基底鎖を通じて高速なスカラー倍算を提案した。また、サイドチャネル解析防止も非常に重要な課題であり、Chevallier-Mames, Ciet, Joye らは、単純サイドチャネル解析を防止するため、サイドチャネルアトミシティを提案した。上記に挙げたこれらの既存研究においては、すべてスカラー倍算はあらかじめ決められた順序でポイント演算が実行される。しかし、演算順序を変更することで、計算途中に得られる中間値を再利用した高速化が可能だと思われる。

我々は射影座標上の twisted Hessian curves におけるポイント二倍方式を改善した。この新たな方式では、演算順序変更と中間値再利用の手法を利用した。既存研究では、$i^2 = -1$ の仮定のもと、ある体上で i 倍の乗算が速く計算できる事実を利用した方式が提案された。それに対し、我々は二つの数値の和の二乗を展開して得られる式を利用して、計算が遅い乗算を計算が速い二乗に置換することでより高速に演算する方式を提案した。また、i に関する仮定を削除した。これにより、乗算の代わりに二乗を計算し、i 倍の乗算の代わりに定数乗算を実行することができた。すなわち、我々はスカラー倍算において頻繁に実行される演算であるポイント二倍を改善したと結論付けられる。

その他にも、(2,3) 二重基底鎖上での高速なスカラー倍算を提案した。ポイント演算順序変更と共に計算した中間値の再利用で、不必要な演算を減少させることができた。既存研究では、ポイント三倍を計算したのちに、ポイント二倍を計算する手法が提案されたが、我々はポイント二倍を先に計算することで中間値の再利用を増やし、より高速にスカラー倍算が行えることを発見した。また、すでに提案されているアトミックブロックを直接応用できるので、我々の方法は単純サイドチャネル解析を防止可能である。さらに、素数体上に定義される曲線以外に、位数２の数体上に定義される曲線についても研究した。位数２の数体上に定義される曲線における中間値の再利用性については今まで研究がなされていないと思われる。位数２の数体の場合、ポイント三倍を先に計算すれば、中間値再利用の回数が増えるとの結果が得られた。

一方、より速くスカラー倍算を行う目的で、サイドチャネル解析を防止するために必要なアトミックブロックについて、従来のシングルパターンアトミックブロックに代えて、マルチパターンアトミックブロックを使用する新しい方法を導入した。全ての演算順序を一つのパターンに集約させるのは非常に難しいことから、我々は二つのパターン以上の使用

することを提案した。マルチパターンに合致するよう演算を調整し、ダミー演算を減少させた結果、より優れた計算時間を得た。我々の提案したマルチパターンアトミックブロックは等価に観測されるように設計したので、サイドチャネルから情報の窃取を試みても情報漏洩することがなく、重要なアトミシティの特徴を保っている。我々は用途に応じた三つのアトミックパターンを紹介した : left-to-right 二進スカラー倍算、資源が限られたシステムにおける right-to-left 二進スカラー倍算、ポイント三倍に関する二進・三進スカラー倍算。

Acknowledgements

I am deeply grateful to my supervisor Hiroshi Imai who patiently provided invaluable feedback. His guidance has truly been helpful for my accomplishment. Without his advice, works presented in this thesis would not have been possible.

I especially thank Vorapong Suppakitpaisarn for his kind mentorship. He gave constructive comment which helped me to improve my work substantially. I am also highly thankful for his continually support and encouragement.

I would like to extend my appreciation to Tanja Lange, Daniel Bernstein, and Craig Costello for their generous hospitality during my visit to Eindhoven University of Technology. Having opportunities to discuss with them allowed me to understand more about the concept behind the elliptic curve cryptography and pairing-based cryptography.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"If he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out." — Suetonius describes in De Vita Caesarum (The Lives of the Caesars) how encryption system used by Caesar works. This indicates that an attempt to communicate securely has a long history dating back to the time of the Roman Emperor Julius Caesar. It also means that cryptography has been existing for more than 2000 years.

Cryptography is used broadly in applications involving sensitive information in order to protect those data. Recently, cryptography plays a crucial role in security over the Internet. One example is an online shopping where cryptographic algorithm, such as Triple-DES and AES, is used to encrypt credit card information in order to prevent unauthorized parties from perceiving essential data. Other examples include internet banking and electronic mail. To give a concrete example, Google uses RC4 for encryption (making messages incomprehensible to irrelevant persons but authorized parties) and SHA1 for message authentication (checking messages upon arrival whether modification had been taken place) [Lan]. This indicates that cryptography is actually being used in our daily life, and we may even use it without realizing it.

1.1 Motivation

Speaking about cryptography, many people might think of big prime number which is exactly one of the crucial concepts of the well-known RSA cryptosystem. Due to the fact that RSA has been used for a long time, plenty of research has been conducted. However, those studies are not only on how to improve the algorithms but also on how to break the cryptosystem. As a consequence, prime numbers that can withstand various kinds of cryptographic attack are getting larger and larger. This also means that the size of the key or the bit length of the key is getting longer and longer. According to [BR], RSA-based key agreement of 2048 bits is acceptable while 1024 bits shall not be used after the year 2013.

By using different approach, elliptic curve cryptography [Was08] [HMV03] [ACD+06] allows shorter key length compared to the RSA cryptosystem. This is achieved by considering elliptic curve group and elliptic curve discrete logarithm. It is this major advantage of requiring smaller key size without declining the security level that has drawn much attention into exploring this alternative kind of cryptography. As stated in [GPW+04], ECC requires only 224-bit key size for a comparable security to 2048-bit RSA. In fact, shorter key length implies smaller space to store and easier to handle the key.

Key size is, indeed, directly related to mathematical security. However, there exists a security threat in which no matter how mathematically secure the cryptosystem is, the attack is always possible. That type of attack is conducted by analyzing information leak via physical devices. Therefore, it is generally called side-channel attack or side-channel analysis [Joy03] [KGSL10] [Ava05] [FGM+10] [FV12] [EBJ02] [KY10] [KHM+05] [FV03] [YKMJ06] [AT03]. Side-channel analysis is a very serious threat because even a mathematically secure cryptosystem also leaks information if not carefully implemented. Thus, countermeasure against this attack has also been intensively studied as well. [CMCJ04] [TMX09] [MMM04] [CJ01] [CJ03] [JT01] [Cor99] [BJ03] [JY02] [IIT06].

One solution to prevent side-channel analysis is to rewrite sequences of operations in the way such that they appear equivalently observed from side-channel so that adversary can no longer be able to derive significant information. This method is done by using side-channel atomic block [CMCJ04]. However, using atomic block resulted in dummy operations being added which slowed down scalar multiplication.

When elliptic curve cryptography was firstly introduced, it was already slower than RSA cryptosystem. To make elliptic curve cryptography secure over side-channel analysis even makes it slower. Since fast and secure cryptosystem is greatly desired, a number of existing research on elliptic curve cryptography aim at both speeding up scalar multiplication [BL07a] [LM08] [BL07b] [Joy08] [Lon07] [HCD07] and enhancing the security level [GV10]. A major reason scalar multiplication has been being focused is that this operation has to be executed very frequently.

1.2 Previous Work

To compute scalar multiplication, double-and-add algorithm is commonly used. This algorithm can be executed in two ways depending on which direction the input scalar is scanned, namely, from left to right or from right to left. In double-and-add algorithm, point doubling is repeatedly executed for every bit of the input scalar while point addition is executed only if the current bit is 1. The aforementioned point operations — point doubling and point addition — consist of field operations such as field multiplication, field squaring, and field addition. In nearly all cases, cost of field addition is significantly less than field multiplication and thus can be neglected. Cost of field squaring S is also less than field multiplication M and an approximation $S = 0.8M$ is usually assumed.

The number of field operations in point operation formulas corresponds to the speed of scalar multiplication. Decreasing the number of field operations obviously makes scalar multiplication faster. Various approaches on reducing field operations have been studied, and a number of techniques have been proposed.

First example is expressing number in multi-based [MD07] [DIM08] [DI08] [DIM05b] [DI06] [DIM05a] [DJM99]. For instance, scalar represented in (2,3) double-base chain [DIM08] involving point tripling allows fewer number of field operation count owning to the fact that point tripling uses $10M + 6S$ while performing one point doubling then one point addition uses $(4M + 6S) + (12M + 4S) = 16M + 10S$.

Second example is using different atomic block patterns. For instance, cost of scalar multiplication per input bit using atomic pattern proposed in [CMCJ04] is $20M$ while atomic pattern proposed in [GV10] requires only $16M$ assuming $S = 0.8M$. The cost per bit reduction was achieved by decreasing dummy field operations by making the patterns more suitable for point operations.

Third example is representing curve in other shapes [BL09] [JQ01] [Sma01] [HWCD09] [LWX09] [BBJ+08] [HWCD08]. For instance, point doubling on twisted Hessian curves requires $8M$ [BL] while on Weierstrass curves requires $4M + 6S$ [DIM08].

1.3 Our Study

We conducted our research based on the above mentioned previous works which we tried to improve their algorithms. Our investigation on point doubling formulas on twisted Hessian curves revealed that it is possible to rewrite the formulas so that earlier computed values can be reused to obtain faster point doubling formulas. Results of our study on double-base chain scalar multiplication suggested that the concept of rewriting and reusing values can also be applied to avoid unnecessary computation and make scalar multiplication faster.

In addition to examining the already proposed techniques on accelerating scalar multiplication, we also analyzed possibilities to apply our techniques of rearranging operation sequences and reusing prior computed values to scalar multiplication which was slowed down by the effect of using side-channel atomic block. Together with rearranging and reusing techniques, we introduced a new concept of multi-pattern atomic block into side-channel atomicity. Our multi-pattern reduces unnecessary dummy operations which leads to side-channel atomic block that requires less computational time.

1.4 Summary of Contributions

We focused our study on improving the speed of scalar multiplication. Main techniques that we used were rearranging operation sequences and reusing previously computed values.

Our contributions can be summarized as follows:

1. Faster point doubling on twisted Hessian curves

   We proposed new formulas that change an expensive field multiplication into a cheaper field squaring, namely, from $8M$ to $7M + 1S$ where M and S are the cost of field multiplication and field squaring respectively. These new formulas also reduce two field additions, that is, from 9 additions down to 7 additions. In brief, we reduced cost per bit by $1.43 - 2.75\%$.

2. Double-base chain scalar multiplication reordering

   Our investigation on consecutive operations revealed that it is more efficient to compute point doubling prior to point tripling for scalar multiplication represented using $(2, 3)$ double-base chain in case of prime field. Our experimental results showed a speed-up of $1.55 - 1.95\%$. On the other hand, it is better to compute point tripling before point doubling in case of binary field. In this case, $0.28 - 0.33\%$ speed-up is obtained.

3. Side-channel atomicity and multi-pattern atomic block

   We introduced an idea of multi-pattern atomic block and proposed three different atomic patterns for different usages. Our atomic block for left-to-right binary scalar multiplication achieves 1.36% improvement while the right-to-left binary scalar multiplication atomic block obtains $0.22 - 0.65\%$ improvement. In case of binary-ternary scalar multiplication involving point tripling, we decreased 1 negation per each atomic block.

1.5 Thesis Outline

Chapter 1 provides a general information about cryptography and existing research on elliptic curve cryptography. Contribution summary and organization of this thesis are also contained in this chapter.

We review a basic mathematical background necessary for understanding the content in this thesis in Chapter 2. Readers who already have a strong background on elliptic curve cryptography, double-base number system, and side-channel analysis may skip this part.

Our contributions are presented in Chapter 3–5. We explain in Chapter 3, our faster point doubling formulas for twisted Hessian curves in projective coordinates. Chapter 4 describes a better point operation sequences for scalar multiplication representing in double-base chain. In Chapter 5, we elaborate our improved atomic pattern for secure and faster scalar multiplication. Each of this chapter is independent and can be read separately.

We wrap up this thesis with a conclusion in Chapter 6. Our ideas for future work are also presented as well.

Chapter 2

Preliminaries

Faster elliptic curve scalar multiplication is the main focus of our study. We investigated various forms of elliptic curve shape including different coordinate system. We also studied an approach of integrating double-base chain number representation into elliptic curve cryptography. Security concern over information leakage via physical devices was examined as well. For a better understanding the contents contained in this thesis, we summarized relate mathematical background of all the above mentioned topics in this chapter.

First, we briefly reviewed the elliptic curve cryptography such as curve equations, point operations, and related algorithms. Then, we explained the definition and provided some examples of double-base number system (DBNS) — a different way to represent numbers — which has long been studied and was recently introduced to elliptic curve cryptography. Finally, we described side-channel analysis and its countermeasures in the last part.

2.1 Elliptic Curve Cryptography

An elliptic curve E over a field K is defined by the following equation

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.1}$$

where $a_1, a_2, a_3, a_4, a_6 \in K$, and $\Delta \neq 0$, where Δ is the discriminant of E.

Equation (2.1) is referred to as the **Weierstrass equation** for an elliptic curve. Weierstrass equation can be rewritten into simpler forms depending on the characteristic of the underlying fields. (see also [Was08] [ACD$^+$06])

If binary field is used, Equation (2.1) can be simplified to

$$y^2 + xy = x^3 + a_2x^2 + a_6 \tag{2.2}$$

where $a_2, a_6 \in K$, and $\Delta = a_6 \neq 0$.

If prime field is used, Equation (2.1) can be further simplified to

$$y^2 = x^3 + a_4x + a_6 \tag{2.3}$$

where $a_4, a_6 \in K$, and $\Delta = 4a_4^3 + 27a_6^2 \neq 0$.

In general, elliptic curves defined over finite field are used. However, it is easier to comprehend if depicting a curve over $\mathbb{R}$. Figure 2.1 illustrates an elliptic curve $E : y^2 = x^3 - 3x + 4$ defined over $\mathbb{R}$.

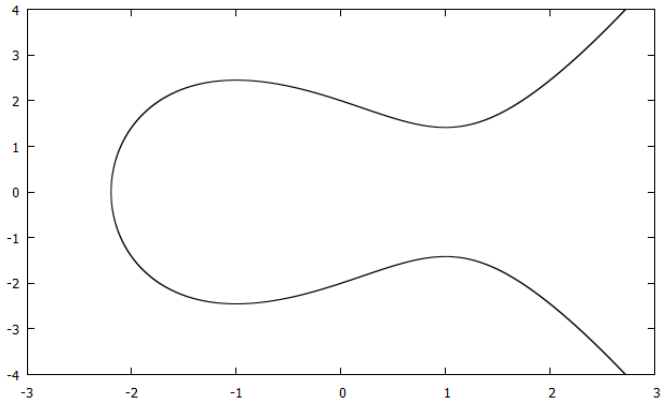

Figure 2.1: Elliptic curve $y^2 = x^3 - 3x + 4$ over $\mathbb{R}$

2.1.1 Point Arithmetic Operation

This subsection summarizes point arithmetic operation for addition, doubling, and tripling. All point operation formulas presented in this subsection are for Weierstrass curves.

Addition in Jacobian Coordinates

Let $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, Z_2)$ be two points in Jacobian coordinates on the elliptic curve E. Point addition $P_3 = (X_3, Y_3, Z_3) = P_1 + P_2$ can be computed as follows:

$$\begin{aligned} X_3 &= \alpha_3^2 - \beta_3^3 - 2X_1Z_2^2\beta_3^2, \\ Y_3 &= \alpha_3(X_1Z_2^2\beta_3^2 - X_3) - Y_1Z_2^3\beta_3^3, \\ Z_3 &= Z_1Z_2\beta_3 \end{aligned}$$

where

$$\begin{aligned} \alpha_3 &= Y_2Z_1^3 - Y_1Z_2^3, \\ \beta_3 &= X_2Z_1^2 - X_1Z_2^2 \end{aligned}$$

Thus, the cost of point addition is $12M + 4S$ where M and S are cost of field multiplication and field squaring respectively.

Addition in Mixed Jacobian-Affine Coordinates

Instead of carrying out the computation in a single coordinate system, Cohen, Miyaji, and Ono [CMO98] introduced a computation in mixed coordinate systems in order to increase the performance by exploiting advantages of each coordinate system. Addition in mixed Jacobian-affine coordinates or *mixed point addition* is computed by adding one point in Jacobian coordinates to the other point in affine coordinates. Recall that points in affine coordinates have its Z value equals to 1.

Let $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, Z_2)$ be points on the elliptic curve E in Jacobian and affine coordinates respectively. Point addition $P_3 = (X_3, Y_3, Z_3) =$

$P_1 + P_2$ can be computed as follows:

$$\begin{aligned} X_3 &= \alpha_3^2 - \beta_3^3 - 2X_1\beta_3^2, \\ Y_3 &= \alpha_3(X_1\beta_3^2 - X_3) - Y_1\beta_3^3, \\ Z_3 &= Z_1\beta_3 \end{aligned}$$

where

$$\begin{aligned} \alpha_3 &= Y_2Z_1^3 - Y_1, \\ \beta_3 &= X_2Z_1^2 - X_1 \end{aligned}$$

Since point P_2 is in affine coordinates which has its Z coordinate or Z_2 equals to 1, computing $Z_2^2, Z_2^3, Z_1Z_2, X_1Z_2^2$ can be omitted. Hence, the cost of mixed point addition is reduced to $8M + 3S$ where M and S are cost of field multiplication and field squaring respectively.

Doubling in Jacobian Coordinates

Let $P_1 = (X_1, Y_1, Z_1)$ be a point in Jacobian coordinates on the elliptic curve E. Point doubling $[2]P_1 = (X_2, Y_2, Z_2)$ can be computed as follows:

$$\begin{aligned} X_2 &= \alpha_2^2 - 2\beta_2, \\ Y_2 &= \alpha_2(\beta_2 - X_2) - 8Y_1^4, \\ Z_2 &= 2Y_1Z_1 \end{aligned}$$

where

$$\begin{aligned} \alpha_2 &= 3X_1^2 + aZ_1^4, \\ \beta_2 &= 4X_1Y_1^2 \end{aligned}$$

The cost of point doubling is $4M + 6S$ where M and S are cost of field multiplication and field squaring respectively. In the special case of $a = -3$, α_2 can be computed more efficiently as follows:

$$\alpha_2 = 3X_1^2 + aZ_1^4 = 3(X_1 + Z_1^2)(X_1 - Z_1^2)$$

We shall refer to point doubling that uses the above equation as *fast point doubling*. Accordingly, the cost of fast point doubling is decreased to $4M + 4S$.

Tripling in Jacobian Coordinates

Let $P_1 = (X_1, Y_1, Z_1)$ be a point in Jacobian coordinates on the elliptic curve E. Point tripling $[3]P_1 = (X_3, Y_3, Z_3)$ can be computed as follows:

$$\begin{aligned} X_3 &= 8Y_1^2(\beta_3 - \alpha_3) + X_1\omega_3^2, \\ Y_3 &= Y_1[4(\alpha_3 - \beta_3)(2\beta_3 - \alpha_3) - \omega_3^3], \\ Z_3 &= 2Z_1\omega_3 \end{aligned}$$

where

$$\begin{aligned} \alpha_3 &= \theta_3\omega_3, \\ \beta_3 &= 8Y_1^4, \\ \theta_3 &= 3X_1^2 + aZ_1^4, \\ \omega_3 &= 12X_1Y_1^2 - \theta_3^2 \end{aligned}$$

The cost of point tripling is $9M + 7S$ where M and S are cost of field multiplication and field squaring respectively. Again, in the special case of $a = -3$, θ_3 can be computed more efficiently as follows:

$$\theta_3 = 3X_1^2 + aZ_1^4 = 3(X_1 + Z_1^2)(X_1 - Z_1^2)$$

We shall refer to point tripling that uses the above equation as *fast point tripling*. Therefore, the cost of fast point tripling is decreased to $9M + 5S$.

2.1.2 Twisted Hessian Curves

An elliptic curve E over a field K in Hessian form is defined by the following equation (see also [JQ01] [LWX09] [Sma01])

$$u^3 + v^3 + 1 = 3Duv \tag{2.4}$$

where $D \in K$ and $D^3 \neq 1$.

If using projective coordinates, the Equation (2.4) can be rewritten as

$$U^3 + V^3 + W^3 = 3DUVW \tag{2.5}$$

where $D \in K$ and $D^3 \neq 1$.

There exists a generalization form of Hessian curve which is the *twisted* Hessian curve. Twisted Hessian curve E over a field K is defined by the following equation (see also [BL])

$$ax^3 + y^3 + 1 = dxy \tag{2.6}$$

where $a, d \in K$.

If using projective coordinates, the Equation (2.6) can be rewritten as

$$aX^3 + Y^3 + Z^3 = dXYZ \tag{2.7}$$

where $x = \frac{X}{Z}$, $y = \frac{Y}{Z}$, and $a, d \in K$.

Please note that Hessian curve is a special case of twisted Hessian curve where parameter $a = 1$. Consider the twisted Hessian curve in the Equation (2.6), if a has a cube root in K then there exists b such that $a = b^3$. It is obvious to see that $b^3 \cdot x^3 = a \cdot x^3$. By defining $t = b \cdot x$, the Equation (2.6) can be rewritten into

$$t^3 + y^3 + 1 = dxy \tag{2.8}$$

According to the fact that Hessian curve is birationally equivalent to the Weierstrass curve, twisted Hessian curve is birationally equivalent to the Weierstrass curve as well (see also [JQ01] [LWX09] [Sma01]). Hessian curve given by the Equation (2.4) is birationally equivalent to the Weierstrass equation

$$y^2 + x^3 - 27D(D^3 + 8)x + 54(D^6 - 20D^3 - 8) \tag{2.9}$$

under transformations

$$(u, v) = (\eta(x + 9D^2), -1 + \eta(3D^3 - Dx - 12)) \tag{2.10}$$

and

$$(x, y) = (-9D^2 + \xi u, 3\xi(v - 1)) \tag{2.11}$$

where $\eta = \frac{6(D^3(y+9D^3-3Dx-36))}{(x+9D^2)63+(3D^3-Dx-12)^3}$ and $\xi = \frac{12(D^3-1)}{Du+v+1}$.

Twisted Hessian Point Doubling in Projective coordinates

Assume $i^2 = -1$ and let $P_1 = (X_1, Y_1, Z_1)$ be a point in projective coordinates on twisted Hessian curve E. Point doubling $[2]P_1 = (X_2, Y_2, Z_2)$ can be computed as follows:

$$\begin{aligned} X_2 &= -2X_1\theta, \\ Y_2 &= (\theta_2 - \omega_2)Z_1, \\ Z_2 &= (\theta_2 + \omega_2)Y_1, \\ \text{where} & \\ \alpha_2 &= (Y_1 - iZ_1)(Y_1 + iZ_1), \\ \beta_2 &= Y_1Z_1, \\ \gamma_2 &= (\alpha_2 - \beta_2)(Y_1 + Z_1), \\ \theta_2 &= (\alpha_2 + \beta_2)(Z_1 - Y_1), \\ \omega_2 &= 3\gamma_2 - 2dX_1\theta_2 \end{aligned}$$

The cost of point doubling is $8M$ where M is the cost of field multiplication.

2.1.3 Double-and-add Algorithm

Scalar multiplication is one of the main operations in elliptic curve cryptography. This operation computes

$$nP = P + \cdots + P \ (n \text{ times})$$

for a given point P on the elliptic curve E and a secret scalar n. A commonly used method for computing scalar multiplication is *double-and-add* algorithm which generally consists of two point arithmetic operations, namely, point doubling and point addition. However, if the scalar n is expressed in double-base chain representation [DIM08] [DI08] [DIM05b] [DI06] [DIM05a] [DJM99] using mixed power of two and three, algorithms computing scalar multiplication also involve another point arithmetic operation called point tripling.

Double-and-add algorithm has two major varieties of direction when performing the computation, that is, either computing from left to right or computing from right to left. Selection on which direction shall be used partly depends upon whether it is easier to scan a bit from left to right or from right to left. Generally, the left-to-right double-and-add algorithm is used because it requires less memory.

Giving a point P on an elliptic curve E and a scalar n of length $k-1$ bits represented in binary representation, scalar multiplication nP scanning bits from left to right using double-and-add method can be computed as shown in Algorithm 1.

Similarly, Algorithm 2 shows double-and-add algorithm reading bits from right to left for computing scalar multiplication nP given a point P on an elliptic curve E and a scalar n of length $k-1$ bits represented in binary representation.

Algorithm 1 Left-to-right double-and-add algorithm [HMV03]

Input: $n = (n_{k-1}, \ldots, n_1, n_0)_2, P \in E(\mathbb{F}_q)$
Output: nP

```
Q ← O
for i = k − 1 downto 0 do
    Q ← 2Q
    if n_i = 1 then
        Q ← Q + P
    end if
end for
return (Q)
```

Algorithm 2 Right-to-left double-and-add algorithm [HMV03]

Input: $n = (n_{k-1}, \ldots, n_1, n_0)_2, P \in E(\mathbb{F}_q)$
Output: nP

```
Q ← O
for i = 0 to k − 1 do
    if n_i = 1 then
        Q ← Q + P
    end if
    P ← 2P
end for
return (Q)
```

2.1.4 Operation Cost

Point addition (ADD) and point doubling (DBL) are two main operations that usually execute when computing scalar multiplication. If double-base representation using base (2,3) is used, point tripling (TPL) is also usually computed. In this thesis, we considered two commonly used fields, namely, prime field and binary field. Cost of field arithmetic operation of prime field [DIM05b] [DIM08] [MD07] and binary field [DIM08] is summarized in Table 2.1 and Table 2.2 respectively. Superscript $\mathcal{J}$ denotes Jacobian coordinate while superscript $\mathcal{J}+\mathcal{A}$ denotes mixed Jacobian-affine coordinates, i.e., $\text{ADD}^{\mathcal{J}+\mathcal{A}}$ means adding one point in Jacobian coordinates to the other point in affine coordinates. Since the cost of field addition is significantly less than that of field multiplication M and field squaring S, we only considered the cost of the last two.

Table 2.1: Operation cost for curves defined over prime field

Point operation	Field operation cost
$\text{ADD}^{\mathcal{J}}$	$12M+4S$
$\text{ADD}^{\mathcal{J}+\mathcal{A}}$	$8M+3S$
$\text{DBL}^{\mathcal{J}}$	$4M+6S$
$\text{TPL}^{\mathcal{J}}$	$10M+6S$

Table 2.2: Operation cost for curves defined over binary field

Point operation	Field operation cost
$\text{ADD}^{\mathcal{J}}$	$16M+3S$
$\text{ADD}^{\mathcal{J}+\mathcal{A}}$	$11M+3S$
$\text{DBL}^{\mathcal{J}}$	$5M+5S$
$\text{TPL}^{\mathcal{J}}$	$15M+7S$

Consecutive Operations

Focusing on the double-base chain representation using base (2,3), point doubling and point tripling operations are usually and repeatedly executed. Considering Jacobian coordinates, Cohen, Miyaji, and Ono explained in [CMO98] that when point doubling ($\text{DBL}^{\mathcal{J}}$) is computed consecutively, i.e., doubling a point for w' times (w'-$\text{DBL}^{\mathcal{J}}$), some intermediate values from prior point doubling can be reused so that each of the following point doubling can be saved up to two field squarings. Dimitrov, Imbert, and Mishra showed in [DIM05b] [DIM08] that when point tripling ($\text{TPL}^{\mathcal{J}}$) is computed consecutively, i.e., tripling a point for w times (w-$\text{TPL}^{\mathcal{J}}$) or when point doubling is computed after point tripling, by using intermediate values from previous computations, one field squaring can be reduced. Table 2.3 summarizes the cost of point doubling and point tripling in both individual and consecutive cases for curves defined over prime field [DIM05b] [DIM08].

Table 2.3: Consecutive operations cost for curves defined over prime field

Point operation	Field operation cost
$\text{DBL}^{\mathcal{J}}$	$4M+6S$
w'-$\text{DBL}^{\mathcal{J}}$	$(4w')M+(4w'+2)S$
$\text{TPL}^{\mathcal{J}}$	$10M+6S$
w-$\text{TPL}^{\mathcal{J}}$	$(11w-1)M+(4w+2)S$
w-$\text{TPL}^{\mathcal{J}}/w'$-$\text{DBL}^{\mathcal{J}}$	$(11w+4w'-1)M+(4w+4w'+3)S$

Multiplication-Squaring Trade-off

Squaring can be thought of a special case of multiplication where both multiplicand and multiplier are the same. Generally, field squaring can be computed with less cost than field multiplication. It is sometimes assumed $S = 0.8M$ or $S = 0.67M$ where S and M are the cost of field squaring and field multiplication respectively.

Longa and Miri [LM08] noticed that multiplication can be converted into squaring by using the equality of square of the sum of two numbers. That is, they considered the following equality:

$$ab = \frac{1}{2}[(a+b)^2 - a^2 - b^2] \tag{2.12}$$

where a, b are elements in the underlying field.

They observed that if the value of a^2 and b^2 have already been calculated at some prior steps, a multiplication of ab can be computed from a squaring of $(a+b)^2$ and use the equality above to obtain the value of ab. By using this conversion, they proposed faster point addition, faster point doubling, and faster point tripling formulas in [LM08]. Sometimes, those formulas are called *fast addition*, *fast double*, and *fast triple* respectively.

We shall now refer to the above multiplication-squaring conversion technique as *M-S trade-off*.

2.2 Double-Base Number System

In double-base number system (DBNS) [DJM99] [DI06], a number n can be represented as a summation of mixed power of two co-prime integers as follows:

$$n = \sum_i d_i p^{a_i} q^{b_i} \tag{2.13}$$

where $d_i \in \{-1, 1\}$, $\gcd(p, q) = 1$, and $a_i, b_i \geq 0$.

Representation in double-base number system is obviously not unique. The following example shows how to write 314159 in double-base format using mixed power of 2 and 3:

$$\begin{aligned} 314159 &= 2^7 3^7 + 2^{10} 3^3 + 2^1 3^8 + 2^2 3^1 + 2^1 \\ &= 2^4 3^9 - 2^8 3^1 - 1 \end{aligned}$$

One advantage of this representation over a single-base format is that fewer number of terms in the expansion can be achieved. This is also the reason why the double-base number system has recently gained much attention in a research area of elliptic curve cryptography.

Double-Base Chain Representation

If extra restrictions over the exponent requiring that $a_1 \geq a_2 \geq \cdots \geq a_\ell$ and $b_1 \geq b_2 \geq \cdots \geq b_\ell$ are also considered, this special case of DBNS is called double-base *chain* which was introduced in [DIM05b] [DI06] [DIM05a]. An example of double-base chain representation is as follows:

$$314159 = 2^4 3^9 - 3^6 - 3^3 - 3^2 - 1$$

Double-base chain representation is more desirable since it allows better algorithm to compute $[n]P$ and Horner-like method can also be applied. Below is an example of how to compute $[314159]P$ using Horner's method.

$$[314159]P = [3^2]([3^1]([3^3]([2^4 3^3]P - P) - P) - P) - P$$

To use double-base number system, the original integers need to be converted into double-base format. Algorithm 3 is an example of algorithms (see also [DIM08] [DIM05b]) converting an integer n into (2,3) double-base *chain* representation.

Algorithm 3 Algorithm converting an integer n into $(2, 3)$ double-base *chain* representation

Input: $n > 0$ and a_{max}, b_{max}[1] > 0
Output: sequence (d_i, a_i, b_i) such that $n = \sum_i d_i 2^{a_i} 3^{b_i}$ with $a_1 \geq a_2 \geq \cdots \geq a_\ell$ and $b_1 \geq b_2 \geq \cdots \geq b_\ell$

```
d ← 1
while n > 0 do
  r ← 2^a 3^b, best approximation of n with 0 ≤ a ≤ a_max and 0 ≤ b ≤ b_max
  a_max ← a, b_max ← b
  print (d, a, b)
  if r > n then
    d ← −1
  end if
  n ← |n − r|
end while
```

[1] a_{max} and b_{max} are the maximum values allowed for binary and ternary exponents respectively

2.3 Side-Channel Analysis and Countermeasures

To break a cryptosystem, apart from investigating mathematical or theoretical weaknesses of algorithms, there exists another type of attack which analyzes information leakage through physical devices. This kind of attack is referred to as *side-channel attack* or *side-channel analysis* or simply *SCA* for short (see also [Ava05] [Joy03] [KGSL10] [CJ03] [EBJ02] [TMX09] [MMM04] [IIT06] [MV06] [FGM+10] [FV12] [JY02] [BJ03]). Examples of side-channel analysis include power analysis [KY10] [KHM+05] [CJ01], doubling attack [YKMJ06] [FV03], and zero-value point attack [AT03].

Simple Side-Channel Analysis refers to processes where only a single input is used to obtain side-channel information and to derive the underlying secret. Since simple SCA observes only once and tries to interpret the meaning of those data acquired, typical countermeasure for this type of attack is to make algorithms behave indistinguishably regardless of the actual underlying operations, for instance, adding dummy operations [CMCJ04], using unified formulas [BL09].

Differential Side-Channel Analysis, on the other hand, uses several inputs together with statistical techniques to analyze patterns observed from the leakage. Examples of countermeasure for this type of attack are randomized projective coordinates [Cor99] and randomized elliptic curve isomorphisms [JT01]. The former has Z coordinate of input points randomized; hence mixed addition cannot be applied but fast doubling still possible. The latter has a randomized; therefore using fast doubling is not possible and must use the general one instead.

2.3.1 Side-Channel Atomicity

Chevallier-Mames, Ciet, and Joye [CMCJ04] proposed a new method to prevent simple side-channel analysis by introducing a concept of *side-channel atomicity*. In contrast to a commonly used approach which tries to make the entire process indistinguishable from one other, their new approach divides each process into small indistinguishable block of instructions called *side-channel atomic block*. Then, the original process is rewritten in a form of sequences of indistinguishable block of instructions. The atomic block structure proposed by Chevallier-Mames, Ciet, and Joye in [CMCJ04] is as follows:

$$\begin{aligned} T_1 &\leftarrow T_2 \cdot T_3 \\ T_4 &\leftarrow T_5 + T_6 \\ T_7 &\leftarrow -T_8 \\ T_9 &\leftarrow T_{10} + T_{11} \end{aligned}$$

By rewriting point operation using the above atomic block, point addition requires 16 blocks while point doubling requires 10 blocks. One drawback of this atomic pattern is that it does not contain field squaring which means field squaring has to be computed at the cost of field multiplication.

Dimitrov, Imbert, and Mishra [DIM08] applied this atomic block to their new formulas for (consecutive) point tripling(s). They showed that point tripling requires 16 blocks or 15 blocks in case of consecutive point triplings.

Longa and Miri [LM08] improved the pattern proposed by Chevallier-Mames, Ciet, and Joye [CMCJ04] and suggested the following pattern:

$$\begin{aligned} T_1 &\leftarrow T_2^2 \\ T_3 &\leftarrow -T_4 \\ T_5 &\leftarrow T_6 + T_7 \\ &* \\ T_8 &\leftarrow T_9 \cdot T_{10} \\ T_{11} &\leftarrow -T_{12} \\ T_{13} &\leftarrow T_{14} + T_{15} \\ T_{16} &\leftarrow T_{17} + T_{18} \end{aligned}$$

Dummy field addition is inserted at $*$ if point tripling is also involved.

With this improved atomic pattern, they showed that mixed point addition, fast point doubling, and fast point tripling require only 6, 4, and 8 atomic blocks respectively. Please note that they introduced field squaring into their pattern. This comes with an advantage that field squaring is no longer need to be executed at an expensive cost of field multiplication. Instead, field squaring can be executed at its own cost of field squaring which is cheaper.

2.3.2 Atomicity in Embedded Devices

It is generally assumed that the cost of field addition is relatively small and negligible compared to both the cost of field squaring and field multiplication. However, in a context of resource constrained system or embedded devices, this assumption is no longer true. Giraud and Verneuil described that the time ratio when performing multiplication and addition can be as large as 0.3. Table 2.4 shows addition-multiplication ratio or A/M corresponding to the bit-length used [GV10].

Table 2.4: A/M ratio for different bit-length

Bit-length	A/M
160	0.36
192	0.30
224	0.25
256	0.22
320	0.16
384	0.13
512	0.09
521	0.09

Under the concern of non-negligible cost of field addition, Giraud and Verneuil [GV10] derived new atomic pattern for embedded devices. Their proposed pattern is as follows:

$$
\begin{aligned}
T_1 &\leftarrow T_2^2 \\
T_3 &\leftarrow T_4 + T_5 \\
T_6 &\leftarrow T_7 \cdot T_8 \\
T_9 &\leftarrow T_{10} + T_{11} \\
T_{12} &\leftarrow T_{13} \cdot T_{14} \\
T_{15} &\leftarrow T_{16} + T_{17} \\
T_{18} &\leftarrow T_{19} \cdot T_{20} \\
T_{21} &\leftarrow T_{22} + T_{23} \\
T_{24} &\leftarrow T_{25} + T_{26} \\
T_{27} &\leftarrow T_{28}^2 \\
T_{29} &\leftarrow T_{30} \cdot T_{31} \\
T_{32} &\leftarrow T_{33} + T_{34} \\
T_{35} &\leftarrow T_{36} - T_{37} \\
T_{38} &\leftarrow T_{39} \cdot T_{40} \\
T_{41} &\leftarrow T_{42} - T_{43} \\
T_{44} &\leftarrow T_{45} - T_{46} \\
T_{47} &\leftarrow T_{48} \cdot T_{49} \\
T_{50} &\leftarrow T_{51} - T_{52}
\end{aligned}
$$

Please notice that their atomic pattern is quite large compared to the previously proposed atomic patterns. With this new pattern, point addition requires 2 blocks while point doubling in modified Jacobian projective coordinates [CMO98] requires only 1 block.

2.4 Chapter Summary

In this chapter, we reviewed some background on elliptic curve cryptography and related mathematics. We also described previous works concerning our study.

We began with the notation of elliptic curve. Then, we explained point arithmetic formulas for different curve forms and coordinate systems. We especially provided the explanation on twisted Hessian curve and point doubling formulas

for this curve form. We also discussed double-and-add algorithm which used for computing scalar multiplication. To give an idea of computational time, we also presented cost of arithmetic operation.

Next, we described double-base number system and double-base chain representation along with examples of how numbers can be represented in those forms. Algorithm converting numbers into (2,3) double-base chain representation was also described as well.

We finished this chapter with a discussion on security concern over an analysis of leakage via physical devices. Explanation on countermeasure against this analysis such as side-channel atomicity and previously proposed atomic blocks were also included.

Chapter 3

Faster Point Doubling on Twisted Hessian Curves

In this chapter, we described how point doubling formulas on twisted Hessian curves in projective coordinates can be modified to make doubling operation faster. First, we stated our observations over dbl-2009-bkl-3 doubling formulas together with our approach on how to reduce the operation cost. Then, our new point doubling formulas were presented. We ended this chapter with cost comparison and analysis.

3.1 Overview

It is very common to represent elliptic curve in terms of Weierstrass equations. However, the same elliptic curve can be represented in different curve forms and/or coordinate systems. Each representation has different advantages and disadvantages. For instance, Hessian curve has an advantage of allowing the same algorithm to be used for both point addition and point doubling [JQ01]. Twisted Hessian is a generalization of Hessian curve that allow faster point addition and point doubling than curves in Weierstrass form. The Explicit-Formulas Database [BL] contains many point operation formulas on various forms of curve shapes and coordinates. This database also keeps the record of best-operation-count for each curve form and coordinate system.

Bernstein, Kohel, and Lange [BL] proposed the dbl-2009-bkl-3 doubling formulas for twisted Hessian curves in projective coordinates. These formulas cost $8M + 1 * minustwo + 1 * i + 1 * 2d + 9add + 1 * 3$. That is, these formulas require 8 field multiplications, 1 multiplication by $minustwo$ where $minustwo = -2$, 1 multiplication by i under the assumption that $i^2 = -1$, 1 multiplication by $2d$ where $2d = 2*d$ and d is a curve parameter, 9 field additions, and 1 multiplication by constant 3.

Our study revealed that the assumption $i^2 = -1$ can be removed and formulas can be modified to result in a faster point doubling. To be more precisely, we derived formulas which cost only $7M+1S+1*minustwo+1*d+7add+1*2+1*3$. That is, these new formulas require 7 field multiplications, 1 field squaring, 1 multiplication by $minustwo$ where $minustwo = -2$, 1 multiplication by d where d is a curve parameter, 7 field additions, 1 multiplication by constant 2, and 1 multiplication by constant 3. We would like to emphasize that in our formulas, one expensive field multiplication is traded into one cheaper field squaring, amount of field addition drops from 9 to 7, and the assumption concerning i is no longer required.

To calculate a cost per bit reduction, we assumed point doubling is performed every bit while point addition is performed one-third of the bit length. Our

new formulas reduced a cost per bit by 1.67% compared to the dbl-2009-bkl-3 doubling formulas assuming $S = 0.8M$ where S and M are the cost of field squaring and field multiplication respectively and neglecting the cost of point addition, multiplication by curve parameter, and multiplication by constant. In case of assuming $S = 0.67M$, cost per bit is decreased by 2.75%.

3.2 Observation and Our Approach

We studied point doubling formulas on twisted Hessian curves in projective coordinates proposed by Bernstein, Kohel, and Lange [BL]. The dbl-2009-bkl-2 formulas cost $6M + 3S + 1 * a$ but require no extra assumption while the dbl-2009-bkl-3 formulas cost $8M+1*minustwo+1*i+1*2d$ but require assumptions that $i^2 = -1$, $minustwo = -2$, and $2d = 2 * d$ where d is a curve parameter. Our focus was on the latter one, namely, the dbl-2009-bkl-3 formulas.

Each step in the dbl-2009-bkl-3 formulas was investigated. We carefully analyzed values obtained during the computation before reaching the desired solution of point doubling. We shall now refer to those values as *intermediate values*.

For example, to compute $(Y_1 - iZ_1)(Y_1 + iZ_1)$, we first need to compute iZ_1. Then, we minus iZ_1 from Y_1 and add iZ_1 to Y_1 to obtain $Y_1 - iZ_1$ and $Y_1 + iZ_1$ respectively. Finally, we multiply $(Y_1 - iZ_1)$ and $(Y_1 + iZ_1)$. In this example, iZ_1, $Y_1 - iZ_1$, and $Y_1 + iZ_1$ are intermediate values. The value iZ_1 is obtained by multiplying Z_1 by i while the value $(Y_1 - iZ_1)$ and $(Y_1 + iZ_1)$ are obtained by doing subtraction/addition between Y_1 and iZ_1.

We observed that it is possible to rewrite some parts of the dbl-2009-bkl-3 formulas so that one computation of field multiplication can be changed into one computation of field squaring. This trade-off is achieved by reusing intermediate values occurred during other steps in point doubling computation. Recall point doubling in projective coordinates for twisted Hessian curves in Subsection 2.1.2, it is assumed $i^2 = -1$ and α_2 is defined as:

$$\alpha_2 = (Y_1 - iZ_1)(Y_1 + iZ_1) \tag{3.1}$$

In the above equation, α_2 is actually equal to $Y_1^2 + Z_1^2$. The following shows the details:

$$\begin{aligned}
\alpha_2 &= (Y_1 - iZ_1)(Y_1 + iZ_1) \\
&= (Y_1)(Y_1) + (Y_1)(iZ_1) - (Y_1)(iZ_1) - (iZ_1)(iZ_1) \\
&= (Y_1)(Y_1) - (iZ_1)(iZ_1) + (Y_1)(iZ_1) - (Y_1)(iZ_1) \\
&= (Y_1)(Y_1) - (iZ_1)(iZ_1) + [(Y_1)(iZ_1) - (Y_1)(iZ_1)] \\
&= (Y_1)(Y_1) - (iZ_1)(iZ_1) \\
&= Y_1^2 - (i^2)(Z_1^2) \\
&= Y_1^2 - (-1)(Z_1^2) \\
&= Y_1^2 + Z_1^2
\end{aligned}$$

It is not difficult to see that computing α_2 according to the formula $\alpha_2 = (Y_1 - iZ_1)(Y_1 + iZ_1)$ requires 2 field additions, 1 multiplication by i, and 1 field multiplication; while computing α_2 using the formula $\alpha_2 = Y_1^2 + Z_1^2$ requires 1 field addition and 2 field squarings. At first glance, it may not seem to gain any benefits from the rewritten formula. However, we can reuse some intermediate values and change 1 field multiplication into 1 field squaring.

By using the equality of square of the sum of two numbers:

$$(a+b)^2 = a^2 + 2ab + b^2$$

This allows us to compute $\alpha_2 = Y_1^2 + Z_1^2$ as:

$$\begin{aligned}\alpha_2 &= Y_1^2 + Z_1^2 \\ &= Y_1^2 + Z_1^2 + [2Y_1Z_1 - 2Y_1Z_1] \\ &= [Y_1^2 + Z_1^2 + 2Y_1Z_1] - 2Y_1Z_1 \\ &= (Y_1 + Z_1)^2 - 2Y_1Z_1\end{aligned}$$

Recall Subsection 2.1.2, β_2 is defined as:

$$\beta_2 = Y_1Z_1$$

The value $\beta_2 = Y_1Z_1$ is used in other steps during point doubling, and it has to be computed anyway whether or not we compute α_2 by $\alpha_2 = (Y_1 - iZ_1)(Y_1 + iZ_1)$ or by $\alpha_2 = Y_1^2 + Z_1^2$. This means that we can reuse β_2 when computing α_2. That is, α_2 can be computed as:

$$\alpha_2 = (Y_1 + Z_1)^2 - 2\beta_2 \tag{3.2}$$

To express the computational cost, we counted how many sub-operations are necessary to compute α_2 if the Equation 3.2 is used. In this case it requires 2 additions, 1 multiplication by constant 2, and 1 squaring. Compared to the cost in Equation 3.1 which requires 2 field additions, 1 multiplication by i, and 1 field multiplication, we changed 1 multiplication by i to multiplication by constant 2 and 1 field multiplication to 1 field squaring. Please notice that our formulas do not contain i. This means that we removed the necessary of defining i to meet the assumption $i^2 = 1$.

3.3 New Point Doubling Formulas

We derived new point doubling formulas by using the equality of square of the sum of two numbers and reusing intermediate values. Our new formulas are significantly faster than the previously proposed formulas if field squaring can be computed faster than field multiplication. This condition generally holds in most cases since it is usually assumed $S = 0.8M$ or $S = 0.67M$ where S and M are cost of field squaring and field multiplication respectively.

The dbl-2012-c doubling formulas

By using the Equation 3.2, we were able to change one field multiplication into field squaring. However, other costs, such as field addition, stay unchanged. We especially thank Daniel J. Bernstein for revising and suggesting further improvements. As a result, 2 more field additions were eliminated. The revised formulas are as follows: [1]

[1] We thank Daniel J. Bernstein for revising and putting the formulas into Explicit-Formulas Database. The revised formulas and codes are available at `http://hyperelliptic.org/EFD/g1p/data/twistedhessian/projective/doubling/dbl-2012-c`.

Let $P_1 = (X_1, Y_1, Z_1)$ be a point in projective coordinates on twisted Hessian curve E. Point doubling $[2]P_1 = (X_2, Y_2, Z_2)$ can be computed as follows:

$$\begin{aligned} X_2 &= -2X_1\theta_2 \\ Y_2 &= (\theta_2 - \omega_2)Z_1 \\ Z_2 &= (\theta_2 + \omega_2)Y_1 \end{aligned}$$

where

$$\begin{aligned} \eta_2 &= Y_1 + Z_1, \\ \alpha_2 &= \eta_2^2 - \beta_2, \\ \beta_2 &= Y_1 Z_1, \\ \gamma_2 &= (\alpha_2 - 2\beta_2)\eta_2, \\ \theta_2 &= \alpha_2(Z_1 - Y_1), \\ \omega_2 &= 3\gamma_2 - dX_1(2\beta_2) \end{aligned}$$

We shall refer to the above formulas as the "dbl-2012-c" doubling formulas.

3.4 Comparison and Analysis

It is commonly described a cost of point operation by the number of field operations required. We also used the same method to express the cost of our new point doubling formulas.

Table 3.1 shows the amount of arithmetic operations required for the dbl-2012-c formulas. We categorized into the number of field multiplication, field squaring, field addition, and multiplication by constant.

Table 3.1: Operation cost analysis of the dbl-2012-c formulas

Operation	Cost			
	Multiplication	Squaring	addition	constant
$\eta_2 = Y_1 + Z_1$	–	–	1	–
$\alpha_2 = \eta^2 - \beta_2$	–	1	1	–
$\beta_2 = Y_1 Z_1$	1	–	–	–
$\gamma_2 = (\alpha_2 - 2\beta_2)\eta_2$	1	–	1	1
$\theta_2 = \alpha_2(Z_1 - Y_1)$	1	–	1	1
$\omega_2 = 3\gamma_2 - dX_1(2\beta_2)$	1	–	1	1
$X_2 = -2X_1\theta_2$	1	–	–	1
$Y_2 = (\theta_2 - \omega_2)Z_1$	1	–	1	–
$Z_2 = (\theta_2 + \omega_2)Y_1$	1	–	1	–
SUM	7	1	7	4

The dbl-2012-c doubling formulas cost $7M + 1S + 1 * minustwo + 1 * d + 7add + 1 * 2 + 1 * 3$. That is, these formulas use 7 field multiplications, 1 field squaring, 1 multiplication by $minustwo$ where $minustwo = -2$, 1 multiplication by d where d is a curve parameter, 7 field additions, 1 multiplication by constant 2, and 1 multiplication by constant 2.

To illustrate improvements achieved by our new point doubling formulas, we compared our formulas to other twisted Hessian point doubling formulas. In other words, we counted the number of field operations used in our formulas and compared to the previously proposed ones.

According to the Explicit-Formulas Database [BL], there are currently three point doubling formulas available for twisted Hessian curve in projective coordinates. One of them considers cubing operation which is not in our context of study. Therefore, we only compared our formulas to the other two, namely, dbl-2009-bkl-2 and dbl-2009-blk-3 formulas.

Table 3.2 shows operation count for each twisted Hessian point doubling formula. We denoted our new formulas by dbl-2012-c and the previously proposed formulas by dbl-2009-bkl-2 and dbl-2009-blk-3. To keep the table compact, we abbreviated field multiplication, field squaring, field addition, multiplication by constant, and multiplication by curve parameter to Mul, Sqr, add, const, and param respectively.

Table 3.2: Operation count on twisted Hessian point doubling formulas

Formula	Mul	Sqr	add	const	param	(assumption)
dbl-2012-c	7	1	7	2	2	–
dbl-2009-bkl-3 [BL]	8	–	9	1	3	$i^2 = -1$
dbl-2009-bkl-2 [BL]	6	3	3	–	1	–

Generally, cost of field multiplication and field squaring are significantly larger than cost of field addition and multiplication by constant or curve parameter. Thus, we ignored the cost of field addition and multiplication by constant or curve parameter when we compared our formulas with other formulas for different assumptions on the cost of field multiplication and field squaring.

Table 3.3 shows cost comparison between our formulas, dbl-2012-c, and previously proposed formulas, dbl-2009-bkl-2 and dbl-2009-blk-3. This comparison considered three cases of $S = M$, $S = 0.8M$, and $S = 0.67M$ where S and M designate the cost of field squaring and field multiplication respectively.

Table 3.3: Twisted Hessian point doubling cost comparison

Formula	$S = M$	$S = 0.8M$	$S = 0.67M$
dbl-2012-c	$8M$	$7.8M$	$7.67M$
dbl-2009-bkl-3	$8M$	$8M$	$8M$
dbl-2009-bkl-2	$9M$	$8.3M$	$8.01M$

According to Table 3.3, it is quite unfortunate that our new formulas show insignificant improvement under the assumption $S = M$. That is, our formulas cost $8M$ which is the same as the best-operation-count of the previously proposed formulas. However, our new formulas achieved 2.5% and 4.125% cost reduction compared to dbl-2009-bkl-3 formulas under assumptions $S = 0.8M$ and $S = 0.67M$ respectively. In other word, our new formulas reduce cost per bit by 1.43% for $S = 0.8M$ and 2.36% for $S = 0.67M$ assuming point doubling is performed every bit while point addition is performed half of the bit length and cost of point addition is $12M$. In case of assuming point addition is performed only one-third of the bit length, the cost per bit reduction increases to 1.67% and 2.75% for $S = 0.8M$ and $S = 0.67M$ respectively.

3.5 Concluding Remarks

Almost every time that elliptic curve is mentioned, Weierstrass equation is also mentioned. However, the Weierstrass curve is not the only option to be used for elliptic curve. There exist other different curve shapes that are birationally equivalent to elliptic curve in Weierstrass form. The aim for faster point operation on elliptic curve has led to the investigation in arithmetic on other curve forms hoping that those alternative curves would allow more efficient point operation.

Among various kinds of curve shapes, we were particular interested in twisted Hessian curves on projective coordinates. Our study revealed that it is possible to rewrite point doubling formulas so that intermediate values occurred during calculation in prior steps can be reused. In this way, unnecessary computation of having to compute new values can be avoided if the already computed ones are efficiently utilized. This also leads to faster formulas.

Speeding up by reusing values is a method that does not require any additional cost. Our result remarked that significant improvement of removing redundant computation can be achieved if sequence of operations is properly arranged and previously computed values are well utilized.

Chapter 4

Double-Base Chain Scalar Multiplication Reordering

In this chapter, we explained our techniques, namely, reordering field arithmetic operation sequences and reusing prior computed intermediate values, applying to elliptic curve scalar multiplication represented in (2,3) double-base chain. We separately described our method for curves defined over prime field and binary field. Experimental results in both cases were presented at the end of the chapter.

4.1 Overview

Speed of any elliptic curve cryptography protocols is related to speed of computing scalar multiplication which is known to be a very time-consuming operation. One of the crucial factors that affects speed of scalar multiplication relies on the underlying layer called field arithmetic operation.

Dimitrov, Imbert, and Mishra [DIM08] showed how to reuse intermediate values to reduce some computations when operations on curves defined over prime field in Jacobian coordinates are executed consecutively. That is, when point tripling is computed consecutively (w-TPL$^{\mathcal{J}}$), except for the first point tripling, one field squaring can be saved for each of the following point tripling. Similarly, when computing consecutive point triplings followed by consecutive point doublings (w-TPL$^{\mathcal{J}}$/w'-DBL$^{\mathcal{J}}$), one field squaring can be saved for the first point doubling that follows point tripling.

According to our study, the sequences of field operation are crucial to how intermediate values can be reused. We found that by altering the order, i.e., computing consecutive point doublings *before* consecutive point triplings (w'-DBL$^{\mathcal{J}}$/w-TPL$^{\mathcal{J}}$), *more* intermediate values can be reused. In other words, two computations (compared to one in the previous work) of field squaring can be reduced if computing point tripling after point doubling. We would like to emphasize that computing *point tripling after point doubling* is the main difference between previous work and our work.

We conducted several experiments using $10,000$ randomly chosen 256-bit integers to evaluate improvement achieved by applying our new method of reordering sequences of field arithmetic operation. Our experimental results showed that for curves defined over prime field in Jacobian coordinates, 1.55% of a computational cost was reduced if computing point doubling *before* point tripling under an assumption that $S = 0.8M$ where S and M are the cost of field squaring and field multiplication respectively. In case of using mixed addition, the reduction increased to 1.71%. If we assume $S = M$ in order to prevent simple side-channel analysis, the reduction improved to 1.77% and 1.95% for general addition and mixed addition respectively.

Our technique of reusing intermediate value can also be applied together with the technique of converting expensive field multiplication into cheaper field squaring or M-S trade-off proposed by Longa and Miri [LM08]. That is, speed-up from two techniques, namely, M-S trade-off and reusing intermediate value, is obtainable at the same time.

In addition to curves defined over prime field, we also examined field arithmetic operation sequences on curves defined over binary field in Jacobian coordinates. Our study indicated that consecutive point triplings (w-TPL$^{\mathcal{J}}$) or consecutive point triplings before consecutive point doublings (w-TPL$^{\mathcal{J}}$/w'-DBL$^{\mathcal{J}}$) allows some intermediate values to be reused. In the former case, one computation of field squaring can be reduced for each of the following point tripling. In the latter case, one computation of field squaring can be reduced for point doubling that follows point tripling.

Our experimental results on curves defined over binary field in Jacobian coordinates showed reductions of 0.28% and 0.31% using general addition and mixed addition respectively. Even though these numbers suggested only small improvements, it is still interesting from a theoretical view point that the idea of reusing intermediate value is applicable for curves defined over binary field. It is true for general cases that cost of field squaring is relatively small compared to cost of field multiplication. However, advancement in technology nowadays has made multiplication faster which affects the assumption that field squaring is relatively free compared to field multiplication.

4.2 Elliptic Curves over Prime Field

For curves defined over prime field in Jacobian coordinates, [DIM08] explained that by computing point tripling *before* point doubling and reusing intermediate values, one computation of field squaring could be reduced. We observed that not only computing point tripling prior to point doubling, but also computing point doubling followed by point tripling can save field squaring. Our study revealed that by slightly reordering, namely, computing point tripling *after* point doubling, two computations of field squaring could be reduced. Figure 4.1 illustrates our idea of switching point doubling and point tripling sequences.

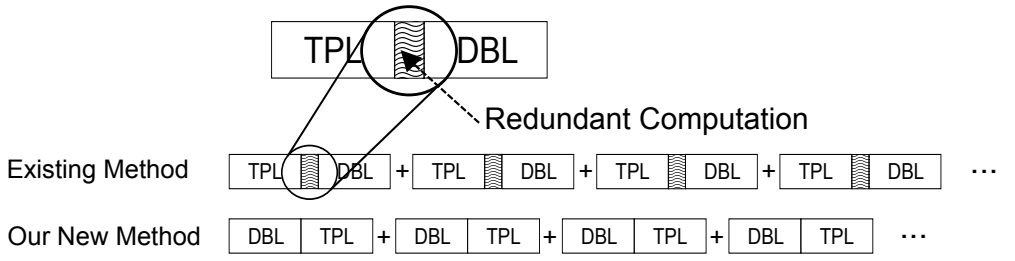

Figure 4.1: New sequence of point operations

We investigated point doubling and point tripling formulas for curves defined over prime field in Jacobian coordinates and analyzed values that have to be computed in each step before reaching the final solutions of point doubling and point tripling. We shall now refer to those values as *intermediate values*.

For example, to compute $2X_1Y_1^2$, we first need to compute Y_1^2. Then, we multiply X_1 by Y_1^2 to obtain $X_1Y_1^2$. Finally, we add $X_1Y_1^2$ to itself and get $2X_1Y_1^2$. In this example, Y_1^2 and $X_1Y_1^2$ are intermediate values. The value Y_1^2

is obtained by squaring Y_1 while the value $X_1Y_1^2$ is obtained by multiplying X_1 and Y_1^2 together.

Let $P_1 = (X_1, Y_1, Z_1)$ be a point in Jacobian coordinates on the elliptic curve E, and let $P_2 = [2]P_1 = (X_2, Y_2, Z_2)$ be the result of doubling point P_1. Table 4.1 summarizes intermediate values occurred during each step of computing point doubling. We categorized them into values obtained by field multiplication and field squaring.

Table 4.1: Intermediate values of point doubling for curves defined over $\mathbb{F}_p$

Operation	Intermediate Values	
	Multiplication	Squaring
$\alpha_2 = 3X_1^2 + aZ_1^4$	aZ_1^4	X_1^2, Z_1^2, Z_1^4
$\beta_2 = 4X_1Y_1^2$	$X_1Y_1^2$	Y_1^2
$X_2 = \alpha_2^2 - 2\beta_2$	–	α_2^2
$Y_2 = \alpha_2(\beta_2 - X_2) - 8Y_1^4$	$\alpha_2(\beta_2 - X_2)$	Y_1^4
$Z_2 = 2Y_1Z_1$	Y_1Z_1	–

According to Table 4.1, to compute α_2, it is necessary that aZ_1^4 is computed. Similarly, Y_1^4 needs to be computed in order to compute Y_2. These two intermediate values, namely, aZ_1^4 and Y_1^4, will appear again when point tripling is computed afterwards.

Let $P_3 = [3]P_2 = (X_3, Y_3, Z_3)$ be the result of tripling point P_2. Table 4.2 summarizes intermediate values occurred during each step of computing point tripling. We also categorized them into values obtained by field multiplication and field squaring.

Table 4.2: Intermediate values of point tripling for curves defined over $\mathbb{F}_p$

Operation	Intermediate Values	
	Multiplication	Squaring
$\theta_3 = 3X_2^2 + aZ_2^4$	aZ_2^4	X_2^2, Z_2^2, Z_2^4
$\omega_3 = 12X_2Y_2^2 - \theta_3^2$	$X_2Y_2^2$	Y_2^2, θ_3^2
$\alpha_3 = \theta_3\omega_3$	$\theta_3\omega_3$	–
$\beta_3 = 8Y_2^4$	–	Y_2^4
$X_3 = 8Y_2^2(\beta_3 - \alpha_3) + X_2\omega_3^2$	$Y_2^2(\beta_3 - \alpha_3), X_2\omega_3^2$	ω_3^2
$Y_3 = Y_2[4(\alpha_3 - \beta_3)(2\beta_3 - \alpha_3) - \omega_3^3]$	$(\alpha_3 - \beta_3)(2\beta_3 - \alpha_3), \omega_3^3,$ $Y_2[4(\alpha_3 - \beta_3)(2\beta_3 - \alpha_3) - \omega_3^3]$	–
$Z_3 = 2Z_2\omega_3$	$Z_2\omega_3$	–

In this work, we observed that the value $Z_2 = 2Y_1Z_1$ obtained when computing point doubling could be substituted into point tripling formulas when computing θ_3, i.e., computing θ_3 from $3X_2^2 + 16aZ_1^4Y_1^4$. Consequently, the intermediate values aZ_1^4 and Y_1^4 occurred during point doubling can be reused to save two computations of field squaring. That is, θ_3 can be computed using only one squaring for X_2^2 and one multiplication for $(aZ_1^4)(Y_1^4)$. Comparing to direct computation from point tripling formulas, it requires three squarings for X_2^2, Z_2^2, Z_2^4 and one multiplication for aZ_2^4. Table 4.3 shows our method of reusing intermediate values to reduce two computations of field squaring when point tripling is computed after point doubling.

Table 4.3: Reducing two squarings on curves defined over $\mathbb{F}_p$

Method	Operation	Intermediate Values	
		Multiplication	Squaring
Direct computation (Previous results) [DIM08]	$\theta_3 = 3X_2^2 + aZ_2^4$	aZ_2^4	$\boldsymbol{X_2^2}, \boldsymbol{Z_2^2}, Z_2^4$
Reusing Values (Our results)	$\theta_3{=}3X_2^2 + a[2Y_1Z_1]^4$ $\theta_3{=}3X_2^2 + 2^4 \cdot aZ_1^4Y_1^4$	$(aZ_1^4)(Y_1^4)$	X_2^2

Note that in [DIM08], they considered the cost of point tripling to be $10M+6S$ where S and M are the cost of field squaring and field multiplication respectively. However, there exist *revised* tripling formulas which cost $9M+7S$. We derived our method from these revised tripling formulas. Hence, we compared our results to the previous idea [DIM08] of computing point tripling *before* point doubling (w-TPL$^{\mathcal{J}}$/w'-DBL$^{\mathcal{J}}$) combining with the revised tripling formulas. Table 4.4 shows the cost comparison between previous work [DIM08] [LM08], namely, computing point tripling *before* point doubling (w-TPL$^{\mathcal{J}}$/w'-DBL$^{\mathcal{J}}$) with revised tripling formulas, and this work, namely, computing point tripling *after* point doubling (w'-DBL$^{\mathcal{J}}$/w-TPL$^{\mathcal{J}}$).

In Subsection 2.1.4, we mentioned M-S trade-off introduced by Longa and Miri [LM08]. This M-S trade-off changes one computation of field multiplication into one field squaring plus three extra field additions. By applying this idea of M-S trade-off, our results could further be improved.

Table 4.4 shows the cost comparison between previous results [DIM08], namely, computing point tripling *before* point doubling (w-TPL$^{\mathcal{J}}$/w'-DBL$^{\mathcal{J}}$), and our results, namely, computing point tripling *after* point doubling (w'-DBL$^{\mathcal{J}}$/w-TPL$^{\mathcal{J}}$), both with and without M-S trade-off [LM08].

Table 4.4: Cost comparison between different point operation sequence on curves defined over $\mathbb{F}_p$

	Cost
Previous results [DIM08] [1]	$(9w + 4w'M){+}(6w + 4w'{+}\mathbf{2})S$
Our results I [2]	$(9w + 4w'M){+}(6w + 4w'{+}\mathbf{1})S$
Our results II [3]	$(7w + 3w' - 1)M{+}(8w + 5w' + 2)S$

[1] (w-TPL$^{\mathcal{J}}$/w'-DBL$^{\mathcal{J}}$)
[2] (w'-DBL$^{\mathcal{J}}$/w-TPL$^{\mathcal{J}}$) without M-S trade-off
[3] (w'-DBL$^{\mathcal{J}}$/w-TPL$^{\mathcal{J}}$) with M-S trade-off

Example: The following explains how our method works compared to previous method [DIM08] [LM08]. In this example, we would like to compute $[1739]P$ from a given double-base chain:

$$1739 = 2^6 3^3 + 2^2 3^1 - 2^0 3^0 = [2^2 3^1]([2^4 3^2]P + P) - P$$

In previous work, $[1739]P$ would be computed as follows:

1. $[3]P$ is computed using $9M + 7S$, then
   $[3]([3]P)$ is computed using $9M + 6S$.
   Total cost of $18M + 13S$ is required at step 1.

2. $[2]([3^2]P)$ is computed using $4M + 5S$, then
   $[2]([2^1 3^2]P)$, $[2]([2^2 3^2]P)$, $[2]([2^3 3^2]P)$ is computed
   using $4M + 4S$ at each step.
   Total cost of $16M + 17S$ is required at step 2.

3. $[2^4 3^2]P + P$ is computed using $8M + 3S$.
   Total cost of $8M + 3S$ is required at step 3.

4. $[3^1]([2^4 3^2]P + P)$ is computed using $9M + 7S$.
   Total cost of $9M + 7S$ is required at step 4.

5. $[2^1]([3^1]([2^4 3^2]P + P))$ is computed using $4M + 5S$, then
   $[2^2]([2^1 3^1]([2^4 3^2]P + P))$ is computed using $4M + 4S$.
   Total of $8M + 9S$ is used at step 5.

6. $[2^2 3^1]([2^4 3^2]P + P) + P$ is computed using $8M + 3S$.
   Total cost of $8M + 3S$ is required at step 6.

Summation of field multiplication and field squaring needed from step 1 to 3 is $42M + 33S$. Including the rest of the steps, total cost of $67M + 52S$ is required.

On the other hand, in our method, $[1739]P$ would be computed as follows:

1. $[2]P$ is computed using $4M + 6S$, then
   $[2]([2]P)$, $[2]([2^2]P)$, $[2]([2^3]P)$ is computed
   using $4M + 4S$ at each step.
   Total cost of $16M + 18S$ is required at step 1.

2. $[3]([2^4]P)$ is computed using $9M + 5S$, then
   $[3]([2^4 3^1]P)$ is computed using $9M + 6S$.
   Total cost of $18M + 11S$ is required at step 2.

3. $[2^4 3^2]P + P$ is computed using $8M + 3S$.
   Total cost of $8M + 3S$ is required at step 3.

4. $[2]([2^4 3^2]P + P)$ is computed using $4M + 6S$, then
   $[2]([2]([2^4 3^2]P + P))$ is computed using $4M + 4S$ at each step.
   Total of $8M + 10S$ is used at step 4.

5. $[3^1]([2^4]([2^4 3^2]P + P))$ is computed using $9M + 5S$.
   Total cost of $9M + 5S$ is required at step 5.

6. $([2^2 3^1]([2^4 3^2]P + P) + P)$ is computed using $8M + 3S$.
   Total cost of $8M + 3S$ is required at step 6.

Summation of field multiplication and field squaring needed from step 1 to 3 is $42M + 32S$. Including the rest of the steps, total cost of $67M + 50S$ is required.

We can see that step 1 to 3 computes one term in double-base chain ($[2^4 3^2]P + P$ in this case). In these three steps, our method reduces one squaring; that is, $32S$ compared to $33S$. If the entire chain is computed, two squarings are reduced; that is, $50S$ compared to $52S$. This is because there are three terms in this expansion.

4.3 Elliptic Curves over Binary Field

We also investigated point doubling and point tripling formulas for curves defined over binary field in Jacobian coordinates and analyzed values that have to be computed in each step before reaching the final solutions of point doubling and point tripling. Our study discovered that by reusing intermediate values, one computation of field squaring can be reduced for either computing point tripling follows by point tripling or computing point tripling follows by point doubling. In other words, one field squaring can be omitted for each of the following point tripling in consecutive point triplings (w-TPL$^{\mathcal{J}}$), and the first point doubling that follows point tripling in consecutive point triplings then consecutive point doublings (w-TPL$^{\mathcal{J}}$/w'-DBL$^{\mathcal{J}}$).

Let $P_1 = (X_1, Y_1, Z_1)$ be a point in Jacobian coordinates on the elliptic curve E defined over binary field, let $P_2 = [2]P_1 = (X_2, Y_2, Z_2)$ be the result of doubling point P_1, and let $P_3 = [3]P_1 = (X_3, Y_3, Z_3)$ be the result of tripling point P_1. Point P_3 is computed by adding the doubling of point P_1 to the original point P_1, i.e., $P_3 = [2]P_1 + P_1$. Table 4.5 summarizes intermediate values occurred during each step of computing point tripling on curves defined over binary field in Jacobian coordinates. We categorized them into values obtained by field multiplication and field squaring.

We can see from Table 4.5 that γ_3^2 is calculated when computing α_3. By using the fact that $[3]P_1 = (X_3, Y_3, Z_3) = (\alpha_3 Z_1^6, \beta_3 Z_1^9, \gamma_3 Z_1^3) = (\alpha_3, \beta_3, \gamma_3)$, calculating γ_3^2 is equivalent to calculating Z_3^2. If point tripling or point doubling is computed afterwards (computing point tripling after point tripling or point doubling after point tripling), the value Z_3^2 will appear again. In such cases, we can reuse the value γ_3^2 so that computing Z_3^2 can be omitted.

Let $P_4 = [2]P_3 = (X_4, Y_4, Z_4)$ be the result of doubling point P_3, namely, computing point doubling after point tripling. Table 4.6 summarizes intermediate values occur during point doubling on curved defined over binary field in Jacobian coordinates. Since the reusing intermediate values concept in point tripling is similar to point doubling, we leave out the explanation for the case of point tripling.

Table 4.6: Intermediate values of point doubling on curves defined over $\mathbb{F}_{2^m}$

Operation			Intermediate Values	
			Multiplication	Squaring
X_4	=	$X_3^4 + a_6 Z_3^8$	$a_6 Z_3^8$	$X_3^2, X_3^4,$ Z_3^2, Z_3^4, Z_3^8
Y_4	=	$X_3^4 Z_4 + (X_3^2 + Y_3 Z_3 + Z_4) X_4$	$X_3^4 Z_4, Y_3 Z_3,$ $(X_3^2 + Y_3 Z_3 + Z_4) X_4$	–
Z_4	=	$X_3 Z_3^2$	$X_3 Z_3^2$	–

According to Table 4.6, the value Z_3^8 must be computed in order to obtain the value X_4. This implicitly implies that the value Z_3^2 needs to be calculated. We found that the value Z_3^2 does not need to be computed at this step because it has already been computed when computing γ_3^2. Table 4.7 shows how we reused intermediate values to reduce one computation of field squaring.

Table 4.5: Intermediate values of point tripling on curves over $\mathbb{F}_{2^m}$

Operation	Intermediate Values	
	Multiplication	Squaring
$X_2 = X_1^4 + a_6 Z_1^8$	$a_6 Z_1^8$	$X_1^2, X_1^4, Z_1^2, Z_1^4, Z_1^8$
$Y_2 = X_1^4 Z_2 + (X_1^2 + Y_1 Z_1 + Z_2) X_2$	$X_1^4 Z_2, Y_1 Z_1, (X_1^2 + Y_1 Z_1 + Z_2) X_2$	–
$Z_2 = X_1 Z_1^2$	$X_1 Z_1^2$	–
$\alpha_3 = a_2 \gamma_3^2 + \omega_3(\omega_3 + \gamma_3) + \theta_3^3$	$a_2 \gamma_3^2, \omega_3(\omega_3 + \gamma_3), \theta_3^3$	$\boldsymbol{\gamma_3^2}, \theta_3^2$
$\beta_3 = (\omega_3 + \gamma_3)\alpha_3 + \theta_3^2(\omega_3 X_2 + \theta_3 Y_2)$	$(\omega_3 + \gamma_3)\alpha_3, \omega_3 X_2, \theta_3 Y_2, \theta_3^2(\omega_3 X_2 + \theta_3 Y_2)$	–
$\gamma_3 = \theta_3 Z_2$	$\theta_3 Z_2$	–
$\theta_3 = X_1^3 Z_1^2 + X_2$	$X_1^3 Z_1^2$	–
$\omega_3 = Y_1 X_1^3 Z_1^3 + Y_2$	$(X_1^3 Z_1^2)(Y_1 Z_1)$	–
$X_3 = \alpha_3 Z_1^6$	$*$	–
$Y_3 = \beta_3 Z_1^9$	$*$	–
$Z_3 = \gamma_3 Z_1^3$	$*$	–

$*$ $(X_3, Y_3, Z_3) = (\alpha_3 Z_1^6, \beta_3 Z_1^9, \gamma_3 Z_1^3) = (\alpha_3, \beta_3, \gamma_3)$

Table 4.7: Reducing one squaring on curves defined over $\mathbb{F}_{2^m}$

Method	Operation	Intermediate Values	
		Multiplication	Squaring
Direct computation (Previous results) [DIM08]	$X_4{=}X_3^4 + a_6 Z_3^8$	$a_6 Z_3^8$	$X_3^2, X_3^4, \boldsymbol{Z_3^2}, Z_3^4, Z_3^8$
Reusing Values (Our results)	$\alpha_3 {=} a_2\gamma_3^2 + \omega_3(\omega_3+\gamma_3) + \theta_3^3$ $X_4{=}X_3^4 + a_6 Z_3^8$	$a_2\gamma_3^2, \omega_3(\omega_3+\gamma_3), \theta_3^3$ $a_6 Z_3^8$	$\boldsymbol{\gamma_3^2}, \theta_3^2$ $X_3^2, X_3^4, Z_3^4, Z_3^8 \quad (\gamma_3^2 = Z_3^2)$

Table 4.8 summarizes the cost of consecutive operations, namely, consecutive point triplings and consecutive point triplings followed by consecutive point doublings for curves defined over binary field in Jacobian coordinates after applying the method of reusing intermediate value. For references, cost of (single) point doubling and (single) point tripling are also displayed.

Table 4.8: Cost of consecutive operations on curves defined over $\mathbb{F}_{2^m}$ after reusing intermediate values

Operation	Cost
$\text{DBL}^{\mathcal{J}}$	$5M+5S$ [1]
$\text{TPL}^{\mathcal{J}}$	$15M+7S$ [1]
w-$\text{TPL}^{\mathcal{J}}$	$(15w)M+(6w+1)S$
w-$\text{TPL}^{\mathcal{J}}/w'$-$\text{DBL}^{\mathcal{J}}$	$(15w+5w')M+(6w+5w')S$

[1] Previous results [DIM08]

4.4 Experimental Results

To compare the efficiency of our proposed method with previous results, several experiments have been conducted. In each experiment, we used $10,000$ randomly chosen 256-bit integers. Those integers were then converted into double-base *chain* representation of mixed power of two and three by the algorithm described in Algorithm 3 of Section 2.2.

Since we were finding the double-base *chain* representation, the restrictions $a_1 \geq a_2 \geq \cdots \geq a_\ell$ and $b_1 \geq b_2 \geq \cdots \geq b_\ell$ must also satisfy. Each experiment, we set a_{max} and b_{max} to 150 and 100 respectively. The a_{max} and b_{max} values do have effect on the efficiency because setting them too small or too large resulted in longer expansion. Longer expansion also means longer time required when computing scalar multiplication. According to [DIM08], the best a_{max} and b_{max} for 160-bit integers are 95 and 41 respectively. We adjusted those numbers to fit 256-bit integers in our experiments.

For curves defined over prime field, it is generally assumed that $S = 0.8M$ where S and M are the cost of field squaring and field multiplication respectively. However, if using side-channel atomic block introduced by Chevallier-Mames, Ciet, and Joye [CMCJ04] in order to prevent simple side-channel analysis, we have to consider $S = M$. We conducted the experiment concerning both cases of $S = M$ and $S = 0.8M$.

Table 4.9 shows cost comparison of curves defined over prime field in Jacobian coordinates between previous results, namely, consecutive point triplings followed by consecutive point doublings (w-$\text{TPL}^{\mathcal{J}}/w'$-$\text{DBL}^{\mathcal{J}}$), and our results, namely, consecutive point doublings followed by consecutive point triplings. (w'-$\text{DBL}^{\mathcal{J}}/w$-$\text{TPL}^{\mathcal{J}}$). $\text{ADD}^{\mathcal{J}}$ denotes general addition in Jacobian coordinates while $\text{ADD}^{\mathcal{J}+\mathcal{A}}$ denotes mixed addition in Jacobian-affine coordinates. Improvements compared to previous results are shown in parentheses.

Experimental results using general addition in Jacobian coordinates showed reductions of 1.77% and 1.55% assuming $S = M$ and $S = 0.8M$ respectively. If mixed addition was used, the reduction increased to 1.95% and 1.71%.

We also conducted experiments on applying the technique of converting some field multiplications into field squarings. In this case, we compared our results to the combined results from [DIM08] and [LM08]. That is, we inferred what would be the results if applying M-S trade-off to the previous work of consecutive point

Table 4.9: Cost comparison of the results on curves defined over $\mathbb{F}_p$

		Cost	$S = M$	$S = 0.8M$
$\mathrm{ADD}^{\mathcal{J}}$	Previous results [1]	$1932.55M$	3268.55	3001.35
		$+1336.00S$	$(-)$	$(-)$
	Our results [2]	$1932.55M$	3210.32	2954.77
		$+1277.77S$	(1.77%)	(1.55%)
	Our results II [3]	$1614.14M$	3093.48	2797.61
		$+1479.34S$	(1.78%)	(1.59%)
$\mathrm{ADD}^{\mathcal{J}+\mathcal{A}}$	Previous results [1]	$1699.63M$	2977.40	2721.85
		$+1277.77S$	$(-)$	$(-)$
	Our results [2]	$1699.63M$	2919.17	2675.26
		$+1219.54S$	(1.95%)	(1.71%)
	Our results II [3]	$1318.55M$	2802.75	2518.51
		$+1421.20S$	(1.96%)	(1.76%)

[1] w-TPL$^{\mathcal{J}}$/w'-DBL$^{\mathcal{J}}$ [DIM08]
[2] w'-DBL$^{\mathcal{J}}$/w-TPL$^{\mathcal{J}}$
[3] w'-DBL$^{\mathcal{J}}$/w-TPL$^{\mathcal{J}}$ with M-S trade-off

tripling followed by consecutive point doubling (w-TPL$^{\mathcal{J}}$/w'-DBL$^{\mathcal{J}}$) and considered these as previous results. We then applied M-S trade-off to our method of consecutive point doubling followed by consecutive point tripling (w'-DBL$^{\mathcal{J}}$/w-TPL$^{\mathcal{J}}$) and considered these as our results. Reduction of 1.78% and 1.59% were obtained assuming $S = M$ and $S = 0.8M$ respectively if using addition in Jacobian coordinates. If addition in mixed Jacobian-affine coordinates was used, the reduction were 1.96% and 1.76% We would like to emphasize that these reductions were the *second step* reductions, i.e., these results were compared to the ones that applied M-S trade off. If we compared to the results that did not apply any techniques, the total speed-up would be $1.77 - 4.25\%$.

For curves defined over binary field, it has been assumed that the cost of field squaring is very small compared to the cost of field multiplication and is neglected in many cases. However, with the continuous improvement of multiplication, this assumption becomes invalid in some situations. Consequently, the cost of field squaring tends to be taken into consideration. In our experiment, we assumed the cost of field squaring and field multiplication to be approximately $S = 0.1M$ according to [HHM00].

Table 4.10 shows the cost comparison for computing consecutive point triplings followed by consecutive point doublings (w-TPL$^{\mathcal{J}}$/w'-DBL$^{\mathcal{J}}$) on curves defined over binary field in Jacobian coordinates between applying and not applying our technique of reusing intermediate values. Improvements are shown in parentheses.

Table 4.10: Cost comparison of w-TPL$^{\mathcal{J}}$/w'-DBL$^{\mathcal{J}}$ on curves defined over $\mathbb{F}_{2^m}$

		Cost	$S = 0.1M$
ADD$^{\mathcal{J}}$	Previous results [DIM08] [1]	$2785.18M$ $+1365.84S$	2921.76 $(-)$
	Our results [2]	$2785.18M$ $+1282.86S$	2913.46 (0.28%)
ADD$^{\mathcal{J}+\mathcal{A}}$	Previous results [DIM08] [1]	$2494.60M$ $+1365.84S$	2631.19 $(-)$
	Our results [2]	$2494.60M$ $+1282.86S$	2622.89 (0.31%)

[1] without reusing intermediate values

[2] with reusing intermediate values

According to the experimental results, if using general addition in Jacobian coordinates, we achieved a reduction of 0.28%. Again, if mixed addition was used, the reduction increased to 0.31%.

4.5 Concluding Remarks

Same integer n with the same double-base chain representation but computed in different order leads to different amount of time required. We raised a discussion on sequences of field arithmetic operation for computing elliptic curve scalar multiplication in double-base chain representation. For curves defined over prime field, a traditional method to compute scalar multiplication in (2,3) double-base chain representation is to compute point tripling then point doubling. It is trivial to see that by switching the sequences, that is, computing point doubling before point tripling, also yields the same result. Nevertheless, its consequence of less computation time required has not yet been revealed.

Scalar multiplication enhancement has long been studied. However, modifications on upper layers are more application specific and can be utilized only in narrow scopes. Unlike our method, we aimed to improve at fundamental layers in order to be applicable to more extensive range of areas. Because field arithmetic operation is a building block for many computations including scalar multiplication in elliptic curve cryptography, speeding up at field arithmetic level can further improve many applications in higher levels.

Our ideas of solving problem in another direction, i.e., *reordering* field arithmetic operation sequences and reusing *intermediate* values, are expected to be inspirations to others within the same and different areas of research on how to look at and understand problems.

Chapter 5

Side-Channel Atomicity and Multi-Pattern Atomic Block

In this chapter, we introduced new concepts of multi-pattern atomic block and ahead computation. We proposed three different atomic patterns for different usages. Our new patterns were explained in the following order: left-to-right binary scalar multiplication, right-to-left binary scalar multiplication for resource constrained system, and binary-ternary scalar multiplication involving point tripling.

5.1 Overview

Side-channel analysis, a threat in which adversaries observe information leak via physical devices and try to determine the underlying secret, has increasingly gained much attention and become one of major concerns in cryptography. There are various kinds of side-channel analysis. A lot of efforts have been put into finding countermeasures, and many algorithms have already been proposed.

Chevallier-Mames, Ciet, and Joye [CMCJ04] proposed a solution to prevent simple side-channel analysis by introducing a concept of *atomicity*. Their method divides each operation into small blocks of sub-operations called *side-channel atomic block*. Dummy operations are inserted to ensure that each block is identical so that it appears indistinguishably observed from side-channel.

Side-channel atomic block was improved once by Longa and Miri [LM08] to lower the cost and was made suitable for binary-ternary base scalar multiplication involving point tripling. Giraud and Verneuil [GV10] further improved atomic block for right-to-left binary scalar multiplication used in embedded devices.

We studied the atomic patterns proposed in [CMCJ04] [LM08] [GV10] and found that some dummy operations can be removed with neither losing its nice property of indistinguishability nor weakening its security. In this thesis, we introduced three new atomic patterns for three different contexts.

Our first contribution is new atomic patterns for left-to-right binary scalar multiplication. These patterns are designed for general point doubling and mixed point addition. Unlike in all of the previous works, we no longer use just only one pattern. In other words, we introduced an idea of using two patterns together. These two patterns are executed alternately so that they reveal no differences of which point operation is being computed. With our new patterns, 1.36% of a cost per bit of when computing scalar multiplication was decreased.

Our second contribution is new atomic patterns for mixed coordinates right-to-left binary scalar multiplication in embedded devices. That is, scalar multiplication computing point doubling in *modified* Jacobian projective coordinates and general point addition in Jacobian projective coordinates were considered. We also investigated the possibility to profit from trading field multiplication for

field squaring in the resource constrained environment. Again, we introduced a usage of two patterns for this situation which reduced $0.22 - 0.65\%$ of a cost per bit of when computing scalar multiplication.

Our third contribution is new atomic patterns applicable to binary-ternary base scalar multiplication involving point tripling. We examined atomic structures proposed in [LM08] and found that they could be more compact. As a result of rearranging those field operations, as well as converting some operations into another, we successfully reduced four dummy field negations for point doubling, one non-dummy and five dummy field negations for point addition, and two non-dummy and six dummy field negations for point tripling.

To summarize, we propose new atomic patterns for the following cases:

- Left-to-right binary scalar multiplication using general point doubling and mixed point addition
- Right-to-left binary scalar multiplication using modified Jacobian projective coordinates point doubling and general point addition
- Scalar multiplication where curves parameter $a_4 = -3$ using fast point doubling, mixed point addition, fast point tripling

Our first two contributions are for scalar n represented as binary number. Thus, point arithmetic operations involved are point doubling and point addition. On the other hand, our third contribution is for scalar n either represented as binary number or double-base chain using mixed power of two and three. Hence, point arithmetic involved might include point tripling if double-base chain representation is used.

5.2 Left-to-right Binary Scalar Multiplication

It has previously been described in [GV10] [JT01] that if using random curve isomorphism as a countermeasure for differential side-channel analysis, it is not possible to use fast point doubling because value a_4 (as appears in Equation 2.1 and 2.3) is random. This implies that general point doubling must be used instead. After examining the general point doubling and mixed point addition formulas, we observed that the number of field multiplications plus field squarings in those two formulas differs by one and the total number of field operations differs by four.

To be more precisely, general point doubling requires 4 field multiplications, 6 field squarings, 13 field additions, and 3 field negations which sum up to 26 field operations; mixed point addition requires 8 field multiplications, 3 field squarings, 7 field additions, and 4 field negations which sum up to 22 field operations. The number of each field operation required for general point doubling and mixed point addition is summarized in Table 5.1.

Table 5.1: Operation cost of general doubling and mixed addition

	General point doubling	Mixed point addition
Multiplication	4	8
Squaring	6	3
Addition	13	7
Negation	3	4
Sum	26	22

Because field squaring can be computed using field multiplication, we first focused our attention to the sum of field multiplication and field squaring necessary. By replacing some field squarings by field multiplications, it is possible to rearrange field arithmetic operations of general point doubling and mixed point addition into similar sequences.

Another important observation that we noticed was mixed point addition is never consecutively computed twice. In other words, mixed point addition never follows mixed point addition. It always follows general point doubling. Also, scalar multiplication never starts with mixed point addition. It always starts with general point doubling.

Our observations are summarized as follows:

- The total number of field operations in general point doubling and mixed point addition are very much similar.
- The sum of field multiplication and field squaring required for both formulas only differs by one.
- Rearranging field arithmetic operations of both formulas into similar sequences of field operations is possible.
- Mixed point addition is never consecutively computed twice.

Since the total number of field multiplications plus field squarings is *nearly* but *not exactly* the same, we did not want to add any extra field multiplications to make both formulas have the total number of field multiplications plus field squarings equal. Instead, we defined *two* different atomic patterns. We shall refer to as:

- Pattern 1: $M + S = 10$, having the number of field multiplications plus field squarings equals to 10
- Pattern 2: $M + S = 11$, having the number of field multiplications plus field squarings equals to 11

It is obvious that general point doubling can be expressed by using either of those patterns because the total number of field multiplications plus field squarings required for point doubling is 10 and either of those patterns has at least 10. However, mixed point addition cannot complete its computation unless it uses pattern 2 having $M + S = 11$.

Due to the fact that mixed point addition is never computed consecutively, that is, mixed point addition is always computed after general point doubling, the $M + S = 11$ pattern is never needed for two consecutive blocks. In this way, we can simply use those two patterns alternately. Figure 5.1 illustrates the idea of using two patterns alternately.

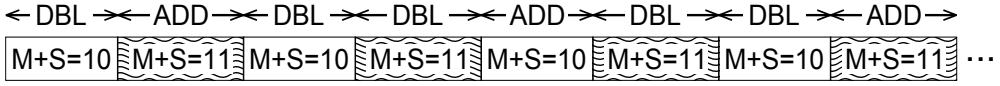

Figure 5.1: Two-pattern atomic block

If general point doubling is executed using pattern $M + S = 11$, we would use the extra field multiplication to compute one multiplication from mixed point addition formulas as if mixed point addition is really executed afterwards. In

a lucky case where mixed point addition is computed afterwards, we waste no computation. In a not-so-lucky case where mixed point addition is not computed afterwards, we waste one ahead-computation of field multiplication. Figure 5.2 illustrates the idea of ahead computation.

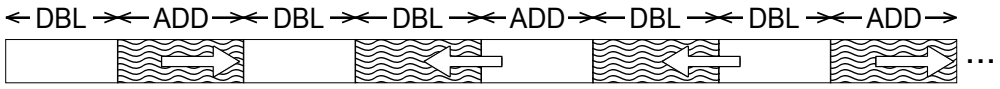

Figure 5.2: Ahead computation

Table 5.2 shows the atomic patterns for general point doubling both pattern 1 and 2. These two patterns are quite similar for point doubling. The atomic pattern 1: $M + S = 10$ consists of 29 field operations while the atomic pattern 2: $M + S = 11$ consists of 30 field operations. The difference between these two patterns is at the last field operation shown in parenthesis. That is, pattern 1 does not contain this operation. Please note that the last operation ($T_6 = T_2 \times Z_1 = Y_2 Z_1$) is for mixed point addition. If mixed point addition is not performed afterwards, this values would be discarded. In our patterns, $\star$ indicates dummy operations.

Table 5.2: General point doubling

Input: $P = (X_1, Y_1, Z_1)$

Output: $2P = (X_2, Y_2, Z_2)$

$T_1 \leftarrow X_1, T_2 \leftarrow Y_1, T_3 \leftarrow Z_1$

$T_4 = T_1^2$	(X_1^2)		$T_3 = T_2 \times T_3$	$(Y_1 Z_1)$	
$T_5 = T_2^2$	(Y_1^2)		$T_7 = T_7 + T_7$	$(4X_1 Y_1^2) = \beta$	
$\star$			$T_7 = -T_7$	$(-\beta)$	
$T_6 = T_4 + T_4$	$(2X_1^2)$		$T_3 = T_3 + T_3$	$(2Y_1 Z_1 = Z_2)$	
$T_7 = T_3^2$	(Z_1^2)		$T_1 = T_7 + T_7$	(-2β)	
$T_7 = T_7 \times T_7$	(Z_1^4)		$T_1 = T_1 + T_6$	$(\alpha^2 - 2\beta = X_2)$	
$T_7 = a \times T_7$	(aZ_1^4)		$T_6 = T_1 + T_7$	$(X_2 - \beta)$	
$\star$			$T_6 = -T_6$	$(\beta - X_2)$	
$T_4 = T_4 + T_6$	$(3X_1^2)$		$T_5 = T_5^2$	(Y_1^4)	
$T_4 = T_4 + T_7$	(α)	$*1$	$T_4 = T_4 \times T_6$	$(\alpha(\beta - X_2))$	
$T_6 = T_4^2$	(α^2)		$T_5 = T_5 + T_5$	$(2Y_1^4)$	
$T_7 = T_1 \times T_5$	$(X_1 Y_1^2)$		$T_5 = T_5 + T_5$	$(4Y_1^4)$	
$\star$			$T_5 = T_5 + T_5$	$(8Y_1^4)$	
$T_7 = T_7 + T_7$	$(2X_1 Y_1^2)$		$T_5 = -T_5$	$(-8Y_1^4)$	
			$T_2 = T_4 + T_5$	(Y_2)	$*2$
			$(T_6 = T_2 \times Z_1)$	$(Y_2 Z_1)$	

$*1 : 3X_1^2 + aZ_1^4 = \alpha$

$*2 : \alpha(\beta - X_2) - 8Y_1^4 = Y_2$

Table 5.3 shows the atomic pattern for mixed point addition using pattern 1: $M + S = 10$. Due to the fact that pattern 1 and 2 are used alternately and there is no consecutive mixed point additions, the previous operation before this mixed point addition using pattern $M + S = 10$ must be general point doubling using pattern $M + S = 11$. In this case, the ahead-computed value Y_2Z_1 in the previous general point doubling will be used in this atomic block of mixed point addition.

Table 5.3: Mixed point addition using pattern 1: $M + S = 10$

Input: $P_1 = (X_1, Y_1, Z_1), P_2 = (X_2, Y_2, 1)$

Output: $P_3 = (X_3, Y_3, Z_3) = P_1 + P_2$

$T_1 \leftarrow X_1, T_2 \leftarrow Y_1, T_3 \leftarrow Z_1$

$T_4 \leftarrow X_2, T_5 \leftarrow Y_2$

$T_7 = T_3^2$	(Z_1^2)	$T_8 = T_4 \times T_8$	$(-\beta^3)$
$T_4 = T_4 \times T_7$	$(X_2Z_1^2)$	$T_7 = T_7 + T_8$	$(\alpha^2 - \beta^3 - X_1\beta^2)$
$T_1 = -T_1$	$(-X_1)$	$\star$	
$T_4 = T_1 + T_4$	$(\beta) \quad *1$	$T_7 = T_1 + T_7$	$(X_3) \quad *3$
$T_8 = T_4^2$	(β^2)	$T_1 = T_1 + T_7$	$(X_3 - X_1\beta^2)$
$T_3 = T_3 \times T_4$	$(Z_1\beta = Z_3)$	$\star$	
$T_6 = T_6 \times T_7$	$(Y_2Z_1^3)$	$\star$	
$T_2 = -T_2$	$(-Y_1)$	$T_1 = -T_1$	$(X_1\beta^2 - X_3)$
$T_6 = T_2 + T_6$	$(\alpha) \quad *2$	$T_5 = T_1 \times T_6$	$(\alpha(X_1\beta^2 - X_3))$
$\star$		$T_8 = T_2 \times T_8$	$(Y_1\beta^3)$
$T_7 = T_6^2$	(α^2)	$\star$	
$T_1 = T_1 \times T_8$	$(-X_1\beta^2)$	$\star$	
$T_8 = -T_8$	$(-\beta^2)$	$T_8 = -T_8$	$(-Y_1\beta^3)$
$T_7 = T_1 + T_7$	$(\alpha^2 - X_1\beta^2)$	$T_2 = T_5 + T_8$	$(Y_3) \quad *4$
		$\star$	

$*1 : X_2Z_1^2 - X_1 = \beta$

$*2 : Y_2Z_1^3 - Y_1 = \alpha$

$*3 : \alpha^2 - \beta^3 - 2X_1\beta^2 = X_3$

$*4 : \alpha(X_1\beta^2 - X_3) - Y_1\beta^3 = Y_3$

Table 5.4 shows the atomic pattern for mixed point addition using pattern 2: $M + S = 11$. In this case, mixed point addition can complete its computation within its block. No value from previous block is used. In fact, there is no ahead-computation in the previous block because operation that comes before mixed point addition using pattern $M + S = 11$ is general point doubling using pattern $M + S = 10$ which has no extra field multiplication.

According to [GV10], they assumed the cost of field operation to be $A/M = 0.2$ and $S/M = 0.8$, and they also used NAF representation, i.e., k-bit scalar having an average Hamming weight of $k/3$. Thus, it implies that point doubling is executed k times and point addition is executed $k/3$ times. With these assumptions, the best average cost per bit to compute left-to-right scalar multiplication

Table 5.4: Mixed point addition using pattern 2: $M + S = 11$

Input: $P_1 = (X_1, Y_1, Z_1), \quad P_2 = (X_2, Y_2, 1)$

Output: $P_3 = (X_3, Y_3, Z_3) = P_1 + P_2$

$T_1 \leftarrow X_1, \quad T_2 \leftarrow Y_1, \quad T_3 \leftarrow Z_1$

$T_4 \leftarrow X_2, \quad T_5 \leftarrow Y_2$

$T_6 = T_3^2$	(Z_1^2)	$T_7 = T_4 \times T_7$	$(-\beta^3)$
$T_4 = T_4 \times T_6$	$(X_2 Z_1^2)$	$T_6 = T_6 + T_7$	$(\alpha^2 - \beta^3 - X_1\beta^2)$
$T_1 = -T_1$	$(-X_1)$	$\star$	
$T_4 = T_1 + T_4$	$(\beta) \quad *1$	$T_6 = T_1 + T_6$	$(X_3) \quad *3$
$T_7 = T_4^2$	(β^2)	$T_1 = T_1 + T_6$	$(X_3 - X_1\beta^2)$
$T_6 = T_3 \times T_6$	(Z_1^3)	$\star$	
$T_5 = T_5 \times T_6$	$(Y_2 Z_1^3)$	$\star$	
$T_2 = -T_2$	$(-Y_1)$	$T_1 = -T_1$	$(X_1\beta^2 - X_3)$
$T_5 = T_2 + T_5$	$(\alpha) \quad *2$	$T_5 = T_1 \times T_5$	$(\alpha(X_1\beta^2 - X_3))$
$\star$		$T_2 = T_2 \times T_7$	$(Y_1\beta^3)$
$T_6 = T_5^2$	(α^2)	$\star$	
$T_1 = T_1 \times T_7$	$(-X_1\beta^2)$	$\star$	
$T_7 = -T_7$	$(-\beta^2)$	$T_2 = -T_2$	$(-Y_1\beta^3)$
$T_6 = T_1 + T_6$	$(\alpha^2 - X_1\beta^2)$	$T_2 = T_2 + T_5$	$(Y_3) \quad *4$
		$\star$	
		$T_3 = T_3 \times T_4$	$(Z_1\beta = Z_3)$

$*1 : X_2 Z_1^2 - X_1 = \beta$

$*2 : Y_2 Z_1^3 - Y_1 = \alpha$

$*3 : \alpha^2 - \beta^3 - 2X_1\beta^2 = X_3$

$*4 : \alpha(X_1\beta^2 - X_3) - Y_1\beta^3 = Y_3$

is $17.7M$ [GV10].

Our proposed atomic pattern 1: $M+S = 10$ consists of 7 field multiplications, 3 field squarings, 13 field additions, and 6 field negations. Thus, the cost per one block is $(7 \times 1M) + (3 \times 0.8M) + (13 \times 0.2M) + (6 \times 0.1M) = 12.6M$. Our proposed atomic pattern 2: $M + S = 11$ consists of 8 field multiplications, 3 field squarings, 13 field additions, and 6 field negations. Thus, the cost per one block is $(8 \times 1M) + (3 \times 0.8M) + (13 \times 0.2M) + (6 \times 0.1M) = 13.6M$.

Consider atomic block that allows to compute both point doubling and point addition, we simply think of it as the average of these two patterns. Therefore, the average cost is computed as $\frac{1}{2}(12.6M + 13.6M) = 13.1M$. Now, we can roughly say that one block of our atomic pattern is sufficient to compute either point doubling or point addition. Regarding NAF representation, average cost per bit to compute left-to-right scalar multiplication using our atomic patterns is $13.1M + \frac{1}{3}(13.1M) = 17.46M$. Comparing this cost to the previous result, we achieved an improvement of 1.36%

5.3 Right-to-left Binary Scalar Multiplication

Giraud and Verneuil [GV10] proposed new atomic patterns right-to-left scalar multiplication for embedded devices. They described that in embedded devices the cost of field addition could not be neglected and the ratio between modular addition and modular multiplication (A/M) could be as large as 0.3. Due to the non-negligible cost of field addition and the fact that three extra field additions were required for each M-S trade-off [LM08], they thought that the trade-off was unprofitable.

However, we examined their patterns and found that there were many dummy field additions in point addition pattern. We also noticed that in their doubling pattern, they used 6 field multiplications and 2 field squarings ($6M+2S$) instead of 4 field multiplications and 4 field squarings ($4M+4S$) as are commonly used. This means that some field squarings were unnecessarily replaced by field multiplications. Please note that they used modified Jacobian projective coordinates for point doubling, and they merged field negation and field addition into field subtraction. We also followed their setting.

Our idea is to utilize those dummy operations for three extra field additions so that M-S trade-off can still be profitable. We observed that the number of field multiplications plus field squarings needed for point addition is twice as many as for point doubling, i.e., 16 and 8. Hence, the number of atomic blocks required for point addition is two times that of point doubling. If M-S trade-off is applied to point addition, the number of field multiplications and field squarings required would change to 11 and 5 respectively. Since 11 and 5 are obviously not even numbers, we can not simply express atomic blocks for point doubling as half the number of the atomic blocks for point addition.

To cope with this problem, we defined *two* atomic patterns, that is, one containing 5 field multiplications and 3 field squarings ($5M+3S$) and the other containing 6 field multiplications and 2 field squarings ($6M+2S$). We shall now refer to as pattern 1 and pattern 2 respectively. It is trivial to see that point addition can be expressed using the combination of pattern 1 and pattern 2 because $(5M+3S)+(6M+2S) = 11M+5S$ which is exactly the same number of field multiplications and field squarings required. It is also possible to express point doubling using either pattern 1: $5M+3S$ or pattern 2: $6M+2S$ because point doubling requires $4M+4S$ and field squaring can always be executed using field multiplication.

Since we now have two different atomic patterns, in order to withstand the simple SCA, it is necessary to ensure that sequences of these two patterns reveal no information related to which patterns belong to which point operations. One solution is to use pattern 1 and pattern 2 alternately regardless of what point operation is being executed. Figure 5.3 illustrates the idea of alternately switching two patterns.

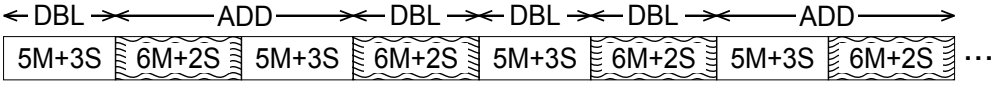

Figure 5.3: Alternating two patterns preventing simple SCA

In this way, point addition must be able to begin its execution with either pattern 1 or pattern 2. As a consequence, the following atomic patterns are required:

- Point doubling using $5M + 3S$
- Point doubling using $6M + 2S$
- Point addition using $5M + 3S$ then $6M + 2S$
- Point addition using $6M + 2S$ then $5M + 3S$

We shall now present our new atomic patterns for the above four cases. Table 5.5 shows the atomic patterns for point doubling. Table 5.6 shows the atomic pattern for point addition using pattern 1: $5M + 3S$ then pattern 2: $6M + 2S$. Table 5.7 shows the atomic pattern for point addition using pattern 2: $6M + 2S$ then pattern 1: $5M + 3S$. Again, $\star$ indicates dummy operations.

Note that the differences between patterns 1: $5M+3S$ and pattern 2: $6M+2S$ are at step 7 and 8. Field squaring at step 7 and field addition at step 8 of pattern 1: $5M + 3S$ are combined to field multiplication in step 7 of pattern 2: $6M + 2S$.

Again, we used NAF representation and assumed the cost of field operation to be $A/M = 0.2$ and $S/M = 0.8$. With these assumptions, average cost per bit to compute scalar multiplication using their atomic patterns is $16M$ [GV10].

In our atomic patterns, there are 5 field multiplications, 3 field squarings, and 11 field additions in pattern 1; and there are 6 field multiplications, 2 field squarings, and 10 field additions in pattern 2. Under the above assumption, cost per pattern for both patterns is $9.6M$. One pattern is needed for point doubling while two are needed for point addition. Therefore, the cost for point doubling and point addition are $9.6M$ and $19.2M$ respectively. With NAF representation, average cost per bit to compute scalar multiplication using our atomic patterns is $16M$.

The reason our average cost per bit equals to theirs even though we traded one field squaring for one field multiplication is that using 5 field multiplications for point doubling resulted in an extra field addition required, that is, 11 field additions are needed instead of 10. Thus, $0.2M$ gained from M-S trade-off was used up for the extra field addition.

However, our atomic patterns are still profitable if the cost of field addition plus field squaring is less than that of field multiplication, that is, $S + A < M$. According to [GV10], the A/M ratio on smart cards with crypto-coprocessor is less than 0.2 when the bit-length used is not shorter than 320. In other words, our atomic patterns show improvement when using at least 320-bit scalar. Table 5.8 shows the improvements according to the bit-length used.

Table 5.8: Improvement of right-to-left atomic pattern

Bit-length	A/M	Cost per bit		Improvement
		[GV10]	Our results	
320	0.16	15.33	15.3	0.22%
384	0.13	14.83	14.78	0.39%
512	0.09	14.17	14.08	0.65%
521	0.09	14.17	14.08	0.65%

Table 5.5: Two atomic patterns for point doubling

Input: $P = (X_1, Y_1, Z_1, W_1)$

Output: $2P = (X_2, Y_2, Z_2, W_2)$

$T_1 \leftarrow X_1, \quad T_2 \leftarrow Y_1, \quad T_3 \leftarrow Z_1, \quad T_4 \leftarrow W_1$

Pattern 1: $5M + 3S$			Pattern 2: $6M + 2S$	
$T_5 = T_1^2$	(X_1^2)	1	$T_5 = T_1^2$	(X_1^2)
$T_6 = T_2 + T_2$	$(2Y_1)$	2	$T_6 = T_2 + T_2$	$(2Y_1)$
$T_3 = T_3 \times T_6$	$(2Y_1Z_1 = Z_2)$	3	$T_3 = T_3 \times T_6$	$(2Y_1Z_1 = Z_2)$
$T_7 = T_5 + T_5$	$(2X_1^2)$	4	$T_7 = T_5 + T_5$	$(2X_1^2)$
$T_2 = T_2 \times T_6$	$(2Y_1^2)$	5	$T_2 = T_2 \times T_6$	$(2Y_1^2)$
$T_6 = T_2 + T_2$	$(4Y_1^2)$	6	$T_6 = T_2 + T_2$	$(4Y_1^2)$
$T_2 = T_2^2$	$(4Y_1^4)$	7	$T_2 = T_2 \times T_6$	$(8Y_1^4 = \gamma)$
$T_2 = T_2 + T_2$	$(8Y_1^4 = \gamma)$	8	$T_5 = T_5 + T_7$	$(3X_1^2)$
$T_5 = T_5 + T_7$	$(3X_1^2)$	9	$T_5 = T_5 + T_4$	$(3X_1^2 + W_1 = \alpha)$
$T_5 = T_4 + T_5$	$(3X_1^2 + W_1 = \alpha)$	10		
$T_7 = T_5^2$	(α^2)	11	$T_7 = T_5^2$	(α^2)
$T_6 = T_1 \times T_6$	$(4X_1Y_1^2 = \beta)$	12	$T_6 = T_1 \times T_6$	$(4X_1Y_1^2 = \beta)$
$T_4 = T_4 + T_4$	$(2W_1)$	13	$T_4 = T_4 + T_4$	$(2W_1)$
$T_7 = T_7 - T_6$	$(\alpha^2 - \beta)$	14	$T_7 = T_7 - T_6$	$(\alpha^2 - \beta)$
$T_4 = T_2 \times T_4$	$(2\gamma W_1 = W_2)$	15	$T_4 = T_2 \times T_4$	$(2\gamma W_1 = W_2)$
$T_1 = T_7 - T_6$	$(\alpha^2 - 2\beta = X_2)$	16	$T_1 = T_7 - T_6$	$(\alpha^2 - 2\beta = X_2)$
$T_6 = T_6 - T_1$	$(\beta - X_2)$	17	$T_7 = T_6 - T_1$	$(\beta - X_2)$
$T_6 = T_5 \times T_6$	$(\alpha(\beta - X_2))$	18	$T_5 = T_5 \times T_7$	$(\alpha(\beta - X_2))$
$T_2 = T_6 - T_2$	$(Y_2) \quad *1$	19	$T_2 = T_5 - T_2$	$(Y_2) \quad *2$

$*1 : \alpha(\beta - X_2) - 8Y_1^4 = Y_2$

$*2 : \alpha(\beta - X_2) - 8Y_1^4 = Y_2$

Table 5.6: Addition using pattern $5M + 3S$ then $6M + 2S$

Input: $P_1 = (X_1, Y_1, Z_1), \quad P_2 = (X_2, Y_2, Z_2)$

Output: $P_3 = (X_3, Y_3, Z_3) = P_1 + P_2$

$T_1 \leftarrow X_1, \quad T_2 \leftarrow Y_1, \quad T_3 \leftarrow Z_1$

$T_4 \leftarrow X_2, \quad T_5 \leftarrow Y_2, \quad T_6 \leftarrow Z_2$

Pattern 1: $5M + 3S$		
$T_7 = T_3^2$	(Z_1^2)	
$T_8 = T_3 + T_6$	$(Z_1 + Z_2)$	
$T_4 = T_4 \times T_7$	$(X_2 Z_1^2)$	
$\star$		
$T_3 = T_3 \times T_7$	(Z_1^3)	
$\star$		
$T_9 = T_6^2$	(Z_2^2)	
$\star$		
$\star$		
$\star$		
$T_8 = T_8^2$	$((Z_1 + Z_2)^2)$	
$T_1 = T_1 \times T_9$	$(X_1 Z_2^2)$	
$T_8 = T_8 - T_7$	$(*1)$	
$T_8 = T_8 - T_9$	$(2Z_1 Z_2)$	$*2$
$T_6 = T_6 \times T_9$	(Z_2^3)	
$T_4 = T_4 - T_1$	(β)	$*3$
$T_7 = T_4 + T_4$	(2β)	
$T_5 = T_3 \times T_5$	$(Y_2 Z_1^3)$	
$\star$		

Pattern 2: $6M + 2S$		
$T_7 = T_7^2$	$(4\beta^2)$	
$\star$		
$T_2 = T_2 \times T_6$	$(Y_1 Z_2^3)$	
$T_5 = T_5 - T_2$	(α)	$*4$
$T_1 = T_1 \times T_7$	$(4X_1 Z_2^2 \beta^2)$	
$T_5 = T_5 + T_5$	(2α)	
$T_7 = T_4 \times T_7$	$(4\beta^3)$	
$T_3 = T_7 + T_7$	$(8\beta^3)$	
$T_9 = T_1 + T_1$	$(8X_1 Z_2^2 \beta^2)$	
$T_6 = T_5^2$	$(4\alpha^2)$	
$T_2 = T_2 \times T_3$	$(8Y_1 Z_2^3 \beta^3)$	
$T_6 = T_6 - T_7$	$(4\alpha^2 - 4\beta^3)$	
$T_6 = T_6 - T_9$	(X_3)	$*5$
$T_8 = T_4 \times T_8$	$(2Z_1 Z_2 \beta = Z_3)$	
$T_1 = T_1 - T_6$	$(4X_1 Z_2^2 \beta^2 - X_3)$	
$\star$		
$T_5 = T_1 \times T_5$	$(*6)$	
$T_2 = T_5 - T_2$	(Y_3)	$*7$

$*1 : (Z_1 + Z_2)^2 - Z_1^2$

$*2 : (Z_1 + Z_2)^2 - Z_1^2 - Z_2^2 = 2Z_1 Z_2$

$*3 : X_2 Z_1^2 - X_1 Z_2^2 = \beta$

$*4 : Y_2 Z_1^3 - Y_1 Z_2^3 = \alpha$

$*5 : 4\alpha^2 - 4\beta^3 - 8X_1 Z_2^2 \beta^2 = X_3$

$*6 : 2\alpha(4X_1 Z_2^2 \beta^2 - X_3)$

$*7 : 2\alpha(4X_1 Z_2^2 \beta^2 - X_3) - 8Y_1 Z_2^3 \beta^3 = Y_3$

Table 5.7: Addition using pattern $6M + 2S$ then $5M + 3S$

Input: $P_1 = (X_1, Y_1, Z_1), \quad P_2 = (X_2, Y_2, Z_2)$

Output: $P_3 = (X_3, Y_3, Z_3) = P_1 + P_2$

$T_1 \leftarrow X_1, \quad T_2 \leftarrow Y_1, \quad T_3 \leftarrow Z_1$

$T_4 \leftarrow X_2, \quad T_5 \leftarrow Y_2, \quad T_6 \leftarrow Z_2$

Pattern 2: $6M + 2S$		
$T_7 = T_3^2$	(Z_1^2)	
$T_8 = T_3 + T_6$	$(Z_1 + Z_2)$	
$T_4 = T_4 \times T_7$	$(X_2 Z_1^2)$	
$\star$		
$T_3 = T_3 \times T_7$	(Z_1^3)	
$\star$		
$T_5 = T_3 \times T_5$	$(Y_2 Z_1^3)$	
$\star$		
$\star$		
$T_9 = T_6^2$	(Z_2^2)	
$T_1 = T_1 \times T_9$	$(X_1 Z_2^2)$	
$T_4 = T_4 - T_1$	(β)	$*1$
$T_3 = T_4 + T_4$	(2β)	
$T_6 = T_6 \times T_9$	(Z_2^3)	
$\star$		
$\star$		
$T_2 = T_2 \times T_6$	$(Y_1 Z_2^3)$	
$T_5 = T_5 - T_2$	(α)	$*2$

Pattern 1: $5M + 3S$		
$T_3 = T_3^2$	$(4\beta^2)$	
$T_5 = T_5 + T_5$	(2α)	
$T_1 = T_1 \times T_3$	$(4X_1 Z_2^2 \beta^2)$	
$T_6 = T_1 + T_1$	$(8X_1 Z_2^2 \beta^2)$	
$T_3 = T_3 \times T_4$	$(4\beta^3)$	
$T_{10} = T_3 + T_3$	$(8\beta^3)$	
$T_8 = T_8^2$	$((Z_1 + Z_2)^2)$	
$T_8 = T_8 - T_7$	$(*3)$	
$T_8 = T_8 - T_9$	$(2Z_1 Z_2)$	$*4$
$\star$		
$T_7 = T_5^2$	$(4\alpha^2)$	
$T_2 = T_2 \times T_{10}$	$(8Y_1 Z_2^3 \beta^3)$	
$T_7 = T_7 - T_3$	$(4\alpha^2 - 4\beta^3)$	
$T_7 = T_7 - T_6$	(X_3)	$*5$
$T_8 = T_4 \times T_8$	$(2Z_1 Z_2 \beta = Z_3)$	
$T_1 = T_1 - T_7$	$(4X_1 Z_2^2 \beta^2 - X_3)$	
$\star$		
$T_5 = T_1 \times T_5$	$(*6)$	
$T_5 = T_5 - T_2$	(Y_3)	$*7$

$*1 : X_2 Z_1^2 - X_1 Z_2^2 = \beta$

$*2 : Y_2 Z_1^3 - Y_1 Z_2^3 = \alpha$

$*3 : (Z_1 + Z_2)^2 - Z_1^2$

$*4 : (Z_1 + Z_2)^2 - Z_1^2 - Z_2^2 = 2Z_1 Z_2$

$*5 : 4\alpha^2 - 4\beta^3 - 8X_1 Z_2^2 \beta^2 = X_3$

$*6 : 2\alpha(4X_1 Z_2^2 \beta^2 - X_3)$

$*7 : 2\alpha(4X_1 Z_2^2 \beta^2 - X_3) - 8Y_1 Z_2^3 \beta^3 = Y_3$

5.4 Scalar Multiplication where $a_4 = -3$

Longa and Miri [LM08] proposed new atomic patterns for the special case where curve parameter a_4 as appears in Equation 2.1 and 2.3 equals to -3. Their atomic structure for traditional scalar multiplication, namely, involving only point doubling and point addition was S-N-A-M-N-A-A. This structure composed of 1 field multiplication, 1 field squaring, 3 field additions, and 2 field negations. For scalar multiplication involving point tripling, their atomic structure was S-N-A-A-M-N-A-A. This structure had one more field addition, that is, 1 field multiplication, 1 field squaring, 4 field additions, and 2 field negations.

We studied their atomic structures and found that there were 4 dummy negations in point doubling and 5 dummy negations in point addition. We also observed that they used 4 and 6 atomic blocks to express point doubling and point addition respectively.

After carefully examining the number of field multiplications, field squarings, field additions, and field negations required for each point operation, we discovered that those operations can be rearranged so that point doubling and mixed point addition use $4M + 4S + 12A + 4N$ and $6M + 6S + 17A + 6N$ field operations respectively (M, S, A, and N designate field multiplication, field squaring, field addition, and field squaring respectively). For this reason, we introduced new atomic patterns for the traditional scalar multiplication consisting of 1 field multiplication, 1 field squaring, 3 field additions, and 1 field negation, that is, having the structure S-A-N-A-M-A.

For point tripling, we found that it could be expressed using $7M + 7S + 23A + 7S$ field operations. It is easy to see that if each block composed of 1 field multiplication, 1 field squaring, 3 field additions, and 1 field negation, eight blocks would be required to compute point tripling. Moreover, there exists dependency among field operation sequences. That is, some certain operations must be computed before some certain operations. As a consequence, eight blocks were unavoidably required.

Our atomic structure for scalar multiplication involving point tripling has a structure of S-A-N-A-M-A-A. Note that there is one more field addition compared to the structure involving only point doubling and point addition.

Table 5.9, Table 5.10, and Table 5.11 show the atomic patterns for point doubling, point addition, and point tripling respectively.

Atomic structure for scalar multiplication involving only point doubling and point addition proposed by Longa and Miri [LM08] is S-N-A-M-N-A-A. On the other hand, our new atomic structure is S-A-N-A-M-A. That is, we reduced one field negation for *each* block. To express point doubling and point addition, four and six atomic blocks are required. Therefore, we removed the total of *four* and *six* field negations for computing point doubling and point addition.

For scalar multiplication that involves point tripling, their atomic structure is S-N-A-A-M-N-A-A. Our atomic structure is S-A-N-A-M-A-A. In other words, we removed one field negation for *each* block. Four, six, and eight atomic blocks are required to express point doubling, point addition, and point tripling respectively. Hence, we decreased *four*, *six*, and *eight* field negations for computing point doubling, point addition, and point tripling respectively.

Table 5.9: Atomic point doubling

Input: $P = (X_1, Y_1, Z_1)$

Output: $2P = (X_2, Y_2, Z_2)$

$T_1 \leftarrow X_1, \quad T_2 \leftarrow Y_1, \quad T_3 \leftarrow Z_1$

$T_4 = T_3^2$	(Z_1^2)
$T_5 = T_1 + T_4$	$(X_1 + Z_1^2)$
$T_4 = -T_4$	$(-Z_1^2)$
$T_6 = T_1 + T_4$	$(X_1 - Z_1^2)$
$T_5 = T_5 \times T_6$	$(X_1^2 - Z_1^4)$
$T_6 = T_5 + T_5$	$(2(X_1^2 - Z_1^4))$
$T_4 = T_2^2$	(Y_1^2)
$T_4 = T_4 + T_4$	$(2Y_1^2)$
$T_1 = -T_1$	$(-X_1)$
$T_1 = T_1 + T_1$	$(-2X_1)$
$T_7 = T_1 \times T_4$	$(-\beta) \quad *1$
$T_5 = T_5 + T_6$	$(\alpha) \quad *2$

$T_6 = T_5^2$	(α^2)
$T_6 = T_6 + T_7$	$(\alpha^2 - \beta)$
$T_5 = -T_5$	$(-\alpha)$
$T_6 = T_6 + T_7$	$(\alpha^2 - 2\beta = X_2)$
$T_3 = T_2 \times T_3$	$(Y_1 Z_1)$
$T_3 = T_3 + T_3$	$(2Y_1 Z_1 = Z_2)$
$T_4 = T_4^2$	$(4Y_1^4)$
$T_4 = T_4 + T_4$	$(8Y_1^4)$
$T_4 = -T_4$	$(-8Y_1^4)$
$T_1 = T_6 + T_7$	$(X_2 - \beta)$
$T_5 = T_1 \times T_5$	$(\alpha(\beta - X_2))$
$T_2 = T_4 + T_5$	$(Y_2) \quad *3$

$*1 : -4X_1 Y_1^2 = -\beta$

$*2 : 3(X_1^2 - Z_1^4) = \alpha$

$*3 : \alpha(\beta - X_2) - 8Y_1^4 = Y_2$

Table 5.10: Atomic point addition

Input: $P_1 = (X_1, Y_1, Z_1), \quad P_2 = (X_2, Y_2, 1)$

Output: $P_3 = (X_3, Y_3, Z_3) = P_1 + P_2$

$T_1 \leftarrow X_1, \quad T_2 \leftarrow Y_1, \quad T_3 \leftarrow Z_1$

$T_4 \leftarrow X_2, \quad T_5 \leftarrow Y_2, \quad Y_2^2$ is pre-computed

$T_6 = T_3^2$	(Z_1^2)	$T_3 = T_3^2$	$((Z_1+\beta)^2)$
$T_2 = T_2 + T_2$	$(2Y_1)$	$T_3 = T_3 + T_8$	$(2Z_1\beta = Z_3) * 4$
$T_1 = -T_1$	$(-X_1)$	$T_4 = -T_4$	$(-\beta)$
$\star$		$T_{10} = T_1 + T_1$	$(-4X_1\beta^2)$
$T_4 = T_4 \times T_6$	$(X_2 Z_1^2)$	$T_4 = T_4 \times T_7$	$(-2\beta^3)$
$T_4 = T_1 + T_4$	$(\beta) \quad *1$	$T_4 = T_4 + T_4$	$(-4\beta^3)$
$T_7 = T_4^2$	(β^2)	$T_7 = T_6^2$	(Z_1^6)
$T_8 = T_6 + T_7$	$(Z_1^2 + \beta^2)$	$T_1 = T_{10} + T_{10}$	$(-8X_1\beta^2)$
$T_9 = -T_2$	$(-2Y_1)$	$T_7 = -T_7$	$(-Z_1^6)$
$T_7 = T_7 + T_7$	$(2\beta^2)$	$T_5 = T_5 + T_7$	$(2\alpha) \quad *5$
$T_6 = T_3 \times T_6$	(Z_1^3)	$T_2 = T_2 \times T_4$	$(-8Y_1\beta^3)$
$T_5 = T_5 + T_6$	$(Y_2 + Z_1^3)$	$T_4 = T_1 + T_4$	$(-4\beta^3 - 8X_1\beta^2)$
$T_5 = T_5^2$	$((Y_2 + Z_1^3)^2)$	$T_6 = T_5^2$	$(4\alpha^2)$
$T_5 = T_5 - Y_2^2$	$(*2)$	$T_6 = T_4 + T_6$	$(X_3) \quad *6$
$T_8 = -T_8$	$(-Z_1^2 - \beta^2)$	$T_5 = -T_5$	(-2α)
$T_5 = T_5 + T_9$	$(*3)$	$T_1 = T_6 + T_{10}$	$(*7)$
$T_1 = T_1 \times T_7$	$(-2X_1\beta^2)$	$T_5 = T_1 \times T_5$	$(*8)$
$T_3 = T_3 + T_4$	$(Z_1 + \beta)$	$T_2 = T_2 + T_5$	$(Y_3) \quad *9$

$*1 : X_2 Z_1^2 - X_1 = \beta$

$*2 : (Y_2 + Z_1^3)^2 - Y_2^2$

$*3 : (Y_2 + Z_1^3)^2 - Y_2^2 - 2Y_1$

$*4 : (Z_1 + \beta)^2 - Z_1^2 - \beta^2 = 2Z_1\beta$

$*5 : 2Y_2 Z_1^3 - 2Y_1 = 2\alpha$

$*6 : 4\alpha^2 - 4\beta^3 - 8X_1\beta^2 = X_3$

$*7 : X_3 - 4X_1\beta^2$

$*8 : 2\alpha(4X_1\beta^2 - X_3)$

$*9 : 2\alpha(4X_1\beta^2 - X_3) - 8Y_1\beta^3 = Y_3$

Table 5.11: Atomic point tripling

Input: $P = (X_1, Y_1, Z_1)$

Output: $3P = (X_2, Y_2, Z_2)$

$T_1 \leftarrow X_1, \quad T_2 \leftarrow Y_1, \quad T_3 \leftarrow Z_1$

$T_4 = T_3^2$	(Z_1^2)
$T_5 = T_1 + T_4$	$(X_1 + Z_1^2)$
$T_6 = -T_4$	$(-Z_1^2)$
$T_7 = T_1 + T_6$	$(X_1 - Z_1^2)$
$\star$	
$T_2 = T_2 + T_2$	$(2Y_1)$
$\star$	
$T_8 = T_2^2$	$(4Y_1^2)$
$T_9 = T_8 + T_8$	$(8Y_1^2)$
$T_3 = -T_3$	$(-Z_1)$
$T_{10} = T_8 + T_9$	$(12Y_1^2)$
$T_5 = T_5 \times T_7$	$(X_1^2 - Z_1^4)$
$T_7 = T_5 + T_5$	$(2(X_1^2 - Z_1^4))$
$T_5 = T_5 + T_7$	$(\theta) \quad *1$
$T_7 = T_5^2$	(θ^2)
$T_2 = T_2 + T_2$	$(4Y_1)$
$T_{10} = -T_{10}$	$(-12Y_1^2)$
$T_2 = T_2 + T_2$	$(8Y_1)$
$T_{10} = T_1 \times T_{10}$	$(-12X_1Y_1^2)$
$T_{10} = T_7 + T_{10}$	$(-\omega) \quad *2$
$\star$	
$T_6 = T_{10}^2$	(ω^2)
$T_7 = T_6 + T_7$	$(\theta^2 + \omega^2)$
$T_{11} = -T_{10}$	(ω)
$T_4 = T_4 + T_6$	$(Z_1^2 + \omega^2)$
$T_{12} = T_6 \times T_{10}$	$(-\omega^3)$
$T_3 = T_3 + T_{10}$	$(-Z_1 - \omega)$
$T_5 = T_5 + T_{10}$	$(\theta - \omega)$

$T_5 = T_5^2$	$((\theta - \omega)^2)$
$T_1 = T_1 + T_1$	$(2X_1)$
$T_7 = -T_7$	$(-\theta^2 - \omega^2)$
$T_1 = T_1 + T_1$	$(4X_1)$
$T_1 = T_1 \times T_6$	$(4X_1\omega^2)$
$T_5 = T_5 + T_7$	$(-2\theta\omega = -2\alpha) * 3$
$\star$	
$T_8 = T_8^2$	$(16Y_1^4 = 2\beta)$
$T_6 = T_5 + T_8$	$(2\beta - 2\alpha)$
$T_7 = -T_6$	$(2\alpha - 2\beta)$
$T_9 = T_9 + T_9$	$(16Y_1^2)$
$T_9 = T_6 \times T_9$	$(16Y_1^2(2\beta - 2\alpha))$
$T_1 = T_1 + T_9$	$(X_2) \quad *4$
$\star$	
$T_3 = T_3^2$	$((Z_1 + \omega)^2)$
$T_6 = T_6 + T_8$	$(4\beta - 2\alpha)$
$T_4 = -T_4$	$(-Z_1^2 - \omega^2)$
$T_3 = T_3 + T_4$	$(2Z_1\omega = Z_2)$
$T_6 = T_6 \times T_7$	$(*5)$
$T_6 = T_6 + T_{12}$	$(*6)$
$\star$	
$(T_4 = T_3^2)$	$(Z_2^2)^{**}$
$(T_5 = T_1 + T_4)$	$(X_2 + Z_2^2)^{**}$
$(T_6 = -T_4)$	$(-Z_2^2)^{**}$
$(T_7 = T_1 + T_6)$	$(X_2 - Z_2^2)^{**}$
$T_2 = T_2 \times T_6$	$(Y_2) \quad *7$
$(T_2 = T_2 + T_2)$	$(2Y_2)^{**}$
$\star$	

$*1 : 3(X_1^2 - Z_1^4) = \theta$

$*2 : \theta^2 - 12X_1Y_1^2 = -\omega$

$*3 : (\theta - \omega)^2 - \theta^2 - \omega^2 = -2\theta\omega$

$*4 : 16Y_1^2(2\beta - 2\alpha) + 4X_1\omega^2 = X_2$

$*5 : 4(\alpha - \beta)(2\beta - \alpha)$

$*6 : 4(\alpha - \beta)(2\beta - \alpha) - \omega^3$

$*7 : 8Y_1[4(\alpha - \beta)(2\beta - \alpha) - \omega^3] = Y_2$

** indicates a merge of last and first block when point tripling is computed consecutively

5.5 Concluding Remarks

Side-channel atomic block has previously been proposed as a countermeasure for side-channel analysis. However, many dummy operations were introduced to make the overall process appeared as a sequence of identical atomic blocks. To deal with this problem, we introduced the idea of two-pattern atomic block and ahead computation.

Two-pattern atomic block is a technique of using two patterns together in a systematic way to preserve the property of indistinguishability observed from side-channel. By increasing the choice of patterns, we decreased the number of dummy operations required to balance each atomic block.

Ahead computation is a technique of bringing operations in following blocks into previous blocks and computing them. "Bringing operations into previous blocks" can be done by either replacing dummy operations or rearranging operations sequences.

With these two techniques, we successfully reduced a number of operations required to compute scalar multiplication. This result can be applied to many scalar multiplication related algorithms.

Chapter 6

Conclusion

Main focus of our research was on improving the speed of scalar multiplication. We examined both point operation on elliptic curve and side-channel atomicity for preventing side-channel analysis. Major techniques that we used were rearranging operation sequence and reusing intermediate values. Our study revealed that reusing the already computed values can significantly reduce computational time because redundant computation can be avoided. In order to efficiently reuse the intermediate values, rewriting and/or rearranging operation sequences may become necessary. We applied these techniques of rearranging operation sequence and reusing intermediate values to twisted Hessian curve, double-base chain scalar multiplication, and side-channel atomic block.

For twisted Hessian point doubling formulas in projective coordinates, we rearranged operations at *field* arithmetic level using equality of square of the sum of two numbers together with reusing previously computed value. On the other hand, we suggested a different *point* operation sequence when performing scalar multiplication on number represented in double-base chain. In the case of side-channel atomic block, we introduced new concepts of multi-pattern atomic block and ahead computation. These two techniques perform at *field* arithmetic operations.

Our new twisted Hessian point doubling formulas achieve a reduction of 2.5% assuming the relationship between the cost of field squaring S and field multiplication M is $S = 0.8M$. We calculated cost per bit reduction assuming point doubling is performed every bit while point addition is performed one-third of the bit length. Our new formulas reduced the cost by 1.67%. Results on double-base chain scalar multiplication showed 1.95% speed up for curves defined over prime field. Our proposed atomic patterns allow the cost per bit reduction of 1.36% for left-to-right binary scalar multiplication and $0.22 - 0.65\%$ for right-to-left binary scalar multiplication.

We believe the concept of our proposed techniques is very general and can be applied to not only results presented in this thesis but also other applications. As for one of the future work, we would like to investigate whether our techniques of rearranging operation sequence and reusing intermediate values can be applied to pairing-based cryptography. To the best of our knowledge, Miller's algorithm in pairing-based cryptography can be compared to scalar multiplication in elliptic curve cryptography. We think techniques that work with scalar multiplication may also work with Miller's algorithm.

In this thesis, we investigated only one layer of scalar multiplication at a time. That is, we only examined either point operation layer or field operation layer. However, the are above layers such as how scalar should be represented, and also below layers such as how scalar multiplication is actually carried out on hardware.

Analysis across layer may suggest how point operations and/or field operations should be arranged so that they would benefit or would be benefited from the above and/or below layers.

Another idea for future work is to create a program that helps optimize the number of field operations required for each point operation. Obviously, computers can easily store intermediate values occurred and can quickly try the permutation of operation sequences along with checking whether intermediate values can be reused or not. This program should also allow adjustment of technique used. That is, other techniques such as M-S trade-off and equality of square of the sum of two numbers should be able to be inputted into the program and to be checked for applicability.

References

[ACD+06] Roberto M. Avanzi, Henri Cohen, Christophe Doche, Gerhard Frey, Tanja Lange, Kim Nguyen, and Frederic Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Chapman & Hall/CRC, 2006.

[AT03] Toru Akishita and Tsuyoshi Takagi. Zero-value point attacks on elliptic curve cryptosystem. In *Proceedings of the 6th International Conference on Information Security*, pages 218–233, 2003.

[Ava05] Roberto M. Avanzi. Side channel attacks on implementations of curve-based cryptographic primitives. *IACR Cryptology ePrint Archive*, 2005:17, 2005.

[BBJ+08] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. Twisted edwards curves. In *Proceedings of the Cryptology in Africa 1st International Conference on Progress in Cryptology*, pages 389–405, 2008.

[BJ03] Olivier Billet and Marc Joye. The jacobi model of an elliptic curve and side-channle analysis. In *Proceedings of the 15th International Conference on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 34–42, 2003.

[BL] Daniel J. Bernstein and Tanja Lange. Explicit-formulas database. `http://hyperelliptic.org/EFD/`.

[BL07a] Daniel J. Bernstein and Tanja Lange. Analysis and optimization of elliptic-curve single-scalar multiplication. *IACR Cryptology ePrint Archive*, 2007, 2007.

[BL07b] Daniel J. Bernstein and Tanja Lange. Faster addition and doubling on elliptic curves. In *Proceedings of the Advances in Cryptology 13th International Conference on Theory and Application of Cryptology and Information Security*, pages 29–50, 2007.

[BL09] Daniel J. Bernstein and Tanja Lange. A complete set of addition laws for incomplete edwards curves. *IACR Cryptology ePrint Archive*, 2009:580, 2009.

[BR] Elaine Barker and Allen Roginsky. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. `http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf`.

[CJ01] Christophe Clavier and Marc Joye. Universal exponentiation algorithm. In *Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems*, pages 300–308, 2001.

[CJ03] Mathieu Ciet and Marc Joye. (virtually) free randomization techniques for elliptic curve cryptography. In *Proceedings of the 5th International Conference on Information and Communications Security*, pages 348–359, 2003.

[CMCJ04] Benoît Chevallier-Mames, Mathieu Ciet, and Marc Joye. Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. *IEEE Transactions on Computers*, 53:760–768, 2004.

[CMO98] Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*, pages 51–65, 1998.

[Cor99] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *Proceedings of the 1st International Workshop on Cryptographic Hardware and Embedded Systems*, pages 292–302, 1999.

[DI06] Christophe Doche and Laurent Imbert. Extended double-base number system with applications to elliptic curve cryptography. In *Proceedings of the 7th International Conference on Cryptology in India*, pages 335–348, 2006.

[DI08] Christophe Doche and Laurent Imbert. The double-base number system in elliptic curve cryptography. In *In Conference Records of the 42nd Asilomar Conference on Signals, Systems and Computers*, pages 777–780, 2008.

[DIM05a] V. S. Dimitrov, L. Imbert, and P. K. Mishra. Fast elliptic curve point multiplication using double-base chains. *IACR Cryptology ePrint Archive*, 2005:69, 2005.

[DIM05b] Vassil Dimitrov, Laurent Imbert, and Pradeep Kumar Mishra. Efficient and secure elliptic curve point multiplication using double-base chains. In *Proceedings of the 11th International Conference on Theory and Application of Cryptology and Information Security*, pages 59–78, 2005.

[DIM08] Vassil Dimitrov, Laurent Imbert, and Pradeep K. Mishra. The double-base number system and its application to elliptic curve cryptography. *Mathematics of Computation*, 77:1075–1104, 2008.

[DJM99] Vassil S. Dimitrov, Graham A. Jullien, and William C. Miller. Theory and applications of the double-base number system. *IEEE Transactions on Computers*, 48:1098–1106, 1999.

[EBJ02] Éric Brier and Marc Joye. Weierstrass elliptic curves and side-channel attacks. In *Proceedings of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems: Public Key Cryptography*, pages 335–345, 2002.

[FGM+10] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. State-of-the-art of secure ecc implementations: a survey on known side-channel attacks and countermeasures. In *Proceedings of the 2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 76–87, 2010.

[FV03] Pierre-Alain Fouque and Frederic Valette. The doubling attack - why upwards is better than downwards. In *Proceedings of the 5th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 269–280, 2003.

[FV12] Junfeng Fan and Ingrid Verbauwhede. An updated survey on secure ecc implementations: Attacks, countermeasures and cost. In *Cryptography and Security*, pages 265–282, 2012.

[GPW+04] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In *Proceedings of the 6th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 119–132, 2004.

[GV10] Christophe Giraud and Vincent Verneuil. Atomicity improvement for elliptic curve scalar multiplication. In *CARDIS*, pages 80–101, 2010.

[HCD07] Huseyin Hisil, Gary Carter, and Ed Dawson. New formulae for efficient elliptic curve arithmetic. In *Proceedings of the Cryptology 8th International Conference on Progress in Cryptology*, pages 138–151, 2007.

[HHM00] Darrel Hankerson, Julio López Hernandez, and Alfred Menezes. Software implementation of elliptic curve cryptography over binary fields. In *Proceedings of the 2nd International Workshop on Cryptographic Hardware and Embedded Systems*, pages 1–24, 2000.

[HMV03] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2003.

[HWCD08] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Twisted edwards curves revisited. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security*, pages 326–343, 2008.

[HWCD09] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Jacobi quartic curves revisited. In *Proceedings of the 14th Australasian Conference on Information Security and Privacy*, pages 452–468, 2009.

[IIT06] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka. Improving the randomized initial point countermeasure against dpa. In *Proceedings of the 4th International Conference on Applied Cryptography and Network Security*, pages 459–469, 2006.

[Joy03] Marc Joye. Elliptic curves and side-channel analysis. *ST Journal of System Research*, 4:17–21, 2003.

[Joy08] Marc Joye. Fast point multiplication on elliptic curves without precomputation. In *Proceedings of the 2nd International Workshop on Arithmetic of Finite Fields*, pages 36–46, 2008.

[JQ01] Marc Joye and Jean-Jacques Quisquater. Hessian elliptic curves and side-channel attacksk. In *Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems*, pages 402–410, 2001.

[JT01] Marc Joye and Christophe Tymen. Protections against differential analysis for elliptic curve cryptography. In *Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems*, pages 377–390, 2001.

[JY02] Marc Joye and Sung-Ming Yen. The montgomery powering ladder. In *Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 291–302, 2002.

[KGSL10] D. Karakoyunlu, F.K. Gurkaynak, B. Sunar, and Y. Leblebici. Efficient and side-channel-aware implementations of elliptic curve cryptosystems over prime fields. *IET Information Security*, 4:30–43, 2010.

[KHM+05] ChangKyun Kim, JaeCheol Ha, SangJae Moon, Sung-Ming Yen, Wei-Chih Lien, and Sung-Hyun Kim. An improved and efficient countermeasure against power analysis attacks. *IACR Cryptology ePrint Archive*, 2005, 2005.

[KY10] Fanyu Kong and Jia Yu. Two efficient algorithms against power attacks for elliptic curve cryptosystems. In *Proceedings of the 2nd International Conference on Signal Processing Systems*, pages V2–148–V2–152, 2010.

[Lan] Adam Langley. Protecting data for the long term with forward secrecy. `http://googleonlinesecurity.blogspot.jp/2011/11/protecting-data-for-long-term-with.html`.

[LM08] Patrick Longa and Ali Miri. Fast and flexible elliptic curve point arithmetic over prime fields. *IEEE Transactions on Computers*, 57:289–302, 2008.

[Lon07] Patrick Longa. Accelerating the scalar multiplication on elliptic curve cryptosystems over prime fields. Master's thesis, University of Ottawa, 2007.

[LWX09] Leibo Li, Mingqiang Wang, and Zhanjiang Xia. New addition operation and its application for scalar multiplication on hessian curves over prime fields. *IACR Cryptology ePrint Archive*, 2009:573, 2009.

[MD07] Pradeep Kumar Mishra and Vassil S. Dimitrov. Efficient quintuple formulas for elliptic curves and efficient scalar multiplication using multibase number representation. In *Proceedings of the 10th International Conference on Information Security*, pages 390–406, 2007.

[MMM04] Hideyo Mamiya, Atsuko Miyaji, and Hiroaki Morimoto. Efficient countermeasures against rpa, dpa, and spa. In *Proceedings of the 6th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 343–356, 2004.

[MV06] Frédéric Muller and Frédéric Valette. High-order attacks against the exponent splitting protection. In *Proceedings of the 9th International Conference on Theory and Practice of Public-Key Cryptography*, pages 315–329, 2006.

[Sma01] Nigel P. Smart. The hessian form of an elliptic curve. In *Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems*, pages 118–125, 2001.

[TMX09] Zhang Tao, Fan Mingyu, and Zheng Xiaoyu. Secure and efficient elliptic curve cryptography resists side-channel attacks. *Journal of Systems Engineering and Electronics*, 20:660–665, 2009.

[Was08] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. Chapman & Hall/CRC, 2008.

[YKMJ06] Sung-Ming Yen, Lee-Chun Ko, SangJae Moon, and Jae. Relative doubling attack against montgomery ladder. In *Proceedings of the 8th International Conference on Information Security and Cryptology*, pages 117–128, 2006.