DYNAMIC PROJECTIVE COORDINATE SYSTEM FOR ELLIPTIC CURVE CRYPTOGRAPHY

BY

THEEB AYEDH AL-GAHTANI

A Dissertation Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

In

COMPUTER SCIENCE AND ENGINEERING

May 2006

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA
DEANSHIP OF GRADUATE STUDIES

This dissertation, written by **Theeb Ayedh Al-Gahtani** under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE AND ENGINEERING**

Dissertation Committee

Dr. Mohammad Al-Suwaiyel

(Dissertation Advisor)

Dr. Mohammad K. Ibrahim

(Member)

Dr. Alaaeldin A. Amin

(Member)

College Dean

Dr. Jarallah S. AlGhamdi

Dean of Graduate Studies

Dr. Mohammad A. Al-Ohali

Date: 14-5-2006

DEDICATION

TO MY PARENTS AND MY FAMILY

ACKNOWLEDGMENTS

Acknowledgement is due to the King Fahd University of Petroleum & Minerals for supporting this research.

Firstly, I would like to thank my major adviser **Dr. Mohammad Al-Suwaiyel** for his inspiration, encouragement and trust.

I wish to express my appreciation to **Professor M. K. Ibrahim** for his guidance and valuable help and encouragement throughout the period of the research and the writing-up.

I also wish to thank **Dr. Alaaeldin A. Amin** for his support throughout my work.

I feel deeply indebted to **Major General Aqeel S. Alaqeel** for his valuable support and encouragement to making this thesis possible.

Warm thanks to **my family** who supported me till the last day of completing this work.

Table of Contents

List of Tables xi

List of Figures xiii

List of Algorithms xiv

Abstract (English) xvi

Abstract (Arabic) xviii

1 Introduction 1

1.1 Scope of the Thesis 6

1.2 Organization of the Thesis 8

2 Finite Field Arithmetic 11

2.1 Introduction 11

2.2 Finite Fields 12

2.3 Finite Field *GF(p)* 13

2.3.1 Finite Field Arithmetic in *GF(p)* 14

2.4 Finite Field $GF(2^m)$ 18

2.4.1 Finite Field Arithmetic in $GF(2^m)$ Using Polynomial Basis 19

2.5 Conclusions 22

3 Elliptic Curve Arithmetic 24

3.1 Introduction 24

3.2 Introduction to Elliptic Curves 25

3.3 Group Law 25

3.4 Elliptic Curve Over Prime Field *GF(p)* 26

3.5 Elliptic Curve Over Binary Field $GF(2^m)$ 31

3.6 Conclusions 35

4 Elliptic Curve Cryptography 36

4.1 Introduction 36

4.2 Elliptic Curve Discrete Logarithm Problem (ECDLP) 37

4.3 ECC Domain Parameters 39

4.4 Elliptic Curve Cryptosystem 40

4.4.1 Symmetric Elliptic Curve Cryptography 41

4.4.2 Public Key Elliptic Curve Cryptography 42

4.5 Scalar Multiplication 44

4.5.1 Binary Methods 45

4.5.2 Window Methods 46

4.5.3 Scalar Recoding Methods 50

4.5.4 Lim/Lee Method 55

4.6 Conclusions 56

5 Coordinate Systems 58

5.1 Introduction 58

5.2 Affine Coordinates 60
5.3 Homogenous Projective Coordinates 61
5.4 Jacobian Coordinates 62
5.5 Chudnovsky-Jacobian Coordinates 63
5.6 Modified Jacobian Coordinates 64
5.7 Mixed Coordinates 65
5.8 Conclusions 67
6 Side Channel Attacks and Countermeasures 69
6.1 Introduction 69
6.2 Classification of Side Channel Attacks 71
6.3 Fault Analysis Attacks 75
6.4 Timing attack 77
6.5 Power Analysis Attacks 78
6.5.1 Simple Power Analysis (SPA) Attack 79
6.5.2 Differential Power Analysis (DPA) Attack 80
6.5.3 Refined Power Analysis (RPA) Attack 82
6.5.4 Zero-value Point Attack (ZPA) 83
6.5.5 Doubling Attack 83
6.5.6 Address-Bit Differential Power Analysis Attack 84
6.6 Electromagnetic Analysis Attacks 85
6.7 Projective Coordinates Leak 87
6.8 Countermeasures 88

6.8.1 Fault Attack Countermeasures 88
6.8.2 Timing Attack Countermeasures 88
6.8.3 SPA Attack Countermeasures 89
6.8.4 DPA Attack Countermeasures 90
6.8.5 Doubling Attack Countermeasures 92
6.8.6 RPA & ZPA Attacks Countermeasures 92
6.8.7 Address-Bit Differential Power Analysis Attack Countermeasures 93
6.8.8 Electromagnetic Attacks Countermeasures 94
6.8.9 Projective Coordinates Leak Countermeasures 94
6.9 Classification of Countermeasures 95
6.10 Conclusions 95
7 Dynamic Projective Coordinate (DPC) System 97
7.1 Introduction 97
7.2 Dynamic Projecting Parameters 100
7.3 Dynamic Projective Coordinate System for *E/GF(p)* 102
7.3.1 General Dynamic Projective Coordinate System for *E/GF(p)* 103
7.3.2 Mixed Dynamic Projective Coordinate System for *E/GF(p)* 106
7.3.3 Optimized Dynamic Projective Coordinate System for *E/GF(p)* 108
7.4 Dynamic Projective Coordinate System for $E/GF(2^m)$ 111
7.4.1 General Dynamic Projective Coordinate System for $E/GF(2^m)$ 112
7.4.2 Mixed Dynamic Projective Coordinate System for $E/GF(2^m)$ 114
7.4.3 Optimized Dynamic Projective Coordinate System for $E/GF(2^m)$ 116
7.5 Conclusions 119

8 Performance and Using of DPC 121

8.1 Introduction 121

8.2 Calculating the Number of Field Operations in DPC System 122

8.3 Performance of DPC for *E/GF(p)* 125

8.4 Performance of DPC for $E/GF(2^m)$ 134

8.5 Using DPC System 141

8.6 Conclusions 143

9 Scalar Multiplication Security In Presence of DPC 144

9.1 Introduction 144

9.2 Countermeasures for Operation and Data Dependent Attacks 145

9.3 Countermeasures for Address-Dependent Attacks 161

9.3.1 Add-Add Algorithm 161

9.3.2 Transition-Based Algorithm 163

9.4 Conclusions 166

10 General Conclusions 167

10.1 Introduction 167

10.2 Overview and Summary of The Work in The Thesis 167

10.2.1 DPC System 168

10.2.2 Performance of DPC System 169

10.2.3 Using DPC System 170

10.2.4 Scalar Multiplication Security in Presence of DPC System 171

10.3 Suggestions for Future Work 174

Appendices 175

Appendix A-I: Derivation of DPC General Addition Formula for *E/GF(p)* 176

Appendix B-I: Derivation of DPC General Doubling Formula for *E/GF(p)* 179

Appendix C-I: Derivation of DPC Optimized Addition Formula for *E/GF(p)* 183

Appendix D-I: Derivation of DPC Optimized Doubling Formula for *E/GF(p)* 185

Appendix A-II: Derivation of DPC General Addition Formula for $E/GF(2^m)$ 187

Appendix B-II: Derivation of DPC General Doubling Formula for $E/GF(2^m)$ 191

Appendix C-II: Derivation of DPC Optimized Addition Formula for $E/GF(2^m)$ 194

Appendix D-II: Derivation of DPC Optimized Doubling Formula for $E/GF(2^m)$ 197

References 200

List of Tables

Table 4. 1 : Diffie-Hellman key agreement scheme....43

Table 5. 1: Costs of Addition and Doubling operations....66

Table 5. 2: Point Conversions among different coordinates....67

Table 6. 1: Classification of side channel attacks. 73

Table 6. 2: Codes of side channel attacks. 74

Table 6. 3: Countermeasures classification, protection, advantages and disadvantages....96

Table 8. 1: Number of field operations in addition formula 7.7 124

Table 8. 2: Computation times for DPC addition operation in *E/GF(p)*. $a \in (0,1)$ 126

Table 8. 3: Computation times for DPC doubling operation in *E/GF(p)*. $a \in (0,1)$ 127

Table 8. 4: Comparisons of field operations using DPC in *E/GF(p)*. 129

Table 8. 5: Hopping cost in DPC system (*E/GF(p)* Addition operation)....131

Table 8. 6: Hopping cost in DPC system (*E/GF(p)* Doubling operation)....131

Table 8. 7: Possible values of L_x and L_y for addition operation in *E/GF(p)*....133

Table 8. 8: Possible values of L_x and L_y for doubling operation in *E/GF(p)*....133

Table 8. 9: Computation times for addition in DPC/*GF*(2^m). $a \in (0,1)$....135

Table 8. 10: Computation times for doubling in DPC/*GF*(2^m). $a \in (0,1)$....136

Table 8. 11: Comparisons of field operations using DPC in *E/GF*(2^m). 137

Table 8. 12: Hopping cost in DPC system (*E/GF*(2^m) Addition operation)....138

Table 8. 13: Hopping cost in DPC system (*E/GF*(2^m) Doubling operation)....138

Table 8. 14: Possible values of L_x and L_y for addition operation in *E/GF*(2^m)....140

Table 8. 15: Possible values of L_x and L_y for doubling operation in $E/GF(2^m)$ 140

Table 9. 1: Expected running times of algorithm 9.1 for specified DPC systems 152

Table 9. 2: Expected running times of algorithm 9.2 for specified DPC systems 153

List of Figures

Figure 2. 1: Representation of $A \in GF(p)$ as an array of W-bits..... 14

Figure 3. 1: Hierarchal organization of elliptic curve arithmetic. 24

Figure 6. 1: Side channel leak Information. 69

Figure 6. 2: Address-bit differential power analysis attack 85

Figure 9. 1: Two examples of Add-Add algorithm. 162

Figure 9. 2: Two examples Transition-Based algorithm. 164

List of Algorithms

Algorithm 2. 1: Multiprecision addition 15

Algorithm 2. 2: Addition in *GF(p)* 15

Algorithm 2. 3: Multiprecision subtraction 16

Algorithm 2. 4: Subtraction in *GF(p)* 16

Algorithm 2. 5: Shift-and-add method for modular multiplication in *GF(p)* 17

Algorithm 2. 6: Inversion using extended Euclidean algorithm in *GF(p)*. 17

Algorithm 2. 7: Bit-level method for addition in $GF(2^m)$ 20

Algorithm 2. 8: Bit-level method for modular reduction in $GF(2^m)$ 20

Algorithm 2. 9: Shift-and-add method for modular multiplication in $GF(2^m)$ 21

Algorithm 2. 10: Bit-level method for squaring in $GF(2^m)$ 21

Algorithm 2. 11: Inversion using extended Euclidean algorithm in $GF(2^m)$ 22

Algorithm 4. 1: Least-to-Most (LM) binary algorithm for scalar multiplication 46

Algorithm 4. 2: Most-to- Least (ML) binary algorithm for scalar multiplication 46

Algorithm 4. 3: *m-ary* method for scalar multiplication 47

Algorithm 4. 4: Modified *m-ary* method for scalar multiplication 48

Algorithm 4. 5: Sliding window method for scalar multiplication 50

Algorithm 4. 6: Computation of NAF(K) 51

Algorithm 4. 7: Computation of $NAF_w(K)$ 52

Algorithm 4. 8: Binary NAF algorithm (addition-subtraction) for scalar multiplication .. 53

Algorithm 4. 9: *width-w* window method for scalar multiplication 54

Algorithm 4. 10: Lim/Lee method for scalar multiplication 56

Algorithm 6. 1: *Double-and-ADD always* Most-to-Least (ML) binary algorithm. 89

Algorithm 6. 2: *Double-and-ADD always* Least-to-Most (LM) binary algorithm. 90

Algorithm 6. 3: Takagi's ML algorithm for scalar multiplication. 90

Algorithm 9. 1: Binary ML algorithm with countermeasure1 149

Algorithm 9. 2: Binary NAF algorithm with countermeasure1 149

Algorithm 9. 3: Binary ML algorithm with countermeasure2 155

Algorithm 9. 4: Binary NAF algorithm with countermeasure2 155

Algorithm 9. 5: Binary ML algorithm with countermeasure3 157

Algorithm 9. 6: Binary NAF algorithm with countermeasure3 158

Algorithm 9. 7: Add-Add algorithm 162

Algorithm 9. 8: Transition-based algorithm 163

Abstract (English)

Student's Name: Theeb Ayedh Al-Gahtani

Title: DYNAMIC PROJECTIVE COORDINATE SYSTEM FOR ELLIPTIC CURVE CRYPTOGRAPHY

Major Field: Computer Science and Engineering

Date of Graduate 5-2006

Scalar multiplication is the basic operation in elliptic curve cryptography that can be performed by many algorithms. These algorithms multiply a scalar value *K* with an elliptic curve base point *P*. One of the crucial decisions when implementing an efficient elliptic curve cryptosystem is deciding which point coordinate system to use. The point coordinate system used for addition and doubling of points on the elliptic curve determines the efficiency of these routines, and hence the efficiency of the basic cryptographic operation, scalar multiplication. Although using a *fixed* coordinate system enhances the performance of the scalar multiplication, (by removing the intermediate inversion operations), it becomes a security weakness since it can be exploited by projective coordinates leak attacks to reveal some secure information. Therefore, finding a coordinate system that can enhance the performance of the scalar multiplication and being secure against such attacks is desired goal.

This thesis introduces a new approach called ***Dynamic Projective Coordinate (DPC) system.*** DPC provides a *framework* that automates the selection of the projective coordinate system and uses a single mathematical formulation/software code to

implement different projective coordinate systems. This framework allows the computing/encrypting device to select the projective coordinate either at random, or according to a certain rule.

DPC uses dynamic transformation functions to convert coordinates of any point on the elliptic curve to any projective coordinates by using the same mathematical formula. These transformation functions are used to develop dynamic addition and doubling formulas for elliptic curve over the prime field *GF(p)* and over the binary field $GF(2^m)$.

Also, this thesis proposes a new classification method for Side Channel Attacks (SCA). This classification is based on the type of information being leaked which can be *Operation-dependent, Data-dependent* , *Address-dependent* or any combination of them. New countermeasures for data-dependent, data-and-operation dependent and address-dependent attacks are proposed. These countermeasures are based on the fact that DPC lends itself to randomize both the data being manipulated and the number of operations being performed by randomizing the coordinate system used.

Abstract (Arabic)
ملخص بحث درجة الدكتوراه في الفلسفة

الاســـــم: ذيب عايض القحطاني

عنوان الرسالة: نظام الإحداثيات الحركي لأنظمة التشفير باستخدام المنحنى البيضاوي

التخصص: علوم وهندسة الحاسب الآلي

تاريخ التخرج: 2006/5 م

إن العملية الأساسية في نظام التشفير باستخدام المنحنيات البيضاوية هي عملية ضرب نقطة مـن نقاط المنحنى بعدد سري صحيح يرمز له بالرمز (ك). هذه العملية تتم عن طريق سلسلة مـن عمليـات إضافة نقطتين إلى بعضهما وإضافة نقطة إلى نفسها (مضاعفة النقطة) وذلك عبر خطوات كـل خطـوة تعطي نتيجة مرحلية حتى يتم الحصول على النتيجة النهائية. وجميع هذه العمليات تتم عن طريق حـساب معادلات رياضية على إحداثيات النقاط. وسرعة تنفيذ هذه المعادلات الرياضية تعتمد بدرجة كبيرة علـى نظام الإحداثيات المستخدم لتمثيل النقاط على المنحنى البيضاوي. ففي حالة اسـتخدام نظـام الإحـداثيات العادية فإن كل خطوة من خطوات ضرب نقطة بعدد صحيح تحتاج إلى عملية قسمة والتي من المعروف أنها تحتاج إلى وقت أكبر بكثير مقارنة بالعمليات الحسابية الأخرى كالضرب والجمع. ولحل هذه المشكلة فإنه يتم استخدام نظام الإحداثيات الإسقاطية بدلاً من نظام الإحداثيات العادية والتي مـن خلالهـا يمكـن الاستغناء عن عمليات القسمة في النتائج المرحلية. وبرغم أن هذه الطريقة أدت إلى تحـسين أداء عمليـة ضرب نقطة بعدد صحيح بشكل كبير جداً إلا أنها تمثل نقطة ضعف أمنية يمكن من خلالها النفـاذ إلـى

معرفة الرقم السري (ك). ولذلك فإن الحاجة ماسة إلى إيجاد طريقة يمكن من خلالها الاستغناء عن عمليات القسمة في النتائج المرحلية وتكون في نفس الوقت آمنه لإتمام عملية ضرب نقطة من نقاط المنحنى البيضاوي بعدد صحيح.

تقدم هذه الرسالة طريقة مبتكرة تسمى نظام الإحداثيات الإسقاطية الحركي (الديناميكي) والذي يتم عن طريقه اختيار نظام الإحداثيات الإسقاطية بشكل آني وتلقائي بدلاً من استخدام نظام إحداثي واحد يكون عرضة للاختراق. وهذه الطريقة تستخدم معادلة رياضية موحدة وبالتالي برمجيات موحدة لعملية إضافة نقطتين إلى بعضهما. وكذلك تستخدم معادلة رياضية موحدة وبالتالي برمجيات موحدة لعملية إضافة نقطة إلى نفسها. وبذلك يمكن من خلال هذه المعادلات الرياضية والبرمجيات الموحدة تمثيل إي من الإحداثيات الإسقاطية المعروفة دون الحاجة إلى استخدام معادلة رياضية مستقلة وبرمجيات مستقلة لكل نظام إحداثيات.

وتقدم هذه الرسالة أيضاً طريقة جديدة لتصنيف طرق اختراق أنظمة التشفير بناء على نوع المعلومات المستغلة في عملية الاختراق. فمنها ما يعتمد على نوع العملية المنفذة (كإضافة نقطتين أو مضاعفة نقطة)، ومنها ما يعتمد على البيانات المدخلة إلى نظام التشفير، ومنها ما يعتمد على مواقع تخزين المعلومات في ذاكرة أجهزة التشفير. وبناء على هذا التصنيف، فإن هذه الرسالة تقدم طرق جديدة لمقاومة هذه الأنواع من الاختراقات باستخدام نظام الإحداثيات الحركي المقترح. وتعتمد طرق المقاومة المقترحة على خاصية مهمة لنظام الإحداثيات الإسقاطية الحركي وهي إمكانية اختيار نظام الإحداثيات الإسقاطية بشكل عشوائي وآني خلال خطوات ضرب نقطة من نقاط المنحنى بالعدد السري (ك) مما يجعل المعلومات الجانبية التي يجمعها المخترق عديمة الفائدة.

CHAPTER 1

Introduction

Cryptography provides methods of providing privacy and authenticity for remote communications and data storage. Privacy is achieved by encryption of data, usually using the techniques of symmetric cryptography (so called because the same mathematical key is used to encrypt and decrypt the data). Authenticity is achieved by the functions of user identification, data integrity, and message non-repudiation. These are best achieved via asymmetric (or public-key) cryptography.

In particular, public-key cryptography enables encrypted communication between users that have not previously established a shared secret key between them. This is most often done using a combination of symmetric and asymmetric cryptography: public-key techniques are used to establish user identity and a common symmetric key, and a symmetric encryption algorithm is used for the encryption and decryption of the actual messages. The former operation is called key agreement. Prior establishment is necessary in symmetric cryptography, which uses algorithms for which the same key is used to encrypt and decrypt a message. Public-key cryptography, in contrast, is based on key pairs. A key pair consists of a private key and a public key. As the names imply, the private key

is kept private by its owner, while the public key is made public (and typically associated to its owner in an authenticated manner). In asymmetric encryption, the encryption step is performed using the public key, and decryption using the private key. Thus the encrypted message can be sent along an insecure channel with the assurance that only the intended recipient can decrypt it.

User identification is most easily achieved using what are called identification protocols. A related technique, that of digital signatures, provides data integrity and message non-repudiation in addition to user identification.

The public key is used for encryption or signature verification of a given message, and the private key is used for decryption or signature generation of the given message.

Koblitz [1] and Miller [2] proposed a method by which public key cryptosystems can be constructed on a *group* of points of an elliptic curve. This group comes from a setting called *finite fields* (chapter 2).

Elliptic Curve Cryptosystem (ECC) relies upon the difficulty of the *Elliptic Curve Discrete Logarithm Problem (ECDLP)* to provide its effectiveness as a cryptosystem. Using multiplicative notation, ECDLP can be described as (section 4.2): given elliptic curve points P and Q in the group, find a number K such that $P^K=Q$; where K is called the discrete logarithm of Q to the base P. Using additive notation, the problem becomes: given two points P and Q in the group, find a number K such that $KP=Q$.

In an ECC, the large integer K is kept private and is often referred to as the secret key. The point Q together with the base point P are made public and are referred to as the

public key. The security of the system, thus, relies upon the difficulty of deriving the secret *K*, knowing the public points *P* and *Q*. The main factor that determines the security strength of such a system is the size of its underlying finite field. In a real cryptographic application, the underlying field is made so large that it is computationally infeasible to determine *K* in a straightforward way by computing all the multiples of *P* until *Q* is found.

The core of the elliptic curve cryptography is an operation called *scalar multiplication* which computes *KP* by adding together *K* copies of the point *P*. Thus, the efficiency of elliptic curve cryptosystems heavily depends on the implementation of the scalar multiplication. The scalar multiplication is performed through a combination of *point-doubling* and *point-addition* operations. The point-addition operation adds two distinct points together and the point doubling operation adds two copies of a point together. To compute, for example, 11P = (2*(2*(2P)))+3P = Q, it would take 3 point-doublings and 1 point-addition.

Point addition and doubling operations require field inversion operations which usually have very high cost (i.e. number of finite field operations required) compared to the multiplication operation (see section 5.1). Its cost ranges from 9 to 30 field multiplications for a field element with bit length grater than 100 [23]. Moreover, it must be (without projective coordinate) performed in each iteration of the scalar multiplication. Therefore, it is important to represent elliptic curve points using projective coordinates. The idea of projective coordinates is based on transferring the point coordinates into another coordinates that can eliminate the inversion operation while performing addition and doubling operations. By this way, the intermediate inversions within the scalar

multiplication iterations are eliminated. However, still we need one final inversion to return back to the affine coordinates after completion of the scalar multiplication.

Transferring any elliptic curve point to projective coordinates can be achieved by using transformation functions. Different projective coordinates use different transformation functions [23], [24], [25]. In this thesis, the sentence "*projective coordinate system*" is used when referring to the transformation functions as well as the coordinates generated by these functions, and the sentence "*projective coordinates*" is used when referring the values of coordinates of a point.

Every computing device acts also as a source of additional information usually called side channel leak information. Depending on its internal computations, it consumes different amounts of power, emits different amounts of electromagnetic emanations, needs different running times or even produces different types of error messages or sounds. All these additional types of information can and have already been exploited in attacking the cryptodevices.

In the execution of ECC, side channel attacks have become serious threat. One of the most side channel attacks is the power analysis attacks, first introduced in [26], [27]. Power analysis attacks monitor power consumption and exploit the leakage information related to power consumption to reveal bits of a secret key K although K is hidden inside the cryptodevice. Thus, it is a serious issue that the implementation should be resistant against SPA and DPA, and many countermeasures have been proposed in [28] – [37]. We may note here that almost all public key cryptosystems including RSA and DLP-based

cryptosystems also execute an exponentiation algorithm with a secret-key exponent, and, thus, they also suffer from both SPA and DPA in the same way as ECC. Recently, in the case of elliptic curve cryptosystems, DPA is further improved to the Refined Power Analysis (RPA) in [28], which exploits a special point with a zero value and reveals a secret key. An elliptic curve happens to have a special point (0, y) or (x, 0), which can be controlled by an adversary because the order of base point is usually known. RPA utilizes such a feature that the power consumption of 0 is distinguishable from that of an non-zero element. Although ECC are vulnerable to RPA, RPA are not applied to RSA or DLP-based cryptosystems because they don‘t have such a special zero element. Furthermore, RPA is generalized to Zero-value Point Attack (ZPA) in [29]. ZPA makes use of any zero-value register used in the addition formula. To make matters worse, some previous efficient countermeasures of the randomized-projective-coordinate method (RPC) [32] are neither resistant against RPA nor ZPA because, a special point (0, y) or (x, 0) has still a zero value even if it is converted into (0, ry, r) or (rx, 0, r) by using RPC.

In 2004, Nigel Smart et. al. [42] showed that it is possible to leak some information about the secret key (scalar K) through the projective representation of elliptic curve points. Giving that $Q = KP$ is the elliptic-curve double-and-add scalar multiplication of a public base point P by a secret K, they showed that allowing an adversary access to the projective representation of Q, obtained using a particular double and add method, may result in information being revealed about K. A countermeasure for such an attack is proposed also in [42] but they assume that the attacker knows the projective coordinate system used and that the coordinate system is fixed.

1.1 Scope of the Thesis

The existing projective coordinate systems and the countermeasures based on them lack the following issues that can be used to enhance the security and/or performance of the scalar multiplication.

First, issues related to the efficiency of the scalar multiplication:

1. Each coordinate system needs its own mathematical formulation/software code and if a different coordinate system is used, it is required to change the microcode of the scalar multiplication.
2. It is a costly operation to convert from one coordinate system to another during the scalar multiplication since this requires an inversion operation.

Second, issues related to the security:

1. The available projective coordinate systems are very limited in number.
2. Vulnerability to RPA, ZPA and projective coordinate leak [31].
3. Existing countermeasures for power analysis attacks that use randomization of projective coordinates such as those introduced in [32] and the countermeasure proposed in [42] for projective coordinate leakage assume that projective coordinate system is fixed and they do not pursue the direction of changing the projective coordinate system randomly during the scalar multiplication due to the efficiency problems mentioned above.

This thesis introduces a new approach for scalar multiplication called *dynamic projective coordinate (DPC)* system. We mean by dynamic projective coordinate system, is a system that automates the selection of the projective coordinate system and uses a

single mathematical formulation/software code to implement different projective coordinate systems. Also, DPC allows projective coordinates hopping at any time during the scalar multiplication with taking into account the efficiency and security issues mentioned above.

Different projective coordinates are implemented by using two *projecting parameters* where one parameter defines the projection of the *x*-coordinate and a second parameter defines the projection of the *y*-coordinate of an elliptic curve point. This allows different projective coordinates to be used within the same mathematical formulation in calculating the scalar multiplication.

These parameters are used to define dynamic transformation functions that can be used to convert any affine point to any projective coordinates using the same mathematical formula. These transformation functions are used to develop dynamic addition and doubling formulas for elliptic curve over the prime field *GF(p)* and elliptic curve over binary field $E/GF(2^m)$.

In this thesis a survey of side channel attacks for ECC is presented in chapter 6. Based on that survey, we introduce a new classification of side channel attacks that can help in providing new countermeasures to cover the weaknesses of the existing ones. The proposed classification is based on the type of information being leaked. It divides all known attacks into three classes: Class A: *Operation-dependent* attacks that depend on the type of operation being performed (multiply, square, addition, doubling, etc…) such as simple power analysis attacks [26]. Class B: *Data-dependent* attacks that are based on the data being manipulated by the cryptodevice such as fault attacks [34]-[45] and projective

coordinate leaks [42]. Class C: *Address-dependent* attacks that are based on the addresses (locations) of the data being processed such as address-bit differential power analysis attacks [38]. There are, however, some attacks, called data-and-operation dependent attacks, that are both operation-dependent and data-dependent such as timing [27] and DPA [26] attacks.

However, an important feature of DPC is that by randomizing the projecting parameters (mentioned above) in addition and doubling DPC formulas, both the data being manipulated and the number of operations being performed are randomized. This fact is used to propose new countermeasures for data-dependent, data-and-operation dependent and address-dependent attacks.

1.2 Organization of the Thesis

The rest of this thesis is divided into 9 chapters. Chapter 2, presents an introduction to finite fields arithmetic. There are two kinds of finite fields that are especially preferred for the efficient implementation of elliptic curve cryptosystems. These fields are the prime field, *GF(p)*, and the binary field $GF(2^m)$. This chapter presents the definition of these fields and the basic arithmetic operations that can be performed on their elements. Also, various algorithms to perform arithmetic operations in the prime and binary finite fields are addressed in this chapter.

Chapter 3 discusses the mathematical background of elliptic curves over finite fields. Curve arithmetic is defined in terms of underlining field operations. This includes the fundamentals of defining elliptic curve over the prime field *GF(p)* and the binary field

GF(2^m).

Chapter 4 presents the principles of elliptic curve cryptography (ECC). It includes definition of the underlining hard problem, Elliptic Curve Discrete Logarithm Problem (ECDLP), that the security of ECC is based on. Also, it illustrates the domain parameters that are required to set up an ECC and the basic principles of symmetric and public key ECC . Finally, different scalar multiplication algorithms are addressed in this chapter.

Chapter 5 surveys the existing projective coordinate systems, namely, *Affine* (A), *Homogenous Projective* (H), *Jacobian* (J), *Chudnovsky-Jacobian* (C), *Modified* (M) and *mixed* coordinate systems. We start this chapter by showing the cost of inversion operation in some recommended curves to show the motivation behind using projective coordinates. Also, this chapter presents the cost (in terms of the number of field multiplications and squaring) of point addition and doubling for each coordinate system. Furthermore, it gives the cost of converting a point from one projective coordinate to another.

In chapter 6, we survey different types of side channel attacks and the various countermeasures known at the time of writing. Also, the classification methods of the attacks found in the literature are discussed. Based on that, we propose a new classification method according to the type of information being leaked. This classification method is used to classify and analyze both the attacks and countermeasures.

Chapter 7 introduces the proposed *dynamic projective coordinate (DPC)* system for ECC over both finite fields *GF(p)* and *GF(2^m)*. In this chapter, we start by defining

dynamic transformation functions which are used to develop dynamic addition and doubling formulas for elliptic curve over the prime field *GF(p)* and elliptic curve over binary field $E/GF(2^m)$.

Chapter 8 analyzes the performance and discusses the use of DPC. To analyze the performance of DPC, the number of field operations in each formula of the formulas presented in chapter 7 is calculated. We provide the method by which we can calculate the number of field operations in any DPC formula. Also, the issue of how the DPC can be used is discussed in this chapter.

In chapter 9, we propose and analyze countermeasures for operation-and-data dependent, data-dependent and address-dependent attacks. All the proposed countermeasures are based on using the DPC system as the coordinate system. This is because the DPC system lends itself to randomization simply by randomizing the projecting parameters. For each countermeasure, we provide the security and complexity analysis.

Finally, conclusions are drawn in chapter 10. This includes a summary of the results obtained in this thesis. Suggestions for further work are also recommended at the end of this chapter.

CHAPTER 2

Finite Field Arithmetic

2.1 Introduction

Cryptographic mechanisms based on elliptic curves depend on arithmetic involving the points of the curve. Curve arithmetic is defined in terms of underlining field operations which its efficiency is essential. From a practical point of view, the performance of ECC depends on the efficiency of finite field computations and fast algorithms for elliptic scalar multiplications (section 4.5). In addition to the numerous known algorithms for these computations, the performance of ECC can be sped up by selecting particular underlying finite fields and/or elliptic curves. Thus, a fast implementation of a security application based on ECC requires several choices, any of which can have a major impact on the overall performance.

This chapter introduces finite fields and the various algorithms to perform arithmetic operations in these fields. An introduction to groups and finite fields is provided in Section 2.2. There are tow kinds of finite fields that are especially prefer for the efficient implementation of elliptic curve cryptosystems. These fields are the prime

field, *GF(p)*,and the binary field $GF(2^m)$. Sections 2.3 and 2.4 present the definition of these fields and the basic arithmetic operations that can be performed in each of them. Finally, conclusions are presented in section 2.5.

2.2 Finite Fields

In this section we present the definition of groups and finite fields. These mathematical structures are fundamental for the construction of an elliptic curve cryptosystem.

A *group* is an algebraic system consisting of a set G together with a binary operation $\Diamond$ defined on G satisfying the following axioms:

- Closure: for all x , y in G we have $x \Diamond y \in G$.
- Associativity: for all x , y and z in G we have $(x \Diamond y) \Diamond z = x \Diamond (y \Diamond z)$.
- Identity: there exists an e in G such that $x \Diamond e = e \Diamond x = x$ for all x in G.
- Inverse: for all x in G there exists y in G such that $x \Diamond y = y \Diamond x = e$.

If in addition, the binary operation $\Diamond$ satisfies the abelian property:

- abelian: for all x , y in G we have $x \Diamond y = y \Diamond x$,

Then we say that the group G is *abelian*.

A *finite field* is an algebraic system consisting of a finite set $\boldsymbol{F}$ together with two binary operations + and $\times$, defined on $\boldsymbol{F}$ satisfying the following axioms:

- $\boldsymbol{F}$ is an abelian group with respect to "+".
- $\boldsymbol{F} \setminus \{0\}$ is an abelian group with respect to "$\times$"

- distributive: for all *x, y* and *z* in ***F*** we have:

$$x \times (y + z) = (x \times y) + (x \times z)$$

$$(x + y) \times z = (x \times z) + (y \times z).$$

The *order* of a finite field is the number of elements in the field. A fundamental result on the theory of finite fields [6] that characterizes the existence of finite field is the following: there exists a finite field of order *p* if and only if *p* is a prime. In addition, if *p* is a prime, then there is essentially only one finite field of order *p*. this field is denoted by *GF(p)* (or F_p). However, there are many ways of representing the elements of *GF(p)*, and some representations may lead to more efficient implementations of the field arithmetic in hardware or in software.

if $p = q^m$ where *q* is a prime and *m* is a positive integer, then *q* is called the *characteristic* of *GF(p)* and *m* is called the *extension degree* of *GF(p)*. Most standards which specify ECC restrict the order of the underlying finite field to be an odd prime (*p = q, i.e. m=1*) which result in *GF(p)* finite field, or restrict the order to a power of 2 ($p = 2^m$, i.e. *q=2*) which result in what called *characteristic two finite field* and denoted by $GF(2^m)$. In the following sections, we will describe these two finite fields and present the basic algorithms for performing arithmetic operations in each of them.

2.3 Finite Field *GF(p)*

Definition 2.1: *Prime Field GF(p)*.

Let *p* be a prime number. The integers modulo *p*, consisting of the integers {0, 1, 2,

…, $p - 1\}$ with addition and multiplication performed modulo p, is a *finite field* of order p called *prime field* and denoted by *GF(p)*. The prime number p is called the *modulus* of *GF(p)*.

2.3.1 Finite Field Arithmetic in *GF(p)*

This section presents algorithms for performing arithmetic in the prime file *GF(p)*. The algorithms presented here are well suited for software implementation. We assume that the implementation platform has a W-bit architecture where W is a multiple of 8. Let $m = \lceil \log_2 p \rceil$ be the bit length of p, and $t = \lceil m/W \rceil$ be its word length. Figure 2.1 illustrates a binary representation of a field element A as an array of W-bit words. As an integer,

$$A = 2^{(t-1)W} a[t-1] + 2^{(t-2)W} a[t-2] + ... + 2^{2W} a[2] + 2^{W} a[1] + a[0].$$

a[t-1]	…	*a[2]*	*a[1]*	*a[0]*

Figure 2. 1: Representation of $A \in GF(p)$ as an array of W-bits

The following notation is used in algorithms for multiword integers. An assignment of the form "(ε,Z) ← A" for an integer A means:

$Z = A \bmod 2^W$, and

ε = 0 if A in [0, $2^W - 1$], otherwise ε = 1.

ε is called the carry bit from single word addition.

Addition: If $a,b \in GF(p)$, then $a + b = r$, where r is the remainder of the division of $(a+b)$

by p and $0 \leq r \leq p - 1$. This operation is called *addition modulo p*. To perform addition operation for multi-word integers in *GF(p)*, we first perform multiprecision addition followed by an additional step for reduction modulo p. The following two algorithms present multiprecision addition and reduction modulo p respectively.

Input: integers $A,B \in [0,2^{Wt} - 1]$
Output: (ε,C) where $C = A + B \bmod 2^{Wt}$
1. $(\varepsilon,c[0]) \leftarrow a[0] + b[0]$
2. for $i = 1$ to t-1 do
   $(\varepsilon,c[i]) \leftarrow a[i] + b[i] + \varepsilon$
3. return (ε,C)

Algorithm 2. 1: Multiprecision addition

Modular addition in *GF(p)*, ($C = A + B \bmod p$), is adapted directly from the corresponding multiprecision addition algorithm with an additional step for reduction modulo p.

Input: modulus p and integers $A,B \in [0, p - 1]$
Output: $C = (A + B) \bmod p$
1. Use algorithm 2.1 to obtain (ε,C) where $C = A + B \bmod 2^{Wt}$ and ε is the carry bit.
2. if ($\varepsilon = 1$ or $C \geq p$) then
   $C = C - p$ // *subtract modulus*.
3. return (ε,C)

Algorithm 2. 2: Addition in *GF(p)*

Subtraction: If $a,b \in GF(p)$, then $a - b = r$, where r is the remainder of the division of (a-b) by p and $0 \leq r \leq p - 1$. This operation is called *subtraction modulo p*. To perform subtraction operation for multi-word integers in *GF(p)*, we first perform multiprecision subtraction followed by an additional step for reduction modulo p. Note that we need a

reduction step here because we may have a negative result which must be reduced to the range $[0, p-1]$. We mean by reduction here is adding the modulus p to the negative result if any. The following two algorithms present multiprecision subtraction and reduction-for-subtraction modulo p respectively.

Input: integers $A,B \in [0,2^{Wt}-1]$
Output: (ε,C) where $C = A - B \bmod 2^{Wt}$ and ε is the borrow bit
1. $(\varepsilon,c[0]) \leftarrow a[0] - b[0]$
2. for $i = 1$ to t-1 do
   $(\varepsilon,c[i]) \leftarrow a[i] - b[i] - \varepsilon$
3. return (ε,C)

Algorithm 2. 3: Multiprecision subtraction

Modular subtraction in $GF(p)$, ($C = A - B \bmod p$), is adapted directly from the corresponding multiprecision subtraction algorithm with an additional step for reduction modulo p.

Input: modulus p and integers $A,B \in [0, p-1]$
Output: $C = (A + B) \bmod p$
1. Use algorithm 2.3 to obtain (ε,C) where $C = A - B \bmod 2^{Wt}$ and ε is the borrow bit.
2. if ($\varepsilon = 1$) then
   $C = C + p$ // *add modulus*.
3. return (ε,C)

Algorithm 2. 4: Subtraction in $GF(p)$

Multiplication: If $a,b \in GF(p)$, then $a \,.\, b = s$, where s is the remainder of the division of $(a.b)$ by p and $0 \leq s \leq p-1$. This operation is called *multiplication modulo p*.

The basic method for performing a multiplication in *GF(p)* is the "shift-and-add" method. Given $A \in GF(p)$, the shift-left operation, $(A << 1) \bmod p$ can be performed as modulo addition of A to itself using algorithm 2.2. That is: $A = (A + A) \bmod p$. The steps of the "shift-and-add" multiplication method are given below.

Input: $A,B \in GF(p)$ and the modulus p
Output: $C = A \times B \bmod p$
1. set $C = 0$
2. for $i = m-1$ to 0 do
   $C = C + C \bmod p$ //shift left
   If $b_i \neq 0$ then $C = C + A$ //use algorithm 2.2
3. return (C)

Algorithm 2. 5: Shift-and-add method for modular multiplication in *GF(p)*.

Inversion: The inverse of a nonzero element $a \in GF(p)$, denoted $(a)^{-1} \bmod p$ or simply $(a)^{-1}$, is the unique element in *GF(p)* such that $a.x = 1$ in *GF(p)*, i.e. $a.x = 1 \pmod p$. The basic algorithm for computing multiplicative inverses in *GF(p)* is the extended Euclidean algorithm as shown below.

Input: $A \in GF(p)$, $(A \neq 0)$ and the modulus p
Output: $C = A^{-1} \bmod p$
1. set $U = A$, $V = p$
   set $X_1 = 1$, $X_2 = 0$
2. while $U \neq 1$ do
   $Q = \lfloor V/U \rfloor$, $R = V - QU$, $X = X_2 - Q X_1$.
   $V = U$, $U = R$, $X_2 = X_1$, $X_1 = X$.
3. return ($X_1 \bmod p$)

Algorithm 2. 6: Inversion using extended Euclidean algorithm in *GF(p)*.

However, several techniques for implementing the finite field arithmetic in F_p are

described in details in [7], [8], [9], [10], [11], and [12].

2.4 Finite Field *GF(2^m)*

Definition 2.2: *Binary Field* $GF(2^m)$

The finite field $GF(2^m)$, called a *binary finite field*, can be viewed as a vector space of dimension *m* over *GF(2)*. That is, there exist a set of *m* elements $\{\alpha_0, \alpha_1, ..., \alpha_{m-1}\}$ in $GF(2^m)$ such that each $a \in GF(2^m)$ can be written uniquely in the form

$$a = \sum_{i=0}^{m-1} a_i \alpha_i \quad \text{where, } a_i \in \{0,1\}.$$

The set $\{\alpha_0, \alpha_1, ..., \alpha_{m-1}\}$ is called a *basis* of $GF(2^m)$ over *GF(2)*. We can then represent *a* as a binary vector $(a_0, a_1, ..., a_{m-1})$. In the sequel, we introduce the most common basis: *polynomial basis*.

Polynomial basis

Let $F(x) = x^m + \sum_{i=0}^{m-1} f_i x^i$ where $f_i \in \{0,1\}$, for *i = 0,1, ..., m-1* be an *irreducible polynomial*[1] of degree *m* over *GF(2)*. *F(x)* is called the *reduction polynomial*. For each reduction polynomial, there exists a polynomial basis representation. In such a

[1] A polynomial is said to be irreducible if it cannot be factored into nontrivial polynomials over the same field

representation, each element of $GF(2^m)$ corresponds to a binary polynomial of degree less than *m*. That is, for $A \in GF(2^m)$ there exists *m* numbers $a_i \in \{0,1\}$ such that

$$A = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_1 x + a_0$$

The field element $A \in GF(2^m)$ is usually denoted by the bit string $(a_{m-1}a_{m-2}.....a_1a_0)$ of length *m*.

The following procedure is commonly used to choose a reduction polynomial: if an irreducible *trinomial*[2] $x^m + x^k + 1$ exists over *GF(2)*, then the reduction polynomial *F(x)* is chosen to be the irreducible trinomial with the lowest-degree middle term x^k. If no irreducible trinomial exists, then select instead a pentanomial $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, such that k_1 has the minimal value; the value of k_2 is minimal for the given k_1; and k_3 is minimal for given k_1 and k_2.

2.4.1 Finite Field Arithmetic in $GF(2^m)$ Using Polynomial Basis

In this section, we describe algorithms for performing arithmetic operations in the finite field $GF(2^m)$ using polynomial basis representation.

Addition. Addition in $GF(2^m)$ is the usual addition of vectors over *GF(2)*. That is, add the corresponding bits modulo 2, i.e. performing bitwise Xoring.

[2] A polynomial with three terms

Input: $A = a_{m-1}a_{m-2}.....a_1a_0$, $B = b_{m-1}b_{m-2}.....b_1b_0 \in GF(2^m)$
Output: $C = A + B = c_{m-1}c_{m-2}.....c_1c_0 \in GF(2^m)$
4. for *i = 0* to *m-1* do
   $c_i = a_i \oplus b_i$
5. return (C)

Algorithm 2. 7: Bit-level method for addition in $GF(2^m)$

Reduction. By the definition of multiplication in $GF(2^m)$, the result of a polynomial multiplication or squaring has to be reduced modulo a reduction (irreducible) polynomial of degree *m*. This reduction operation is particularly efficient when the irreducible polynomial *F(x)* is a trinomial or pentanomial. The following algorithm for computing *A(x)* mod *F(x)* works by reducing the degree of *A(x)* until it is less than *m*.

Input: $A = a_{2m-2}.....a_1a_0$ and $F = f_m f_{m-1} f_{m-2}.....f_1f_0$
Output: *C = A mod F*
1. for *i = 2m-2* to *m* do
   for *j = 0* to *m-1* do
   If $f_j \neq 0$ then $a_{i-m+j} = a_{i-m+j} + a_i$
2. return ($C = a_{m-1}a_{m-2}.....a_1a_0$)

Algorithm 2. 8: Bit-level method for modular reduction in $GF(2^m)$

Multiplication. The basic method for performing a multiplication in $GF(2^m)$ is the "shift-and-add" method. Given $A(x) \in GF(2^m)$, the shift-left operation *xA(x) mod F(x)* can be performed as follows:

$$xA(x) \bmod F(x) = \begin{cases} \sum_{j=1}^{m-1} a_{j-1}x^j & if\ a_{m-1} = 0 \\ \sum_{j=1}^{m-1} (a_{j-1} + f_i)x^j + f_0 & if\ a_{m-1} \neq 0 \end{cases}$$

Then the steps of the "shift-and-add" method are given below.

Input: $A(x), B(x) \in GF(2^m)$ and $F = f_m f_{m-1} f_{m-2} \ldots\ldots f_1 f_0$
Output: $C = A \times B \bmod F$
4. set $C(x) = 0$
5. for $i = m-1$ to 0 do
   $C(x) = xC(x) \bmod F(x)$
   If $a_i \neq 0$ then $C(x) = C(x) + B(x)$ //*use algorithm 2.7*
6. return ($C(x)$)

Algorithm 2. 9: Shift-and-add method for modular multiplication in $GF(2^m)$.

A faster modular multiplication is proposed in [50] but it requires more temporary storage.

Squaring. This operation can be calculated in an efficient way by observing that the square of a polynomial $A(x)$ is given by:

$$(A(x))^2 = \left(\sum_{i=1}^{m-1} a_i x^i\right)^2 = \sum_{i=1}^{m-1} a_i^2 x^{2i}$$

This equation yields a simple squaring algorithm:

Input: $A = a_{m-1} \ldots\ldots a_1 a_0$ and $F = f_m f_{m-1} f_{m-2} \ldots\ldots f_1 f_0$
Output: $C = A^2 \bmod F$
1. $T = \sum_{i=1}^{m-1} a_i^2 x^{2i}$
2. $C = T \bmod F$ // *use algorithm 2.8*
3. return ($C(x)$)

Algorithm 2. 10: Bit-level method for squaring in $GF(2^m)$

A known technique for speeding up the computation in step 1 is to use a table lookup as in [70].

Inversion. The basic algorithm for computing multiplicative inverses is the extended Euclidean algorithm. A high level description of this method is the following:

Input: $A(x) \in GF(2^m)$, ($A(x) \neq 0$) and $F = f_m f_{m-1} f_{m-2} \ldots\ldots f_1 f_0$
Output: $C = A^{-1} \bmod F$
1. set $B_1(x) = 1$, $B_2(x) = 0$
   set $P_1(x) = A(x)$, $P_2(x) = F(x)$
2. while $degree(P_1(x)) \neq 0$ do
   if $degree(P_1(x)) < degree(P_2(x))$ then
   Exchange $P_1(x), P_2(x)$ and $B_1(x)\ B_2(x)$
   $j = degree(P_1(x)) - degree(P_2(x))$
   $P_1(x) = P_1(x) + x^j P_2(x),\ B_1(x) = B_1(x) + x^j B_2(x)$
3. return ($C(x) = B_1(x)$)

Algorithm 2. 11: Inversion using extended Euclidean algorithm in $GF(2^m)$.

An alternative method for computing inverses, called the *almost inverse algorithm*, was proposed by Schroeppel et al [70]. This method works quite well when the reduction polynomial is a trinomial of the form $x^m + x^k + 1$ with $k > W$ and $m - k > W$, where W is the word size of the computer used. The authors suggested a number of implementation tricks that can be used for improving the speed of this method. Many of these tricks also work for the extended Euclidean algorithm. However, in the context of elliptic curve computations, most of the inversions required can be avoided be using projective coordinates (see chapter 5).

2.5 Conclusions

In this chapter, the basic theory behind finite fields has been presented. The construction of finite fields has been illustrated and the representation of finite field

elements has been considered. Also, the finite fields *GF(p)* and $GF(2^m)$ were defined. The basic arithmetic operations for these two finite fields were studied and the algorithms for performing these arithmetic operations have been presented.

CHAPTER 3

Elliptic Curve Arithmetic

3.1 Introduction

In this chapter, we present fundamentals of the theory of elliptic curves defined over finite fields. Curve arithmetic is defined in terms of underlining field operations discussed in chapter 2. However, based on the group law, elliptic curve can be defined over the prime field *GF(p)* or the binary field *GF(2m)*. In both cases, the two main operations of elliptic curve are the addition and doubling operations. Figure 3.1 shows the hierarchal organization of curve operations in terms of finite field operations.

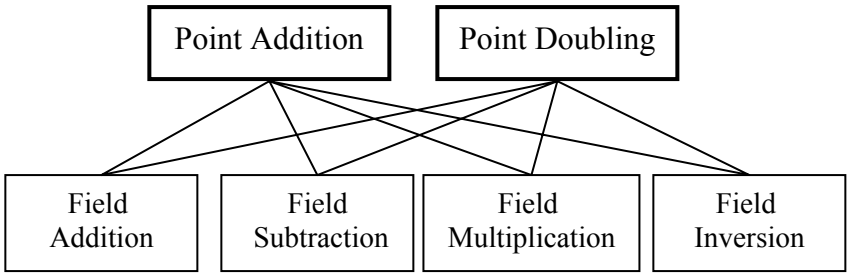


Figure 3. 1: Hierarchal organization of elliptic curve arithmetic.

The remaining of this chapter is organized as follows. Section 3.2 gives an introduction to elliptic curves. Section 3.3 presents the basic fundamentals of group low.

Elliptic curve over the prime field *GF(p)* and the binary field $GF(2^m)$ are discussed in sections 3.4 and 3.5 respectively. Finally, we conclude this chapter in section 3.6.

3.2 Introduction to Elliptic Curves

Definition 3.1: Let *E* be an elliptic curve defined over the finite field *K* denoted by *E/K*. *E/K* is defined by an equation

$$E/K:\ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \qquad 3.1$$

Where, $a_1, a_2, a_3, a_4, a_6 \in K$.

For *GF(p)*, we get the simplified *Weierstrass* of the elliptic curve equation 3.1.

$$E/K:\ y^2 = x^3 + a_4x + a_6 \qquad 3.2$$

However, there are several ways of defining equations for elliptic curves, which depend on whether the field is a prime finite field, F_p, or a binary (characteristic 2) finite field, $GF(2^m)$. The *Weierstrass* equation for both finite fields *GF(p)* and $GF(2^m)$ are described in sections 3.4 and 3.5 respectively.

Additional information on elliptic curves and its applications to cryptography can be found in [9], [13], [14] and [15].

3.3 Group Law

Let *E* be an elliptic curve defined over the field *K* denoted by *E/K*. There is a *chord-and-tangent* rule for adding tow points in *E/K* to give a third point in *E/K*. together with this addition operation, the set of points in *E/K* forms an abelian group with ∞

serving as its identity. The group *(E/K,+)* consists of a finite set of points *P(x,y)* that satisfy the elliptic curve equation 3.2 together with a point at infinity. The *x* and *y* coordinates of any point as well as the coefficients of elliptic curve equation, a_4, a_6, are elements of *K*. The group *(E/K,+)* is the algebraic group that is used to construct elliptic curve cryptosystem.

Addition operation, + , is best explained geometrically. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two distinct points on an elliptic curve *E*. Then the *sum R* of *P* and *Q* is defined as follows:

1. Draw a line through *P* and *Q*. This line intersects the elliptic curve at a third point $\overline{R}$.
2. *R* is the reflection of $\overline{R}$ around the x-axis.

The double *R*, of *P*, is defined as follows:

1. Draw the tangent line to the elliptic curve at *P*. This line intersects the elliptic curve at a third point $\overline{R}$.
2. *R* is the reflection of $\overline{R}$ around the x-axis.

The algebraic formulations of the group law can be derived from the geometric description. In the next two sections, we present the algebraic formulations of the group law of elliptic curve over finite fields *GF(p)* and $GF(2^m)$.

3.4 Elliptic Curve Over Prime Field *GF(p)*

Definition 3.2: Let $P > 3$ be an odd prime and let $a, b \in GF(p)$ satisfy

$4a^3 + 27b^2 \neq 0 \pmod{p}$. Then an *elliptic curve E* over a finite prime field *GF(p)* , denoted by *E/GF(p)*, is defined by an equation:

$$E/GF(p): \; y^2 = x^3 + ax + b \qquad 3.3$$

where parameters *a, b* ∈ *GF(p)*.

Comments in definition 3.2

(i) Equation 3.3 is called *Weierstrass* equation with $a_4 = a$ and $a_6 = b$.

(ii) We say that *E* is *defined over GF(p)* because the coefficients *a* and *b* are elements of *GF(p)*. *GF(p)* is called the *underlining field*.

(iii) The notion *E/GF(p)* (or $E(F_p)$) is used to emphasize that *E* is *defined over GF(p)*.

(iv) The *set* of points of an elliptic curve *E/GF(p)* are the points (or solutions) *P = (x, y)* (where *x, y* ∈ *GF(p)*) that satisfy equation 3.3 together with a special point called the *point at inanity*, ∞.

(v) The point ∞ is the only point on the line at infinity (∞ and –∞) that satisfies the projective form of the Weierstrass equation.

(vi) For a given point $P_1 = (x_1, y_1)$, x_1 is called the *x-coordinate* of P_1 and y_1 is called the *y-coordinate* of P_1 .

The algebraic formulas of group law for *E/GF(p)* are specified as follows:

1. *Identity*: $P + \infty = \infty + P = P$ for all $P \in E/GF(p)$.

2. *Inverse*: if $P = (x, y) \in E/GF(p)$, then $(x, y) + (x, -y) = \infty$. The point $(x, -y)$ is denoted by $-P$ and is called the *inverse* of P. Note that $-P$ is indeed a point in *E/GF(p)*. Also, $-\infty = \infty$.

3. *Point Addition (denoted by ADD)*: Let $P = (x_1, y_1) \in E/GF(p)$ and $Q = (x_2, y_2) \in E/GF(p)$ be two points satisfying the elliptic curve equation 3.3 where $P \neq \pm Q$. Then $R = P + Q = (x_3, y_3)$ is given by:

$$\left.\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ where, \quad \lambda &= (y_2 - y_1)/(x_2 - x_1) \end{aligned}\right\} \qquad 3.4$$

4. *Point doubling (denoted by DBL)*: Let $P = (x_1, y_1) \in E/GF(p)$ be a point satisfying the elliptic curve equation 3.3 where $P \neq -P$. Then $R = 2P = (x_3, y_3)$ is given by:

$$\left.\begin{aligned} x_3 &= \lambda^2 - 2x_1 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ where, \quad \lambda &= (3x_1^2 + a)/2y_1 \end{aligned}\right\} \qquad 3.5$$

From the above formulas, we get the following results:

- If $(x_2, y_2) = -(x_1, y_1)$, then $(x_3, y_3) = (x_1, y_1) + (-(x_1, y_1)) = \infty$.
- If $(x_2, y_2) = \infty$, then $(x_3, y_3) = (x_1, y_1) + \infty = (x_1, y_1)$.
- $-(x_1, y_1) = (x_1, -y_1)$.

Example 3.1: Elliptic curve over the prime field *GF(29)*.

Let P = 29 (hence we have finite field *GF(29) (or* F_{29}*)*) and the elliptic curve coefficients *a* and *b* are 4 and 20 respectively. The elliptic curve equation 3.3 becomes:

$$y^2 = x^3 + 4x + 20$$

First, note that $4a^3 + 27b^2 \neq 0 \pmod{p}$ is satisfied. That is,

$$4 \times 4^3 + 27 \times 20^2 \pmod{29} = 11056 \pmod{29} = 7 \text{ which} \neq (0 \bmod 29).$$

To get the points of *E/GF(29),* consider all possible values of *x* which are in the range from 0 to 28 and compute the corresponding *y* value by using equation 3.3 with *a* = 4 and *b* = 20. Note that all operations are performed modulo 29. For example,

- When x = 0, $y^2 = 0 + 0 + 20 = 20 = 20 \pmod{29}$, and $y = \sqrt{20} \pmod{29}$. There are two solutions:
  - ***y* = 7** since 7 × 7 = 49 = 20 (mod 29). i.e. the first solution of square root of 20 (mod 29) is 7. Therefore, the point **(0,7)** ∈ *E/GF(29).*
  - ***y* = 22** since 22 × 22 = 484 = 20 (mod 29). i.e. the second solution of square root of 20 (mod 29) is 22. Therefore, the point **(0,22)** ∈ *E/GF(29).*
- When $x = 10$, $y^2 = 10^3 + 4 \times 10 + 20 = 1060 = 16 \pmod{29}$, and $y = \sqrt{16} \pmod{29}$. There are two solutions:
  - ***y* = 4** since 4 × 4 = 16 = 16 (mod 29). i.e. the first solution of square root of 16 (mod 29) is 4. Therefore, the point **(10,4)** ∈ *E/GF(29).*
  - ***y* = 25** since 25 × 25 = 625 = 16 (mod 29). i.e. the second solution of square root of 16 (mod 29) is 25. Therefore, the point **(10,25)** ∈ *E/GF(29).*

- When $x = 7,\ y^2 = 7^3 + 4 \times 7 + 20 = 391 = 14 \pmod{29}$. $y = \sqrt{14} \pmod{29}$ is not found. In other words, there is no number in the range from 0 to 29 that when it is multiplied by itself gives 14 (mod 29). Therefore points **(7, *y*)** $\notin$ *E/GF(29)*.

The points in *E/GF(29)* are the following:

∞	(2,6)	(4,19)	(8,10)	(13,23)	(16,2)	(19,16)	(27,2)
(0,7)	(2,23)	(5,7)	(8,19)	(14,6)	(16,27)	(20,3)	(27,27)
(0,22)	(3,1)	(5,22)	(10,4)	(14,23)	(17,10)	(20,26)	
(1,5)	(3,28)	(6,12)	(10,25)	(15,2)	(17,19)	(24,7)	
(1,24)	(4,10)	(6,17)	(13,6)	(15,27)	(19,13)	(24,22)	

Point Addition: Let $P = (x_1, y_1) = (5,22)$ and $Q = (x_2, y_2) = (16,27)$ (note that $P \neq \pm Q$). Then $R = P + Q = (x_3, y_3)$ is given by: (apply addition formula 3.4)

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} = \frac{27 - 22}{16 - 5} = \frac{5}{11} = 5 \times (11)^{-1} = 5 \times (11)^{-1} = 5 \times 8 = 40 = \mathbf{11} \pmod{29}.$$

Note that the inverse of 11 (mod 29) is the number *r* where 11 × *r* = 1 (mod 29). That number, i.e. *r*, is 8 since 8 × 11 = 88 = 1 (mod 29).

$$x_3 = \lambda^2 - x_1 - x_2 = (11)^2 - 5 - 16 = 100 = \mathbf{13} \pmod{29}.$$

$$y_3 = \lambda(x_1 - x_3) - y_1 = 11(5 - 13) - 22 = -110 = \mathbf{6} \pmod{29}.$$

Therefore, ***R = (13,6)*** which is in *E/GF(29)*.

Remark: to get the modulo of a negative number *r* mod *P*, repeat adding *P* to *r* until getting the first positive number in the range from 0 to *P*. for example, to get –110 mod 29, repeat adding 29 to –110 until getting the first positive number in the range from 0 to 29 which is 6.

Point doubling: Let $P = (x_1, y_1)$ = (5,22). Then $R = 2P = (x_3, y_3)$ is given by: (apply doubling formula 3.5)

$$\lambda = \frac{3x_1^2 + a}{2y_1} = \frac{3(5)^2 + 4}{2 \times 22} = 79 \times (44)^{-1} = 21 \pmod{29} \times (15)^{-1} \pmod{29} = 21 \times 2 =$$

42 = **13** (mod 29).

Note that the inverse of 15 (mod 29) is the number *r* where 15 × *r* = 1 (mod 29). That number, i.e. *r*, is 2 since 2 × 15 = 30 = 1 (mod 29).

$$x_3 = \lambda^2 - 2x_1 = (13)^2 - 10 = 159 = \mathbf{14} \pmod{29}.$$

$$y_3 = \lambda(x_1 - x_3) - y_1 = 13(5 - 14) - 22 = -139 = \mathbf{6} \pmod{29}.$$

Therefore, ***R = (14,6)*** which is in *E/GF(29)*. □

3.5 Elliptic Curve Over Binary Field *GF(2^m)*

Definition 3.3: Let $GF(2^m)$ be a finite field of characteristic two. A non-supersingular elliptic curve *E* over $GF(2^m)$, denoted by $E/GF(2^m)$, is defined to be the set of solutions $(x, y) \in GF(2^m)$ to the equation,

$$E/GF(2^m): y^2 + xy = x^3 + ax^2 + b \qquad 3.6$$

where a and $b \in GF(2^m)$ and $b \neq 0$.

Comments in definition 3.3

(i) Equation 3.6 is called *Weierstrass* equation with $a_1 = 1$, $a_4 = a$ and $a_6 = b$.

(ii) We say that E is *defined over* $GF(2^m)$ because the coefficients a and b are elements of $GF(2^m)$. $GF(2^m)$ is called the *underlining field*.

(iii) The notion $E/GF(2^m)$ (or $E(GF(2^m))$) is used to emphasize that E is *defined over* $GF(2^m)$.

(iv) The *set* of points of an elliptic curve $E/GF(2^m)$ are the points (or solutions) $P = (x, y)$ (where $x, y \in GF(2^m)$) that satisfy equation 3.6 together with a special point called the *point at inanity*, ∞.

(v) The point ∞ is the only point on the line at infinity (∞ and –∞) that satisfies the projective form of the Weierstrass equation.

(vi) For a given point $P_1 = (x_1, y_1)$, x_1 is called the *x-coordinate* of P_1 and y_1 is called the *y-coordinate* of P_1.

It is well known that E with the point at infinity, ∞, forms an abelian finite group with ∞ serving as the identity element of the group. The algebraic formulas of group law for $E/GF(2^m)$ are specified as follows:

1. *Identity*: $P + \infty = \infty + P = P$ for all $P \in E/GF(2^m)$.

2. *Inverse*: if $P=(x,y) \in E/GF(2^m)$, then $(x,y)+(x,x+y)=\infty$. The point $(x,x+y)$ is denoted by –*P* and is called the *inverse* of *P*. Note that –*P* is indeed a point in $E/GF(2^m)$. Also, $-\infty=\infty$.

3. *Point Addition (denoted by ADD)*: Let $P=(x_1,y_1) \in E/GF(2^m)$ and $Q=(x_2,y_2) \in E/GF(2^m)$ be two points satisfying the elliptic curve equation 3.6 where $P \neq \pm Q$. Then $R=P+Q=(x_3,y_3)$ is given by:

$$\left.\begin{aligned} x_3 &= \lambda^2+\lambda+x_1+x_2+a \\ y_3 &= \lambda(x_1+x_3)+x_3+y_1 \\ where,\quad \lambda &= (y_2+y_1)/(x_2+x_1) \end{aligned}\right\} \qquad 3.7$$

5. *Point doubling (denoted by DBL)*: Let $P=(x_1,y_1) \in E/GF(2^m)$ be a point satisfying the elliptic curve equation 3.6 where $P \neq -P$. Then $R=2P=(x_3,y_3)$ is given by:

$$\left.\begin{aligned} x_3 &= \lambda^2+\lambda+a \\ y_3 &= \lambda(x_1+x_3)+x_3+y_1 \\ where,\quad \lambda &= x_1+y_1/x_1 \end{aligned}\right\} \qquad 3.8$$

From the above formulas, we get the following results:

- If $(x_2,y_2)=-(x_1,y_1)$, then $(x_3,y_3)=(x_1,y_1)+(-(x_1,y_1))=\infty$.
- If $(x_2,y_2)=\infty$, then $(x_3,y_3)=(x_1,y_1)+\infty=(x_1,y_1)$.
- $-(x_1,y_1)=(x_1,x_1+y_1)$.

Example 3.2: non-supersingular elliptic curve over the binary field $GF(2^4)$.

Consider the finite field $GF(2^4)$ as represented by the reduction polynomial $f(z) = z^4 + z + 1$. An element $a_3z^3 + a_2z^2 + a_1z + a_0 \in GF(2^4)$ is represented by the bit string $(a_3a_2a_1a_0)$ of length 4 bits. For example, (0101) represents $z^2 + 1$.

Let elliptic curve coefficients *a* and *b* are z^3 and $z^3 + 1$ respectively. The elliptic curve equation 3.6 becomes: $E/GF(2^4): y^2 + xy = x^3 + z^3x^2 + z^3 + 1$.

To get the points of $E/GF(2^4)$, consider all possible values of *x* which are in the range from (0000) to (1111) and compute the corresponding *y* value by using the above elliptic equation. Note that all operations are performed modulo the reduction polynomial

$$f(z) = z^4 + z + 1.$$

The points in $E/GF(2^4)$ are the following:

∞	(0011,1100)	(1000,0001)	(1100,0000)
(0000,1011)	(0011,1111)	(1000,1001)	(1100,1100)
(0001,0000)	(0101,0000)	(1001,0110)	(1111,0100)
(0001,0001)	(0101,0101)	(1001,1111)	(1111,1011)
(0010,1101)	(0111,1011)	(1011,0010)	
(0010,1111)	(0111,1100)	(1011,1001)	

Point Addition: Let $P = (x_1, y_1)$ = (0010,1111) and $Q = (x_2, y_2)$ = (1100,1100) (note that $P \neq \pm Q$). Then $R = P + Q = (x_3, y_3)$ = (0001,0001) (apply addition formula 3.7).

Point doubling: Let $P = (x_1, y_1)$ = (0010,1111). Then $R = 2P = (x_3, y_3)$ is (1011,0010) (apply doubling formula 3.8) □

3.6 Conclusions

In this chapter, we have presented the fundamentals of the theory of elliptic curves defined over finite fields. Hierarchal organization of curve operations in terms of finite field operations has been introduced. Also, defining an elliptic curve over the prime field *GF(p)* and over the binary field $GF(2^m)$ has been discussed with providing an example for each case.

CHAPTER 4

Elliptic Curve Cryptography

4.1 Introduction

The security of Elliptic Curve Cryptography (ECC) in based on the apparent intractability of Elliptic Curve Discrete Logarithm Problem (ECDLP) [9]. To date, that there are no sub-exponential algorithms for the ECDLP known. This means that we can use shorter keys (compared to other cryptosystems) for high security levels. However, to establish an ECC, several main aspects need to be discussed. The main purpose of this chapter is to present these main aspect which are necessary for any environment that wishes to use ECC.

To setup an ECC, domain parameters such as the curve coefficients a and b and the base point should be selected and verified. These parameters are used to establish a cryptography system whether this system is a symmetric key or public key cryptography. Also, a curial operation in ECC is the scalar multiplication (or point multiplication) in which a base point P is added to itself K times. This point multiplication is performed based on the group law discussed in chapter 3.

Since this thesis considers both the elliptic curve defined over the prime field *E/GF(p)* and over the binary field *E/GF(2^m)*, we use the following common notation: *E/GF(q)* , where $q = p$ or $q = 2^m$, to denote both cases. Whenever "*E/GF(q)*" appears, it means that the related subject is applicable to both *E/GF(p)* and *E/GF(2^m)*.

This chapter is organized as follows. Section 4.2 discusses the ECDLC. Elliptic curve domain parameters are presented in section 4.3. Elliptic curve cryptosystems namely, elliptic curve symmetric and public cryptography are discussed in section 4.4. Scalar multiplication and the most popular algorithms to perform it are the subject of section 4.5. Finally, conclusions are drawn in section 4.6.

4.2 Elliptic Curve Discrete Logarithm Problem (ECDLP)

ECDLP is defined as follows: Given an elliptic curve *E/GF(q)*, a point $P \in$ *E/GF(q)* of order *n* and a point $Q \in$ *E/GF(q)*, determine the integer *K* satisfying $Q = K P$, provided that such $0 \le K \le n-1$ exists. The integer K is called the discrete logarithm of Q to the base P, denoted $K = \log_P Q$.

To date, the most efficient general algorithm to resolve the ECDLP is Pollard-ρ [17] algorithm, which has the running time $O(\sqrt{n}/r)$, where r is the parallel processor number.

Another possible attack known on the ECDLP is the combination of the Pohlig-Hellman algorithm [16] and Pollard-ρ algorithm where the computation of K is reduced to the problem of computing K modulo each prime factor of n. So if n is a large prime,

the ECDLP becomes harder. In practice, one must carefully select elliptic curve parameters (section 4.2) such as selecting a base point that has large prime order *n* and curve order *#E/GF(q)* = n $\times$ *h*, where *h* is a small integer.

It is well known that the security of any cryptosystems depends mainly on the hardness of the mathematical underlining problem that the cryptosystems is based on. Fore example, Rivest-Shamir-Adleman (RSA) [3] cryptosystem is based on integer factorization problem. An instance of integer factorization problem is an integer *n* that is a product of two *L*/2 bits primes. The best algorithm known for solving the integer factorization problem is the *Number Field Sieve* (NFS) [9] which has sub-exponential time. On the other hand, The best algorithm to solve the ECDLP is the combination of the Pohlig-Hellman [16] and Pollard's ρ algorithms [17], which has a fully-exponential running time. This means that significantly smaller parameters can be used in ECC than in RSA system, but with equivalent levels of security. A typical example of the size in bits of the keys used, is that a l60-bit ECC key is equivalent to RSA with a modulus of 1024 bits. Thus ECC offers potential reductions in the number of required arithmetic operations, storage space, bandwidth and electrical power. These advantages are specially important in applications on constrained devices such as smart cards and cellular phones.

[3] In RSA, one has a public key (*e*, *n*), a prime number *P*, and a private key $K = k_{n-1}...k_0$ .When creating an encrypted message *C* one has to compute $C = P^e \bmod n$. Decryption is done by $P = C^d \bmod n$. The modular exponentiation is usually done by the *square-and-multiply* algorithm.

4.3 ECC Domain Parameters

Before we introduce the ECC domain parameters, It is necessary to present some basic facts and concepts of ECC.

- *Order of point* $P \in E/GF(q)$ is the smallest integer r such that $rP = \infty$.
- *Order of the curve*, is the number of points of *E/GF(q)*, donated by *#E/GF(q)*. Note that the curve order can be computed by Schoof's algorithm [9] or its improvements, which is needed if one selects a random curve. And normally choosing *a* and *b* to make the curve order have a large prime factor can improve the cryptography scheme's security. So, this is an important parameter of the scheme to determine the system's security.
- Hasse Theorem: let *E* be an elliptic curve defined over *GF(p)*. then the curve order *#E/GF(p)* is bounded by:

$$p + 1 - 2\sqrt{p} \leq \#E/GF(p) \leq p + 1 + 2\sqrt{p}$$

Elliptic curve parameters over the finite field *GF(p)* or $GF(2^m)$ can be described by the following 6-tuple:

$T = (q, FR, a, b, G, n, h)$

Where:

- q: the prime p or 2^m that defines the field and at the same time decides the curve form.

- *FR*: the field representation, i.e., using which method to represent the elements in the field (polynomial basis or normal basis for $GF(2^m)$, or normal or Montgomery residue for *GF(p)*).
- *a, b*: the curve coefficients, depending on the security requirement.
- *G*: the base point, $G = (x_G, y_G)$, one element in *E/GF(q)*, which has the largest order *n*.
- *n*: the order of *G*, large prime. Also, the order of the curve, *N = #E/GF(q)*, is divisible by *n*.
- *h*: *# E/GF(q)*/*n*.

These parameters should be chosen to setup an ECC system.

4.4 Elliptic Curve Cryptosystem

Given a message point (x_m, y_m), a base point (x_G, y_G), and a given key, *K*, the cipher point (x_C, y_C) is obtained using the following equation,

$$(x_C, y_C) = (x_m, y_m) + K(x_G, y_G) \qquad 4.1$$

There are two basics steps in the computation of the above equations. The first is to find the scalar multiplication (section 4.5) of the base point with the key, "$K(x_G, y_G)$". The resulting point is then added to the message point, (x_m, y_m) to obtain the cipher point.

At the receiver, the message point is recovered from the cipher point which is usually transmitted, the shared key and the base point, that is

$$(x_m, y_m) = (x_C, y_C) - K(x_G, y_G) \qquad 4.2$$

4.4.1 Symmetric Elliptic Curve Cryptography

The steps of elliptic curve symmetric cryptography can be summarized as follows:

Both the sender and receiver must agree on:

1. A random number, *K*, that will be the shared secret key for communication,
2. A base point, $G = (x_G, y_G)$.

At the sending correspondent:

1 Embed a message bit string into the *x-coordinate* of an elliptic curve point which is designated as the message point, (x_m, y_m).

2 The cipher point (x_c, y_c) is computed using,

$$(x_c, y_c) = (x_m, y_m) + K(x_G, y_G)$$

3 The appropriate bits of the *x*-coordinate and the sign bit of the *y*-coordinate of the cipher point (x_c, y_c) are sent to the receiving entity.

At the receiving correspondent, the following steps are performed,

1. Using the shared key, *K*, and the base point (x_G, y_G), the scalar multiplication $KG = K(x_G, y_G)$ is computed.
2. The message point (x_m, y_m) is computed using,

$$(x_m, y_m) = (x_c, y_c) + (-K(x_G, y_G))$$

3. The secret messages bit string is recovered from x_m.

4.4.2 Public Key Elliptic Curve Cryptography

Before we proceed to see how two entities can communicate using elliptic curve public key cryptography, we first have to show how the private and public keys are generated and verified and then how the sending and receiving entities agree on a key. For the following, let A denotes the sending entity and B denotes the receiving entity.

Key Generation. We mean by key generation is to generate the public and private key pair. Given the domain parameters *(q, FR, a, b, G, n, h)*, each entity does the following:

Sending entity, A:

1. Selects a random integer d_A from the interval [1,n – 1].
2. Computes $Q_A = d_A G$. (It is a scalar multiplication step, $Q_A = d_A(x_G, y_G)$).

d_A is the private key and Q_A is the public key of A.

Similarly, B computes d_B and Q_B as its private and public key pair.

Key Validation. We mean by key validation is to validate the public key's legality. Entity A does the following:

1. Check that $Q_B \neq \infty$.
2. Check that x_{Q_B} , $y_{Q_B} \in$ *E/GF(q),* where x_{Q_B} and y_{Q_B} denote the x-coordinate and y-coordinate of the point Q_B .
3. Check that Q_B lies on the elliptic curve defined by *a* and *b*;
4. Check that $nQ_B = \infty$. (note that, $nQ_B = n(d_B G) = d_B(nG) = d_B \infty = \infty$, because *G*'s order is *n*)

The public key validation without Step 4 is called the partial public-key validation. Without Step 4, the entity could be attacked. However, we can carefully select *h* to reduce the threat.

Key agreement scheme. One of the most popular key agreement schemes is the Diffie-Hellman key agreement scheme [9]. Table 4.1 shows the steps taken by each entity.

By end of step 3, in table 4.1,each entity get the same shared secret point (x_P, y_P). That is, A computes: $P = d_A Q_B = d_A(d_B G) = (d_A d_B)G$ and

B computes: $P = d_B Q_A = d_B(d_A G) = (d_B d_A)G = (d_A d_B)G$.

Table 4. 1 : Diffie-Hellman key agreement scheme

Step	**Description**	**Entity A**	**Entity B**
1	Choose random private key	d_A=*rand*(1,$n-1$)	d_B=*rand*(1,$n-1$)
2	Compute public key from the private key and the base point *G*. Then each entity publishes its public key.	$Q_A = d_A G$	$Q_B = d_B G$
3	Generate Common key. Each entity computes the common key using its private key and the public key of the other entity.	$P = d_A Q_B = (x_P, y_P)$	$P = d_B Q_A = (x_P, y_P)$

The steps of elliptic curve public key cryptography can be summarized as follows: Both the sender and receiver must agree on:

1. An elliptic curve.
2. A base point, $G = (x_G, y_G)$.

At the sending correspondent:

1. Embed a message bit string into the *x-coordinate* of an elliptic curve point which is designated as the message point, (x_m, y_m).

2. Using the steps (entity A) in table 4.1, compute the shared secret point $P = (x_P, y_P)$.
3. Compute a cipher point (x_c, y_c) using: $(x_c, y_c) = (x_m, y_m) + (x_P, y_P)$.
4. Send appropriate bits of the *x*-coordinate and the sign bit of the *y*-coordinate of the cipher point (x_c, y_c) to the receiving correspondent;

At the receiving correspondent:

1. Using the steps (entity B) in table 4.1, compute the shared secret point $P = (x_P, y_P)$.
2. Compute the message point (x_m, y_m) using $(x_m, y_m) = (x_c, y_c) - (x_P, y_P)$.
3. Recover the message bit string from x_m

4.5 Scalar Multiplication

Scalar multiplication (SM) (or point multiplication) is the result of adding the base point[4] *P* to itself *K* times on the elliptic curve over a given finite field, where *K* is a positive integer. That is

$$KP = \underbrace{P + P ..... + P}_{K \text{ times}} \qquad 4.3$$

The integer *K* is referred to as *scalar* and the point *P* as the base point.

However, adding the point *P* to itself *K* times is not an efficient way to compute scalar multiplication. More efficient methods are based on a sequence of *Addition* (ADD)

[4] We mean by base point here, is a base point for the scalar multiplication and not the base point *G* in the domain parameters. This is because scalar multiplication can be performed to any point whether this point is *G* or any other point *P*

and *Doubling* (DBL) operations. Note that doubling operation is simply adding the point to itself. In the literature, there are many methods (or algorithms) for computing *KP* or equivalently performing the scalar multiplication. In the following subsections, we present the most popular scalar multiplication algorithms. However, it is worth to mention that each of theses algorithms can be applied to *E/GF(p)* and *E/GF(2^m)*.

4.5.1 Binary Methods

Let $(k_{n-1}2^{n-1} + k_{n-2}2^{n-2} + ... + k_1 2 + k_0)_2$ be the binary representation of the scalar *K* where $k_i \in \{0,1\}$ is the *i-th* bit and *n* is the total number of bits. Hence, the scalar multiplication *KP* can be written as:

$$KP = \left(\sum_{i=0}^{n-1} k_i 2^i \right) P$$

which can be expanded to one of the following forms:

$$KP = k_{n-1}2^{n-1}P + k_{n-2}2^{n-2}P + ... + k_1 2P + k_0 P \qquad 4.4$$

$$KP = 2(2(...2(2(k_{n-1}P) + k_{n-2}P) + ...) + k_1 P) + k_0 P \qquad 4.5$$

Based on 4.4 and 4.5, there are two main binary methods of calculating *KP*. The first is the Least-to-Most (LM) algorithm, which corresponds to the expansion in 4.4, starts from the least significant bit of *K* to the most significant one. The second is the Most-to-Least (ML) algorithm, which corresponds to the expansion in 4.5, starts from the most significant bit of K. Algorithms 4.1 and 4.2 show the LM and the ML binary algorithms respectively.

```
INPUT        K, P
OUTPUT       KP
    1. Initialize Q[0] = ∞ , Q[1] = P
    2. for i = 0 to n-1
    3.       if k[i] = 1 then
    4.             Q[0] = ADD(Q[0],Q[1])
    5.        end if
    6.       Q[1] = DBL(Q[1])
    7. end for
    8. return Q[0]
```

Algorithm 4. 1: Least-to-Most (LM) binary algorithm for scalar multiplication

```
INPUT        K, P
OUTPUT       KP
    1. Initialize Q[0] = P
    2. for i = n-2 downto 0
    3.       Q[0] = DBL(Q[0])
    4.       if k[i] = 1 then
    5.           Q[0] = ADD(Q[0],P)
    6.       end if
    7. end for
    8. return Q[0]
```

Algorithm 4. 2: Most-to- Least (ML) binary algorithm for scalar multiplication

In both algorithms, *KP* is computed using the straightforward *double-and-add* approach in *n* iterations. The point doubling operation (DBL) is performed in all cases regardless of the scalar bit value, while the ADD operation is conditioned by the scalar bit value. If the scalar bit value is 1, ADD is performed; otherwise it is not performed.

4.5.2 Window Methods

Several generalizations of the binary method work by processing simultaneously a block of digits. In these methods, depending on the size of the blocks (or windows) a

number of precomputed points are required. However, the most popular window methods presented in this subsection are: *m-ary*, modified *m-ary* and sliding window methods.

4.5.2.1 The *m-ary* Method

This method uses the *m-ary* expansion of *K* where $m = 2^r$ for some integer $r \geq 1$. The binary method is a special case of *m-ary* method corresponding to *r = 1*. The scalar *K* is expanded as follows:

$$K = \sum_{j=0}^{n-1} k_j m^j \ , \ k_j \in \{0, 1, 2, \ldots, m\text{-}1\}.$$

The *m-ary* method of computing *KP* is shown in algorithm 4.3.

Input: An integer $K = \sum_{j=0}^{n-1} k_j m^j$ and a point $P = (x,y) \in E/GF(q)$
Output: The point $Q = KP \in E/GF(q)$
// Precomputation:
1. $P_1 = P$
2. for $i = 2$ to $m - 1$ do
    $P_i = P_{i-1} + P$ // (we have $P_i = iP$)
3. Q = ∞
// Main loop
4. **for** $j = n - 1$ **downto** 0 **do**
5. $Q = [m]Q$ //(this requires *r* doublings)
6. $Q = Q + P_{K_j}$
Return (*Q*)

Algorithm 4. 3: *m-ary* method for scalar multiplication

It can be readily verified that the algorithm computes *KP*, following Horner's rule [16]:

$$KP = [m]([m](...[m]([m](k_{n-1}P) + k_{n-2}P) + ...) + k_1 P) + k_0 P$$

The number of doubling in the main loop of the *m-ary* method is $(d - 1)r$ (the first iteration is not counted, as it starts with $Q = \infty$). Since $d = \lceil n / r \rceil$, where *n* is the length

of the binary representation of *K*, the number of doublings in the *m-ary* method may be up to (*r – 1*) less than the (*n – 1*) required by the binary method. However, it needs to pre-compute and store the points *2P* to [*m-1*]*P*.

4.5.2.2 The Modified *m-ary* Method

The main disadvantage of the *m-ary* method is that it requites pre-computing and storing the points *2P*, *3P*, …, [*m-1*]*P*. This disadvantage can be reduced to only computing and saving the odd multiples of *P* only (i.e. skipping the even multiples of *P* in the precomputation phase) resulting in the modified *m-ary* method shown in algorithm 4.4.

Input: An integer $K = \sum_{j=0}^{n-1} k_j m^j$ and a point *P= (x,y) ∈ E/GF(q)*
Output: The point *Q = KP ∈ E/GF(q)*
// Precomputation:
1. $P_1 = P, \quad P_2 = P$
2. for *i* = 1 to *(m – 2) / 2* do
   $P_{2i+1} = P_{2i-1} + P_2$
3. Q = ∞.
// Main loop
4. **for** *j = n - 1* **downto** 0 **do**
5. **If** $k_j \neq 0$ **then**
6. Let s_j, h_j be such that $k_j = 2^{s_j} h_j$, h_j odd.
7. $Q = [2^{r-s_j}]Q$
8. $Q = Q + P_{h_j}$
9. **Else** $s_j = r$
10. $Q = [2^{s_j}]Q$
Return (*Q*)

Algorithm 4. 4: Modified *m-ary* method for scalar multiplication

In the modified *m-ary* method, computation of *mP* (step 5 of algorithm 4.3) is split into two steps (steps 7 and 8) as shown algorithm 4.4. However, in algorithm 4.4, we assume that $r > 1$, otherwise we revert to the original binary method.

4.5.2.3 Sliding Window Method

In the *m-ary* and modified *m-ary* methods, the windows are contiguous and in fixed bit positions. When a window has zeros in the left most bit positions, it is treated as any other window. However, in the sliding window methods, the left most zeros of any window are dropped and corresponding doubling operations are performed in the accumulator point *Q*. Therefore, the window size can shrink and grow up to length *r*.

In the sliding window method, *K* is represented as:

$$K = \sum_{j=0}^{n-1} k_j 2^j \ , \ k_j \in \{0, 1\}.$$

and computing *KP* using this method is shown in algorithm 4.5.

In the main while loop of algorithm 4.5, the bits of the *K* are scanned starting from the most significant bit and based on the value of each bit one of two things may performed:

1. If $k_j = 0$, then perform a double operation on the point *Q* (step 5).
2. If $k_j \neq 0$, (i.e $k_j = 1$) then:

a. Consider a window of size up to *r* bits such that the contents of this window is $h_j = (k_j k_{j-1} \ldots\ldots k_t)_2$ where j is the current bit position and t is the least integer such that $j - t + 1 \leq r$ and $k_t = 1$.

b. Update the value of the point Q as shown in step 9.

Input: An integer $K = \sum_{j=0}^{n-1} k_j 2^j$ and a point $P = (x,y) \in E/GF(q)$
Output: The point $Q = KP \in E/GF(q)$
// Precomputation:
1. $P_1 = P$, $P_2 = 2P$
2. for $i = 1$ to $(2^{r-1} - 1)$ do
   $P_{2i+1} = P_{2i-1} + P_2$
3. $Q = \infty$, $j = n - 1$.
// Main loop
4. **While** $j \geq 0$ **do**
5. **If** $k_j = 0$ **then**
   $Q = [2]Q$; $j = j - 1$;
6. **Else**
7. Let *t* be the least integer such that
   $j - t + 1 \leq r$ and $k_t = 1$
8. $h_j = (k_j k_{j-1} \ldots\ldots k_t)_2$
9. $Q = [2^{j-t+1}]Q + P_{h_j}$
10. $j = t - 1$
Return (*Q*)

Algorithm 4. 5: Sliding window method for scalar multiplication

4.5.3 Scalar Recoding Methods

We main by scalar recoding is transforming the scalar *K* to another form $\overline{K}$ such that it still gives the correct result of computing *KP*. i.e. $KP = \overline{K} P$ but with less

computations. One popular recoding of any integer (rather than the scalar) is the *non-adjacent form* (NAF) recoding. In NAF, every integer K has a unique signed digit representation of the form $K = \sum_{i=0}^{l-1} k_i 2^i$ where $k_i \in \{-1,0,1\}$, such that no two consecutive digits are nonzero [9]. However, there are several algorithms for computing the NAF of K from its binary representation (see for example [8] and [9]). The following algorithm (algorithm 4.6), from Solinas [18] computes the NAF of an integer K.

Input: an integer K
Output: The NAF form of K, NAF(K) = $(u_{l-1} \ldots u_1 u_0)$
1. Set $c = K$, $l = 0$
2. **While** $c > 0$ **do**
  if c *odd* **then**
    Set $u_l = 2 - (c \bmod 4)$
    Set $c = c - u_l$
  Else $u_l = 0$
  Set $c = c/2$, $l = l + 1$
Return (NAF(K) = $(u_{l-1} \ldots u_1 u_0)$)

Algorithm 4. 6: Computation of NAF(K)

A general form of NAF(K) is what is called the *width-w nonadjacent form* or *width-w NAF*. Let w be an integer greater than one. Then every positive number K has a unique *width-w nonadjacent form*:

$K = \sum_{j=0}^{l-1} u_j 2^j$ Where:

- Each nonzero u_j is odd and less than 2^{w-1} in absolute value.
- Among any w consecutive coefficients, at most one is non zero.

The *width-w NAF* is written as $NAF_w(K) = (u_{l-1}u_{l-2}.....u_1u_0)$. A generalized version of algorithm 4.6 for computing $NAF_w(K)$ is described in algorithm 4.7.

Input: an integer K
Output: $NAF_w(K) = (u_{l-1}u_{l-2}.....u_1u_0)$
1. Set $c = K$, $l = 0$
2. **While** $c > 0$ **do**
  if c *odd* **then**
    $u_l = 2 - (c \bmod 2^w)$
    If $u_l > 2^{w-1}$ **then**
      $u_l = u_l - 2^w$
    $c = c - u_l$
  Else $u_l = 0$
  $c = c/2$, $l = l + 1$
Return ($NAF_w(K) \leftarrow (u_{l-1}u_{l-2}.....u_1u_0)$)

Algorithm 4. 7: Computation of $NAF_w(K)$

Many scalar multiplication algorithms have been proposed based on *NAF(K)* and $NAF_w(K)$ representations of the scalar [8], [9], [18] and [19]. Addition-subtraction algorithm (section 4.3.3.1) and width-w window algorithm (section 4.3.3.2) are examples of using these representations respectively.

4.5.3.1 Addition-Subtraction Algorithms

An improved algorithm for computing *KP* can be obtained from the following facts:

- Every integer *K* has a unique NAF representation.
- The expected weight of a NAF of length *l* is *l/3* [9].
- The computation of the negation of a point *P = (x; y)* $\in$ `E/GF(p)` is simply the negation of its *y-coordinate (i.e. – P = (x; –y))* which is virtually free. So the cost

of addition or subtraction is practically the same. In case of $E/GF(2^m)$, $-P$ is computed by replacing *y-coordinate* by $(x+y)$.

Addition-subtraction algorithm requires computing the NAF representation of the scalar K. It performs a point addition or subtraction depending on the sign of each digit of K as shown in Algorithm 4.8. This algorithm scans the NAF representation of the scalar K (which has now l bits rather than n) from left to right and requires l doublings and l /3 additions on average. However, this algorithm can be modified to obtain a right-to-left version [18], which does not need storage for the *NAF(K)*.

```
Input: An integer K an a point P = (x,y) ∈ E/GF(q)
Output: The point Q = KP ∈ E/GF(q)
1. Use algorithm 4.6 to compute NAF(K) = (u_{l-1} ... u_1 u_0)
2. Q = ∞
2. for j = l - 1 downto 0 do
        Q = DBL(Q)
        if u_l = 1  then
            Q = ADD(Q, P)
        if u_l = –1  then
            Q = ADD(Q, – P)
Return (Q)
```

Algorithm 4. 8: Binary NAF algorithm (addition-subtraction) for scalar multiplication

4.5.3.2 Width-w Window Method

Given the *width-w NAF* of an integer K, and a point $\in E/GF(p)$, the calculation of KP can be carried out by using a typical window method called the *width-w window method* [18] shown in algorithm 4.9.

The number of nonzero digits in the $NAF_w(K)$ is on the average $l/(w+1)$ [20]. Therefore, algorithm 4.9 requires $2^{w-2}-1$ additions and one doubling for the

precomputation step, and (*l/(w + 1))* additions and (*l – 1)* doublings for the main computation. Note that although the number of additions can be reduced by selecting an appropriate width *w*, the number of doublings is the same as in the previous methods. The total number of finite fields operations required for computing *KP* depends mainly on the algorithms used for the elliptic operations (affine or projective coordinates), the cost-ratio of inversion to multiplication, and the width *w*.

Input: integers K and w, a point $P = (x,y) \in E/GF(q)$
Output: The point $Q = KP \in E/GF(q)$
// Precomputation:
// Compute uP for u odd and $2 < u < 2^{w-1}$
1. $P_0 = P$, $T = 2P$
2. for $i = 1$ to $2^{w-2} - 1$ do
    $P_i = P_{i-1} + T$
// Main computation
3. Use algorithm 4.7 to compute $NAF_w(K) \leftarrow (u_{l-1}u_{l-2}.....u_1u_0)$
4. $Q = \infty$
5. **for** $j = l - 1$ **downto** 0 **do**
    Q = DBL(Q)
    if $u_j \neq 0$ **then**
        $i = (|u_j| - 1)/2$
        if $u_j > 0$ **then**
            Q = ADD(Q, P_i)
        Else
            Q = ADD(Q,–P_i)
Return (Q)

Algorithm 4. 9: *width-w* window method for scalar multiplication

4.5.4 Lim/Lee Method

This method, developed by Lim and Lee [21], can be used for computing *KP* when *P* is a fixed point, known in advance of the computation. In order to compute *KP*, the *l-bit* integer *K* is divided into *h* blocks K_r, each one of length $a = \lceil l/h \rceil$. In addition, each block K_r is subdivided into *v* blocks of size $b = \lceil a/v \rceil$. Thus *K* can be written as:

$$\sum_{r=0}^{h-1}\sum_{s=0}^{v-1}\sum_{t=0}^{b-1} k_{vbr+bs+t}\, 2^{vbr+bs+t}$$

Then, Lim/Lee's method uses the following expression for computing *KP*:

$$KP = \sum_{t=0}^{b-1} 2^t \left(\sum_{s=0}^{v-1} G[s][I_{s,t}]\right)$$

Where the precomputation array G[s][u] for $0 \le s < v$, $0 \le u < 2^h$ and $u = (u_{h-1}...u_0)_2$, is defined by the following equations:

$$G[0][u] = \sum_{r=0}^{h-1} u_r 2^{rvb} P,$$

$$G[s][u] = 2^{sb} G[0][u]$$

and the number $I_{s,t}$ for $0 \le s < v - 1$ and $0 \le t < b$ is defined by

$$I_{s,t} = \sum_{r=0}^{h-1} k_{vbr+bs+t}\, 2^r$$

A detailed description of Lim/Lee's method is given in algorithm 4.10. This algorithm requires $v(2^h - 1)$ elliptic points of storage, and the average number of operations to perform a scalar multiplication is *(b – 1)* doublings and $((2^h - 1)/2^h vb - 1)$ additions on average, but *(vb – 1)* additions in the worst case. The selection of both

parameters *h* and *v* presents a trade-off between precomputation (memory) and online computations (speed). Some improvements to this algorithm are discussed in [22].

Input: Integers *K, h, v* and an array of points *G[s][u]*,with $0 \le s < v, 1 \le u < 2^h$
Output: The point $Q = KP \in E/GF(q)$
// The array *G* is computed as:
for $u = 1$ to $2^h - 1$ do
for $s = 0$ to $v - 1$ do
$u = (u_{h-1} \ldots u_0)_2$
$G[s][u] = 2^{sb} \sum_{i=0}^{h-1} u_i 2^{vbi} P$
// Main computation
1. $Q = \infty$
2. **for** $t = b - 1$ **downto** 0 **do**
Q = DBL(Q)
For $s = v - 1$ downto 0 do
$I_{s,t} = \sum_{i=0}^{h-1} 2^i k_{vbi+bs+t}$
if $I_{s,t} \neq 0$ **then**
Q = ADD(Q, $G[s][I_{s,t}]$)
Return (Q)

Algorithm 4. 10: Lim/Lee method for scalar multiplication

4.6 Conclusions

In this chapter, the basic aspects behind elliptic curve cryptography has been introduced. ECDLP has been defined as the mathematical underlining problem of ECC. The ECC domain parameters were presented. We concluded that careful selection of these parameters plays a certain role in ECC security. The most important elliptic curve cryptography schemes, symmetric key and public key, are studied. The detailed steps to establish a secure communication between two entities using these two schemes are

addressed. Finally, in this chapter, the main operation in ECC, scalar multiplication, is discussed. Also, The various popular algorithms for scalar multiplication has been presented.

CHAPTER 5

Coordinate Systems

5.1 Introduction

The most difficult finite field operation to implement is inversion. An efficient hardware implementations in $GF(2^m)$ costs [52][5]:

$\lfloor \log_2(m-1) \rfloor + w(m-1) - 1$ multiplications ; $m-1$ squaring

Where $w(m-1)$ denotes the number of ones in the binary representation of $(m-1)$. It is reported in [52] that the number of multiplications and squaring needed to compute inversions in the NIST binary fields $GF(2^{163})$ and $GF(2^{232})$ to be:

m	$\lfloor \log_2(m-1) \rfloor$	$w(m-1)$	Multiplication	Squaring
163	7	3	9	162
233	7	4	10	232

In software implementation, the inversion is estimated to be between 9 and 30 multiplications in case of $GF(p)$ with p larger than 100 bits [23].

[5] It is derived based on the fact: $a^{-1} = a^{2^m-2}$ with a $\in GF(2^m)$. Then recursively compute $a^{-1} = \left(a^{2^{m-1}-1}\right)^2$

Therefore, one of the most important techniques that can be used to enhance the scalar multiplication is the idea of transferring the point coordinates into another coordinates that can eliminate the inversion operation.

Deciding which point *Coordinate System (CS)* to use is also one of the crucial decisions when implementing elliptic curve cryptosystem. The point coordinate system used for addition and doubling of points on the elliptic curve determines the efficiency of these operations, and hence the efficiency of the basic cryptographic operation, scalar multiplication.

This chapter discusses the various coordinates that can be used in order to eliminate the inverse operation in the scalar multiplication and hence increase the speed of calculations. We still need one final inverse operation to return back to the normal (Affine) coordinates after completing the scalar multiplication. However, there are five different coordinate systems [23] - [25]: *Affine* (A), *Homogenous Projective* (H), *Jacobian* (J), *Chudnovsky-Jacobian* (C), *Modified* (M) and *mixed* coordinate systems. The computation times in terms of number of multiplications (M), squaring (S), and inverse (I) operations are computed for each coordinate system. For simplicity we will not consider the addition, subtraction and multiplication by a small constant because they are very fast compared to multiplication, squaring and inversion operations.

Affine coordinates are the simplest to understand and are used for communication between two parties because they require the lowest bandwidth. However, the modular inversions required when adding and doubling points which are represented using Affine coordinates cause them to be highly inefficient for use in addition and doubling of points.

The other coordinate systems require at least one extra value (i.e. z-*coordinate*) to represent a point and do not require the use of modular inversions in point addition and doubling, but extra multiplications and squaring are required instead. When referring to the Affine CS, small liters are used, i.e. *x, y*, and capital liters, i.e. X, Y, Z, are used when referring to the remaining coordinate systems.

This chapter is organized as follows. Affined coordinate system is discussed in section 5.2. Sections 5.3 to 5.7 present homogenous, Jacobian, Chudnovsky-Jacobian, modified Jacobian and mixed coordinate systems. In section 5.8 conclusions are provided.

5.2 Affine Coordinates

Let:

$$ECE: \quad y^2 = x^3 + ax + b \ \ (a,b \in GF(p),\ 4a^3 + 27b^2 \neq 0). \qquad 5.1$$

be the equation of elliptic curve E over F_p. We will refer to this equation as *ECE*.

Let: $P = (x_1,y_1)$, $Q = (x_2,y_2)$ are points on E, and we want to fined $R = P + Q = (x_3,y_3)$. The affine formulas for addition and doubling are given below:

- The addition formulas ($R = P + Q = (x_3,y_3)$ where ($P \neq \pm Q$)) is given by:

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda (x_1 - x_3) - y_1 \qquad 5.2$$

$$\text{Where: } \lambda = (y_2 - y_1)/(x_2 - x_1)$$

- The doubling formulas ($R = 2P = (x_3,y_3)$) is given by:

$$x_3 = \lambda^2 - 2 x_1$$

$y_3 = \lambda (x_1 - x_3) - y_1$ 5.3

Where: $\lambda = (3 x_1^2 + a)/(2 y_1)$

The computation times for addition and doubling operations using affine coordinates are (1I + 2M + 1S) and (1I + 2M + 2S) respectively.

5.3 Homogenous Projective Coordinates

In homogenous projective coordinates the following transformation functions are used to get the projected X & Y coordinates:

$$x = \frac{X}{Z} \text{ and } y = \frac{Y}{Z}$$

The ECE becomes:

$$Y^2Z = X^3 + aXZ^2 + bZ^3 \quad 5.4$$

In this CS, the points P, Q, and R are represented as follows:

P = (X_1,Y_1,Z_1), Q = (X_2,Y_2,Z_2), and R = P + Q = (X_3,Y_3,Z_3).

- The addition formulas are given by:

$$X_3 = vA, \quad Y_3 = u\left(v^2 X_1 Z_2 - A\right) - v^3 Y_1 Z_2, \quad Z_3 = v^3 Z_1 Z_2 \quad 5.5$$

where:

$u = Y_2 Z_1 - Y_1 Z_2$, $v = X_2 Z_1 - X_1 Z_2$ and $A = u^2 Z_1 Z_2 - v^3 - 2v^2 X_1 Z_2$

- The doubling formulas are given by (R = 2P):

$$X_3 = 2hs, \quad Y_3 = w\left(4B - h\right) - 8Y_1^2 s^2, \quad Z_3 = 8s^3 \quad 5.6$$

where:

$w = aZ_1^2 + 3X_1^2$, $s = Y_1Z_1$, $B = X_1Y_1s$ and $h = w^2 - 8B$

The computation times for addition and doubling operations using homogenous coordinates are (12M + 2S) and (7M + 5S) respectively.

5.4 Jacobian Coordinates

In Jacobian CS, the following transformation functions are used:

$$x = \frac{X}{Z^2} \text{ and } y = \frac{Y}{Z^3}$$

The ECE becomes:

$$Y^2 = X^3 + aXZ^4 + bZ^6 \qquad 5.7$$

In this CS, the points P, Q, and R are represented as follows:

P = (X_1, Y_1, Z_1), Q = (X_2, Y_2, Z_2), and R = P + Q = (X_3, Y_3, Z_3).

- The addition formulas are given by:

$$X_3 = -H^3 - 2U_1H^2 + r^2, \quad Y_3 = -S_1H^3 + r(U_1H^2 - X_3), \quad Z_3 = Z_1Z_2H \qquad 5.8$$

where:

$U_1 = X_1Z_2^2$, $U_2 = X_2Z_1^2$, $S_1 = Y_1Z_2^3$, $S_2 = Y_2Z_1^3$, $H = U_2 - U_1$, and $r = S_2 - S_1$

- The doubling formulas are given by (R = 2P):

$$X_3 = T, \quad Y_3 = -8Y_1^4 + M(S - T), \quad Z_3 = 2Y_1Z_1 \qquad 5.9$$

where: $S = 4X_1Y_1^2$, $M = 3X_1^2 + aZ_1^4$, and $T = -2S + M^2$

The computation times for addition and doubling operations using Jacobian coordinates are (12M + 4S) and (4M + 6S) respectively.

5.5 Chudnovsky-Jacobian Coordinates

D. V. Chudnovsky [25] concluded that Jacobian coordinate system provide faster doubling and slower addition compared to projective coordinates. In order to speedup addition, he proposed the Chudnovsky-Jacobian coordinate system. In this CS, a Jacobian point is represented internally as 5-tupel point (X, Y, Z, Z^2, Z^3). The transformation and ECE equations are the same as in Jacobian CS, while the points P, Q, and R represented as follows:

P = (X_1, Y_1, Z_1, Z_1^2, Z_1^3), Q = (X_2, Y_2, Z_2, Z_2^2, Z_2^3), and R = P + Q = (X_3, Y_3, Z_3, Z_3^2, Z_3^3).

The main idea in Chudnovsky-Jacobian coordinate is that the Z_2, Z_3 are already calculated in the previous iteration and no need to calculate them again in the current iteration. In other words, Z_1^2, Z_1^3, Z_2^2, Z_2^3 are computed during the previous iteration and fed to the current iteration as inputs, while Z_3^2, Z_3^3 need to be calculated.

- The addition formulas are given by:

$$X_3 = -H^3 - 2U_1H^2 + r^2, \quad Y_3 = -S_1H^3 + r(U_1H^2 - X_3), \quad Z_3 = Z_1Z_2H \qquad 5.10$$

$$Z_3^2 = Z_3^2, \quad Z_3^3 = Z_3^3$$

where:

$U_1 = X_1Z_2^2$, $U_2 = X_2Z_1^2$, $S_1 = Y_1Z_2^3$, $S_2 = Y_2Z_1^3$, $H = U_2 - U_1$, and

$r = S_2 - S_1$

- The doubling formula is given by (R = 2P):

$$X_3 = T, \quad Y_3 = -8Y_1^4 + M(S - T), \quad Z_3 = 2Y_1Z_1 \qquad 5.11$$

$$Z_3^2 = Z_3^2, \quad Z_3^3 = Z_3^3$$

where: $S = 4X_1Y_1^2$, $M = 3X_1^2 + a(Z_1^2)^2$, and $T = -2S + M^2$

The computation times for addition and doubling operations using Chudnovsky-Jacobian coordinates are (11M + 3S) and (5M + 6S) respectively.

5.6 Modified Jacobian Coordinates

Henri Cohen et. al. modified the Jacobian coordinates and claimed that they got the fastest possible point doubling. The term (aZ^4) is needed in doubling rather than in Addition. Taking this into consideration, they employed the idea of internally representing this term and provide it as input to the doubling formula. The point is represented in 4-tuple representation (X, Y, Z, aZ^4). It uses the same transformation equations used in Jacobian coordinates.

The points P, Q, and R are represented as follows:

P = (X_1, Y_1, Z_1, a Z_1^4), Q = (X_2, Y_2, Z_2, a Z_2^4), and R = P + Q = (X_3, Y_3, Z_3, a Z_3^4)

- The addition formulas are given by:

$$X_3 = -H^3 - 2U_1H^2 + r^2, \quad Y_3 = -S_1H^3 + r(U_1H^2 - X_3), \quad Z_3 = Z_1Z_2H \qquad 5.12$$

$$aZ_3^4 = aZ_3^4$$

where:

$U_1 = X_1Z_2^2$, $U_2 = X_2Z_1^2$, $S_1 = Y_1Z_2^3$, $S_2 = Y_2Z_1^3$, $H = U_2 - U_1$, and $r = S_2 - S_1$

- The doubling formula is given by (R = 2P):

$$X_3 = T, \quad Y_3 = M(S - T) - U, \quad Z_3 = 2Y_1Z_1 \qquad 5.13$$

$$aZ_3^4 = 2U(aZ_1^4)$$

where: $S = 4X_1Y_1^2$, $U = 8Y_1^4$, $M = 3X_1^2 + aZ_1^4$, and $T = -2S + M^2$

The computation times for addition and doubling operations using modified Jacobian coordinates are (13M + 6S) and (4M + 4S) respectively.

5.7 Mixed Coordinates

Henri Cohen et al. [23] recommended the idea of mixed coordinates, where the inputs and outputs to point additions and doublings may be in different coordinates. i.e. with mixed coordinates we can add two points where one point is given in some coordinate system and the other point is in some other coordinate system. Also, the result point can be computed in a third coordinate system.

Consider the coordinate systems discussed so far. We have many choices in order to mix them in one operation. For example, we can select Affine coordinates for input points and the result be in Chudnovsky-Jacobian coordinates. This mixing can be denoted by (AAC), where the first two letters denote the input coordinates (Affine) and the third one represents the result coordinates (Chudnovsky-Jacobian). In case of doubling, (AM) means that the input point is represented in Affine coordinates and the result is in Modified coordinates. However, Cohen does not show the formulas used in case of mixing different coordinates. Therefore, considerable effort needs to be spent to derive

these equations. He provides the cost of mixed coordinates in terms of number of multiplication, squaring and inversion operations required for Addition and Doubling operations as shown in Table 5.1 [23].

Table 5. 1: Costs of Addition and Doubling operations using mixed coordinates

Coordinates	S	M	I
Point Addition			
AAC	4	2	
AAM	5	3	
AJJ	8	3	
AHH	9	2	
ACC	8	3	
AJM	9	5	
AMM	9	5	
CCC	11	3	
HHH	12	2	
JJJ	12	4	
JJM	13	6	
MMM	13	6	
AAA	2	1	1
Point Doubling			
AJ	5	2	
MJ	3	4	
MM	4	4	
AC	3	5	
AM	5	4	
CC	5	6	
JJ	4	6	
HH	7	5	
AA	2	2	1

In order to use mixed coordinates it is necessary to be able to convert a point representation from one coordinate system to another. Table 5.2 presents the number of

multiplications, squaring, and inversions required to convert a point representation among the discussed five coordinate systems.

Table 5. 2: Point Conversions among different coordinates

From \ To	Affine	Projective	Jacobean	Chudnovsky	Modified
Affine	-	-	-	-	-
Projective	*2M + I*	-	*2M + I*	*2M + I*	*2M + I*
Jacobean	*3M+S+I*	*3M+S+I*	-	*2M*	*3M*
Chudnovsky	*3M+S+I*	*3M+S+I*	-	-	*3M*
Modified	*3M+S+I*	*3M+S+I*	-	*2M*	-

Table 5.2 shows that the conversion from Affine coordinates to any of the other coordinate systems is very efficient because the conversions only consist of setting all of the Z, Z^2 and Z^3 coordinates to one, and the aZ^4 coordinate to *a* (the elliptic curve parameter). Conversion to or from homogenous projective coordinates is inefficient because of the inversion required, as is converting from any of the other coordinate systems to affine coordinates.

5.8 Conclusions

This chapter has discussed the various coordinates that can be used in order to eliminate the inverse operation in the scalar multiplication. Five different coordinate systems were studied: *Affine* (A) CS, *Homogenous Projective* (P) CS, *Jacobian* (J) CS, *Chudnovsky-Jacobian* (C) CS, and *Modified* (M) CS. The computation times in terms of number of multiplications (M), squaring (S), and inverses (I) operations were computed for each coordinate system. Also, mixed coordinates system in which the inputs and

outputs to point additions and doublings may be in different coordinates has been illustrated. Comparisons among different coordinate systems and the required operations to convert a point form one coordinate system to another were provided.

CHAPTER 6

Side Channel Attacks and Countermeasures

6.1 Introduction

Every computing device acts also as a source of additional information usually called side channel leak information (figure 6.1). Depending on its internal computations, it consumes different amounts of power, emits different amounts of electromagnetic radiations, needs different running times or even produces different types of error messages or sounds. All these additional types of information can and have already been exploited in attacks.

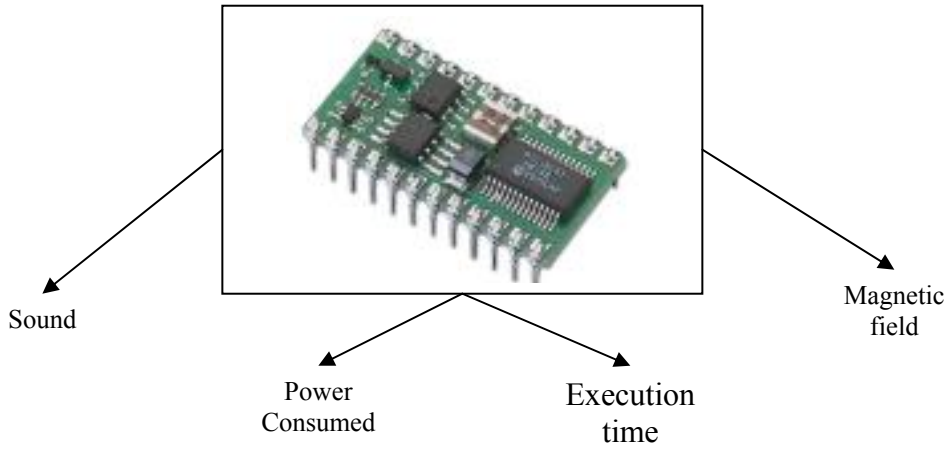


Figure 6. 1: Side channel leak Information.

Side-channel cryptanalysis takes advantage of implementation-specific characteristics to recover the secret parameters involved in the computation. It is therefore much less general than classical cryptanalysis – since it is specific to a given implementation – but often much more powerful, and is considered very seriously by cryptographic devices' implementers.

In this chapter, we survey different types of side channel attacks and the various countermeasures known at the time of writing. Also, the classification methods of the attacks found in the literature are discussed. Based on that, we propose a new classification method according to the type of information being leaked. This classification method is used to classify and analyze both the attacks and countermeasures.

The remaining of this chapter is organized as follows. Section 6.2 gives a classification of the various attacks found in the literature. It also presents the proposed classification method. Sections 6.4 to 6.8 describe the various side channel attacks, namely, fault attacks, timing attacks, power analysis attacks, electromagnetic attacks and projective coordinates leak. Section 6.9 presents countermeasures for these attacks. In section 6.9, we classify the countermeasures according to the proposed classification. Also in this section, we analyze each countermeasure via providing the attacks that it can defend, attacks that it cannot defend, its advantages and weaknesses. Finally, conclusions are drawn in section 6.10.

6.2 Classification of Side Channel Attacks

The literature usually classifies side channel attacks depending on the way they affect the attacked device. This result in the following two orthogonal axes.

Invasive vs. non-invasive: invasive attacks require depackaging the chip to get direct access to its components; a typical example of this is the connection of a wire on a data bus to see the data transfers. A non-invasive attack only exploits externally available information such as running time and power consumption. In [80], Skorobogatov and Anderson add a new distinction with what they call semi-invasive attacks. These attacks have the specificity that they require depackaging of the chip to get access to the chip surface, but do not tamper with the passivation layer – they do not require electrical contact to the metal surface.

Active vs. passive: active attacks try to tamper with the device's proper functioning; for example, fault-induction attacks will try to induce errors in the computation. As opposed, passive attacks will simply observe the device's behavior during its processing, without disturbing it.

Although these classifications help in organizing the attacks into groups, it does not help in providing the type of information being leaked. Therefore, we propose the following classification based on the type of information being leaked so that it is possible to devise some countermeasures to protect against attacks of certain class. This classification divides all known attacks into three classes: Class A: *Operation-dependent* attacks that depend on the type of operation being performed (multiply, square, addition,

doubling, etc…) such as timing attacks and simple power analysis attacks. Class B: *Data-dependent* attacks that are based on the data being manipulated by the cryptodevice such as fault attacks and projective coordinate leaks. Class C: *Address-dependent* attacks that are based on the addresses (locations) of the data being processed such as and address-bit differential power attacks. Table 6.1 presents the various side channel attacks according to the above proposed classification.

Note that some attacks exploit both the data being processed and a certain operation such as doubling certain point to leak some information. Examples of these attacks are DPA and DEMA. This will be illustrated in more details when we discuss each attack alone.

Let the type of information being leaked be represented by a binary variable that equals "1" when this type of information is leaked and "0" when it is not. For example, let O denotes operation-dependent information, D denotes data-dependent information and A denotes Address-dependent information. Then, there are seven possible classes of attacks each of which exploits one or more kind of leaked information. These classes range from ADO = 001 to 111. The code 000 means no attacks while 111 means an attack that exploits operations, data and locations of data. Table 6.2 lists the side channel attacks and the code of each one according to this general classification.

SPA attack has the code 001 because it is based on the conditional ADD operation whether it is performed or not (section 6.5.1). DPA attack has the code 011 because it is based on operations being performed on classified input points (section 6.5.1). ABDPA

has the code 100 because it is based on the addresses (or locations) of data being manipulated.

Table 6. 1: Classification of side channel attacks.

Class	Attack	Year of discovery[6]	Target
A: **Operation-dependent**	Timing Attack (TA)	1996 [27]	-Conditional operations. - Small differences obtained from feeding the operations with classified input points
	Simple Power Analysis (SPA) attack	1999 [26]	- Conditional operations. -Optimization techniques
	Differential Power Analysis (DPA) attack	1999 [26]	-Small differences obtained from feeding the operations with classified input points.
	Simple Electromagnetic Analysis (SEMA) attack	2000 [46]-[48]	- Conditional operations. -Optimization techniques
	Differential Electromagnetic Analysis (DEMA) attack	2000 [46]-[48]	-Small differences obtained from feeding the operations with classified input points.
	Doubling Attack (DA)	2003 [30]	-Zeros in the scalar.
B: Data-dependent	Fault Attacks (FA)	1997 [43]-[45]	-Registers (variables) content.
	Timing Attack (TA)	1996 [27]	- Small differences obtained from feeding the operations with classified input points
	DPA attack	1999 [26]	Small differences obtained from feeding the operations with classified input points
	DEMA attack	2000 [46]-[48]	Small differences obtained from feeding the operations with classified input points
	Refined Power Analysis (RPA) attacks	2003 [28]	-Coordinates of a point.
	Doubling Attack (DA)	2003 [30]	-Zeros in the scalar.
	Zero-value Point Attack (ZPA)	2003 [29]	-Registers (variables) content.
	Projective Coordinates Leak (PCL)	2004 [42]	-Projective coordinates of a point. (not affine)
C: Address-dependent	Address-bit DPA (ABDPA)	2002 [38],[39]	-Addresses (Locations) of variables.

[6] The year shown is either the discovery year of the attack or its application to ECC.

The three classes in table 6.1 are special cases of the general classification in table 6.2. However, since most of classes in this general classification are empty (at the time of writing) especially classes from 101 to 111, we stick to the proposed classification presented in table 6.1.

In the following sections, we discuss all side channel attacks listed in table 6.2 in the same order they appear in the table.

Table 6. 2: Codes of side channel attacks.

Attack	Code (ADO)	Description
Fault Attacks (FA)	010	Based on faults induced to the data being manipulated.
Timing Attack (TA)	011	Based on the variation in execution time for classified input points.
Simple Power Analysis (SPA) attack	001	Based on the conditional ADD operation, i.e. whether it is performed or not.
Differential Power Analysis (DPA) attack	011	Based on operations being performed on classified input points.
Refined Power Analysis (RPA) attacks	010	Exploits a special point with zero-value such as (0, y) or (x, 0).
Zero-value Point Attack (ZPA)	010	A generalization of RPA where it exploits any zero-value auxiliary register.
Doubling Attack (DA)	011	Based on detecting when the same operation is performed on the same operands.
Address-bit DPA (ABDPA)	100	Based on the idea that accessing the same location is correlated to the scalar bit value.
Simple Electromagnetic Analysis (SEMA) attack	001	Based on the conditional ADD operation, i.e. whether it is performed or not.
Differential Electromagnetic Analysis (DEMA) attack	011	Based on operations being performed on classified input points.
Projective Coordinates Leak (PCL)	010	Based on knowing the projective representation of a point obtained using a particular projective coordinate system.

6.3 Fault Analysis Attacks

Fault attacks were introduced by Boneh et al in [43]. Fault attacks are based on tampering with a device in order to have it perform some erroneous operations, hoping that the result of that erroneous behavior will leak information about the secret parameters involved – for example by changing some bits in the internal memory.

Boneh et al classified the faults into three categories. The first type is transient faults which can occur randomly causing a faulty computation to be executed. The second type is latent faults, which are hardware or software bugs that are difficult to locate. The third type is induced faults for which physical access to the hardware is necessary. Induced faults are the most interesting because of the active role of the attacker. For example, optical fault induction attacks, as introduced by Scorobogatov and Anderson [44], use a flashgun targeting a transistor to change the state of a memory cell in a microcontroller. The authors have proven this optical probing to be feasible as they managed to change an arbitrary bit of an SRAM array.

Differential fault attacks (DFA) on ECC cryptosystems were outlined in the work of Biehl et al. [45]. They presented three types of attacks on ECC that can be used to derive information about the secret key if bit errors can be inserted into the elliptic curve computations in a tamper-proof device. They also estimate the effectiveness of the attacks using a software simulation.

Their methods require very precise placement and timing of the faults and depend on the ability to change the coordinates of a point at any specific iteration of the scalar multiplication. Based on that, the scenario of DFA on ECC is the following:

Let the binary representation of the scalar K is,

$$K = k_{n-1}2^{n-1} + k_{n-2}2^{n-2} + ... + k_1 2 + k_0 \qquad 6.1$$

And let P be the base point, and the right-to-left scalar multiplication algorithm is:

H = P; Q = 0;
for i = 0 to n-1 do
    if (k_i = 1) then Q = Q + H;
    H = 2 H;
end for;
return Q;

Assume that we know the binary length n of the unknown scalar K (note that an attacker can easily guess this length). Denote by Q[i], H[i] the value stored in the variable Q, H in the algorithm above before iteration i. The final result will then be Q[n-1]. The attacker proceeds as follows:

1. Use the tamper-proof device with some input P_e to get the correct result Q[n-1] = $K P_e$.
2. Restart scalar multiplication with the same input P_e but enforce a random register fault to get a faulty result $\tilde{Q}[n-1]$. Assume that we enforce the register fault in beginning of the last iteration, n-1, and that this fault changes the variable H.
3. If the final result is unchanged, then there was no addition in the last iteration and k_{n-1} = 0, otherwise there was an addition and k_{n-1}= 1 (remember that the final result is in the variable Q, see the above algorithm).

   Clearly, we can do this for each bit of the scalar.

Fault attacks can be considered as one of the biggest threat of all implementation attacks as countermeasures usually include more complex techniques which are not easy to implement on constraint environment such as smart cards.

6.4 Timing attack

In 1996 Kocher [27] described timing based attacks on public key algorithms such as RSA. Timing attacks are based on the fact that algorithms with a non-constant execution time can leak secret information. A non-constant execution time can be caused by conditional branches in the algorithm, various optimization techniques, cache hits, etc. For example, the binary algorithm 4.1 (in chapter 4) of the scalar multiplication performs the addition operation only if the current bit of the scalar is 1. Hence there will be different execution times when the current bit is 0 or 1.

Assume that the scalar *K* is constant throughout the attack and that the attacker can choose the input points. The scenario of timing attack on ECC is the following:

Let the scalar *K* be represented by the binary representation 6.1. Assume that algorithm 4.1 is used for the scalar multiplication. Suppose that the bits $k_{n-1}, k_{n-2}, ..., k_{j+1}$ are known. The attacker wants to find the *j-th* bit, k_j. He proceeds as follows:

1. The attacker first makes a guess: $k_j = 1$ (or 0).
2. He takes several input points $D_1, ... D_t$ and divides these points into two subsets based on the following rule: based in his knowledge about the scalar multiplication

algorithm, he knows (via simulation for example) that some points need more time than the others to be doubled and added to a fixed base point *P*. This difference in time comes due to the fact that doubling certain point and adding the result to the base point needs more modular reductions than other points. Based on that, he selects input points $D_1,...D_t$ and classify them into two subsets: S1 for which the computation of DBL(D_i) and ADD(D_i+*P*) will induce a modular reduction and S2 for which it will not.

3. For each input point D_i, he computes a full scalar multiplication KD_i. If k_j is really one, then we can expect the computation times for the points from S1 to be slightly higher than the corresponding times for S2. On the other hand, if the actual value of k_j is zero, then the ADD operation will not be performed and the separation into two subsets should look random and we should not observe any distinguishable difference in the computation times.

6.5 Power Analysis Attacks

The power consumption of a cryptographic device may provide much information about the operations that take place and the involved parameters. This is the idea of simple and differential power analysis, first introduced by Kocher et al. in [26] and [27]. After publication of these two main types, other power analysis attacks have been discovered. At the time of writing there are six types of power analysis attacks. These attacks are: Simple Power Analysis (SPA) attack [26], Differential Power Analysis

(DPA) Attack [26] and [32], Refined Power Analysis (RPA) attack (also known as Goubin attack) [28], Zero-value Point Attack (ZPA) [29], Doubling Attack (DA) [30] and Address-Bit Differential Power Analysis (ABDPA) Attack [38], [39]. Sections 6.6.1 to 6.6.6 discuss each of these attacks.

6.5.1 Simple Power Analysis (SPA) Attack

SPA makes direct use of one power consumption measurement. A *trace* refers to a measurement (*i.e.*, a dataset) taken for one execution of the cryptographic operation under attack. In a simple power analysis attack, only a single measurement is used to gain information about the secret key of a device. Obviously, to perform such an attack the side-channel information needs to be strong enough to be directly visible in the trace. Additionally, the secret key needs to have some simple, exploitable relationship with the operations visible in the power trace. Such an attack typically targets implementations which use key dependent operations in the implementation.

An important characteristic of simple power attacks is the assumption that the attacker is supposed to have a detailed knowledge about the implementation of the cryptographic algorithm under attack. Furthermore, the part(s) of the trace corresponding to the operation under attack needs to be clearly distinguishable from the whole trace.

In elliptic curve cryptography, SPA attack consists of observing the power consumption during a single execution of an elliptic curve cryptographic algorithm. The power consumption analysis may enable one to distinguish between point addition and

point doubling in the non-immune scalar multiplication algorithm. As shown in scalar multiplication algorithms presented in section 4.3 namely Algorithms 4.1 and 4.2, performing the ADD operation is conditioned by the scalar (key) bit. If the scalar bit value is ONE, an ADD operation is performed, otherwise, an ADD operation is not performed. Therefore, a simple power analysis will produce different power traces that distinguish between the existence of an ADD operation or not. This can reveal the bit values of the scalar.

6.5.2 Differential Power Analysis (DPA) Attack

Even if an algorithm is protected against SPA attack, it may be vulnerable to the more sophisticated differential power analysis (DPA) attack. DPA attack is based on the same basic concept as a SPA attack, but makes use of several measurements and statistical analysis to extract very small differences in the power consumption signals.

Assume that the scalar multiplication algorithm is immune against SAP by using double-and-add always method (algorithms 6.2 or 6.3). Let the scalar *K* be represented by 6.1 where k_i is the *i-th* bit of the binary representation of *K*, and *n* is the total number of bits. If one knows the binary representation of the computed points one can again mount a successful attack. At step *i* the processed point P depends only on the first bits $k_{n-1}...k_i$ of the secret scalar *K*. When *P* is processed, power consumptions is correlated to the bits of *P*. No correlation will be observed if the point is not computed. For example, the second most significant bit can be learned by calculating the correlation between the power

consumption and any specific bit of the binary representation of *4P*. If k_{n-2} = 0, *4P* is computed during the binary algorithm. Otherwise if k_{n-2} = 1, *4P* is never computed and thus there will be no correlation observed. This correlation method is used to classify power traces of several input points chosen by the attacker. In the following we present a possible scenario of DPA.

Assume that an attacker already knows the highest bits, $k_{n-1}, k_{n-2}.....k_{j+1}$, of *K*. (i.e. the bits from position *j+1* up to *n-1* where *j* is the current position) and he wants to find k_j. The scenario of DPA on ECC is the following:

1. The attacker first makes a guess: $k_j = 0$ (or 1).
2. He chooses several input points $D_1,...D_t$ and computes $Q_i = 2\left(\sum_{d=j}^{n-1} k_d 2^{d-j}\right)D_i$. The attacker can compute these points using a small program. For example, in attacking bit k_{n-2} if the attacker guess that $k_j = 0$, then he will computes (He will compute not the cryptodevice) $Q_1,...Q_t = 4D_1,...4D_t$.
3. He picks a certain bit in the binary representation of $Q_1,...Q_t$ (fixed for all points) as a boolean selection function *g* to construct the following two index sets:

$$S_t = \{i : g(Q_i) = true\} \qquad and \qquad S_f = \{i : g(Q_i) = false\}$$

For example, *g* is chosen to be a specific bit of the binary representation of $4D_1,...4D_t$ in case of attacking bit k_{n-2}. Note that the same bit must be chosen for all points.

4. Let $C_i = C_i(\tau)$ = power trace obtained from the computation of a full scalar multiplication KD_i. This is a function of the time τ.

5. Let $\langle C_i \rangle_{i \in S}$ denote the average of the functions C_i for the $i \in S$, $S = S_t \cup S_f$. If the guess of k_j was incorrect then

$$\langle C_i \rangle_{i \in S_t} - \langle C_i \rangle_{i \in S_f} \approx 0$$

i.e. the two sets are uncorrelated.

On the other hand, if the guess of k_j was correct then the difference

$\langle C_i \rangle_{i \in S_t} - \langle C_i \rangle_{i \in S_f}$ will present spikes, i.e. deviations from zero.

6.5.3 Refined Power Analysis (RPA) Attack

In 2003, DPA is further improved to the Refined Power Analysis (RPA) by Goubin et al [28]. RPA exploits a special point with a zero value and reveals a secret key. An elliptic curve happens to have a special point (0, y) or (x, 0), which can be controlled by an adversary because the order of base point is usually known. RPA utilizes such a feature that the power consumption of 0 is distinguishable from that of a non-zero element. Although elliptic curve cryptosystems are vulnerable to RPA, RPA is not applied to RSA or DLP-based cryptosystems because they don‘t have such a special zero element. In general, the RPA attack assumes that the attacker can input adaptively chosen messages or elliptic curve points to the victim scalar multiplication algorithm.

Smart analyzed the RPA attack in detail and discounted its effectiveness in a large number of order [37]. However, the RPA attack is still a threat to most elliptic curve cryptosystems.

6.5.4 Zero-value Point Attack (ZPA)

RPA is generalized to Zero-value Point Attack (ZPA) in [29]. ZPA makes use of any zero-value register used in addition or doubling formula. ZPA utilizes a special feature of elliptic curves that addition and doubling formulas need a lot of each different operations stored in auxiliary registers, one of which happens to become zero.

In ZPA, the attacker utilizes an auxiliary register which might take a zero-value in the definition field. This auxiliary register will take a value of zero for certain operations that are some how correlated to the scalar bit values. Hence, some secret bits may be revealed.

6.5.5 Doubling Attack

In 2003, a new attack known as Doubling attack is proposed by Fouque et al [30]. DA only works for the ML binary method. The main idea of this attack is based on the fact that, even if an adversary cannot see whether the computation being done is doubling or addition, he can still detect when the same operation is done twice. More precisely, if a device computes 2A and 2B in any operation, the attacker is not able to guess the value of A or B but he can check if $A = B$. This assumption is reasonable since this kind of computation usually takes many clock cycles and depends greatly on the value of the

operands. If the noise is negligible, a simple comparison of the two power traces during the doubling will be efficient to detect this equality.

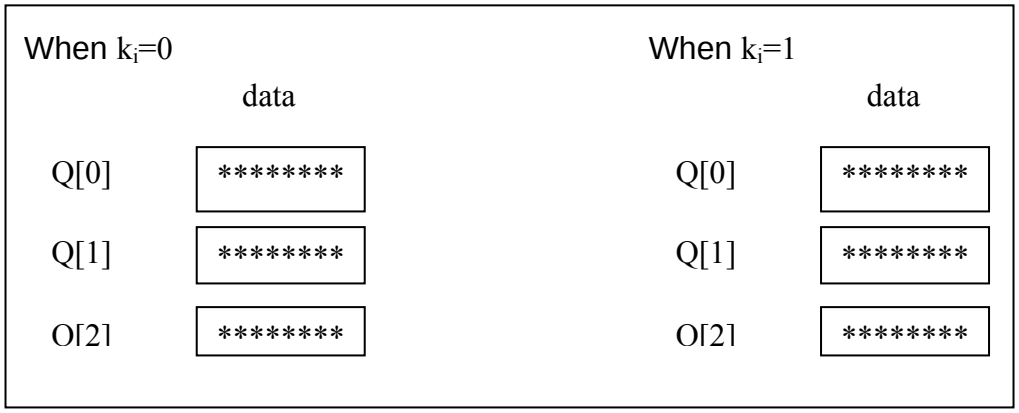
6.5.6 Address-Bit Differential Power Analysis Attack

In 1999, Messerges et al. proposed a new attack against the secret key cryptosystems, the *address-bit DPA* (ABDPA), which analyzes a correlation between the secret information and addresses of registers [38]. Then, in 2002, Itoh et al. extended the attack to Elliptic Curve based Cryptosystems [39].

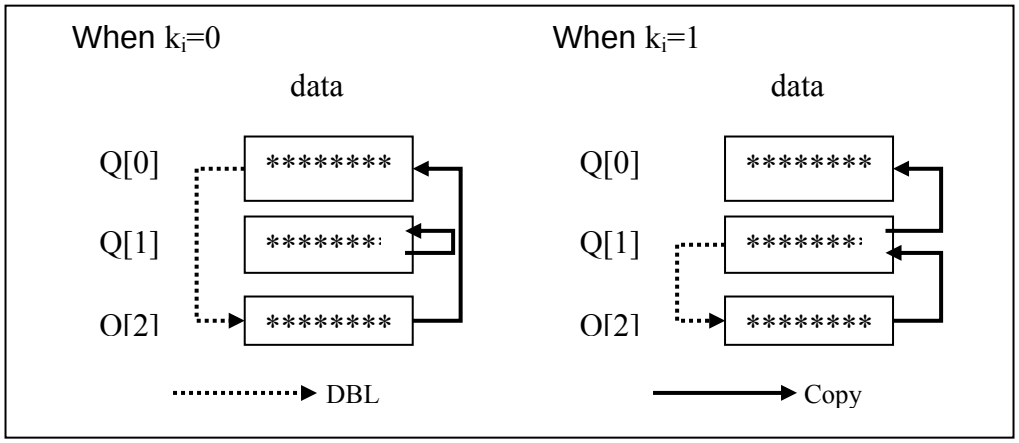
Address-bit Differential Power Analysis Attack is based on the correlation between bit values of the scalar and the location (address) of the variables used in a scalar multiplication algorithm. Consider for example Takagi's algorithm (algorithm 6.3). The values of variables Q[0], Q[1] and Q[2] can be randomized by randomizing the projective coordinates (or the base point) as shown in Figure 6.2(a). However, Figure 6.2(b) shows that the location of input operand of DBL operation (dotted line) and the data transfer from either Q[1] or Q[2] to Q[0] (solid line) are correlated to the bit value of the scalar. This Figure shows that, in Takagi's algorithm, the following data transfer is performed based on the bit value of the scalar:

$$Q[0] = \begin{Bmatrix} Q[2] & k_i = 0 \\ Q[1] & k_i = 1 \end{Bmatrix}$$

$$Q[1] = \begin{Bmatrix} Q[1] & k_i = 0 \\ Q[2] & k_i = 1 \end{Bmatrix}$$



(a) Randomizing data by using randomized projective coordinates



(b) Correlation still exists between the addresses and the bit values of the scalar

Figure 6. 2: Address-bit differential power analysis attack

6.6 Electromagnetic Analysis Attacks

Any movement of electric charges is accompanied by an electromagnetic (EM) field. The currents going through a processor can characterize it according to its spectral signature. Electromagnetic attacks, first introduced by Quisquater and Samyde [46], and further developed in [47], [48] exploit this side channel by placing coils in the neighborhood of the chip and studying the measured electromagnetic field.

The information measured can be analyzed in the same way as power consumption (simple and differential electromagnetic analysis – SEMA and DEMA), but may also provide much more information and are therefore very useful, even when power consumption is available. Agrawal et al [49] show that EM emanations consist of a multiplicity of signals, each leaking somewhat different information about the underlying computation. They sort the EM emanations in two main categories: direct emanations, i.e. emanations that result from intentional current flow, and unintentional emanations, caused by coupling effects between components in close proximity. According to them, unintentional emanations, which have been somewhat neglected so far, can prove much more useful than direct emanations. Moreover, some of them have substantially better propagation than direct emanations, which enables them to be observed without resorting to invasive attacks (and even, in some cases, to be carried out at pretty large distances - 15 feet! - which comes back to the field of tempest-like attacks [50]). Finally, they argue that EM emanations can even be used to break power analysis countermeasures, and illustrate this by sketching a practical example.

Electromagnetic attacks are powerful attacks especially when combined with other side channel attacks. For example, Quisquater and Samyde recently showed [51] that it was possible to build a dictionary of instructions and their power/electromagnetic traces, and, using correlation techniques and neural networks, to recognize the instructions executed by a processor.

EMA is a non-invasive attack, as it consists in measuring the near field. However, this attack is made much more efficient by de-packaging the chip first, to allow nearer measurements and to avoid perturbations due to the passivation layer.

6.7 Projective Coordinates Leak

In 2004, Nigel Smart et. al. [42] showed that it is possible to leak some information about the secret key (scalar *K*) through the projective representation of elliptic curve points. Giving that $Q = KP$ is the elliptic-curve double-and-add scalar multiplication of an elliptic curve point *P* by a secret *K*, they showed that allowing an adversary access to the projective representation of *Q* may result in information being revealed about *K*.

In [42], they restrict projective coordinates leak to Jacobian projective coordinates in *GF(p)* (although it can be applied to other coordinates). For each affine point there are *P-1* representatives in Jacobian projective coordinates, one for every non-zero value of *Z*. By knowing the projective coordinates of a point *G*, they consider the least significant bit of the scalar and guess its value. Once this is done, it is possible to compute a set of candidates for the coordinates of the previous intermediate values handled by the double-and-add algorithm while processing that bit. This is achieved by reversing computations: reversing doubling is Halving while reversing addition is subtracting. In other words, they apply a backtracking algorithm that can reveal whether the final bit was zero or not.

This attack requires a special backtracking formulas for each projective coordinate system. Thus, formulas used to half (subtract) a point in homogenous projective coordinates cannot be used to half (subtract) a point in Jacobian projective coordinates.

6.8 Countermeasures

This section presents countermeasures found in the literature for side channel attacks . We organized the countermeasures in the same way as we did for attacks.

6.8.1 Fault Attack Countermeasures

The most obvious way that comes to mind in order to protect against fault attacks is to check the computation for errors, for example by repeating the computation and comparing the results. However, it must be noted that this policy is very costly, either in time (repeat computation) or in hardware (double hardware and perform both computations in parallel). Moreover, repeating the computation is not always satisfactory as, in the case of a permanent fault induction, it will yield identical, although wrong, results.

Another way to check for the presence of faults is, in the case of public-key cryptography, to re-encrypt the message. This is usually less time-consuming, as the public exponent is usually chosen to be small.

6.8.2 Timing Attack Countermeasures

The obvious way to prevent timing attacks is to implement cryptographic algorithms with a constant execution time. In case of elliptic curve cryptography, this idea can be implemented by adding a dummy operation to balance all operations in all iterations.

Almost all modern implementations are resistant against timing attacks, which makes a timing-only attack very difficult. However, the threat remains in combining timing information with other side-channels. For example, timing information can be used by an attacker in order to locate specific parts of the algorithm.

6.8.3 SPA Attack Countermeasures

To protect against SPA attack, Coron [32] proposed a simple SPA countermeasure which consisted of modifying the binary methods shown in algorithms 4.1 and 4.2 to be as in algorithms 6.1 and 6.2 respectively. The basic idea of these countermeasures is to perform the ADD operation in all cases regardless of the scalar bit value. Therefore, the ADD operation is no longer conditioned by the scalar bit values. However, if the ADD operation is originally not required (i.e. in case of the scalar bit is 0), the result of ADD operation is simply discarded. Since none of the instructions in algorithms 6.1 and 6.2 depend on the scalar bit value, these algorithms are resistant to a SPA attack. These algorithms are called *Double-and-ADD always* algorithms since it computes a point addition and point doubling in each iteration without regard to the secret key *K*. However, even though this scheme is resistant to SPA attack, it remains vulnerable to DPA attack.

INPUT *K*, *P*
OUTPUT *KP*

1. Initialize Q[2] = P
2. **for** $i = n\text{-}2$ **down** to 0
3. Q[0] = DBL(Q[2])
4. Q[1] = ADD(Q[0], P)
5. Q[2] = Q[k_i]
6. **end for**

return Q[2]

Algorithm 6. 1: *Double-and-ADD always* Most-to-Least (ML) binary algorithm.

```
INPUT K, P
OUTPUT      KP
        1.  Initialize Q[0] = P; Q[1] = P
        2.  for i = 1 to n-1
        3.      Q[0] = DBL(Q[0])
        4.      Q[2] = ADD(Q[0], Q[1])
        5.      Q[1] = Q[1+ k_i]
            end for
        return Q[1]
```

Algorithm 6. 2: *Double-and-ADD always* Least-to-Most (LM) binary algorithm.

Another ML algorithm to avoid this kind of leak was proposed by Takagi et al [33]. This algorithm uses extra ADD operations to assure that the sequence of DBL and ADD operations is carried out in each iteration. We refer to this algorithm as Takagi's algorithm and it is shown in algorithm 6.3.

```
INPUT       K, P
OUTPUT      KP
        1.  Initialize Q[0] = P; Q[1] = 2P
        2.  for i = n-2 down to 0
        3.      Q[2] = DBL(Q[k_i])
        4.      Q[1] = ADD(Q[0], Q[1])
        5.      Q[0] = Q[2- k_i],
        6.      Q[1] = Q[1+ k_i]
        7.  end for
        return Q[0]
```

Algorithm 6. 3: Takagi's ML algorithm for scalar multiplication.

6.8.4 DPA Attack Countermeasures

In order for an algorithm to be resistant to a DPA attack, some system parameters or computation procedures must be randomized. Coron et. al [32] suggested three countermeasures to protect against a classical DPA: randomizing the scalar, randomizing

the base point P, and randomizing the projective coordinates. Brief summary of how these countermeasures can be realized is given below:

1. Randomizing the scalar K

   If $n = \text{ord}_E(P)$ denotes the order of $P \in E/GF(p)$, then $Q = KP$ can be computed as $Q = (k + r\,n)P$ for a random r. Alternatively, one can replace n by the order of the elliptic curve, $\#E/GF(p)$.

2. Randomizing the base-point P

   The base point P to be multiplied by K is randomized by adding a secret random point R for which we know $S = KR$. Scalar multiplication is done by computing the point $(R + P)K$ and subtracting $S = KR$ to get $Q = KP$.

3. Using randomized projective coordinates

   Randomized projective coordinates can use the Homogenous or Jacobian coordinate to randomize a point $P = (x, y)$. For homogenous projective coordinate, P can be randomized to (rx, ry, r) for a random number $r \in GF(p)$. Similarly, P can be randomized to (r^2x, r^3y, r) in case of using Jacobian coordinates where r is a random in $GF(p)$.

However, the main goal of all these countermeasures, and others proposed in [33] - [36], is to randomize the power traces collected by the attacker and hence make it difficult for him to exploit the differences between these traces.

6.8.5 Doubling Attack Countermeasures

According to [30], two of Coron's three proposed countermeasures against DPA attacks, discussed in the previous section, fail to protect against a doubling attack: randomizing the scalar and randomizing the base point. However, his third countermeasure, the randomized projective coordinate does protect against a doubling attack as does a randomized exponentiation algorithm such as the Ha-Moon algorithm which maps a given scalar to one of various representations [34]. Since the positions of the zeros in the Ha-Moon algorithm vary in each representation, the doubling attack cannot detect the positions of the zeros for the doubling operation.

To enhance the Coron's 2ed countermeasure, to protect against a doubling attack, the secret random point R should be randomly updated. A regularly updated method shouldn't be used.

6.8.6 RPA & ZPA Attacks Countermeasures

To protect against RPA and ZPA attacks, the base point P or the secret scalar d should be randomized. For example, Coron's first two counter-measures (but not the 3rd) protect against these attacks. Projective coordinates randomization does not protect against RPA and ZPA because it cannot randomize the zero-value operands.

Mamiya et al [31] recently proposed a countermeasure (called BRIP) which uses a random initial point (RIP) R. They computes $KP + R$ using a special algorithm and then subtracts R to get KP.

6.8.7 Address-Bit Differential Power Analysis Attack Countermeasures

The countermeasures used to protect against simple power analysis and differential power analysis that are based on randomization of the base point or the projective coordinate do not provide countermeasure against address-bit analysis attacks. Therefore, these countermeasures do not remove the correlation between the bit values of a scalar and the location (address) of the variables used in a scalar multiplication algorithm.

Itoh et al gave several countermeasures against the ABDPA attack in [39]. But those countermeasures require at least twice computing time than without them [39].

A hardware-based DPA countermeasure proposed by May et al. [40] is based on Randomized Register Renaming (RRR). RRR is supposed to be implemented on a processor that can execute instructions in parallel. In other words, it requires a special hardware to work [41].

In 2003, Itoh et al. proposed a countermeasure [41], called *randomized addressing method* (*RA*), which is similar to RRR but does not require special hardware because it can be implemented by only software with a program code. In RA, they randomize addresses of registers by a one-time random number $r_{n-1}2^{n-1} + r_{n-2}2^{n-2} + ... + r_1 2 + r_0$ where $\left(r_i \in \{0,1\}\right)$. They change each bit, k_i, of the scalar to $k_i \oplus r_i$, where $\oplus$ denotes the XOR operation. Then all addresses of registers are randomized so that the side channel information will be randomized for each scalar exponentiation. Of course this change in the scalar bits requires a special algorithm to calculate the correct point of the scalar multiplication *KP*. They provided such an algorithm in [41].

6.8.8 Electromagnetic Attacks Countermeasures

Electromagnetic attacks and power attacks are, in many respects, very similar. Although the way the side channel leaks information differs, but the type of leaking information is roughly the same. Countermeasures do not try to reduce the signal amplitude, but rather to make the information it conveys useless by obscuring the internal parameters. Therefore, any countermeasure for SPA and DPA can be used for SEMA and DEMA respectively.

6.8.9 Projective Coordinates Leak Countermeasures

Nigel Smart et al [42] suggested two methods to resist this attack. First, we call it Smart's trick, which is done by randomly replacing the output (X, Y, Z) of the computation by (X, εY, εZ), with ε = ±1. Although, this method does not lend itself to a formal proof, they claim that it can defend the PCL. However, this method does not protect against PCL if the attacker obtains intermediate points. Second, is by replacing (*x, y, z*) representation of Q by $(\lambda^2 x, \lambda^3 y, \lambda)$, where λ is randomly chosen among the non zero elements of the base field. This method, identical to Coron's 3-ed countermeasure, provides a randomly chosen set of projective coordinates for the result and, therefore, cannot leak additional information.

However, it is worth mentioning that they assume that the attacker knows the projective coordinate system used and that the coordinate system is fixed.

6.9 Classification of Countermeasures

In this section, we provide a classification of countermeasures according to the proposed classification of the attacks presented in section 6.2. Table 6.3 shows the proposed classification. In addition, table 6.3 contains the attacks that each countermeasure can help in defending them and those it cannot. Also, table 6.3 contrast the advantages and disadvantages of each countermeasure.

6.10 Conclusions

In this chapter, we have surveyed different types of side channel attacks and the various countermeasures for defending them. Also, according to the type of information being leaked, a new classification method of attacks has been proposed. This classification method was used to classify and analyze both the attacks and countermeasures. Three classes were proposed: Class A: *Operation-dependent* attacks that depend on the type of operation being performed. Class B: *Data-dependent* attacks that are based on the data being manipulated. Class C: *Address-dependent* attacks that are based on the addresses (locations) of the data being processed.

In this chapter, we analyze and contrast the existed countermeasures in terms of what attacks each countermeasure can defend and what it cannot, its advantages and disadvantages. A summary of this analyze is presented in table 6.3

We conclude that there are powerful side channel attacks that exploit more that one type of leaked information. Therefore, sophisticated countermeasures to protect

against each type of information are mandatory. We recommend that at least one countermeasure from each class should be involved in any ECC implementation.

Table 6. 3: Countermeasures classification, protection, advantages and disadvantages.

Class	Countermeasure/ (Code)	Help in protect	Not protect	Advantages	Disadvantages
A: Operation-dependent	Operations balancing by adding a dummy operation. (001)	SPA TA SEMA	DPA DEMA ABDPA DA RPA ZPA PCL	-Simple and can be plugged to any scalar multiplication algorithm.	-The dummy operation is extra operation, that increases the execution time.
	Randomizing the Scalar (001)	DPA FAs DEMA RPA ZPA	DA PCL ABDPA	-Simple and can be plugged to any scalar multiplication algorithm.	-Requires a word length multiplication and an addition operations. -Requires knowing order of the base point or the curve.
B: Data-dependent	Randomizing the base point. (Coron's 2-ed countermeasure) (010)	DPA FAs DEMA RPA ZPA PCL	DA ABDPA	-Simple and can be plugged to any scalar multiplication algorithm.	-$S = KR$ of the secret random point R must be known. Otherwise it needs to be computed hence duplicating scalar the multiplication time. -Weak since R needs to be updated.
	Randomizing projective coordinates (010)	DPA FAs DEMA DA PCL	RPA ZPA ABDPA	-Simple and can be plugged to any scalar multiplication algorithm.	-Each coordinate system requires its own randomization method. -Requires 2 multiplications in H coordinate system and 3 multiplications and one squaring in J coordinate system.
	Random initial point (RIP) (010)	DPA FAs DEMA RPA ZPA PCL	DA	-Does not require storing RK of the random point R. -Does not require updating R.	-Complex. -Needs a special scalar multiplication algorithm.
	Error detection technique (010)	FAs	The rest	- The only way to detect errors.	-Complex. -Needs special techniques. -Increase scalar multiplication time dramatically.
	N. Smart's trick (010)	PCL	The rest	-Simple.	-Does not protect PCL if the attacker obtain intermediate points.
C: Address-dependent	Randomized register renaming (RRR) (100)	ABDPA	The rest	-Faster than RA.	-Requires special hardware.
	Randomized addressing (RA) (100)	ABDPA	The rest	-Does not requires special hardware.	-Requires special scalar multiplication algorithm.

CHAPTER 7

Dynamic Projective Coordinate (DPC) System

7.1 Introduction

Using projective coordinates in point addition and doubling operations is an important requirement to remove the need for intermediate inversion operations in the scalar multiplication. The usual way used in the literature to achieve this is by using a fixed coordinate system that is selected in the design stage. The selected system is used in a fixed manner for all scalar multiplication iterations. However, although using a fixed coordinate system removes the intermediate inversion operations, it becomes a security weakness since it can be exploited by projective coordinates leak attacks to reveal some secure information (section 6.7 in chapter 6). Therefore, finding a coordinate system that can satisfy both requirements: removing the intermediate inversions and being secure against such attacks is mandatory.

Although, mixed coordinates (section 5.7) provide efficient addition and doubling operations, most of them cannot be used for the following reasons:

- It is necessary to convert a point representation from one coordinate system to another to have the input in the required format for the addition or doubling

operation. For example, using Jacobian coordinate for addition operation and homogenous coordinates for doubling operation requires converting the addition result to homogenous coordinates. This conversion requires an inversion operation. Same thing happens if using homogenous for addition and Jacobian for doubling.

- It requires separate mathematical formulas for each coordinate system.

However, using different projective coordinates for different runs and/or different phases of the scalar multiplication is not used yet as a randomization method to resist many operation-dependent and data-dependent attacks.

In this chapter, we introduce the ***Dynamic Projective Coordinate (DPC) system*** which is proposed to overcome the above difficulties and has the following properties:

- It automates the selection of the projective coordinate system and uses a single mathematical formulation/software code to implement different projective coordinate systems.
- It allows the computing/encrypting device to select the projective coordinate either at random, or according to a certain rule.
- Different projective coordinates can be implemented by using two parameters where one parameter defines the projection of the x-coordinate and a second parameter defines the projection of the y-coordinate of an elliptic curve point. This allows different projective coordinates to be used within the same mathematical formulation in calculating the scalar multiplication.

- The computation of the scalar multiplication can be randomized by simply varying either the *x*-coordinate projecting parameter and/or the *y*-coordinate projecting parameter.
- It allows projective coordinates hopping at any time during the scalar multiplication.
- With DPC system, different projective coordinate systems can be used for different phases of the scalar multiplication. Fore example, a certain coordinate system can be used for the pre-computation phase of the scalar multiplication while other coordinate systems can be used for addition and/or doubling operations in the main loop. Furthermore, different blocks (or windows) of the scalar *K* can use different projective coordinate systems.
- It does not require the sending and receiving correspondents to use to the same projective coordinates in computing the same scalar multiplication.

In this chapter, we start by defining dynamic transformation functions that are used to convert any affine point to any projective coordinates using the same mathematical formula. Then these transformation functions are used to develop dynamic addition and doubling formulas for elliptic curve over the prime field *GF(p)* and elliptic curve over binary field $E/GF(2^m)$.

The rest of this chapter is organized as follows. Section 7.2 introduces the proposed dynamic projecting parameters and transformation functions. In section 7.3, DPC is used to propose dynamic addition and doubling formulas for elliptic curve over finite field *GF(P)*. Similarly, in section 7.4, DPC is used to propose dynamic addition and

doubling formulas for elliptic curve over finite field $GF(2^m)$. Finally, conclusions are presented in section 7.5

7.2 Dynamic Projecting Parameters

In DPC, we use two values Z^{L_x} and Z^{L_y} for projecting the x-coordinate and the y-coordinate of a point respectively. L_x and L_y are projecting parameters (powers) that can be chosen either at random or according to a certain criteria such as a criteria for reducing the computation complexity.

To formulate the Dynamic Projective Coordinate system, consider that there are multiple degrees of powers for the *Z*-coordinate, as follows:

Degree-0 is the affine coordinate system *P = (x,y)*

In Degree-1, $x = \frac{X}{Z}$, $y = \frac{Y}{Z}$

In Degree-2, $x = \frac{X}{Z^2}$, $y = \frac{Y}{Z^2}$

… … …

In Degree-i, $x = \frac{X}{Z^i}$, $y = \frac{Y}{Z^i}$

In DPC system the *x* and *y* coordinates can be projected to any degree of the above degrees and not necessarily to the same degree. In other words, *x*-coordinate can be in one degree while *y*-coordinate in another one resulting in many combinations of coordinate systems.

Projecting parameters L_x and L_y are used to define the following Dynamic Transformation Functions:

$$x = \frac{X}{Z^{L_x}} \text{ and } y = \frac{Y}{Z^{L_y}} \qquad 7.1$$

where, L_x and L_y are positive integers.

However, in any projective coordinate system, each affine point (x, y) can be converted to many projective points (X, Y, Z), one for each non-zero value of *Z*. This means that we have the freedom to select *Z*. However, *Z* should be selected in a way that clears the denominators and minimizes the computations of X_3 and Y_3. For example, consider addition operation using homogenous coordinate system in which the point $P_3 = (X_3, Y_3, Z_3)$ is the result of point addition $P_1 = (X_1, Y_1, Z_1) + P_2 = (X_2, Y_2, Z_2)$. The Z-coordinate of the result point, Z_3, is chosen to be $V^3 Z_1 Z_2$, where $V = X_2 Z_1 - X_1 Z_2$, which is the best choice to unify Z-coordinate and minimize the computations of X_3 and Y_3 (see equation 5.5 in section 5.3). Similarly, in case of addition using Jacobian coordinate system, Z_3 is chosen to be HZ_1Z_2, where $H = X_2 Z_1^2 - X_1 Z_2^2$ (equation 5.8 in section 5.4).

Therefore, in order to have a general method for choosing Z_3 in DPC, a third parameter, called *d-parameter*, is used to control choosing the *Z*-coordinate of the resulting point of addition and doubling operations. For example, Z_3 can be chosen to be $V^d Z_1 Z_2$. By setting *d=3* we get the same definition of Z_3 in homogenous coordinate

system while by setting *d=1* we get the same definition of Z_3 in Jacobian coordinate system. However, it is worth mentioning that *d* is not used to project neither *x* nor *y* coordinates. It is only used to help in choosing *Z*-coordinate of the resulting point of addition and doubling operations. Furthermore, Z_3 of addition operation and Z_3 of doubling operation are different because each operation has its own formula.

However, using the *d*-parameter in the way discussed above introduces a powerful and very efficient projective coordinates randomization method by simply randomizing *d* itself. This method is discussed in chapter 8.

7.3 Dynamic Projective Coordinate System for *E/GF(p)*

Let *E/GF(P)* denotes elliptic curve defined over the *prime* field *GF(P)* (see section 3.3 in chapter 3). By substituting for *x* and *y* from 7.1 in the elliptic curve equation 3.3, we get:

$$Y^2 Z^{3L_x - 2L_y} = X^3 + aXZ^{2L_x} + bZ^{3L_x} \qquad 7.2$$

Note that if we set $L_x = L_y = 1$ in 7.2, we get: $Y^2 Z = X^3 + aXZ^2 + bZ^3$ which is identical to the standard projective equation of the elliptic curve equation over prime field found in [4].

This equation is satisfied by all projective points with Z ≠ 0 for which the corresponding affine points satisfy the affine equation 3.3. Now the question is which points on the line at infinity satisfy equation 7.2? Setting Z = 0 in the equation leads to

$X^3 = 0$, i.e. $X = 0$. The only point with both *X* and *Z* zero is the point (0, 1, 0). This point is called the point at infinity and denoted as ∞. It is the point on the intersection of the *y*-axis with the line at infinity

Lemma 7.1: Any point $Q = (x, y) \in E/GF(p)$ represented in affine coordinates can be transferred to a *4-tuple projective point* $P = (X, Y, Z^{L_x}, Z^{L_y}) \in E/GF(p)$ where, Z^{L_x} and $Z^{L_y} \neq 0$.

Proof: Since the two values, Z^{L_x} and Z^{L_y}, are available within the 4-tuple representation of the point, the affine point (*x,y*) can be obtained by direct application of 7.1.

The following subsections present the addition and doubling formulas for *GF(p)* using DPC. However, several DPC formulas are introduced. These formulas are: General formulas in which L_x and L_y can be selected to be any positive integers without any restriction. Optimized formulas in which L_x and L_y are selected according to certain rule to reduce the number of computations required. Mixed formulas in which each coordinate of each point has its own projecting parameter.

7.3.1 General Dynamic Projective Coordinate System for *E/GF(p)*

Formulations for Elliptic curve point addition and doubling, over *GF(p)*, using DPC are presented in this section. We develop point addition mathematical formulas that can be used to implement any projective coordinate system simply by varying the projecting parameters L_x and L_y. Similarly, point doubling formulas are also presented.

However, one of the most important features of the DPC system for *E/GF(p)* is that the same mathematical formulas, either for point addition or doubling, can implement any projective coordinate system without the need to recode or reprogram the cryptodevice.

Point Addition Formula

Theorem 7.1: Given two elliptic curve points represented in DPC, $P=(X_1,Y_1,Z_1^{L_x},Z_1^{L_y}) \in E/GF(P)$, $Q=(X_2,Y_2,Z_2^{L_x},Z_2^{L_y}) \in E/GF(p)$, and denoting the point $R=(X_3,Y_3,Z_3^{L_x},Z_3^{L_y}) \in E/GF(p)$ as the addition of the two points P and Q, i.e. $R=P+Q$, the dynamic projective coordinates of the point R is given by:

$$\left.\begin{aligned}
&X_3 = X_3^{'} R^{dL_x-2} \\
&Y_3 = Y_3^{'} R^{dL_y-3} \\
&Z_3^{L_x} = R^{dL_x} T_1 \\
&Z_3^{L_y} = R^{dL_y} T_2 \\
&\textit{where},\quad U_1 = Y_2 Z_1^{L_y},\quad U_2 = Y_1 Z_2^{L_y},\quad U = U_1 - U_2, \\
&\qquad V_1 = X_2 Z_1^{L_x},\quad V_2 = X_1 Z_2^{L_x},\quad V = V_1 - V_2, \\
&\qquad T_1 = Z_1^{L_x} Z_2^{L_x},\quad T_2 = Z_1^{L_y} Z_2^{L_y},\quad R = VT_2 \\
&\qquad X_3^{'} = UT_1^3 - R^2 V_1 - R^2 V_2, \\
&\qquad Y_3^{'} = T_2\left(U(R^2 V_2 - X_3^{'}) - U_2(R^2 V_1 - R^2 V_2)\right) \\
&\qquad d \geq 3,\quad L_x > 0,\quad L_y > 0
\end{aligned}\right\} \qquad 7.3$$

Proof: According to lemma 7.1, since $P=(X_1,Y_1,Z_1^{L_x},Z_1^{L_y})$, $Q=(X_2,Y_2,Z_2^{L_x},Z_2^{L_y})$ and $R=(X_3,Y_3,Z_3^{L_x},Z_3^{L_y})$ are elliptic curve projective points $\in$ *E/GF(p)*, one can use the addition formula 3.4 for *E/GF(p)* in affine coordinates to compute $R=P+Q$ (addition

operation). The projective coordinates $(X_3, Y_3, Z_3^{L_x}, Z_3^{L_y})$ of the point *R* can be derived using the dynamic transformation functions 7.1. This is shown in appendix A-I to obtain the formulas 7.3 for computing $R = P + Q$.

Point Doubling Formula

Theorem 7.2: Given an elliptic curve point represented in DPC, $P = (X_1, Y_1, Z_1^{L_x}, Z_1^{L_y}) \in$ *E/GF(p)*, and denoting the point $R = (X_3, Y_3, Z_3^{L_x}, Z_3^{L_y}) \in$ *E/GF(p)* as the addition of the point *P* to itself, i.e. $R = 2P$, the coordinates of the point *R* is given by:

$$\left.\begin{aligned}
&X_3 = X_3' Z_1^{L_x} S^{dL_x - 2} \\
&Y_3 = Y_3' S^{dL_y - 3} \\
&Z_3^{L_x} = S^{dL_x} Z_1^{L_x} \\
&Z_3^{L_y} = S^{dL_y} Z_1^{L_y} \\
&\textit{Where}, \quad W = 3X_1^2 + aZ_1^{2L_x}, \quad S = 2Z_1^{2L_x} Y_1, \\
&\qquad T = WZ_1^{2L_y}, \quad T_1 = SY_1 Z_1^{L_x}, \quad T_2 = 2T_1 X_1 \\
&\qquad X_3' = WT - 2T_2 \\
&\qquad Y_3' = T(T_2 - X_3') - 2T_1^2 \\
&\qquad d \geq 3, \quad L_x > 0, \quad L_y > 0
\end{aligned}\right\} \qquad 7.4$$

Proof: According to lemma 7.1, let $P = (X_1, Y_1, Z_1^{L_x}, Z_1^{L_y})$, and $R = (X_3, Y_3, Z_3^{L_x}, Z_3^{L_y})$ be elliptic curve projective points $\in$ *E/GF(p)*. We can use the doubling formula 3.5 for *E/GF(p)* in affine coordinates to compute $R = 2P$ (doubling operation). The projective coordinates $(X_3, Y_3, Z_3^{L_x}, Z_3^{L_y})$ of the point *R* can be derived using the dynamic transformation

functions 7.1. This is shown in appendix B-I to obtain the formulas 7.4 for computing $R = 2P$.

7.3.2 Mixed Dynamic Projective Coordinate System for *E/GF(p)*

Formulas 7.3 are obtained using *uniform* transformation in which L_x and L_y are the same for the three points *P, Q* and *R*. More general addition formulas can be obtained by using *mixed* transformation where each coordinate in each point has its own projecting parameter. In this case, projecting parameters L_{x1}, L_{y1} are used for *P*, L_{x2}, L_{y2} are used for *Q* and L_{x3}, L_{y3} are used for *R*.

Theorem 7.3: Given two elliptic curve points represented in DPC, $P = (X_1, Y_1, Z_1^{L_{x1}}, Z_1^{L_{y1}}) \in E/GF(p)$, $Q = (X_2, Y_2, Z_2^{L_{x2}}, Z_2^{L_{y2}}) \in E/GF(p)$, and denoting the point $R = (X_3, Y_3, Z_3^{L_{x3}}, Z_3^{L_{y3}}) \in E/GF(p)$ as the addition of the two points *P* and *Q*, i.e. $R = P + Q$, the coordinates of the point *R* is given by:

$$
\left.
\begin{aligned}
&X_3 = X_3' R^{dL_{x3}-2} T_1^{L_{x3}-1} \\
&Y_3 = Y_3' R^{dL_{y3}-3} T_1^{L_{y3}-1} \\
&Z_3^{L_{x3}} = R^{dL_{x3}} T_1^{L_{x3}} \\
&Z_3^{L_{y3}} = R^{dL_{y3}} T_1^{L_{y3}} \\
&\textit{where}, \quad U_1 = Y_2 Z_1^{L_{y1}}, \quad U_2 = Y_1 Z_2^{L_{y2}}, \quad U = U_1 - U_2, \\
&\qquad V_1 = X_2 Z_1^{L_{x1}}, \quad V_2 = X_1 Z_2^{L_{x2}}, \quad V = V_1 - V_2, \\
&\qquad T_1 = Z_1^{L_{x1}} Z_2^{L_{x2}}, \quad T_2 = Z_1^{L_{y1}} Z_2^{L_{y2}}, \quad R = VT_2 \\
&\qquad X_3' = U^2 T_1^3 - R^2 V_1 - R^2 V_2, \\
&\qquad Y_3' = T_1 \left(U(R^2 V_2 - X_3') - U_2 (R^2 V_1 - R^2 V_2) \right) \\
&\qquad d \geq 3, \quad L_{x3} > 0, \quad L_{y3} > 0
\end{aligned}
\right\} \qquad 7.5
$$

Proof: The proof of Theorem 7.3 is similar to the proof of Theorem 7.1 with replacing each $Z_i^{L_x}$ by $Z_i^{L_{xi}}$ and each $Z_i^{L_y}$ by $Z_i^{L_{yi}}$ where, *i = 1,2*.

Formulas 7.4 are obtained using *uniform* transformation functions in which L_x and L_y are the same for *P* and *R*. More general addition formulas can be obtained by using *mixed* transformation, where different projecting parameters for each point, i.e. L_{x1}, L_{y1} for *P* and L_{x3}, L_{y3} for *R*.

Theorem 7.4: Given an elliptic curve point represented in DPC, $P=(X_1,Y_1,Z_1^{L_{x1}},Z_1^{L_{y1}}) \in E/GF(p)$, and denoting the point $R=(X_3,Y_3,Z_3^{L_{x3}},Z_3^{L_{y3}}) \in E/GF(p)$ as the addition of the point *P* to itself, i.e. $R=2P$, the coordinates of the point *R* is given by:

$$
\left.
\begin{aligned}
&X_3 = X_3' S^{dL_{x3}-2} (Z_1^{L_{y1}})^{L_{x3}-1} \\
&Y_3 = Y_3' S^{dL_{y3}-3} (Z_1^{L_{y1}})^{L_{y3}-1} \\
&Z_3^{L_{x3}} = S^{dL_{x3}} (Z_1^{L_{y1}})^{L_{x3}} \\
&Z_3^{L_{y3}} = S^{dL_{y3}} (Z_1^{L_{y1}})^{L_{y3}} \\
&\textit{Where}, \quad W = 3X_1^2 + aZ_1^{2L_{x1}}, \quad S = 2Z_1^{2L_{x1}} Y_1, \\
&\qquad T = WZ_1^{2L_{y1}}, \quad T_1 = SY_1 Z_1^{L_{x1}}, \quad T_2 = 2T_1 X_1 \\
&\quad X_3' = T^2 - 2T_2 \\
&\quad Y_3' = T(T_2 Z_1^{L_{y1}} - X_3') - 2T_1^2 \\
&\quad d \geq 3, \quad L_{x3} > 0, \quad L_{y3} > 0
\end{aligned}
\right\} \qquad 7.6
$$

Proof: The proof of Theorem 7.4 is similar to the proof of Theorem 7.2 with replacing each $Z_i^{L_x}$ by $Z_i^{L_{xi}}$ and each $Z_i^{L_y}$ by $Z_i^{L_{yi}}$ where, *i = 1,2*.

7.3.3 Optimized Dynamic Projective Coordinate System for *E/GF(p)*

Addition and doubling formulas 7.3 and 7.4 are the most general homogenous formulas for *E/GF(p)* without any restriction on the values of the projecting parameters L_x and L_y. However, their computation complexity can be reduced by reproducing these formulas with taking Z_1 and Z_2 as common factors in each equation (whenever it is possible) and simplify the resultant formulas by eliminating the unnecessary terms. This results in the existence of terms such as $Z_1^{L_x - L_y}$, in which its power is a relation between L_x and L_y. Existence of such terms requires providing either pure *Z-coordinate* (i.e. not raised to any power) or the required term as a ready computed value in the point representation. This can be achieved with the help of the following lemma.

Lemma 7.2: Any point $Q = (x, y)$ *E/GF(p)* represented in affine coordinates can be transferred to a *5-tuple projective point* $P = (X, Y, Z, Z^{L_x}, Z^{L_y})$ *E/GF(p)* where, Z, Z^{L_x} and $Z^{L_y} \neq 0$.

Proof: Since the values, Z, Z^{L_x} and Z^{L_y}, are available within the 5-tuple representation of the point, proof follows directly from 7.1.

Appendixes C-I and D-I present the derivation of optimized addition and doubling formulas respectively. The optimized addition formulas are:

$$
\left.
\begin{aligned}
&X_3 = X_3' V^{dL_x-2} \\
&Y_3 = Y_3' V^{dL_y-3} \\
&Z_3 = V^d T \\
&Z_3^{L_x} = (V^d T)^{L_x} \\
&Z_3^{L_y} = (V^d T)^{L_y} \\
&\textit{where}, \quad U_1 = Y_2 Z_1^{L_y}, \quad U_2 = Y_1 Z_2^{L_y}, \quad U = U_1 - U_2, \\
&\qquad V_1 = X_2 Z_1^{L_x}, \quad V_2 = X_1 Z_2^{L_x}, \quad V = V_1 - V_2, \\
&\qquad T = Z_1 Z_2, \quad T_1 = V T^{L_y - L_x} \\
&\qquad X_3' = U^2 T^{3L_x - 2L_y} - V^3 - 2V^2 V_2, \\
&\qquad Y_3' = U(V^2 V_2 - X_3') - U_2 V^3 \\
&L_y - L_x \geq 0, \quad 3L_x - 2L_y \geq 0, \quad dL_x - 2 \geq 0, \quad dL_y - 3 \geq 0
\end{aligned}
\right\} \qquad 7.7
$$

and the optimized doubling formulas are:

$$
\left.
\begin{aligned}
&X_3 = X_3' S^{dL_x-2} \\
&Y_3 = Y_3' S^{dL_y-3} \\
&Z_3 = S^d \\
&Z_3^{L_x} = (S^d)^{L_x} \\
&Z_3^{L_y} = (S^d)^{L_y} \\
&\textit{Where}, \quad W = 3X_1^2 + aZ_1^{2L_x}, \quad S = 2Z_1^{L_x} Y_1, \\
&\qquad T = W Z_1^{L_y - L_x}, \quad T_1 = S Y_1, \quad T_2 = 2T_1 X_1, \\
&\qquad X_3' = T^2 - 2T_2 \\
&\qquad Y_3' = T(T_2 - X_3') - 4T_1 Y_1^2 Z_1^{2L_x - L_y} \\
&L_y - L_x \geq 0, \quad 2L_x - L_y \geq 0, \quad dL_x - 2 \geq 0, \quad dL_y - 3 \geq 0
\end{aligned}
\right\} \qquad 7.8
$$

Formulas 7.7 and 7.8 are obtained using *uniform* transformation functions. Similar *mixed* optimized formulas can be obtained using the same way as in appendixes C-I and D-I with replacing each $Z_i^{L_x}$ by $Z_i^{L_{xi}}$ and each $Z_i^{L_y}$ by $Z_i^{L_{yi}}$ where, *i = 1,2*. The *mixed* optimized addition formulas are:

$$
\left.
\begin{aligned}
&X_3 = X_3^{'} R^{dL_{x3}-2} \\
&Y_3 = Y_3^{'} R^{dL_{y3}-3} \\
&Z_3 = R^d \\
&Z_3^{L_{x3}} = (R^d)^{L_{x3}} \\
&Z_3^{L_{y3}} = (R^d)^{L_{y3}} \\
&\textit{where}, \quad U_1 = Y_2 Z_1^{L_{y1}}, \quad U_2 = Y_1 Z_2^{L_{y2}}, \quad U = U_1 - U_2, \\
&\qquad V_1 = X_2 Z_1^{L_{x1}}, \quad V_2 = X_1 Z_2^{L_{x2}}, \quad V = V_1 - V_2, \\
&\qquad T_1 = Z_1^{L_{x1}} Z_2^{L_{x2}}, \quad T_2 = Z_1^{L_{y1}} Z_2^{L_{y2}}, \quad R = VT_2, \quad T_3 = UT_1 \\
&\qquad X_3^{'} = T_3^2 - VR(Z_1^{L_{y1}-L_{x1}} Z_2^{L_{y2}-L_{x2}})(V_1 + V_2), \\
&\qquad Y_3^{'} = R^2(UV_2 - VU_2) - X_3^{'} T_3 \\
&\quad L_{y1} - L_{x1} \geq 0, \quad L_{y2} - L_{x2} \geq 0, \quad dL_{x3} - 2 \geq 0, \quad dL_{y3} - 3 \geq 0
\end{aligned}
\right\} \qquad 7.9
$$

and the *mixed* optimized doubling formulas are:

$$
\left.
\begin{aligned}
&X_3 = X_3^{'} S^{dL_{x3}-2} \\
&Y_3 = Y_3^{'} S^{dL_{y3}-3} \\
&Z_3 = S^d \\
&Z_3^{L_{x3}} = (S^d)^{L_{x3}} \\
&Z_3^{L_{y3}} = (S^d)^{L_{y3}} \\
&\textit{Where}, \quad W = 3X_1^2 + aZ_1^{2L_{x1}}, \quad S = 2Z_1^{L_{x1}} Y_1, \quad T = WZ_1^{L_{y1}-L_{x1}} \\
&\qquad X_3^{'} = T^2 - 4SY_1 X_1 \\
&\qquad Y_3^{'} = T(2SY_1 X_1 - X_3^{'}) - 4(SY_1) Y_1^2 Z_1^{2L_{x1}-L_{y1}} \\
&\qquad L_{y1} - L_{x1} \geq 0, \quad 2L_{x1} - L_{y1} \geq 0, \quad dL_{x3} - 2 \geq 0, \quad dL_{y3} - 3 \geq 0
\end{aligned}
\right\} \qquad 7.10
$$

7.4 Dynamic Projective Coordinate System for *E/GF(2^m)*

Dynamic Projective Coordinate system can be used to get addition and doubling formulas, similar to those obtained in section 7.4, in case of defining ECC over the *binary* field $GF(2^m)$.

Transformation functions 7.1 are used to formulate the DPC in $E/GF(2^m)$. By substituting for *x* and *y* from 7.1 in the elliptic curve equation 3.6, we get:

$$Y^2 Z^{3L_x - 2L_y} + XYZ^{2L_x - L_y} = X^3 + aX^2 Z^{L_x} + bZ^{3L_x} \qquad 7.11$$

Note that if we set $L_x = L_y = 1$ in 7.11, we get: $Y^2 Z + XYZ = X^3 + aX^2 Z + bZ^3$ which is identical to the standard projective form of the elliptic curve equation over binary field found in [52]. Also, If $Z = 0$, then $X^3 = 0$, i.e. $X = 0$. Therefore, (0,1,0) is the only projective point that satisfies this equation. This point is called the point at infinity and denoted as ∞.

Lemma 7.3: Any point $Q = (x, y) \in E/GF(2^m)$ represented in affine coordinates can be transferred to a *4-tuple projective point* $P = (X, Y, Z^{L_x}, Z^{L_y}) \in E/GF(2^m)$ where, Z^{L_x} and $Z^{L_y} \neq 0$.

Proof: Since the two values, Z^{L_x} and Z^{L_y}, is available within the 4-tuple representation of the point, proof follows directly from 7.1.

7.4.1 General Dynamic Projective Coordinate System for *E/GF(2^m)*

Formulations for Elliptic curve point addition and doubling, over $GF(2^m)$, using DPC are presented in this section. We develop point addition mathematical formulas that can be used to implement any projective coordinate system simply by varying the projecting parameters L_x and L_y. Similarly, point doubling formulas are also presented. However, one of the most important features of the DPC system for $E/GF(2^m)$ is that the same mathematical formulas, either for point addition or doubling, can implement any projective coordinate system without the need to recode or reprogram the cryptodevice.

Point Addition Formula

Theorem 7.5: Given two elliptic curve points represented in DPC, $P=(X_1,Y_1,Z_1^{L_x},Z_1^{L_y}) \in E/GF(2^m)$, $Q=(X_2,Y_2,Z_2^{L_x},Z_2^{L_y})$ $E/GF(2^m)$, and denoting the point $R=(X_3,Y_3,Z_3^{L_x},Z_3^{L_y}) \in E/GF(2^m)$ as the addition of the two points *P* and *Q*, i.e. $R=P+Q$, the coordinates of the point *R* is given by:

$$
\left.
\begin{aligned}
&X_3 = X_3' R^{dL_x-2} T^{L_x-1} \\
&Y_3 = Y_3' R^{dL_y-3} T^{L_y-1} \\
&Z_3^{L_x} = (R^d T)^{L_x} \\
&Z_3^{L_y} = (R^d T)^{L_y} \\
&\textit{where}, \quad U_1 = Y_2 Z_1^{L_y}, \quad U_2 = Y_1 Z_2^{L_y}, \quad U = U_1 + U_2, \\
&\qquad V_1 = X_2 Z_1^{L_x}, \quad V_2 = X_1 Z_2^{L_x}, \quad V = V_1 + V_2, \\
&\qquad R = V Z_1^{L_y} Z_2^{L_y}, \quad T = Z_1^{L_x} Z_2^{L_x}, \quad T_1 = UT, \\
&\qquad X_3' = T_1 T (T_1 + R) + R^2 (V + aT), \\
&\qquad Y_3' = T_2 (UV_2 + VU_2) + X_3' (T_1 + R) \\
&\qquad d \geq 3, \quad L_x > 0, \quad L_y > 0
\end{aligned}
\right\} \qquad 7.12
$$

Proof: According to lemma 7.3, since $P = (X_1, Y_1, Z_1^{L_x}, Z_1^{L_y})$, $Q = (X_2, Y_2, Z_2^{L_x}, Z_2^{L_y})$ and $R = (X_3, Y_3, Z_3^{L_x}, Z_3^{L_y})$ are elliptic curve projective points $\in E/GF(2^m)$, one can use the addition formula 3.7 for $E/GF(2^m)$ in affine coordinates to compute $R = P + Q$ (addition operation). The dynamic projective coordinates $(X_3, Y_3, Z_3^{L_x}, Z_3^{L_y})$ of the point R can be derived using the dynamic transformation functions 7.1. This is shown in appendix A-II to obtain the formulas in equation 7.12 for computing $R = P + Q$.

Point Doubling Formula

Theorem 7.6: Given an elliptic curve point represented in DPC, $P = (X_1, Y_1, Z_1^{L_x}, Z_1^{L_y}) \in E/GF(2^m)$, and denoting the point $R = (X_3, Y_3, Z_3^{L_x}, Z_3^{L_y}) \in E/GF(2^m)$ as the addition of the point P to itself, i.e. $R = 2P$, the coordinates of the point R is given by:

$$
\left.
\begin{aligned}
&X_3 = X_3^{'} S^{dL_x - 2} \\
&Y_3 = Y_3^{'} S^{dL_y - 3} \\
&Z_3^{L_x} = \left(S^d\right)^{L_x} \\
&Z_3^{L_y} = \left(S^d\right)^{L_y} \\
&\textit{Where}, \quad T_1 = X_1^2 Z_1^{L_y}, \quad T_2 = Y_1 Z_1^{2L_x}, \quad W = T_1 + T_2, \quad S = X_1 Z_1^{L_x} Z_1^{L_y} \\
&\qquad X_3^{'} = W\left(W + S\right) + aS^2 \\
&\qquad Y_3^{'} = ST_1\left(W + T_2\right) + X_3^{'}\left(W + S\right) \\
&\qquad d \geq 3, \quad L_x > 0, \quad L_y > 0
\end{aligned}
\right\} \qquad 7.13
$$

Proof: According to lemma 7.3, let $P = (X_1, Y_1, Z_1^{L_x}, Z_1^{L_y})$, and $R = (X_3, Y_3, Z_3^{L_x}, Z_3^{L_y})$ be elliptic curve projective points $\in E/GF(2^m)$. We can use the doubling formula 3.8 for $E/GF(2^m)$ in affine coordinates to compute $R = 2P$ (doubling operation). The dynamic projective coordinates $(X_3, Y_3, Z_3^{L_x}, Z_3^{L_y})$ of the point R can be derived using the dynamic transformation functions 7.1. This is shown in appendix B-II to obtain the formulas in equation 7.13 for computing $R = 2P$.

7.4.2 Mixed Dynamic Projective Coordinate System for *E/GF(2^m)*

Formulas 7.12 are obtained using *uniform* transformation in which L_x and L_y are the same for the three points *P, Q* and *R*. More general addition formulas can be obtained by using *mixed* transformation where each coordinate in each point has its own projecting parameter. In this case, projecting parameters L_{x1}, L_{y1} are used for *P*, L_{x2}, L_{y2} are used for *Q* and L_{x3}, L_{y3} are used for *R*.

Theorem 7.7: Given two elliptic curve points represented in DPC, $P=(X_1,Y_1,Z_1^{L_{x1}},Z_1^{L_{y1}})\in E/GF(2^m)$, $Q=(X_2,Y_2,Z_2^{L_{x2}},Z_2^{L_{y2}})\in E/GF(2^m)$, and denoting the point $R=(X_3,Y_3,Z_3^{L_{x3}},Z_3^{L_{y3}})\in E/GF(2^m)$ as the addition of the two points *P* and *Q*, i.e. $R=P+Q$, the coordinates of the point *R* is given by:

$$\left.\begin{aligned}
&X_3=X_3^{'}R^{dL_{x3}-2}T^{L_{x3}-1}\\
&Y_3=Y_3^{'}R^{dL_{y3}-3}T^{L_{y3}-1}\\
&Z_3^{L_x}=(R^dT)^{L_{x3}}\\
&Z_3^{L_y}=(R^dT)^{L_{y3}}\\
&\textit{where},\quad U_1=Y_2Z_1^{L_{y1}},\quad U_2=Y_1Z_2^{L_{y2}},\quad U=U_1+U_2,\\
&\qquad V_1=X_2Z_1^{L_{x1}},\quad V_2=X_1Z_2^{L_{x2}},\quad V=V_1+V_2,\\
&\qquad R=VZ_1^{L_{y1}}Z_2^{L_{y2}},\quad T=Z_1^{L_{x1}}Z_2^{L_{x2}},\quad T_1=UT,\\
&\qquad X_3^{'}=T_1T(T_1+R)+R^2(V+aT),\\
&\qquad Y_3^{'}=T_2(UV_2+VU_2)+X_3^{'}(T_1+R)\\
&\qquad d\geq 3,\quad L_{x3}>0,\quad L_{y3}>0
\end{aligned}\right\} \qquad 7.14$$

Proof: The proof of Theorem 7.7 is similar to the proof of Theorem 7.5 with replacing each $Z_i^{L_x}$ by $Z_i^{L_{xi}}$ and each $Z_i^{L_y}$ by $Z_i^{L_{yi}}$ where, *i = 1,2*.

Formulas 7.13 are obtained using *uniform* transformation in which L_x and L_y are the same for *P* and *R*. More general doubling formulas can be obtained by using *mixed* transformation. In this case, projecting parameters L_{x1},L_{y1} are used for *P*, and L_{x3},L_{y3} are used for *R*.

Theorem 7.8: Given an elliptic curve point represented in DPC, $P=(X_1,Y_1,Z_1^{L_{x1}},Z_1^{L_{y1}})\in E/GF(2^m)$, and denoting the point $R=(X_3,Y_3,Z_3^{L_{x3}},Z_3^{L_{y3}})\in E/GF(2^m)$ as the addition of the point P to itself, i.e. $R=2P$, the coordinates of the point R is given by:

$$\left.\begin{aligned}
&X_3=X_3^{'}S^{dL_{x3}-2}\\
&Y_3=Y_3^{'}S^{dL_{y3}-3}\\
&Z_3^{L_x}=\left(S^d\right)^{L_{x3}}\\
&Z_3^{L_y}=\left(S^d\right)^{L_{y3}}\\
&Where,\quad T_1=X_1^2Z_1^{L_{y1}},\quad T_2=Y_1Z_1^{2L_{x1}},\quad W=T_1+T_2,\quad S=X_1Z_1^{L_{x1}}Z_1^{L_{y1}}\\
&\qquad X_3^{'}=W\left(W+S\right)+aS^2\\
&\qquad Y_3^{'}=ST_1\left(W+T_2\right)+X_3^{'}\left(W+S\right)\\
&\qquad d\geq 3,\quad L_{x3}>0,\quad L_{y3}>0
\end{aligned}\right\}\qquad 7.15$$

Proof: The proof of Theorem 7.8 is similar to the proof of Theorem 7.6 and is omitted here for space limitations.

7.4.3 Optimized Dynamic Projective Coordinate System for *E/GF(2^m)*

Addition and doubling formulas 7.12 and 7.13 are the most general homogenous formulas for $E/GF(2^m)$ without any restriction in the values of the projecting parameters L_x and L_y. However, their computation complexity can be reduced by reproducing these formulas with taking Z_1 and Z_2 as common factors in each equation (whenever it is possible) and simplify the resultant formulas by eliminating the unnecessary terms. This results in the existence of terms such as $Z_1^{L_x-L_y}$, in which its power is a relation between

L_x and L_y. Existence of such terms requires providing either pure *Z-coordinate* (i.e. not raised to any power) or the required term as a ready computed value in the point representation. This can be achieved with the help of the following lemma.

Lemma 7.4: Any point $Q=(x,y) \in E/GF(2^m)$ represented in affine coordinates can be transferred to a *5-tuple projective point* $P=(X,Y,Z,Z^{L_x},Z^{L_y}) \in E/GF(2^m)$ where, Z, Z^{L_x} and $Z^{L_y} \neq 0$.

Proof: Since the values, Z, Z^{L_x} and Z^{L_y}, are available within the 5-tuple representation of the point, proof is follows directly from 7.1.

Appendixes C-II and D-II present the derivation of optimized addition and doubling formulas respectively. The optimized addition formulas are:

$$\left.\begin{aligned}
&X_3 = X_3' V^{dL_x-2}\\
&Y_3 = Y_3' V^{dL_y-3}\\
&Z_3 = V^d T\\
&Z_3^{L_x} = (V^d T)^{L_x}\\
&Z_3^{L_y} = (V^d T)^{L_y}\\
&\textit{where},\quad U_1 = Y_2 Z_1^{L_y},\quad U_2 = Y_1 Z_2^{L_y},\quad U = U_1 + U_1,\\
&\qquad V_1 = X_2 Z_1^{L_x},\quad V_2 = X_1 Z_2^{L_x},\quad V = V_1 + V_1,\\
&\qquad T = Z_1 Z_2,\quad T_1 = V T^{L_y - L_x}\\
&\qquad X_3' = U T^{3L_x - 2L_y}(U + T_1) + aV^2 T^{L_x} + V^3\\
&\qquad Y_3' = U(V^2 V_2 + X_3') + V^3 U_2 + X_3' T_1\\
&\qquad L_y - L_x \geq 0,\quad 3L_x - 2L_y \geq 0,\quad dL_x - 2 \geq 0,\quad dL_y - 3 \geq 0
\end{aligned}\right\} \qquad 7.16$$

and the optimized doubling formulas are:

$$
\left.
\begin{aligned}
&X_3 = X_3^{'} S^{dL_x - 2} \\
&Y_3 = Y_3^{'} S^{dL_y - 3} \\
&Z_3 = S^d \\
&Z_3^{L_x} = \left(S^d\right)^{L_x} \\
&Z_3^{L_y} = \left(S^d\right)^{L_y} \\
&\textit{Where}, \quad T = Y_1 Z_1^{2L_x - L_y}, \quad W = X_1^2 + T, \quad S = X_1 Z_1^{L_x} \\
&\qquad X_3^{'} = W\left(W + S\right) + aS^2 \\
&\qquad Y_3^{'} = SX_1^2\left(W + T\right) + X_3^{'}\left(W + S\right) \\
&\qquad 2L_x - L_y \geq 0, \quad dL_x - 2 \geq 0, \quad dL_y - 3 \geq 0
\end{aligned}
\right\} \qquad 7.17
$$

Formulas 7.16 and 7.17 are obtained using *uniform* transformation functions. Similar *mixed* optimized formulas can be obtained using the same way as in appendixes C-II and D-II with replacing each $Z_i^{L_x}$ by $Z_i^{L_{xi}}$ and each $Z_i^{L_y}$ by $Z_i^{L_{yi}}$ where, *i = 1,2*. The *mixed* optimized addition formulas are:

$$
\left.
\begin{aligned}
&X_3 = X_3^{'} R^{dL_{x3} - 2} \\
&Y_3 = Y_3^{'} \; R^{dL_{y3} - 3} \\
&Z_3 = R^d \\
&Z_3^{L_{x3}} = \left(R^d\right)^{L_{x3}} \\
&Z_3^{L_{y3}} = \left(R^d\right)^{L_{y3}} \\
&\textit{where}, \quad U_1 = Y_2 Z_1^{L_{y1}}, \quad U_2 = Y_1 Z_2^{L_{y2}}, \quad U = U_1 + U_2, \\
&\qquad V_1 = X_2 Z_1^{L_{x1}}, \quad V_2 = X_1 Z_2^{L_{x2}}, \quad V = V_1 + V_2, \\
&\qquad T_1 = Z_1^{L_{x1}} Z_2^{L_{x2}}, \quad T_2 = Z_1^{L_{y1}} Z_2^{L_{y2}}, \quad R = VT_2, \quad R_1 = V(Z_1^{L_{y1} - L_{x1}} Z_2^{L_{y2} - L_{x2}}) \\
&\qquad X_3^{'} = UT_1^2\left(U + R_1\right) + RR_1\left(V + aT_1\right) \\
&\qquad Y_3^{''} = R^2\left(UV_2 + VU_2\right) + X_3^{'}\left(UT_1 + VT_2\right) \\
&\qquad L_{y1} - L_{x1} \geq 0, \quad L_{y2} - L_{x2} \geq 0, \quad dL_{x3} - 2 \geq 0, \quad dL_{y3} - 3 \geq 0
\end{aligned}
\right\} \qquad 7.18
$$

and the *mixed* optimized doubling formulas are:

$$
\left.
\begin{aligned}
&X_3 = X_3^{'} S^{dL_{x3}-2} \\
&Y_3 = Y_3^{'} S^{dL_{y3}-3} \\
&Z_3 = S^d \\
&Z_3^{L_{x3}} = \left(S^d\right)^{L_{x3}} \\
&Z_3^{L_{y3}} = \left(S^d\right)^{L_{y3}} \\
&\textit{Where}, \quad T = Y_1 Z_1^{2L_{x1}-L_{y1}}, \quad W = X_1^2 + T, \quad S = X_1 Z_1^{L_{x1}} \\
&\qquad X_3^{'} = W\left(W+S\right) + aS^2 \\
&\qquad Y_3^{'} = SX_1^2\left(W+T\right) + X_3^{'}\left(W+S\right) \\
&\qquad 2L_{x1} - L_{y1} \geq 0, \quad dL_{x3} - 2 \geq 0, \quad dL_{y3} - 3 \geq 0
\end{aligned}
\right\} \qquad 7.19
$$

7.5 Conclusions

In this chapter, a new approach called ***Dynamic Projective Coordinate (DPC) system*** was presented. In DPC, we first proposed a general transformation functions that can be used to project *x* and *y* coordinates of any point to any projective coordinates. Then these transformation functions are used to derive dynamic addition and doubling formulas for both *E/GF(p)* and *E/GF(2^m)*. However, three types of formulas for both addition and doubling operations were presented. First, general formulas in which there is no constraints on the projecting parameters L_x and L_y with $d \geq 3$. Second, optimized formulas that reduce the number of required computations by selecting projecting parameters according to certain rules. Third, mixed formulas in which each coordinate can be projected using its own projecting parameter resulting in the most mixing degree of coordinates ever. By this way, coordinates of the same point can be represented in

different coordinate systems. The detailed steps for deriving each type of these formula are presented in appendices.

The resulting DPC allows the computing/encrypting device to select the projective coordinate either at random, or according to a certain rule. Therefore, DPC automates the selection of the projective coordinate system and uses a single mathematical formulation/software code to implement different projective coordinate systems.

CHAPTER 8

Performance and Using of DPC

8.1 Introduction

We mean by performance of DPC system is the number of required field arithmetic operations (computations) for addition and doubling operations. The less the number of required computations the faster the system we get. As in [23]-[25], for simplicity, we neglect addition, subtraction and multiplication by a small constant because they are much faster than multiplication and inversion operations.

To analyze the performance of DPC, we have to compute the number of field operations in each formula of the formulas presented in chapter 7. Therefore, a method for computing the number of computations in a dynamic formula is required. In this chapter we provide such a method that can determine the number of computations as a function of the projecting parameters Z^{L_x} and Z^{L_y} and d parameter.

As shown in chapter 7, the conventional homogenous and Jacobian coordinate systems are special cases of DPC. Hence, by selecting the appropriate Z^{L_x} and Z^{L_y} and d

parameters, we compare the DPC with these coordinate systems. Moreover, Mixed DPC system is compared with the mixed coordinates (section 5.7 in chapter5).

The rest of this chapter is organized as follows. Section 8.2 presents a method of computing the number of field operations that can be applied for both *E/GF(p)* and *E/GF(2^m)*. The performance of DPC in *E/GF(p)* and in *E/GF(2^m)* is discussed in sections 8.3 and 8.4 respectively. Using DPC is addressed in section 8.5. Finally, conclusions are given in section 8.6

8.2 Calculating the Number of Field Operations in DPC System

To calculate the number of field operations in any DPC formula of the addition and doubling formulas presented in chapter 7, the following points should be noticed.

- *First*, the number of field operations in a DPC formula consists of two parts. Part1 is a constant number of operations that must be performed regardless of the values of L_x, L_y and d. Examples of part1 are the field operations required to compute the auxiliary variables U and V in all addition formulas (i.e. formulas 7.3, 7.5, 7.7, 7.9, 7.12, 7.14, 7.16 and 7.18) and compute the auxiliary variables W and S in all doubling formulas (i.e. formulas 7.4, 7.6, 7.8, 7.10, 7.13, 7.15, 7.17 and 7.19). Part2 is the number of field operations required to compute the terms that are raised to some powers and these powers are functions of L_x, L_y and d. Examples of part2 are the field operations required to compute $Z_3^{L_x}$ and $Z_3^{L_y}$ in all formulas.

- *Second*, the total number of field operations in any formula is the summation of part1 and part2. Hence the total number of field operations is a function of L_x, L_y and *d* even of a part of it is a constant number.

Let $\alpha(T,B)$ and $\beta(T,B)$ be two functions that calculate, respectively, the number of multiplication and squaring operations required to raise some term, *T*, to the power of *B*. Then these *alpha* and *beta* functions are used to determine part2 of the total number of required field operations in any formula. Let the binary representation of *B* is: $B = b_{l-1}2^{l-1} + b_{l-2}2^{l-2} + ... + b_1 2 + b_0$ with *l* bit length. Then the average number of ones in *B* is *l/2*. Hence, according to the square and multiply method, the average values (E) of $\alpha(T,B)$ and $\beta(T,B)$ are given by:

E($\alpha(T,B)$) = *l/2* multiplications; E($\beta(T,B)$) = *l* squaring. 8.1

However, without loss of generality, L_x and L_y can be selected in a way that minimizes part2 computations such as selecting them to be powers of 2. In this case, part2 computations become squaring only which are faster than multiplications.

In the following, we present a full example of how the number of field operations are calculated in a DPC formula. Consider the optimized addition formula 7.7. The number of field operations in this formula is computed as in table 8.1:

Table 8. 1: Number of field operations in addition formula 7.7

Term	# of Multiplications (M)	# of Squaring (S)
U	2	
V	2	
T	1	
T_1	$\begin{cases}0 & if\ L_y - L_x = 0\\ 1+\alpha(T, L_y - L_x) & otherwise\end{cases}$	$\begin{cases}0 & if\ L_y - L_x = 0\\ \beta(T, L_y - L_x) & otherwise\end{cases}$
X_3'	$2+\begin{cases}0 & if\ 3L_x - 2L_y = 0\\ 1+\alpha(T, 3L_x - 2L_y) & otherwise\end{cases}$	$2+\begin{cases}0 & if\ 3L_x - 2L_y = 0\\ \beta(T, 3L_x - 2L_y) & otherwise\end{cases}$
Y_3'	2	
X_3	$\begin{cases}0 & if\ dL_x - 2 = 0\\ 1+\alpha(V, dL_x - 2) & otherwise\end{cases}$	$\begin{cases}0 & if\ dL_x - 2 = 0\\ \beta(V, dL_x - 2) & otherwise\end{cases}$
Y_3	$\begin{cases}0 & if\ dL_y - 3 = 0\\ 1+\alpha(V, dL_y - 3) & otherwise\end{cases}$	$\begin{cases}0 & if\ dL_y - 3 = 0\\ \beta(V, dL_y - 3) & otherwise\end{cases}$
$Z_3^{L_x}$ $Z_3^{L_y}$	$1+\max(\alpha(V^d T, L_x), \alpha(V^d T, L_y)$	$\max(\beta(V^d T, L_x), \beta(V^d T, L_y)$
Totals	$10+\begin{cases}0 & if\ L_y - L_x = 0\\ 1+\alpha(T, L_y - L_x) & otherwise\end{cases}$ $+\begin{cases}0 & if\ 3L_x - 2L_y = 0\\ 1+\alpha(T, 3L_x - 2L_y) & otherwise\end{cases}$ $+\begin{cases}0 & if\ dL_x - 2 = 0\\ 1+\alpha(V, dL_x - 2) & otherwise\end{cases}$ $+\begin{cases}0 & if\ dL_y - 3 = 0\\ 1+\alpha(V, dL_y - 3) & otherwise\end{cases}$ $+\max(\alpha(V^d T, L_x), \alpha(V^d T, L_y)$	$2+\begin{cases}0 & if\ L_y - L_x = 0\\ \beta(T, L_y - L_x) & otherwise\end{cases}$ $+\begin{cases}0 & if\ 3L_x - 2L_y = 0\\ \beta(T, 3L_x - 2L_y) & otherwise\end{cases}$ $+\begin{cases}0 & if\ dL_x - 2 = 0\\ \beta(V, dL_x - 2) & otherwise\end{cases}$ $+\begin{cases}0 & if\ dL_y - 3 = 0\\ \beta(V, dL_y - 3) & otherwise\end{cases}$ $+\max(\beta(V^d T, L_x), \beta(V^d T, L_y)$

By setting $L_x = L_y = 1$ and $d = 3$ we get a total number of computations equals to 2+2+1+0+3+2+1+0+1 = 12M and 0+0+0+0+2+0+0+0+0 = 2S which is identical to the number of computations in homogenous coordinate system (section 5.3).

The number of computations in other DPC formulas are computed in the same way discussed above. However, it is important to mention that the above method is applied in both cases when using DPC for *E/GF(p)* and for *E/GF(2^m)*.

8.3 Performance of DPC for *E/GF(p)*

As presented in chapter 7, there are several DPC formulas for *E/GF(p)* for addition and doubling operations. These formulas range from general formulas in which no constraints in selecting L_x and L_y (with $d \geq 3$) to formulas that can be used according to certain selection rules of L_x and L_y such as $3L_x - 2L_y \geq 0$ or $L_y - L_x \geq 0$. However, if the main goal is enhancing the performance, then clever selection of L_x, L_y and d can reduce the number of computations dramatically.

Tables 8.2 and 8.3 show the computation times in terms of the required number of multiplication and squaring operations for addition and doubling operations respectively.

Table 8. 2: Computation times for DPC addition operation in *E/GF(p)*. $a \in (0,1)$

Formula	**Multiplications (M)**	**Squaring (S)**
General–Uniform $d \geq 3$	$18 + \max\left(\alpha(R^d, L_x), \alpha(R^d, L_y)\right)$	$2 + \max\left(\beta(R^d, L_x), \beta(R^d, L_y)\right)$
General–Mixed $d \geq 3$	$18 + \max\left(\alpha(R^d, L_{x3}), \alpha(R^d, L_{y3})\right) + \max\left(\alpha(T_1^{\,d}, L_{x3}), \alpha(T_1^{\,d}, L_{y3})\right)$	$2 + \max\left(\beta(R^d, L_{x3}), \beta(R^d, L_{y3})\right) + \max\left(\beta(T_1^{\,d}, L_{x3}), \beta(T_1^{\,d}, L_{y3})\right)$
Optimized-Uniform $d \geq 3$ $3L_x - 2L_y \geq 0$ $L_y - L_x \geq 0$	$10 + \begin{cases} 0 & \text{if } L_y - L_x = 0 \\ 1 + \alpha(T, L_y - L_x) & \text{otherwise} \end{cases} + \begin{cases} 0 & \text{if } 3L_x - 2L_y = 0 \\ 1 + \alpha(T, 3L_x - 2L_y) & \text{otherwise} \end{cases} + \begin{cases} 0 & \text{if } dL_x - 2 = 0 \\ 1 + \alpha(V, dL_x - 2) & \text{otherwise} \end{cases} + \begin{cases} 0 & \text{if } dL_y - 3 = 0 \\ 1 + \alpha(V, dL_y - 3) & \text{otherwise} \end{cases} + \max(\alpha(V^d T, L_x), \alpha(V^d T, L_y)$	$2 + \begin{cases} 0 & \text{if } L_y - L_x = 0 \\ \beta(T, L_y - L_x) & \text{otherwise} \end{cases} + \begin{cases} 0 & \text{if } 3L_x - 2L_y = 0 \\ \beta(T, 3L_x - 2L_y) & \text{otherwise} \end{cases} + \begin{cases} 0 & \text{if } dL_x - 2 = 0 \\ \beta(V, dL_x - 2) & \text{otherwise} \end{cases} + \begin{cases} 0 & \text{if } dL_y - 3 = 0 \\ \beta(V, dL_y - 3) & \text{otherwise} \end{cases} + \max(\beta(V^d T, L_x), \beta(V^d T, L_y)$
Optimized–Mixed $d \geq 3$, $L_{y1} - L_{x1} \geq 0$, $L_{y2} - L_{x2} \geq 0$	$14 + \begin{cases} 0 & \text{if } L_{y1} - L_{x1} = 0 \text{ AND } L_{y2} - L_{x2} = 0 \\ 1 + \alpha(Z_2, L_{y2} - L_{x2}) & \text{if } L_{y1} - L_{x1} = 0 \text{ AND } L_{y2} - L_{x2} > 0 \\ 1 + \alpha(Z_1, L_{y1} - L_{x1}) & \text{if } L_{y2} - L_{x2} = 0 \text{ AND } L_{y1} - L_{x1} > 0 \\ 2 + \alpha(Z_1, L_{y1} - L_{x1}) + \alpha(Z_2, L_{y2} - L_{x2}) & \text{otherwise} \end{cases} + \begin{cases} 0 & \text{if } dL_{x3} - 2 = 0 \\ 1 + \alpha(R, dL_{x3} - 2) & \text{otherwise} \end{cases} + \begin{cases} 0 & \text{if } dL_{y3} - 3 = 0 \\ 1 + \alpha(R, dL_{y3} - 3) & \text{otherwise} \end{cases} + \max(\alpha(R^d, L_x), \alpha(R^d, L_y)$	$2 + \begin{cases} 0 & \text{if } L_{y1} - L_{x1} = 0 \text{ AND } L_{y2} - L_{x2} = 0 \\ \beta(Z_2, L_{y2} - L_{x2}) & \text{if } L_{y1} - L_{x1} = 0 \text{ AND } L_{y2} - L_{x2} > 0 \\ \beta(Z_1, L_{y1} - L_{x1}) & \text{if } L_{y2} - L_{x2} = 0 \text{ AND } L_{y1} - L_{x1} > 0 \\ \beta(Z_1, L_{y1} - L_{x1}) + \beta(Z_2, L_{y2} - L_{x2}) & \text{otherwise} \end{cases} + \begin{cases} 0 & \text{if } dL_{x3} - 2 = 0 \\ \beta(R, dL_{x3} - 2) & \text{otherwise} \end{cases} + \begin{cases} 0 & \text{if } dL_{y3} - 3 = 0 \\ \beta(R, dL_{y3} - 3) & \text{otherwise} \end{cases} + \max(\beta(R^d, L_x), \beta(R^d, L_y)$
Equivalent homogenous PC $d = 3,\ L_x = L_y = 1$	13	2
Equivalent Jacobian PC $d = 1,\ L_x = 2,\quad L_y = 3$	12	3

Table 8. 3: Computation times for DPC doubling operation in *E/GF(p)*. $a \in (0,1)$

Formula	**Multiplications (M)**	**Squaring (S)**
General–Uniform $d \geq 3$	$11 + \max\left(\alpha(S^d, L_x) + \alpha(S^d, L_y)\right)$	$4 + \max\left(\beta(S^d, L_x) + \beta(S^d, L_y)\right)$
General–Mixed $d \geq 3$	$11 + \max\left(\alpha(S^d, L_{x3}) + \alpha(S^d, L_{y3})\right) + \max\left(\alpha(Z_1^{L_{y1}}, L_{x3}) + \alpha(Z_1^{L_{y1}}, L_{y3})\right)$	$5 + \max\left(\beta(S^d, L_{x3}) + \beta(S^d, L_{y3})\right) + \max\left(\beta(Z_1^{L_{y1}}, L_{x3}) + \beta(Z_1^{L_{y1}}, L_{y3})\right)$
Optimized-Uniform $d \geq 3$, $2L_x - L_y \geq 0$, $L_y - L_x \geq 0$	$5 + \begin{cases} 0 & if\ L_y - L_x = 0 \\ 1 + \alpha(Z_1, L_y - L_x) & otherwise \end{cases} + \begin{cases} 0 & if\ 2L_x - L_y = 0 \\ 1 + \alpha(Z_1, 2L_x - L_y) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_x - 2 = 0 \\ 1 + \alpha(S, dL_x - 2) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_y - 3 = 0 \\ 1 + \alpha(S, dL_y - 3) & otherwise \end{cases} + \max(\alpha(S^d, L_x), \alpha(S^d, L_y))$	$4 + \begin{cases} 0 & if\ L_y - L_x = 0 \\ 1 + \beta(Z_1, L_y - L_x) & otherwise \end{cases} + \begin{cases} 0 & if\ 2L_x - L_y = 0 \\ 1 + \beta(Z_1, 2L_x - L_y) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_x - 2 = 0 \\ 1 + \beta(S, dL_x - 2) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_y - 3 = 0 \\ 1 + \beta(S, dL_y - 3) & otherwise \end{cases} + \max(\beta(S^d, L_x), \beta(S^d, L_y))$
Optimized–Mixed $d \geq 3$, $2L_{x1} - L_{y1} \geq 0$, $L_{y1} - L_{x1} \geq 0$	$5 + \begin{cases} 0 & if\ L_{y1} - L_{x1} = 0 \\ 1 + \alpha(Z_1, L_{y1} - L_{x1}) & otherwise \end{cases} + \begin{cases} 0 & if\ 2L_{x1} - L_{y1} = 0 \\ 1 + \alpha(Z_1, 2L_{x1} - L_{y1}) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_{x3} - 2 = 0 \\ 1 + \alpha(S, dL_{x3} - 2) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_{y3} - 3 = 0 \\ 1 + \alpha(S, dL_{y3} - 3) & otherwise \end{cases} + \max(\alpha(S^d, L_{x3}), \alpha(S^d, L_{y3}))$	$4 + \begin{cases} 0 & if\ L_{y1} - L_{x1} = 0 \\ \beta(Z_1, L_{y1} - L_{x1}) & otherwise \end{cases} + \begin{cases} 0 & if\ 2L_{x1} - L_{y1} = 0 \\ \beta(Z_1, 2L_{x1} - L_{y1}) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_{x3} - 2 = 0 \\ \beta(S, dL_{x3} - 2) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_{y3} - 3 = 0 \\ \beta(S, dL_{y3} - 3) & otherwise \end{cases} + \max(\beta(S^d, L_{x3}), \beta(S^d, L_{y3}))$
Equivalent homogenous PC $d = 3\ L_x = L_y = 1$	8	5
Equivalent Jacobian PC $d = 1\ L_x = 2, L_y = 3,$	8	5
Equivalent Modified Jacobian $d = 1\ L_x = 2, L_y = 3,$	8	4

The first column specifies the DPC system used. The second and third columns specify the number of multiplications and squaring respectively. The number of required multiplications and squaring are calculated using the method introduced in section 8.2. For example, in the case of using general-uniform addition formula (the first row of table 8.2) the number of required multiplications is $18+\max\left(\alpha(R^d,L_x),\alpha(R^d,L_y)\right)$. Thus, it requires 18 multiplications (part1) plus the maximum of $\alpha(R^d,L_x)$ and $\alpha(R^d,L_y)$ (part2). Note that we need only the maximum of these two numbers because the other one (minimum) will be computed in the way while computing the maximum one.

Also, Tables 8.2 and 8.3 show the savings in the number of required operations in optimized formulas compared to the general formulas. However, for further analyzing of performance of DPC in *E/GF(p)*, we compare it with the most popular existing (conventional) coordinate systems, namely, homogenous (H), Jacobian (J), modified (M) and mixed coordinate systems. Table 8.4 shows the exact number of computations in these coordinate systems according to [23] and the corresponding equivalent systems in DPC. The first four rows show the number of computations in the conventional projective coordinates found in [23]. The second four rows present the DPC systems that are equivalent to those conventional ones. The remaining rows show some possible new mixed DPC systems that do not exist in [23]-[25]. An example of such new mixed coordinates is DPC-H_xA_yH. In this system the *x*-coordinate of the input points is represented in homogenous coordinates, the *y*-coordinate is represented in affine

coordinates and the result point is represented in homogenous coordinates. Similar other mixed systems are listed in the table with their computation times.

Table 8. 4: Comparisons of field operations using DPC in *E/GF(p)*.

Projective Coordinate (PC) System	**Addition**	**Doubling**
HHH	12M + 2S	7M + 5S
JJJ	12M + 4S	4M + 6S
MMM	13M + 6S	4M + 4S
AAJ	5M + 3S	2M + 4S
Optimized DPC (DPC-HHH) $d = 3, L_x = L_Y = 1$	12M + 2S	8M + 5S
Optimized DPC (DPC-JJJ) $d = 1, L_x = 2, L_Y = 3$	12M + 3S	8M + 5S
Optimized DPC (DPC-MMM) $d = 1, L_x = 2, L_Y = 3$	12M + 4S	8M + 4S
Mixed DPC (DPC-AAJ) $d = 1$, $L_{x1} = L_{Y1} = 0$, $L_{x2} = L_{Y2} = 0$, $L_{x3} = 2, L_{Y3} = 3$	6M + 2S	4M + 4S
Mixed DPC (DPC-AAH) $d = 3$, $L_{x1} = L_{Y1} = 0$, $L_{x2} = L_{Y2} = 0$, $L_{x3} = 1, L_{Y3} = 1$	7M + 2S	5M + 4S
Mixed DPC (DPC- A_xH_yH) $d = 3$, $L_{x1} = L_{x2} = 0$, $L_{Y1} = L_{Y2} = 1$, $L_{x3} = L_{Y3} = 1$	12M + 2S	7M + 4S
Mixed DPC(DPC- A_xJ_yJ) $d = 1$, $L_{x1} = L_{x2} = 0$, $L_{Y1} = L_{Y2} = 3$, $L_{x3} = 2, L_{Y3} = 3$	13M +3S	7M + 5S
Mixed DPC (DPC- H_xA_yH) $d = 3$, $L_{x1} = L_{x2} = 1$, $L_{Y1} = L_{Y2} = 0$, $L_{x3} = L_{Y3} = 1$	14M + 2S	10M + 4S
Mixed DPC (DPC- J_xA_yJ) $d = 1$, $L_{x1} = L_{x2} = 2$, $L_{Y1} = L_{Y2} = 0$, $L_{x3} = 2, L_{Y3} = 3$	15M + 3S	10M + 5S

(DPC-HHH = Equivalent homogenous DPC, DPC-JJJ = Equivalent Jacobian DPC DPC-MMM = Equivalent modified DPC)

By comparing the number of arithmetic operations of the existing coordinate systems and the corresponding DPC systems, table 8.4 shows that addition using DPC-HHH has the same cost as HHH. In case of Jacobian the DPC-JJJ is faster than JJJ by one square operation. Also, DPC-MMM is faster than MMM by one multiplication and 2 squaring operations.

In the case of doubling operation, HH is faster than DPC-HH by one multiplication while JJ has less multiplications and more squaring than DPC-JJ.

By using *mixed* DPC formulas for *E/GF(p)*, it is possible to hop from one coordinate system to another during the scalar multiplication without the need to perform any inversion operation. We mean by hopping is using a coordinate system in iteration *i* of the scalar multiplication and use another (desired) coordinate system in the next iteration, *i*+1. In conventional coordinate systems, hopping is achieved by first converting the resulting point of iteration *i* to the desired coordinate system and then perform the point doubling (or addition) in iteration *i*+1 using the desired coordinate system formulas. In DPC, hopping is achieved by simply setting the projecting parameters L_{x3} and L_{y3} and *d*-parameter of the resulting point of iteration *i* to the desired values by which point operations in iteration *i+1* will be performed in the desired coordinate system. In other words, hopping in DPC system is achieved by adjusting the projecting parameters L_{x3} and L_{y3} and *d*-parameter of addition and doubling formulas to the values of the desired coordinate system.

Tables 8.5 and 8.6 show the cost of hoping among a set of possible DPC systems. These tables show only the DPC systems that are equivalent to the conventional coordinate systems presented in chapter 5. Other possible coordinate systems can be obtained by using different values of L_{x3}, L_{y3} and d.

However, it should be pointed out that the affine coordinates are used only in the boundaries of the scalar multiplication (bolded areas in tables 8.5 and 8.6). i.e. the affine base point is converted to any DPC system, scalar multiplication is performed and the result is converted back to the affine coordinates. The conversion from affine to any DPC system costs nothing since Z can be initialized to 1; while conversion back to affine coordinates requires an inversion operation. Note that conversion back to affine coordinates requires an inversion operation in all coordinate systems (conventional as well as DPC) regardless of the projective coordinate system used.

Table 8. 5: Hopping cost in DPC system (*E/GF(p)* Addition operation)

From \ To	Affine	DPC-HHH	DPC-JJJ	DPC-CCC	DPC-MMM
Affine	-	-	-	-	-
DPC-HHH	***2M + I***	*16M + 2S*	*15M + 2S*	*16M + 3S*	*15M + 4S*
DPC-JJJ	***3M+S+I***	*18M + 2S*	*17M + 2S*	*17M + 2S*	*17M + 3S*
DPC-CCC	***3M+S+I***	*18M + 2S*	*17M + 2S*	*17M + 2S*	*17M + 3S*
DPC-MMM	***3M+S+I***	*18M + 2S*	*17M + 2S*	*17M + 2S*	*17M + 3S*

Table 8. 6: Hopping cost in DPC system (*E/GF(p)* Doubling operation)

From \ To	Affine	DPC-HHH	DPC-JJJ	DPC-CCC	DPC-MMM
Affine	-	-	-	-	-
DPC-HHH	***2M + I***	*8M + 5S*	*7M + 5S*	*8M + 6S*	*7M + 7S*
DPC-JJJ	***3M+S+I***	*9M + 5S*	*8M + 5S*	*8M + 5S*	*8M + 6S*
DPC-CCC	***3M+S+I***	*9M + 5S*	*8M + 5S*	*8M + 5S*	*8M + 6S*
DPC-MMM	***3M+S+I***	*9M + 4S*	*8M + 4S*	*8M + 4S*	*8M + 4S*

Tables 8.5 and 8.6 show that hoping from one DPC system to another during the scalar multiplication does not require any inversion operation. On the other hand, in conventional coordinate systems, the conversion form homogenous to Jacobian or to any Jacobian variant coordinate system (i.e. C and M) requires an inversion operation as shown in table 5.2. Same thing happens if converting from Jacobian or Jacobian variant coordinate systems to Homogenous. However, conversion among the Jacobian and Jacobian variant coordinate systems does not require inversion operation because they are actually belong to the same coordinate systems (Jacobian). In other words, they use the same transformation functions $x = X / Z^2$ and $x = X / Z^3$, and hence no need to perform the inversion operation. Also, note that table 5.2 shows only the point conversion cost and does not include the cost of addition (or doubling) operation.

Tables 8.7 and 8.8 show the number of multiplications for different values of L_x and L_y for *E/GF(p)* optimized DPC addition and doubling operations respectively. For each value of L_x there are several possible choices of L_y (second column). These choices increase as L_x increases. For example, in case of addition operation, if L_x = 1, then we have only one L_y possible value while if L_x = 10 we have six possible values of L_y. In case of doubling operation, if L_x = 1, then we have two possible values of L_y while if L_x = 5 we have six possible values of L_y. This due to the constraints caused by the relations between L_x and L_y.

Table 8. 7: Possible values of L_x and L_y for addition operation in *E/GF(p)*

L_x	Valid range of L_y	Number of multiplications
1	1	12
2	2	14
	3	15
3	3	18
	4	18
4	4	15
	5	17
	6	18
5	5	18
	6	21
	7	19
6	6	18
	7	17
	8	17
	9	16
7	7	20
	8	21
	9	20
	10	22
8	8	18
	9	19
	10	20
	11	22
	12	17
9	9	18
	10	22
	11	22
	12	20
	13	19
10	10	20
	11	21
	12	19
	13	20
	14	21
	15	21

Table 8. 8: Possible values of L_x and L_y for doubling operation in *E/GF(p)*

L_x	Valid range of L_y	Number of multiplications
1	1	7
	2	9
2	2	9
	3	11
	4	9
3	3	13
	4	13
	5	13
	6	15
4	4	10
	5	13
	6	14
	7	14
	8	11
5	5	13
	6	15
	7	15
	8	15
	9	13
	10	15

In case of addition operation (table 8.7), for a certain L_x, the best choice of L_y is the one with the minimum umber of ones in the binary representation of the terms $(L_y - L_x)$, $(3L_x - 2L_y)$, $(dL_x - 2)$, $(dL_y - 3)$ and $(\max(L_x, L_y))$. For example, if L_x = 5, then the best choice of L_y is 5 while the best choice for L_x = 10 is L_y = 12.

Similarly, in case of doubling operation (table 8.8), the best choice of L_y for a certain L_x is the one with the minimum umber of ones in the binary representation of the terms $(L_y - L_x)$, $(2L_x - L_y)$, $(dL_x - 2)$, $(dL_y - 3)$ and $(\max(L_x, L_y))$. For example, if L_x = 3, then the best choice of L_y is either 3, 4 or 5 while the best choice for L_x = 5 is L_y = 5 or 9 .

8.4 Performance of DPC for *E/GF(2^m)*

There are several DPC formulas for *E/GF(2^m)* for both addition and doubling operations. These formulas range from general formulas in which no constraints in selecting L_x and L_y (with $d \geq 3$) to formulas that can be used according to certain selection rules of L_x and L_y such as $3L_x - 2L_y \geq 0$ or $L_y - L_x \geq 0$. Again, if the main goal is enhancing the performance, then clever selection of L_x, L_y and d can reduce the number of computations dramatically.

Tables 8.9 and 8.10 show the computation times in terms of the required number of multiplication and squaring operations for addition and doubling operations respectively.

Table 8. 9: Computation times for addition in DPC/$GF(2^m)$. $a \in (0,1)$

Formula	Multiplications (M)	Squaring (S)
General–Uniform	$17+\max\left(\alpha(R^dT,L_x),\alpha(R^dT,L_y)\right)$	$1+\max\left(\beta(R^dT,L_x),\beta(R^dT,L_y)\right)$
General–Mixed	$17+\max\left(\alpha(R^dT,L_{x3}),\alpha(R^dT,L_{y3})\right)$	$1+\max\left(\beta(R^dT,L_{x3}),\beta(R^dT,L_{y3})\right)$
Optimized-Uniform $d\geq 3,\ 3L_x-2L_y\geq 0,$ $L_y-L_x\geq 0$	$12+\begin{cases}0 & if\ L_y-L_x=0\\ 1+\alpha(T,L_y-L_x) & otherwise\end{cases}$ $+\begin{cases}0 & if\ 3L_x-2L_y=0\\ 1+\alpha(T,3L_x-2L_y) & otherwise\end{cases}$ $+\begin{cases}0 & if\ dL_x-2=0\\ 1+\alpha(V,dL_x-2) & otherwise\end{cases}$ $+\begin{cases}0 & if\ dL_y-3=0\\ 1+\alpha(V,dL_y-3) & otherwise\end{cases}$ $+\max(\alpha(V^dT,L_x),\alpha(V^dT,L_y)$	$1+\begin{cases}0 & if\ L_y-L_x=0\\ \beta(T,L_y-L_x) & otherwise\end{cases}$ $+\begin{cases}0 & if\ 3L_x-2L_y=0\\ \beta(T,3L_x-2L_y) & otherwise\end{cases}$ $+\begin{cases}0 & if\ dL_x-2=0\\ \beta(V,dL_x-2) & otherwise\end{cases}$ $+\begin{cases}0 & if\ dL_y-3=0\\ \beta(V,dL_y-3) & otherwise\end{cases}$ $+\max(\beta(V^dT,L_x),\beta(V^dT,L_y)$
Optimized–Mixed $d\geq 3,\ ,\ L_y-L_x\geq 0$	$17+\begin{cases}0 & if\ L_{y1}-L_{x1}=0\\ & AND\ L_{y2}-L_{x2}=0\\ 1+\alpha(Z_2,L_{y2}-L_{x2}) & if\ L_{y1}-L_{x1}=0\\ & AND\ L_{y2}-L_{x2}>0\\ 1+\alpha(Z_1,L_{y1}-L_{x1}) & if\ L_{y2}-L_{x2}=0\\ & AND\ L_{y1}-L_{x1}>0\\ 2+\alpha(Z_1,L_{y1}-L_{x1}) & \\ \quad+\alpha(Z_2,L_{y2}-L_{x2}) & otherwise\end{cases}$ $+\begin{cases}0 & if\ dL_{x3}-2=0\\ 1+\alpha(R,dL_{x3}-2) & otherwise\end{cases}$ $+\begin{cases}0 & if\ dL_{y3}-3=0\\ 1+\alpha(R,dL_{y3}-3) & otherwise\end{cases}$ $+\max(\alpha(R^d,L_x),\alpha(R^d,L_y)$	$1+\begin{cases}0 & if\ L_{y1}-L_{x1}=0\\ & AND\ L_{y2}-L_{x2}=0\\ \beta(Z_2,L_{y2}-L_{x2}) & if\ L_{y1}-L_{x1}=0\\ & AND\ L_{y2}-L_{x2}>0\\ \beta(Z_1,L_{y1}-L_{x1}) & if\ L_{y2}-L_{x2}=0\\ & AND\ L_{y1}-L_{x1}>0\\ \beta(Z_1,L_{y1}-L_{x1}) & \\ \quad+\beta(Z_2,L_{y2}-L_{x2}) & otherwise\end{cases}$ $+\begin{cases}0 & if\ dL_{x3}-2=0\\ \beta(R,dL_{x3}-2) & otherwise\end{cases}$ $+\begin{cases}0 & if\ dL_{y3}-3=0\\ \beta(R,dL_{y3}-3) & otherwise\end{cases}$ $+\max(\beta(R^d,L_x),\beta(R^d,L_y)$
Equivalent homogenous PC $d=3,\ L_x=L_y=1$	15	2
Equivalent Jacobian PC $d=1,\ L_x=2,\ \ L_y=3$	13	2

Table 8. 10: Computation times for doubling in DPC/$GF(2^m)$. $a \in (0,1)$

Formula	Multiplications (M)	Squaring (S)
General–Uniform	$10 + \max\left(\alpha(S^d, L_x), \alpha(S^d, L_y)\right)$	$1 + \max\left(\beta(S^d, L_x), \beta(S^d, L_y)\right)$
General–Mixed	$10 + \max\left(\alpha(S^d, L_{x3}), \alpha(S^d, L_{y3})\right)$	$1 + \max\left(\beta(S^d, L_{x3}), \beta(S^d, L_{y3})\right)$
Optimized-Uniform $dL_x \geq 2$, $dL_y \geq 3$	$5 + \begin{cases} 0 & if\ 2L_x - L_y = 0 \\ 1 + \alpha(Z_1, 2L_x - L_y) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_x - 2 = 0 \\ 1 + \alpha(S, dL_x - 2) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_y - 3 = 0 \\ 1 + \alpha(S, dL_y - 3) & otherwise \end{cases} + \max(\alpha(S^d, L_x), \alpha(S^d, L_y))$	$2 + \begin{cases} 0 & if\ 2L_x - L_y = 0 \\ 1 + \beta(Z_1, 2L_x - L_y) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_x - 2 = 0 \\ 1 + \beta(S, dL_x - 2) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_y - 3 = 0 \\ 1 + \beta(S, dL_y - 3) & otherwise \end{cases} + \max(\beta(S^d, L_x), \beta(S^d, L_y))$
Optimized–Mixed $dL_{x3} \geq 2$, $dL_{y3} \geq 3$	$5 + \begin{cases} 0 & if\ 2L_{x1} - L_{y1} = 0 \\ 1 + \alpha(Z_1, 2L_{x1} - L_{y1}) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_{x3} - 2 = 0 \\ 1 + \alpha(S, dL_{x3} - 2) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_{y3} - 3 = 0 \\ 1 + \alpha(S, dL_{y3} - 3) & otherwise \end{cases} + \max(\alpha(S^d, L_{x3}), \alpha(S^d, L_{y3}))$	$2 + \begin{cases} 0 & if\ 2L_{x1} - L_{y1} = 0 \\ \beta(Z_1, 2L_{x1} - L_{y1}) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_{x3} - 2 = 0 \\ \beta(S, dL_{x3} - 2) & otherwise \end{cases} + \begin{cases} 0 & if\ dL_{y3} - 3 = 0 \\ \beta(S, dL_{y3} - 3) & otherwise \end{cases} + \max(\beta(S^d, L_{x3}), \beta(S^d, L_{y3}))$
Equivalent homogenous PC $L_x = L_y = 1$, $d = 3$	8	2
Equivalent Jacobian PC $d = 1,\ L_x = 2, L_y = 3$	7	2

Similar to what we did in case of *E/GF(p)*, we compare DPC for *E/GF(2^m)* with the conventional coordinate systems. Table 8.11 shows the exact number of computations of these coordinate systems and the corresponding equivalent systems in DPC for *E/GF(2^m)*. Although mixed coordinates for *E/GF(2^m)* are not existing in the literature, table 8.11 contains some useful mixed DPC systems.

Table 8. 11: Comparisons of field operations using DPC in *E/GF(2^m)*.

PC System	Addition	Doubling
HHH	15M + 2S	7M + 5S
JJJ	14M + 4S	5M + 5S
Optimized DPC (DPC-HHH) $d=3, L_x = L_Y = 1$	15M + 2S	8M + 2S
Optimized DPC (DPC-JJJ) $d=1, L_x = 2, L_Y = 3$	13M + 2S	7M + 2S
General Mixed DPC (DPC-AAH) $d=3$ $L_{x1} = L_{Y1} = L_{x2} = L_{Y2} = 0$ $L_{x3} = 1, L_{Y3} = 1$	8M + 1S	6M + 2S
General Mixed DPC (DPC-AAJ) $d=1$ $L_{x1} = L_{Y1} = L_{x2} = L_{Y2} = 0$ $L_{x3} = 2, L_{Y3} = 3$	7M + 1S	5M + 2S
General Mixed DPC (DPC- A_xH_yH) $d=3,\ L_{x1} = L_{x2} = 0,\ L_{Y1} = L_{Y2} = 1,\ L_{x3} = L_{Y3} = 1$	12M + 1S	8M + 2S
General Mixed DPC(DPC- A_xJ_yJ) $d=1,\ L_{x1} = L_{x2} = 0,\ L_{Y1} = L_{Y2} = 3,\ L_{x3} = 2, L_{Y3} = 3$	11M +2S	7M + 2S
General Mixed DPC (DPC- H_xA_yH) $d=3,\ L_{x1} = L_{x2} = 1,\ L_{Y1} = L_{Y2} = 0,\ L_{x3} = L_{Y3} = 1$	13M + 1S	8M + 3S
General Mixed DPC (DPC- J_xA_yJ) $d=1,\ L_{x1} = L_{x2} = 2,\ L_{Y1} = L_{Y2} = 0,\ L_{x3} = 2, L_{Y3} = 3$	15M + 2S	7M + 3S

DPC-HHH = Equivalent homogenous DPC, DPC-JJJ = Equivalent Jacobian DPC DPC-MMM = Equivalent modified DPC

In case of addition operation, table 8.11 shows that DPC-HHH has exactly the same number of computations as in HHH and DPC-JJJ is faster than JJJ by one multiplication and two squaring operations.

In doubling operation, DPC-HH is higher than HH by one multiplication but lower by 3 squaring. Hence by considering S = 0.8M, as in [23], DPC-HH is in total faster than HH. Also, DPC-JJ is higher than JJ by two multiplications but lower by 3 squaring. Hence, under the same assumption, i.e. S = 0.8M, DPC-JJ is faster than JJ.

Similar to the case of *E/GF(p)*, by using *mixed* DPC formulas for *E/GF(2^m)*, it is possible to hop from one coordinate system to another during the scalar multiplication without the need to perform any inversion operation. Tables 8.12 and 8.13 show the cost of hopping among a set of possible coordinate systems.

Table 8. 12: Hopping cost in DPC system (*E/GF(2^m)* Addition operation)

From \ To	Affine	DPC-HHH	DPC-JJJ	DPC-CCC	DPC-MMM
Affine	-	-	-	-	-
DPC-HHH	***2M + I***	*19M+2S*	*18M+2S*	*18M+2S*	*18M+3S*
DPC-JJJ	***3M+S+I***	*20M+2S*	*19M+2S*	*19M+2S*	*19M+3S*
DPC-CCC	***3M+S+I***	*20M+2S*	*19M+2S*	*19M+2S*	*19M+3S*
DPC-MMM	***3M+S+I***	*20M+3S*	*19M+3S*	*19M+3S*	*19M+3S*

Table 8. 13: Hopping cost in DPC system (*E/GF(2^m)* Doubling operation)

From \ To	Affine	DPC-HHH	DPC-JJJ	DPC-CCC	DPC-MMM
Affine	-	-	-	-	-
DPC-HHH	***2M + I***	*8M+2S*	*7M+2S*	*8M+3S*	*8M+4S*
DPC-JJJ	***3M+S+I***	*8M+2S*	*7M+2S*	*7M+2S*	*7M+3S*
DPC-CCC	***3M+S+I***	*8M+2S*	*7M+2S*	*7M+2S*	*7M+3S*
DPC-MMM	***3M+S+I***	*8M+3S*	*7M+3S*	*7M+3S*	*7M+3S*

Tables 8.14 and 8.15 show the number of multiplications for different values of L_x and L_y for *E/GF(2^m)* optimized DPC addition and doubling operations respectively. For each value of L_x there are several possible choices of L_y (second column). These choices increase as L_x increases. For example, in case of addition operation, if L_x = 1, then we have only one L_y possible value while if L_x = 10 we have six possible values of L_y. In case of doubling operation, if L_x = 1, then we have two possible values of L_y while if L_x = 5 we have six possible values of L_y. This due to the constraints caused by the relations between L_x and L_y.

In case of addition operation (table 8.14), for a certain L_x, the best choice of L_y is the one with the minimum umber of ones in the binary representation of the terms $(L_y - L_x)$, $(3L_x - 2L_y)$, $(dL_x - 2)$, $(dL_y - 3)$ and $(\max(L_x, L_y))$. For example, if L_x = 5, then the best choice of L_y is 5 while the best choice for L_x = 10 is L_y = 12.

Similarly, in case of doubling operation (table 8.15), the best choice of L_y for a certain L_x is the one with the minimum umber of ones in the binary representation of the terms $(2L_x - L_y)$, $(dL_x - 2)$, $(dL_y - 3)$ and $(\max(L_x, L_y))$. For example, if L_x = 3, then the best choice of L_y is either 3, 4 or 5 while the best choice for L_x = 5 is L_y = 5 or 9.

Table 8. 14: Possible values of L_x and L_y for addition operation in *E/GF(2^m)*

L_x	Valid range of L_y	Number of multiplications
1	1	15
2	2	17
	3	18
3	3	21
	4	21
4	4	18
	5	20
	6	21
5	5	21
	6	24
	7	22
6	6	21
	7	20
	8	20
	9	19
7	7	23
	8	24
	9	23
	10	25
8	8	21
	9	22
	10	23
	11	25
	12	20
9	9	21
	10	25
	11	25
	12	23
	13	22
10	10	23
	11	24
	12	22
	13	23
	14	24
	15	24

Table 8. 15: Possible values of L_x and L_y for doubling operation in *E/GF(2^m)*

L_x	Valid range of L_y	Number of multiplications
1	1	8
1	2	10
2	2	10
2	3	12
2	4	10
3	3	14
3	4	14
3	5	14
3	6	16
4	4	11
4	5	14
4	6	15
4	7	15
4	8	12
5	5	14
5	6	16
5	7	16
5	8	16
5	9	14
5	10	16

8.5 Using DPC System

One of the most important features of DPC is that it automates the selection of the projective coordinate system and uses a single mathematical formulation/software code to implement different projective coordinate systems. In other words, different projective coordinate systems can be implemented by using different values of Z^{L_x} , Z^{L_y} and *d*. For example, consider DPC addition formulas 7.7 and doubling formulas 7.8. By setting $L_x = L_y = 1$ and *d = 3*, we get the following addition and doubling formulas:

Addition:

$$\left.\begin{array}{l} X_3 = X_3' V, \quad Y_3 = Y_3', \quad Z_3 = V^3 T, \\ \textit{where}, \quad U_1 = Y_2 Z_1, \quad U_2 = Y_1 Z_2, \quad U = U_1 - U_1, \quad V_1 = X_2 Z_1, \quad V_2 = X_1 Z_2, \quad V = V_1 - V_1, \\ \qquad T = Z_1 Z_2, \quad T_1 = V, \quad X_3' = U^2 T - V^3 - 2V^2 V_2, \quad Y_3' = U(V^2 V_2 - X_3') - U_2 V^3 \end{array}\right\}$$

Doubling:

$$\left.\begin{array}{l} X_3 = X_3' S, \quad Y_3 = Y_3', \quad Z_3 = S^3 \\ \textit{Where}, \quad W = 3X_1^2 + aZ_1^2, \quad S = 2Z_1 Y_1, \quad T = W, \quad T_1 = SY_1, \quad T_2 = 2T_1 X_1, \\ \qquad X_3' = T^2 - 2T_2, \quad Y_3' = T(T_2 - X_3') - 4T_1 Y_1^2 Z_1 \end{array}\right\}$$

Which are identical to the homogenous projective coordinates system (section 5.3 in chapter 5) in which the transformation functions: $x = X / Z$ and $y = Y / Z$ are used. Also, By setting $L_x = 2, \quad L_y = 3$ and *d = 1*, we get a DPC system that is identical to the Jacobian projective coordinates system (section 5.3 in chapter 5) in which the transformation functions: $x = X / Z^2$ and $y = Y / Z^3$ are used.

DPC system can be plugged to any scalar multiplication algorithm such as those in [3] and [14] without any restriction. The only thing that is needed to be done is selecting the values of the projecting parameters L_x and L_y and the *d*-parameter. However, there are two possible modes for using DPC with any scalar multiplication algorithm. First, is initializing the coordinate system and selecting the projecting and *d* parameters in the beginning of the scalar multiplication and fixing that system for the whole scalar multiplication iterations. Second, is allowing projective coordinates hopping at any time during the scalar multiplication.

In scalar multiplication, it is required to perform a series of doubling and addition operations where the result of one operation is used as input operands to the other. This prevents conventional mixed coordinates from benefiting from the efficient mixed coordinates such as using HHH for addition and JJ for doubling. This is, however, because the result of the ADD operation is represented in H coordinates while the input of the DBL operation must be in J representation. The conversion from H to J representation requires an inversion operation as shown in table 5.2. This kind of problems do not exit in DPC system since it is possible to dynamically change from one coordinate system to another without any inversion operation simply by using mixed DPC formulas with setting L_{x3} and L_{y3} to the desired values.

In window based methods, DPC can use different projective coordinate systems for different phases of the scalar multiplication. Fore example, a certain coordinate system can be used for the pre-computation phase of the scalar multiplication while other coordinate systems can be used for addition and/or doubling operations in the main loop.

Furthermore, different blocks (or windows) of the scalar K can use different projective coordinate systems.

Finally, it worth to mention that each run of the scalar multiplication can start with new coordinate system every time. This is because DPC system lends itself to randomize the scalar multiplication simply by randomizing the projecting parameters.

8.6 Conclusions

This chapter discussed the performance and using of DPC. The performance of DPC for addition and doubling operations in both *E/GF(p)* and *E/GF(2^m)* has been analyzed. We conclude that the number of field operations required is a function of the projecting parameters L_x and L_y and the *d*-parameter. Various tables that show the number for required operations for several coordinate systems were presented.

Also, this chapter studied how the DPC can be used. DPC uses a single mathematical formulation/software code to implement different projective coordinate systems. Hence, we conclude that DPC system can be plugged to any scalar multiplication algorithm. However, two possible modes for using DPC with any scalar multiplication algorithm were been discussed. First, initializing the coordinate system and selecting the projecting and *d* parameters in the beginning of the scalar multiplication and fixing that system for the whole scalar multiplication iterations. Second, is allowing projective coordinates hopping at any time during the scalar multiplication.

CHAPTER 9

Scalar Multiplication Security In Presence of DPC

9.1 Introduction

Since the scalar multiplication is the part of any elliptic curve cryptosystem that is directly correlated to the secret scalar *K*, researcher have become increasingly aware of the possibility of side channel attacks that exploits specific properties of the implementation of the scalar multiplication. As discussed in chapter 6, there are many countermeasures that can be used to protect against these attacks. However, non of these countermeasures are guaranteed to defeat all the side channel attacks. For example, many countermeasures against differential power analysis attacks rely on randomizing the projective coordinates. But all these countermeasures are vulnerable to the projective coordinates leak since they depend on pre-determined projective coordinate systems. Moreover, these countermeasures are vulnerable to the newly proposed attacks such as RPA, ZPA, DA, ABDPA attacks.

According to the proposed classification, presented in chapter 6, of side channel attacks, in this chapter, we propose and analyze countermeasures for operation-and-data dependent and data-dependent attacks. We mean by operation-and-data dependent attacks

is the attacks that are based on both the data being manipulated and the operations being performed on this data. Also, we propose countermeasures for address-dependent attacks. For each of the proposed countermeasure, we provide the security and complexity analysis.

All the proposed countermeasures are based on using the DPC system as the coordinate system. This is because the DPC system lends itself to randomization simply by randomizing the projecting parameters L_x and L_y and/or d-parameter. Also, all the proposed countermeasures are applied to both *E/GF(p)* and *E/GF(2^m)*.

However, the following notations are used through out this chapter. DPC_ADD means any DPC addition formula. DPC_DBL means any DPC doubling formula. Also, we use the word "mixed" or "optimized" in front of these notations to specify the mixed and optimized DPC formulas.

This chapter is organized as follows. Section 9.2 discusses the proposed countermeasures for operation-and-data dependent attacks. The proposed countermeasures for address-dependent attacks are addressed in section 9.3. Finally, section 9.4 gives the conclusions.

9.2 Countermeasures for Operation and Data Dependent Attacks

As discussed in chapter 6, most of attacks are operation-dependent and at the same time data-dependent such as DPA and DA attacks. Some other attacks are data-dependent only such as RPA, ZPA and PCL. The existing countermeasures (section 6.9) do not

defeat all these attacks. More precisely, if a countermeasure defends one attack it may not defend the others. In the following, we show the attacks that each countermeasure cannot defend according to the mentioned reference.

- Randomizing the base point (code = 010). (Coron's 2nd countermeasure) does not protect RPA [30].
- Randomizing projective coordinates (code = 010) does not protect RPA, ZPA [31].
- Randomizing the scalar (code = 001) does not protect PCL [42].
- N. Smart's trick (code = 010) does not protect RPA, ZPA [31] and some cases of PCL [42].
- Non of the above countermeasures protect address bit DPA (ABDPA) [38].

Therefore, it is desired to find countermeasures to protect against these type of attacks. In this chapter, we propose three countermeasures for operation-and-data dependent and data-dependent attacks and two countermeasures for address-dependent attacks. All the proposed countermeasures are based on the following lemma.

Lemma 9.1: By randomizing the projecting parameters L_x and L_y and/or d parameter in any addition and doubling DPC formula, both the data being manipulated and the number of operations being performed are randomized.

Proof: Given that L_x and L_y and/or *d* are initialized randomly. Then the proof consists of the following three parts:

1. Each auxiliary variable *T* in any formula of the formulas presented in chapter 7 is a function of either L_x, L_y and/or *d.* Hence, the *value* of *T* is randomized since L_x and L_y and/or *d* are initialized randomly.

2. Each of the variables X_3, Y_3 $Z_3^{L_x}$ and $Z_3^{L_y}$ which form the resultant point ($X_3, Y_3, Z_3^{L_x}, Z_3^{L_y}$) of any formula of the formulas presented in chapter 7 is a function of either L_x, L_y and/or *d.* Hence, the *values* of these variables are randomized since L_x and L_y and/or *d* are initialized randomly.

3. As shown in tables 8.2, 8.3, 8.9 and 8.10, the number of required operations for each formula of the formulas presented in chapter 7 is a function of either L_x, L_y and/or *d.* Hence, the *number of required operations* are randomized since L_x and L_y and/or *d* are initialized randomly.

In the following, we introduce the proposed countermeasures and for each countermeasure, we do the following:

- Apply the countermeasure to the binary ML and binary NAF algorithms (4.2 and 4.8) respectively. We have chosen these two algorithms because they are the most widely used scalar multiplication algorithms.

- Analyze the security of the countermeasure by showing the attacks that the countermeasure can resist and how; and the attacks that the countermeasure cannot resist and why.
- Analyze the complexity of the countermeasure by showing the cost in terms of number of field operations required for the countermeasure itself and the cost of applying it to the ML and binary NAF algorithms.

Countermeasure 1: This countermeasure uses the DPC system with randomly initialized projecting parameters, L_x, L_y and d. Countermeasure1 randomizes L_x, L_y and d in the beginning of each run of the scalar multiplication. Hence, each execution of the scalar multiplication has its own coordinate system with different data values and different number of field operations. Although any DPC addition or doubling formula can be used for this countermeasure, it is preferred to use the optimized formulas since they require less number of field operations such as using formula 7.7 for addition in *E/GF(p)* and 7.16 for addition in $GF(2^m)$ (see tables 8.2, 8.3, 8.9 and 8.10).

Algorithms 9.1 and 9.2 show the application of this countermeasure to the binary ML and binary NAF algorithms (4.2 and 4.8) respectively (N is positive integer).

```
INPUT  K, P
OUTPUT        KP
    7.  L_x = rand(1..N) , L_y = rand(1..N), d = rand(3..N)
    8.  Set Z = 1 then compute P = (X,Y,1,1)
    9.  Initialize Q[2] = P
    10. for i = n-2 down to 0
    11.     Q[0] = DPC_DBL(Q[2])
    12.     Q[1] = DPC_ADD(Q[0], P)
    13.     Q[2] = Q[k_i]
    14. end for
    15. Convert Q[2] to affine coordinate.
    Return Q[2]
```

Algorithm 9. 1: Binary ML algorithm with countermeasure1

```
Input: An integer K an a point P = (x,y) ∈ E/GF(q)
Output: The point Q = KP ∈ E/GF(q)
    1.  Compute NAF(K) = (u_{l-1} ... u_l u_0)
    2.  L_x = rand(1..N) , L_y = rand(1..N), d = rand(3..N)
    3.  Set Z = 1 then compute P = (X,Y,1,1)
    4.  Q = ∞
    5.  for j = l - 1 downto 0 do
    6.  Q = DPC_DBL(Q)
    7.  if u_l = 1 then
    8.          Q = DPC_ADD(Q, P)
    9.  if u_l = –1 then
    10.         Q = DPC_ADD(Q, – P)
    11. Convert Q to affine coordinate.
  Return (Q)
```

Algorithm 9. 2: Binary NAF algorithm with countermeasure1

Security analysis of Countermeasure1:

The number of field operations in DPC_ADD and DPC_DBL is determined in the beginning of the scalar multiplication when the values of L_x , L_y and d are initialized. These numbers remain fixed during the whole scalar multiplication. In the next run of the scalar multiplication, new values of L_x , L_y and d will be initiated and hence the number

of field operations in DPC_ADD and DPC_DBL will be changed accordingly. Based on that, this countermeasure can resist DPA, DFA, DEMA and DA.

Also, any register used in DPC_ADD and DPC_DBL operations changes at each execution. Hence this countermeasure is resistant against RPA, ZPA and PCL attacks.

Since countermeasure1 has nothing to do with addresses of variables, algorithm 9.1 is not immune against ABDPA. This is because there is still a direct correlation between the register transfer operation in step 7 and the scalar bit value. On the other hand, algorithm 9.2 is immune against ABDPA by its nature since the locations of operands of DPC_ADD and DPC_DBL operations are independent of the scalar bit values.

Finally, it is worth to mention that countermeasure1 resists SPA since it uses double-and-add always method in algorithm 9.1. In algorithm 9.2, the addition operations are not conditioned by the value of the scalar bit.

Complexity analysis of Countermeasure1:

As discussed in chapter 4, let the binary representation of the scalar K is

$K = k_{n-1}2^{n-1} + k_{n-2}2^{n-2} + ... + k_1 2 + k_0$ where n is the number of bits.

Let A and D denotes the number of field operations (multiplications + squaring) in DPC_ADD and DPC_DBL respectively. In other words, A contains the number of multiplications and squaring in DPC_ADD and D contains the number of multiplications

and squaring in DPC_DBL. Fore example, $A = 12M + 2S$ and $D = 8M + 5S$ in case of using DPC-HHH system. Since the DPC_ADD and DPC_DBL operations are performed in each iteration of algorithm 9.1 (double-and-add always), then its Expected Running Time (ERT) is given by [52]:

$$ERT(Algorithm\ 9.1) = An + Dn \qquad 9.1$$

With n being the bit length of the scalar K.

The values of A and D are given in tables 8.2 and 8.3 for $E/GF(p)$ and in tables 8.9 and 8.10 for $E/GF(2^m)$. Note that the number of field operations in A and D differ from one DPC formula to the other. For example, the ERT of algorithm 9.1 when using the general DPC_ADD and DPC_DBL formulas is given by:

$ERT(Algorithm\ 9.1) =$

$$\left(18 + \max\left(E(\alpha(R^d, L_x)), E(\alpha(R^d, L_y))\right) + 11 + \max\left(E(\alpha(S^d, L_x)) + E(\alpha(S^d, L_y))\right)\right) nM +$$

$$\left(2 + \max\left(E(\beta(R^d, L_x)), E(\beta(R^d, L_y))\right) + 4 + \max\left(E(\beta(S^d, L_x)) + E(\beta(S^d, L_y))\right)\right) nS$$

Where the letter E before *alpha* and *beta* functions means their expected values which are given by equation 8.1 (see section 8.2). Note that M denotes multiplication and S denotes squaring.

Table 9.1 shows the expected running times of algorithm 9.1 when using some specific DPC system.

Table 9. 1: Expected running times of algorithm 9.1 for specified DPC systems

Coordinate system	**ERT in case of *E/GF(p)***	**ERT in case of *E/GF(2^m)***
Optimized DPC-HHH	20*n M* + 7*n S*	*23n M* + 4*n S*
Optimized DPC-JJJ	20*n M* + 8*n S*	20*n M* + 4*n S*

n = bit length of the recoded scalar, M = multiplication and S = squaring

In case of algorithm 9.2, given that the binary representation of the recoded scalar U = NAF(K) is given by:

$$U = u_{l-1}2^{l-1} + u_{l-2}2^{l-2} + ... + u_1 2 + u_0 \qquad 9.2$$

Then according to [52] the average density of non zero digits in U is $l/3$ where l is the bit length of U. Based on that, the expected running time of algorithm 9.2 is:

$$ERT(Algorithm\ 9.2) = \frac{l}{3}A + D\,l \qquad 9.3$$

with A and D given in tables 8.2 and 8.3 for *E/GF(p)* and in tables 8.9 and 8.10 for *E/GF(2^m)*. For example, the ERT of algorithm 9.2 when using the general DPC_ADD and DPC_DBL formulas is given by:

ERT(Algorithm 9.2) =

$$\left(\left(18 + \max\left(E(\alpha(R^d, L_x)), E(\alpha(R^d, L_y))\right)\right)\frac{l}{3} + \left(11 + \max\left(E(\alpha(S^d, L_x)) + E(\alpha(S^d, L_y))\right)\right)l\right)M +$$

$$\left(2 + \max\left(E(\beta(R^d, L_x)), E(\beta(R^d, L_y))\right) + 4 + \max\left(E(\beta(S^d, L_x)) + E(\beta(S^d, L_y))\right)\right)Sl$$

Table 9.2 shows the expected running times of algorithm 9.2 when using some specific DPC systems.

Table 9. 2: Expected running times of algorithm 9.2 for specified DPC systems

Coordinate system	**ERT in case of *E/GF(p)***	**ERT in case of *E/GF(2^m)***
Optimized DPC-HHH	*12l M* + 5.66*l S*	13*l M* + 2.66*l S*
Optimized DPC-JJJ	*12l M* + 6*l S*	*11.33l M* + 2.66*l S*

l = bit length of the recoded scalar, M = multiplication and S = squaring

Countermeasure 2: This countermeasure is based in using DPC in conjunction with exponent (scalar) splitting (ES) method as follows:

1. ES splits the scalar *K* into two parts *R* and *(K – R)* using a random number *R*.
2. Computes $P_1 = RP$, $P_2 = (K - R)P$ and then $KP = P_1 + P_2$.

P_1 and P_2 are computed using DPC with randomly initialized projecting parameters. These parameters could be the same for both points (i.e. for P_1 and P_2) or be different. In case of different projecting parameters, the final addition to get $KP = P_1 + P_2$, is performed either using a mixed addition formula that allows using different projective coordinates, or performed using the affine coordinates since it is the last operation and the final result should be presented in the affine coordinates.

Let the number of bits in *R* and (*K* – *R*) be n_1 and n_2 respectively. Then the binary representation of *R* is given by,

$$R = r_{n-1}2^{n-1} + r_{n-2}2^{n-2} + ... + r_1 2 + r_0$$

Algorithms 9.3 and 9.4 show the application of this countermeasure to the binary ML and binary NAF algorithms respectively. Note that in case of binary NAF, countermeasure2 splits the scalar before recoding and then R and $(K - R)$ are recoded separately. In this case, n_1 and n_2 become the bit length of *U1* and *U2* respectively. However, note that the binary representation of the recoded scalar is $U = u_{l-1}2^{l-1} + u_{l-2}2^{l-2} + ... + u_1 2 + u_0$ (see section 4.5.3) with bit length l equals to n or grater by only 1.

Security analysis of Countermeasure2:

The security analysis of countermeasure1 is applicable to phase1 and phase2 of countermeasure2. That is, each phase is immune against DPA, DFA, and DA since the number of operations is randomized and immune against RPA, ZPA, and PCL since the data manipulated is also randomized. Furthermore, countermeasure2 resists SPA and DPA in the same way discussed in countermeasure1. Also, algorithm 9.3 does not resist ABDPA for the same reason addressed in countermeasure1.

However, countermeasure2 has an additional security strength resulting from *random* splitting the scalar into two scalars. This is because in each run of the scalar multiplication the data and the number of operations will be randomized since R and $(K - R)$ will have different values in each run.

INPUT K, P
OUTPUT KP

Phase 1:

1. R = rand*(1..K-1)*
2. L_x = rand(1..N) , L_y = rand(1..N), d = rand(3..N)
3. Set Z = 1 then compute P = *(X,Y,1,1)*
4. Initialize Q[2] = P
5. **for** $i = n_1 - 2$ **down** to 0
6. Q[0] = Optimized_DPC_DBL(Q[2])
7. Q[1] = Optimized_DPC_ADD(Q[0], P)
8. Q[2] = Q[r_i]
9. **end for**
10. P_1 = Q[2]

Phase 2:

11. $K = K - R$
12. L_x = rand(1..N) , L_y = rand(1..N), d = rand(3..N)
13. Set Z = 1 then compute P = *(X,Y,1,1)*
14. Initialize Q[2] = P
15. **for** $i = n_2 - 2$ **down** to 0
16. Q[0] = Optimized_DPC_DBL(Q[2])
17. Q[1] = Optimized_DPC_ADD(Q[0], P)
18. Q[2] = Q[k_i]
19. **end for**
20. $P_1 = P_1 +$ Q[2]
21. Convert P_1 to affine coordinate.

Return (P_1)

Algorithm 9. 3: Binary ML algorithm with countermeasure2

Input: K , P
Output: The point $Q = KP$

1. R = rand*(1..K-1)*

Phase 1:

2. Compute NAF(U) = *(* $u_{n_1-1}\ u_{n_1-2}\ \ldots\ u_0$ *)*
3. L_x = rand(1..N) , L_y = rand(1..N), d = rand(3..N)
4. Set Z = 1 then compute P = *(X,Y,1,1)*
5. $Q = \infty$
6. **for** $i = n_1 - 1$ **downto** 0 **do**
7. Q = Optimized_DPC_DBL(Q)
8. **if** $u_i = 1$ **then**
9. Q = Optimized_DPC_ADD(Q, P)
10. **if** $u_i = -1$ **then**
11. Q = Optimized_DPC_ADD(Q, $-P$)
12. $P_1 = Q$

Phase 2:

13. Compute NAF(K-R) U = *(* $u_{n_2-1}\ u_{n_2-2}\ \ldots\ u_0$ *)*
14. L_x = rand(1..N) , L_y = rand(1..N), d = rand(3..N)
15. Set Z = 1 then compute P = *(X,Y,1,1)*
16. $Q = \infty$
17. **for** $i = n_2 - 1$ **downto** 0 **do**
18. Q = Optimized_DPC_DBL(Q)
19. **if** $u_i = 1$ **then**
20. Q = Optimized_DPC_ADD(Q, P)
21. **if** $u_i = -1$ **then**
22. Q = Optimized_DPC_ADD(Q, $-P$)
23. $P_1 = P_1 + Q$
24. Convert P_1 to affine coordinate.

Return (P_1)

Algorithm 9. 4: Binary NAF algorithm with countermeasure2

Complexity analysis of Countermeasure2:

Countermeasure2 computes *KP* by almost the same cost as countermeasure1 since each phase uses the double-and-add always method. However, there are an extra final addition operation to compute $KP = P_1 + P_2$. Also, computing $K = K - R$ requires one word-length subtraction operation which can be neglected.

Countermeasure3: A third countermeasure uses the ability of DPC to dynamically hop from one coordinate system to another half the way in the scalar multiplication. This hopping can be achieved by using general or optimized *mixed* addition and doubling formulas. This kind of formulas have the ability to perform the addition and doubling operations in totally different projective coordinates. Furthermore, these formulas do not requires any inversion operation to change form one coordinate system to the other. However, dynamic hopping can range from hopping in each iteration of the scalar multiplication (full hopping) to non-hopping which is identical to the case of countermeasure1.

Countermeasure3 can be performed as follows:

1. Randomly initialize the projecting parameters L_{x1}, L_{y1}, L_{x2}, L_{y2}, L_{x3}, L_{y3} and d parameter. Note that we need to use all these parameters since the mixed formulas are used.
2. Start the scalar multiplication.

3. In each iteration, based on the value of a random bit *r*, randomly select new parameters L_{x3}, L_{y3} and *d*. i.e. if *r* = 1, then L_{x3} = rand (1..N), L_{y3} = rand (1..N), and *d* = rand (3..N); otherwise keep the old values. This random selection is called a hop.

Algorithms 9.5 and 9.6 show the application of this countermeasure to the binary ML and binary NAF algorithms respectively.

INPUT *K*, *P*
OUTPUT *KP*

1. L_{x1} = rand(1..N) , L_{y1} = rand(1..N), *d* = rand(3..N)
2. L_{x2} = rand(1..N) , L_{y2} = rand(1..N)
3. L_{x3} = rand(1..N) , L_{y3} = rand(1..N)
4. Set Z = 1 then compute *P = (X,Y,1,1)*
5. Initialize Q[2] = *P*
6. **for** *i = n-2* **down** to 0
7. **if** (*r* = rand(0..1) = 1) **then**
8. L_{x3} = rand(1..N) , L_{y3} = rand(1..N)
9. Q[0] = Mixed_DPC_DBL(Q[2])
10. Q[1] = Mixed_DPC_ADD(Q[0], *P*)
11. Q[2] = Q[k_i]
12. **end for**
13. Convert Q[2] to affine coordinate.

Return Q[2]

Algorithm 9. 5: Binary ML algorithm with countermeasure3

Input: An integer K an a point $P = (x,y) \in E/GF(q)$
Output: The point $Q = KP \in E/GF(q)$

1. Compute NAF(K) = $(u_{l-1} \ldots u_1 u_0)$
2. L_{x1} = rand(1..N) , L_{y1} = rand(1..N), d = rand(3..N)
3. L_{x2} = rand(1..N) , L_{y2} = rand(1..N)
4. L_{x3} = rand(1..N) , L_{y3} = rand(1..N)
5. Set Z = 1 then compute $P = (X,Y,1,1)$
6. $Q = \infty$
7. **for** $j = l - 1$ **downto** 0 **do**
8. **if** (r = rand(0..1) = 1) **then**
9. L_{x3} = rand(1..N) , L_{y3} = rand(1..N)
10. Q = Mixed_DPC_DBL(Q)
11. **if** $u_l = 1$ **then**
12. Q = Mixed_DPC_ADD(Q, P)
13. **if** $u_l = -1$ **then**
14. Q = Mixed_DPC_ADD(Q, $-P$)
15. **end for**
16. Convert Q to affine coordinate.

Return (Q)

Algorithm 9. 6: Binary NAF algorithm with countermeasure3

Security analysis of Countermeasure3:

The security analysis of this countermeasure is similar to that of countermeasure1 except that it uses mixed DPC formulas in which each coordinate of each point has it own different projecting parameters.

According to step1 of countermeasure3, the number of field operations and the data manipulated will be randomized in each run of the scalar multiplication. Hence this countermeasure has the same security as countermeasure1. i.e. it can defend the same attacks defended by countermeasure1. Moreover, in any iteration of the scalar multiplication, one or more of the projecting parameters L_{x3}, L_{y3} and/or d can hop to a new random value. This introduces intermediate randomization inside execution of the

scalar multiplication where it can guard any similarity analysis of different blocks of the scalar multiplication.

Since countermeasure3 has nothing to do with addresses of variables, algorithm 9.5 is not immune against ABDPA. This is because there is still a direct correlation between the register transfer operation in step 11 and the scalar bit value. On the other hand, algorithm 9.6 is immune against ABDPA by its nature since the locations of operands of DPC_ADD and DPC_DBL operations are independent of the scalar bit values.

Algorithm 9.5 resists SPA because of: First, it uses double-and-add always method. Second, the projective coordinates hopping in the intermediate iterations is applied to both the addition and doubling operations to prevent any distinguishably between them. Recall that the addition and doubling operations are performed in each iteration independently from the scalar bit value. Third, projective coordinates hopping happens at random iterations without any correlation between this hopping and the scalar bit value. i.e. the projective coordinates hopping is independent of the scalar bit values.

Also, algorithm 9.6 resists SPA because the addition operations are not conditioned by the value of the scalar bit. Moreover, the "Third" argument above is valid in case of algorithm 9.6 as well.

Complexity analysis of Countermeasure3:

The expected running time of algorithms 9.5 and 9.6 are given by 9.1 and 9.3 respectively with the values of *A* and *D* being the number of filed operations for addition and doubling operations for the mixed DPC formulas only. The number of field operations of such formulas are given in tables 8.2 and 8.3 for *E/GF(p)* and in tables 8.9 and 8.10 for *E/GF(2^m)*. For example, the ERT of algorithm 9.5 when using the general mixed DPC_ADD and DPC_DBL formulas is given by:

ERT(Algorithm 9.5) =

$$\left(\begin{pmatrix}18+\max\left(E(\alpha(R^d,L_{x3})),E(\alpha(R^d,L_{y3}))\right)+\\ \max\left(E(\alpha(T_1^{\,d},L_{x3})),E(\alpha(T_1^{\,d},L_{y3}))\right)\end{pmatrix}+\begin{pmatrix}11+\max\left(E(\alpha(S^d,L_{x3}))+E(\alpha(S^d,L_{y3}))\right)+\\ \max\left(E(\alpha(Z_1^{L_{y1}},L_{x3}))+E(\alpha(Z_1^{L_{y1}},L_{y3}))\right)\end{pmatrix}\right)Mn$$
$$+\left(\begin{pmatrix}2+\max\left(E(\beta(R^d,L_{x3})),E(\beta(R^d,L_{y3}))\right)+\\ \max\left(E(\beta(T_1^{\,d},L_{x3})),E(\beta(T_1^{\,d},L_{y3}))\right)\end{pmatrix}+\begin{pmatrix}5+\max\left(E(\beta(S^d,L_{x3}))+E(\beta(S^d,L_{y3}))\right)+\\ \max\left(E(\beta(Z_1^{L_{y1}},L_{x3}))+E(\beta(Z_1^{L_{y1}},L_{y3}))\right)\end{pmatrix}\right)Sn$$

Where the letter *E* before *alpha* and *beta* functions means their expected values which are given by equation 8.1 (see section 8.2).

On the other hand, the ERT of algorithm 9.6 when using the general mixed DPC_ADD and DPC_DBL formulas is given by:

ERT(Algorithm 9.6) =

$$\left(\begin{pmatrix}18+\max\left(E(\alpha(R^d,L_{x3})),E(\alpha(R^d,L_{y3}))\right)+\\ \max\left(E(\alpha(T_1^{\,d},L_{x3})),E(\alpha(T_1^{\,d},L_{y3}))\right)\end{pmatrix}\frac{l}{3}+\begin{pmatrix}11+\max\left(E(\alpha(S^d,L_{x3}))+E(\alpha(S^d,L_{y3}))\right)+\\ \max\left(E(\alpha(Z_1^{L_{y1}},L_{x3}))+E(\alpha(Z_1^{L_{y1}},L_{y3}))\right)\end{pmatrix}l\right)M$$
$$+\left(\begin{pmatrix}2+\max\left(E(\beta(R^d,L_{x3})),E(\beta(R^d,L_{y3}))\right)+\\ \max\left(E(\beta(T_1^{\,d},L_{x3})),E(\beta(T_1^{\,d},L_{y3}))\right)\end{pmatrix}+\begin{pmatrix}5+\max\left(E(\beta(S^d,L_{x3}))+E(\beta(S^d,L_{y3}))\right)+\\ \max\left(E(\beta(Z_1^{L_{y1}},L_{x3}))+E(\beta(Z_1^{L_{y1}},L_{y3}))\right)\end{pmatrix}\right)Sl$$

9.3 Countermeasures for Address-Dependent Attacks

Since most of the scalar multiplication binary algorithms are vulnerable to address-bit-DPA attack (ABDPA), it is desired to find an immune algorithm to such attack. Here, we prose two ML algorisms called Add-Add algorithm and transition-based algorithm that can be used in conjunction with DPC system. These algorithms can be used to protect against class C attack. Fortunately, these algorithms can also be used to protect against doubling attack. However, it is worth to mention that DPC can be plugged to any of these algorithms. Hence, we will concentrate in describing the proposed algorithms letting the use of DPC to be default argument.

9.3.1 Add-Add Algorithm

This algorithm is a ML algorithm. It performs one ADD operation followed by another ADD operation in each iteration of a scalar multiplication. In any iteration of the scalar multiplication, the first ADD and the second ADD operations are performed in a fixed sequence (ADD → ADD). i.e. they will be performed in all iterations in the same order independently of the scalar bit values. Note that we can get $-P$ by simply negating the y-coordinate of P in case of $GF(p)$ and adding x to y coordinates in case of $GF(2^m)$. Steps of algorithm Add-Add are shown in algorithm 9.7.

INPUT K, P
OUTPUT KP
1. Initialize Q[0] = P(or 2P); Q[1] = 2P(or P); Q[2] = P
2. **for** $i = n\text{-}2$ **down** to 0
3. Q[0] = ADD(Q[1],Q[0])
4. Q[1] = ADD((Q[0], $(-1)^{1-ki}$ Q[2])]
5. **end for**

return Q[1- k_0]

Algorithm 9. 7: Add-Add algorithm

The second ADD operation performs the addition operation on the contents of Q[0] and Q[2]. The result is stored in Q[1]. The effect of $(-1)^{1-k_i}$ in step 4 of the algorithm can be explained as follows. First, note that the contents of Q[2] is always P. If the current bit k_i is 1, P will be added to Q[0]. Otherwise (i.e. for $k_i = 0$), $-P$ is added to Q[0].

Figure 9.1 presents two examples of Add-Add algorithm. The upper table of the Figure shows the values of Q[0], Q[1], and Q[2] in all iterations of calculating 173P. The lower table shows all iterations of calculating 155P.

K	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
Q[2]	1	1	1	1	1	1	1	1
Q[0]	1	3	5	11	21	43	87	**173**
Q[1]	2	2	6	10	22	44	86	174

K	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**
Q[2]	1	1	1	1	1	1	1	1
Q[0]	1	3	5	9	19	39	77	**155**
Q[1]	2	2	4	10	20	38	78	156

Figure 9. 1: Two examples of Add-Add algorithm.
Upper table calculates 173P. Lower table calculates 155P.

Add-Add algorithm resists doubling attack by its nature since no doubling operation at all. It resists ABDPA since it reads its operands from a fixed locations

regardless of the scalar bit value. When –P is needed it is simply computed (it can be computed all the times).

9.3.2 Transition-Based Algorithm

This algorithm is a ML algorithm. In any iteration, doubling and addition operations are performed in a fixed sequence, denoted by DBL→ADD. In other words, DBL and ADD operations are always performed in all iterations in the same order independently of the bit values of a scalar. The most important property of this algorithm is that in the *i-th* iteration of calculating *KP*, the selection of the input operand of DBL operation is dependant on the existence of a transition between bits k_i and k_{i+1} of a scalar *K* and it is not dependant directly on the value of k_i. The steps of the transition-based algorithm are shown in algorithm 9.8.

INPUT K, P
OUTPUT KP
1. Initialize Q[0] = P; Q[1] = 2P
2. **for** *i* = *n-2* **down** to 0
3. Q[2] = DBL(Q[1 – ($k_i \oplus k_{i+1}$)])
4. Q[0] = ADD(Q[1],Q[0]]
5. Q[1] = Q[2]
6. **end for**

return Q[1- k_0]

Algorithm 9. 8: Transition-based algorithm

The choice of input operand of DBL operation in step3 is based on existence of a transition between k_i and k_{i+1} bits of the scalar. If there is a transition from 0 to 1 or from

1 to 0 between bits k_i and k_{i+1}, Q[0] is doubled and the result is stored in Q[2]; otherwise (i.e. k_i and k_{i+1} are both 1's or both are 0's and hence no transition) Q[1] is doubled and the result is stored in Q[2].

Figure 9.2 presents two examples transition-based algorithm. The upper table of the Figure shows the values of Q[0], Q[1], and Q[2] in all iterations of calculating 173P. The lower table shows all iterations of calculating 155P.

K	1	0	1	0	1	1	0	1
Q[2]		2	6	10	22	44	86	174
Q[0]	1	3	5	11	21	43	87	**173**
Q[1]	2	2	6	10	22	44	86	174

K	1	0	0	1	1	0	1	1
Q[2]		2	4	10	20	38	78	156
Q[0]	1	3	5	9	19	39	77	**155**
Q[1]	2	2	4	10	20	38	78	156

Figure 9. 2: Two examples Transition-Based algorithm.
Upper table calculates 173P. Lower table calculates 155P.

Transition-based algorithm resists ABDPA in the sense that the same location (address) is accessed either on a transition from 1 to 0 or from 0 to 1. Therefore, it is difficult to detect whether this transition is from 0 to 1 or from 1 to 0. The same argument can hold in the absence of a transition. In this case, an attacker cannot know whether the previous bit was 1 and remains 1 or was 0 and remains 0 since the same address is used in both cases.

Transition-based algorithm resists DA in the same scenario described above since the operand of the doubling operation is chosen based on the existence/absence of a

transition. The same operand is doubled either on a transition from 1 to 0 or from 0 to 1. Therefore, it is difficult to detect whether this transition is from 0 to 1 or from 1 to 0. On the other hand, in the case of transition absence, the same operand is doubled whether the previous bit was 1 and remains 1 or was 0 and remains 0.

Countermeasure4: Combining Add-Add and Transition-based Algorithms

The first iteration of transition-based algorithm is weak against ABDPA since the most significant bit of the key, k_{n-1}, is always known to be 1. In this case, an attacker can find the value of the second most significant bit k_{n-2} depending on whether the input operand of DBL operation is Q[1] or Q[0] as stated in step 3 of the algorithm. To overcome this difficulty we use the Add-Add algorithm to perform the initial iteration. This is because it has the property that its initial step is independent of the content of Q[0] and Q[1] which could be either the points *P* and *2P* or *2P* and *P* respectively. In other words, when using Add-Add algorithm in the first iteration, an attacker can not detect the value of the next most significant bit, k_{n-2}, even though the value of the most significant bit, k_{n-2}, is always known to be 1. It is this property of Add-Add algorithm that is used to overcome the possible leaking of information about k_{n-2} in the first iteration of transition-based algorithm. This combination of Add-Add and transition-based algorithms is used to prevent any leakage of information about k_{n-2}. Once the value of k_{n-2} is protected against ABDPA in the first iteration, transition-based algorithm is used in subsequent iterations.

9.4 Conclusions

This chapter discussed the security of DPC. We have proposed and analyzed countermeasures for operation-and-data dependent and address-dependent attacks.

All the proposed countermeasures are based on using the DPC system as the coordinate system since it has the ability to lends itself to randomization simply by randomizing the projecting parameters L_x and L_y and/or d-parameter. We conclude that by randomizing the projecting parameters L_x and L_y and/or d parameter in any addition and doubling DPC formula, both the data being manipulated and the number of operations being performed are randomized.

Also, we conclude that all the proposed countermeasures can be applied to both E/GF(p) and E/GF(2^m).

CHAPTER 10

General Conclusions

10.1 Introduction

The main objective of this chapter is to summarize the results obtained in this thesis. Another aim is to provide some suggestions for future work that may be carried out based on the results obtained.

This chapter is subdivided as follows. Section 10.2 summarizes the work undertaken in the thesis. Section 10.3 presents some suggestions for future research.

10.2 Overview and Summary of The Work in The Thesis

The work undertaken in this thesis is mainly in three parts: first, proposing the new Dynamic Projective Coordinate (DPC) system. Second, analyzing performance of the proposed DPC and discussing how it can be used. Third, developing DPC-based countermeasures and algorithms that can cover all the classes of the side channel attacks presented in chapter 6.

10.2.1 DPC System

10.2.1.1 Overview

In this thesis, a new approach, called Dynamic Projective Coordinate (DPC) system was proposed. It allows the computing/encrypting device to select the projective coordinate system either at random, or according to a certain rule.

DPC automates the selection of the projective coordinate system and uses a single mathematical formulation/software code to implement different projective coordinate systems. Different projective coordinates can be implemented by using two parameters where one parameter defines the projection of the *x*-coordinate and a second parameter defines the projection of the *y*-coordinate of an elliptic curve point. This allows different projective coordinates to be used within the same mathematical formulation in calculating the scalar multiplication.

10.2.1.2 Summary of The Results

In this part of the thesis, we obtained the following formulas for elliptic curve defined over finite fields *GF(p)* and $GF(2^m)$:

1. General dynamic addition and doubling formulas that allow different projective coordinate systems to be used within the same mathematical formulation. In these formulas, L_x and L_y can be selected without any restriction. In other words no relation between them.
2. Optimized dynamic addition and doubling formulas that use DPC system and minimize the computation time through reducing the required number of filed

operations. In these formulas, L_x and L_y are selected according to certain rules to minimize the number of required operations.

3. Mixed dynamic addition and doubling formulas in which each coordinate can be projected using its own projecting parameter resulting in the most mixing degree of coordinates ever. In this way, coordinates of the same point can be represented in different coordinate systems

10.2.2 Performance of DPC System

10.2.2.1 Overview

The performance of DPC for addition and doubling operations in both *E/GF(p)* and *E/GF(2^m)* has been analyzed. We conclude that the number of field operations required is a function of the projecting parameters L_x and L_y and the *d*-parameter. Various tables that show the number for required operations for several coordinate systems were presented.

10.2.2.2 Summary of The Results

In this part of the thesis, we obtained the following results:

First*, in case of *E/GF(p)

1. Addition using DPC-HHH has exactly the same number of computations as in HHH.
2. Addition using DPC-JJJ is faster than JJJ by one squaring operation.

3. Addition using DPC-MMM is faster than MMM by one multiplication and 2 squaring operations.

1. Doubling using HH is faster than DPC-HH by one multiplication
2. Doubling using JJ has less multiplications and more squaring than DPC-JJ.

Second*, in case of *E/GF(2^m)

1. Addition using DPC-HHH has exactly the same number of computations as in HHH
2. Addition using DPC-JJJ is faster than JJJ by one multiplication and two squaring operations.
3. Doubling using DPC-HH is higher than HH by one multiplication but lower by 3 squaring. Hence by considering S = 0.8M, as in [23], DPC-HH is in total faster than HH.
4. Doubling using DPC-JJ is higher than JJ by two multiplications but lower by 3 squaring. Hence by considering S = 0.8M, as in [23], DPC-JJ is in total faster than JJ.
5. Various dynamic mixed coordinates for *E/GF(2^m)* for addition and doubling operations. Note that the conventional mixed coordinates for *E/GF(2^m)* are not existed in the literature.

10.2.3 Using DPC System

In this thesis, we studied how the DPC can be used. DPC uses a single mathematical formulation/software code to implement different projective coordinate

systems. Hence, we conclude that DPC system can be plugged into any scalar multiplication algorithm. However, two possible modes for using DPC with any scalar multiplication algorithm were discussed. First, initializing the coordinate system and selecting the projecting and *d* parameters in the beginning of the scalar multiplication and fixing that system for all scalar multiplication iterations. Second, is allowing projective coordinates hopping at any time during the scalar multiplication.

10.2.4 Scalar Multiplication Security in Presence of DPC System

10.2.4.1 Overview

In this thesis, we proposed DPC-based countermeasures for each class of the classes of attacks presented in chapter 6. A common property among the proposed DPC-Based countermeasures is that the scalar multiplication can be randomized by simply varying one of the projecting parameter used. We conclude that by randomizing L_x, L_y and *d* parameters, we randomize both the data being manipulated and the number of operations being performed in the scalar multiplication.

10.2.4.2 Summary of The Results

In this part of the thesis, we obtained the following results:

***First,* Proposed Countermeasures**

Countermeasure 1: This countermeasure uses the DPC system with randomly initialized projecting parameters, L_x, L_y and *d*. It randomizes L_x, L_y and *d* in the beginning of each

run of the scalar multiplication. Hence, each execution of the scalar multiplication has its own coordinate system with different data values and different number of field operations.

Countermeasure 2: This countermeasure is based on using DPC in conjunction with exponent (scalar) splitting (ES) method. ES splits the scalar *K* into two parts *r* and *(K – r)* using a random number *r* . The scalar multiplication is then computed as,

$KP = P_1 + P_2$, where $P_1 = rP$, $P_2 = (K - r)P$

P_1 and P_2 are computed using DPC with randomly initialized projecting parameters. These parameters could be the same for both points (i.e. for P_1 and P_2) or be different.

Countermeasure 3: A third countermeasure uses the ability of DPC to dynamically hop from one coordinate system to another half the way in the scalar multiplication. This hopping can be achieved by using general or optimized *mixed* addition and doubling formulas which have the ability to perform the addition and doubling operations in totally different projective coordinates.

***Second*, proposed algorithms**

1. Add-Add Algorithm

It is a ML algorithm. It performs one ADD operation followed by another ADD operation in each iteration of a scalar multiplication. In any iteration of the scalar multiplication, the first ADD and the second ADD operations are performed in a fixed

sequence. The second ADD operation works as follows: If the current bit k_i is 1, P will be added. Otherwise (i.e. for $k_i = 0$), $-P$ is added.

2. Transition-based Algorithm

It is a ML algorithm. In this algorithm, DBL and ADD operations are always performed in all iterations in the same order independently of the bit values of a scalar. The most important property of this algorithm is that in the *i-th* iteration, the selection of the input operand of DBL operation is dependant on the existence of a transition between bits k_i and k_{i+1} of a scalar *K* and it is not dependant directly on the value of k_i.

Countermeasure 4: This countermeasure is based on Combining the Add-Add and Transition-based Algorithms. The Add-Add algorithm is used to perform the initial iteration of the scalar multiplication because the first iteration of the transition-based algorithm is weak against ABDPA. It is this property of Add-Add algorithm that is used to overcome the possible leaking of information about k_{n-2} in the first iteration of transition-based algorithm. This combination of Add-Add and transition-based algorithms is used to prevent any leakage of information about k_{n-2}. Once the value of k_{n-2} is protected against ABDPA in the first iteration, transition-based algorithm is used in subsequent iterations.

10.3 Suggestions for Future Work

Since the proposed DPC enables the ECC designers to choose from many combinations of DPC systems and/or various scalar multiplication algorithms, we propose the following future work

1. This thesis provides dynamic addition and doubling formulas for *E/GF(p)* based on the DPC system where these formulas are separate. A suggested future research is to provide a *unified dynamic formula* for *E/GF(p)* that can be used for both addition and doubling operations. i.e. getting one dynamic formula that can be used for both addition and doubling operations at the same time. This unified formula should be developed using the DPC transformation functions.
2. This thesis provides dynamic addition and doubling formulas for $E/GF(2^m)$ based on the DPC system where these formulas are separate. A suggested future research is to provide a *unified dynamic formula* for $E/GF(2^m)$ that can be used for both addition and doubling operations. i.e. getting one dynamic formula that can be used for both addition and doubling operations at the same time. This unified formula should be developed using the DPC transformation functions.
3. Study the security-performance tradeoffs of the unified dynamic formula suggested in (1) for different scalar multiplication algorithms for *E/GF(p)*.
4. Study the security-performance tradeoffs of the unified dynamic formula suggested in (2) for different scalar multiplication algorithms for $E/GF(2^m)$.

Appendices

Appendix A-I: Derivation of DPC General Addition Formula for *E/GF(p)*

Transformation functions 7.1 are used to get the dynamic projective coordinates $(X_3, Y_3, Z_3^{L_x}, Z_3^{L_y})$ of the point *R* according to addition formula 3.4 (section 3.3 in chapter 3). The following subsections present the derivation of dynamic projective addition formulas.

A-I.1 Derivation of Dynamic projective *x*-coordinate, X_3.

Let $P = (X_1, Y_1, Z_1^{L_x}, Z_1^{L_y})$, $Q = (X_2, Y_2, Z_2^{L_x}, Z_2^{L_y})$ and $R = (X_3, Y_3, Z_3^{L_x}, Z_3^{L_y})$. Then the dynamic projective coordinate X_3 of the point $R = P + Q$ can be derived as follows:

By applying the dynamic transformation functions 7.1 to the equation of x_3 in 3.4, we get:

$$\frac{X_3}{Z_3^{L_x}} = \left(\frac{\frac{Y_2}{Z_2^{L_y}} - \frac{Y_1}{Z_1^{L_y}}}{\frac{X_2}{Z_2^{L_x}} - \frac{X_1}{Z_1^{L_x}}} \right)^2 - \frac{X_1}{Z_1^{L_x}} - \frac{X_2}{Z_2^{L_x}}$$

Unify denominators to get,

$$= \left(\frac{\frac{Y_2 Z_1^{L_y} - Y_1 Z_2^{L_y}}{Z_1^{L_y} Z_2^{L_y}}}{\frac{X_2 Z_1^{L_x} - X_1 Z_2^{L_x}}{Z_1^{L_x} Z_2^{L_x}}} \right)^2 - \frac{X_1 Z_2^{L_x} + X_2 Z_1^{L_x}}{Z_1^{L_x} Z_2^{L_x}}$$

$$= \left(\frac{(Y_2 Z_1^{L_y} - Y_1 Z_2^{L_y})(Z_1^{L_x} Z_2^{L_x})}{(X_2 Z_1^{L_x} - X_1 Z_2^{L_x})(Z_1^{L_y} Z_2^{L_y})} \right)^2 - \frac{X_1 Z_2^{L_x} + X_2 Z_1^{L_x}}{Z_1^{L_x} Z_2^{L_x}}$$

Let $U = Y_2 Z_1^{L_y} - Y_1 Z_2^{L_y}$ and $V = X_2 Z_1^{L_x} - X_1 Z_2^{L_x}$ then,

$$\frac{X_3}{Z_3^{L_x}} = \frac{U^2 (Z_1^{L_x} Z_2^{L_x})^2}{V^2 (Z_1^{L_y} Z_2^{L_y})^2} - \frac{X_2 Z_1^{L_x} + X_1 Z_2^{L_x}}{Z_1^{L_x} Z_2^{L_x}}$$

$$= \frac{U^2(Z_1^{L_x}Z_2^{L_x})^3 - V^2(Z_1^{L_y}Z_2^{L_y})^2(X_2Z_1^{L_x} + X_1Z_2^{L_x})}{V^2(Z_1^{L_y}Z_2^{L_y})^2 Z_1^{L_x}Z_2^{L_x}}$$

Let $R = (VZ_1^{L_y}Z_2^{L_y})$ and $X_3^{'} = U^2(Z_1^{L_x}Z_2^{L_x})^3 - R^2X_2Z_1^{L_x} - R^2X_1Z_2^{L_x}$, then the above equation can be written as,

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3^{'}}{R^2Z_1^{L_x}Z_2^{L_x}} \qquad \text{A-I.1}$$

A-I.2 Derivation of Dynamic projective *y*-coordinate, Y_3.

By applying the dynamic transformation functions 7.1 to the equation of y_3 in 3.4, we get:

$$\frac{Y_3}{Z_3^{L_y}} = \left(\frac{\frac{Y_2}{Z_2^{L_y}} - \frac{Y_1}{Z_1^{L_y}}}{\frac{X_2}{Z_2^{L_x}} - \frac{X_1}{Z_1^{L_x}}}\right)\left(\frac{X_1}{Z_1^{L_x}} - \frac{X_3}{Z_3^{L_x}}\right) - \frac{Y_1}{Z_1^{L_y}}$$

$$= \left(\frac{(Y_2Z_1^{L_y} - Y_1Z_2^{L_y})(Z_1^{L_x}Z_2^{L_x})}{(X_2Z_1^{L_x} - X_1Z_2^{L_x})(Z_1^{L_y}Z_2^{L_y})}\right)\left(\frac{X_1}{Z_1^{L_x}} - \frac{X_3}{Z_3^{L_x}}\right) - \frac{Y_1}{Z_1^{L_y}}$$

$$= \left(\frac{U(Z_1^{L_x}Z_2^{L_x})}{V(Z_1^{L_y}Z_2^{L_y})}\right)\left(\frac{X_1}{Z_1^{L_x}} - \frac{X_3}{Z_3^{L_x}}\right) - \frac{Y_1}{Z_1^{L_y}}$$

Unify denominators to get,

$$\frac{Y_3}{Z_3^{L_y}} = \left(\frac{U(Z_1^{L_x}Z_2^{L_x})}{V(Z_1^{L_y}Z_2^{L_y})}\right)\left(\frac{X_1Z_3^{L_x} - X_3Z_1^{L_x}}{Z_1^{L_x}Z_3^{L_x}}\right) - \frac{Y_1}{Z_1^{L_y}}$$

$$= \frac{U(Z_1^{L_x}Z_2^{L_x})(X_1Z_3^{L_x} - X_3Z_1^{L_x})}{V(Z_1^{L_y}Z_2^{L_y})Z_1^{L_x}Z_3^{L_x}} - \frac{Y_1}{Z_1^{L_y}}$$

$$= \frac{UZ_2^{L_x}(X_1Z_3^{L_x} - X_3Z_1^{L_x}) - Y_1VZ_2^{L_y}Z_3^{L_x}}{V(Z_1^{L_y}Z_2^{L_y})Z_3^{L_x}}$$

Finally, the above equation can be written as,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Z_3^{L_x}(UZ_2^{L_x}X_1 - Y_1VZ_2^{L_y}) - X_3UZ_2^{L_x}Z_1^{L_x}}{RZ_3^{L_x}} \qquad \text{A-I.2}$$

A-I.3 Choosing Common Z_3.

Let $Z_3 = R^d Z_1 Z_2$, then we can write,

$$\left.\begin{aligned} Z_3^{L_x} &= (R^d Z_1 Z_2)^{L_x} \\ Z_3^{L_y} &= (R^d Z_1 Z_2)^{L_y} \end{aligned}\right\}$$

Based on that, equation A-I.1 can be written as,

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3' R^{dL_x - 2}}{R^{dL_x} Z_1^{L_x} Z_2^{L_x}} \qquad \text{A-I.3}$$

Substitute for $X_3 = X_3' R^{dL_x - 2}$, obtained from A-I.3, in A-I.2 to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Z_3^{L_x}(UZ_2^{L_x}X_1 - Y_1VZ_2^{L_y}) - X_3' R^{dL_x - 2}UZ_2^{L_x}Z_1^{L_x}}{RZ_3^{L_x}}$$

Since $Z_3^{L_x} = R^{dL_x} Z_1^{L_x} Z_2^{L_x}$, $Z_3^{L_x}$ can be taken as a common factor in the numerator and canceled with $Z_3^{L_x}$ in the denominator to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{(UZ_2^{L_x}X_1 - Y_1VZ_2^{L_y}) - X_3' R^{-2}U}{R}$$

Multiply the right hand side by R^2 / R^2 to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{R^2(UZ_2^{L_x}X_1 - Y_1VZ_2^{L_y}) - X_3'U}{R^3}$$

Rearrange the numerator of the above equation to exploit the previously computed terms,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{U(R^2 Z_2^{L_x}X_1 - X_3') - R^2 Z_2^{L_y} Y_1 V}{R^3}$$

According to the selection of $Z_3 = R^d Z_1 Z_2$ which result in $Z_3^{L_y} = (R^d Z_1 Z_2)^{L_y}$, multiply the right hand side of the above equation by $\frac{Z_1^{L_y} Z_2^{L_y}}{Z_1^{L_y} Z_2^{L_y}}$,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Z_1^{L_y} Z_2^{L_y} \left(U(R^2 Z_2^{L_x} X_1 - X_3') - R^2 Z_2^{L_y} Y_1 V\right)}{R^3 Z_1^{L_y} Z_2^{L_y}}$$

Let $Y_3' = Z_1^{L_y} Z_2^{L_y} \left(U(R^2 Z_2^{L_x} X_1 - X_3') - R^2 Z_2^{L_y} Y_1 V\right)$, then the above equation can be written as,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Y_3' R^{dL_y - 3}}{R^{dL_y} (Z_1^{L_y} Z_2^{L_y})} \qquad \text{A-I.4}$$

From equations A-I.3 and A-I.4, we get the following **general dynamic addition formulas**:

$$\left.\begin{aligned}
&X_3 = X_3' R^{dL_x - 2} \\
&Y_3 = Y_3' R^{dL_y - 3} \\
&Z_3^{L_x} = R^{dL_x} T_1 \\
&Z_3^{L_y} = R^{dL_y} T_2 \\
&\textit{where}, \quad U_1 = Y_2 Z_1^{L_y}, \quad U_2 = Y_1 Z_2^{L_y}, \quad U = U_1 - U_2, \\
&\qquad V_1 = X_2 Z_1^{L_x}, \quad V_2 = X_1 Z_2^{L_x}, \quad V = V_1 - V_2, \\
&\qquad T_1 = Z_1^{L_x} Z_2^{L_x}, \quad T_2 = Z_1^{L_y} Z_2^{L_y}, \quad R = VT_2 \\
&\qquad X_3' = UT_1^3 - R^2 V_1 - R^2 V_2, \\
&\qquad Y_3' = T_2\left(U(R^2 V_2 - X_3') - U_2(R^2 V_1 - R^2 V_2)\right)
\end{aligned}\right\} \qquad \text{A-I.5}$$

Appendix B-I: Derivation of DPC General Doubling Formula for *E/GF(p)*

Transformation functions 7.1 are used to get the dynamic projective coordinates $(X_3, Y_3, Z_3^{L_x}, Z_3^{L_y})$ of the point *R* according to doubling formula 3.5 (section 3.3 in chapter 3). The following subsections present the derivation of dynamic projective doubling formulas.

B-I.1 Derivation of Dynamic projective *x*-coordinate, X_3.

Let $P=(X_1,Y_1,Z_1^{L_x},Z_1^{L_y})$ and $R=(X_3,Y_3,Z_3^{L_x},Z_3^{L_y})$. Then the dynamic projective coordinate X_3 of the point *R = 2P* can be derived as follows:

By applying the dynamic transformation functions 7.1 to the equation of x_3 in 3.5, we get:

$$\frac{X_3}{Z_3^{L_x}}=\left(\frac{3\frac{X_1^2}{Z_1^{2L_x}}+a}{2\frac{Y_1}{Z_1^{L_y}}}\right)^2-2\frac{X_1}{Z_1^{L_x}}$$

$$=\left(\frac{\frac{3X_1^2+aZ_1^{2L_x}}{Z_1^{2L_x}}}{2\frac{Y_1}{Z_1^{L_y}}}\right)^2-2\frac{X_1}{Z_1^{L_x}}$$

$$=\left(\frac{\left(3X_1^2+aZ_1^{2L_x}\right)Z_1^{L_y}}{2Z_1^{2L_x}Y_1}\right)^2-2\frac{X_1}{Z_1^{L_x}}$$

Let $W=3X_1^2+aZ_1^{2L_x}$, then

$$\frac{X_3}{Z_3^{L_x}}=\frac{(WZ_1^{L_y})^2}{(2Z_1^{2L_x}Y_1)^2}-2\frac{X_1}{Z_1^{L_x}}$$

$$=\frac{(WZ_1^{L_y})^2-8X_1Z_1^{3L_x}Y_1^2}{(2Z_1^{2L_x}Y_1)^2}$$

Let $S=2Z_1^{2L_x}Y_1$, then

$$\frac{X_3}{Z_3^{L_x}}=\frac{(WZ_1^{L_y})^2-4SX_1Y_1Z_1^{L_x}}{S^2}$$

Let $X_3^{'}=(WZ_1^{L_y})^2-4SX_1Y_1Z_1^{L_x}$, then

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3^{'}}{S^2} \qquad \text{B-I.1}$$

B-I.2 Derivation of Dynamic projective *y*-coordinate, Y_3.

By applying the dynamic transformation functions 7.1 to the equation of y_3 in 3.5, we get:

$$\frac{Y_3}{Z_3^{L_y}} = \left(\frac{3\frac{X_1^2}{Z_1^{2L_x}} + a}{2\frac{Y_1}{Z_1^{L_y}}} \right) \left(\frac{X_1}{Z_1^{L_x}} - \frac{X_3}{Z_3^{L_x}} \right) - \frac{Y_1}{Z_1^{L_y}}$$

$$= \left(\frac{(WZ_1^{L_y})}{(2Z_1^{2L_x}Y_1)} \right) \left(\frac{X_1}{Z_1^{L_x}} - \frac{X_3}{Z_3^{L_x}} \right) - \frac{Y_1}{Z_1^{L_y}}$$

Unify denominators to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{(WZ_1^{L_y})(X_1Z_3^{L_x} - X_3Z_1^{L_x})}{2Z_1^{3L_x}Y_1Z_3^{L_x}} - \frac{Y_1}{Z_1^{L_y}}$$

$$= \frac{Z_1^{L_y}(WZ_1^{L_y})(X_1Z_3^{L_x} - X_3Z_1^{L_x}) - 2Y_1Z_1^{3L_x}Y_1Z_3^{L_x}}{2Z_1^{3L_x}Z_1^{L_y}Y_1Z_3^{L_x}}$$

which can be rearranged to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Z_3^{L_x}(WX_1Z_1^{2L_y} - SZ_1^{L_x}Y_1) - X_3WZ_1^{2L_y}Z_1^{L_x}}{SZ_1^{L_x}Z_1^{L_y}Z_3^{L_x}} \qquad \text{B-I.2}$$

B-I.3 Choosing a common Z_3

Let $Z_3 = S^d Z_1$, then we can write,

$$\left.\begin{aligned} Z_3^{L_x} &= S^{dL_x}Z_1^{L_x} \\ Z_3^{L_y} &= S^{dL_y}Z_1^{L_y} \end{aligned}\right\}$$

Therefore, B-I.1 can be written as,

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3' S^{dL_x-2} Z_1^{L_x}}{S^{dL_x} Z_1^{L_x}} \qquad \text{B-I.3}$$

Substitute for $X_3 = X_3' Z_1^{L_x} S^{dL_x-2}$, obtained in D-I.3, in D-I.2 to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Z_3^{L_x}(WX_1 Z_1^{2L_y} - SZ_1^{L_x} Y_1) - X_3' Z_1^{L_x} S^{dL_x-2} W Z_1^{2L_y} Z_1^{L_x}}{SZ_1^{L_x} Z_1^{L_y} Z_3^{L_x}}$$

Take $Z_3^{L_x}$ as a common factor in the numerator and cancel it with $Z_3^{L_x}$ in the denominator. Note that $Z_3^{L_x} = S^{dL_x} Z_1^{L_x}$,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{(WX_1 Z_1^{2L_y} - SZ_1^{L_x} Y_1) - X_3' S^{-2} W Z_1^{2L_y} Z_1^{L_x}}{SZ_1^{L_x} Z_1^{L_y}}$$

Multiply the right hand side by S^2/S^2 to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{S^2(WX_1 Z_1^{2L_y} - SZ_1^{L_x} Y_1) - X_3' W Z_1^{2L_y} Z_1^{L_x}}{S^3 Z_1^{L_x} Z_1^{L_y}}$$

Take $Z_1^{L_x}$ as a common factor in the numerator and cancel it with $Z_1^{L_x}$ in the denominator (note that $S = 2Z_1^{2L_x} Y_1$),

$$\frac{Y_3}{Z_3^{L_y}} = \frac{2SY_1 Z_1^{L_x}(WZ_1^{2L_y} X_1 - SY_1 Z_1^{L_x}) - X_3' W Z_1^{2L_y}}{S^3 Z_1^{L_y}}$$

Let $T = WZ_1^{2L_y}$. Then rearrange the numerator of the above equation to exploit the previously computed terms.

$$\frac{Y_3}{Z_3^{L_y}} = \frac{T(2SY_1 Z_1^{L_x} X_1 - X_3') - 2(SY_1 Z_1^{L_x})^2}{S^3 Z_1^{L_y}}$$

Let $Y_3' = T(2SY_1 Z_1^{L_x} X_1 - X_3') - 2(SY_1 Z_1^{L_x})^2$, then the above equation can be written as,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Y_3' S^{dL_y-3}}{S^{dL_y} Z_1^{L_y}} \qquad \text{B-I.4}$$

From equations B-I.3 and B-I.4, we get the following **general dynamic doubling formulas**:

$$\left.\begin{aligned} X_3 &= X_3' Z_1^{L_x} S^{dL_x - 2} \\ Y_3 &= Y_3' S^{dL_y - 3} \\ Z_3^{L_x} &= S^{dL_x} Z_1^{L_x} \\ Z_3^{L_y} &= S^{dL_y} Z_1^{L_y} \\ \textit{Where,}\quad W &= 3X_1^2 + aZ_1^{2L_x}, \quad S = 2Z_1^{2L_x} Y_1, \\ T &= WZ_1^{2L_y}, \quad T_1 = SY_1 Z_1^{L_x}, \quad T_2 = 2T_1 X_1 \\ X_3' &= WT - 2T_2 \\ Y_3' &= T(T_2 - X_3') - 2T_1^2 \end{aligned}\right\} \qquad \text{B-I.5}$$

Appendix C-I: Derivation of DPC Optimized Addition Formula for *E/GF(p)*

From equation A-I.2 in appendix A-I, we have:

$$\frac{X_3}{Z_3^{L_x}} = \frac{U^2 (Z_1^{L_x} Z_2^{L_x})^3 - V^2 (Z_1^{L_y} Z_2^{L_y})^2 (X_2 Z_1^{L_x} + X_1 Z_2^{L_x})}{V^2 (Z_1^{L_y} Z_2^{L_y})^2 Z_1^{L_x} Z_2^{L_x}}$$

Take $(Z_1^{L_y} Z_2^{L_y})^2$ as a common factor from the numerator and simplify,

$$\frac{X_3}{Z_3^{L_x}} = \frac{U^2 (Z_1 Z_2)^{3L_x - 2L_y} - X_2 Z_1^{L_x} V^2 - X_1 Z_2^{L_x} V^2}{V^2 Z_1^{L_x} Z_2^{L_x}}$$

Let $X_3' = U^2 (Z_1 Z_2)^{3L_x - 2L_y} - X_2 Z_1^{L_x} V^2 - X_1 Z_2^{L_x} V^2$, then,

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3'}{V^2 Z_1^{L_x} Z_2^{L_x}} \qquad \text{C-I.1}$$

From equation A-I.4 in appendix A-I, we have:

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Z_3^{L_x} (UZ_2^{L_x} X_1 - Y_1 V Z_2^{L_y}) - X_3 U Z_2^{L_x} Z_1^{L_x}}{V (Z_1^{L_y} Z_2^{L_y}) Z_3^{L_x}} \qquad \text{C-I.2}$$

Let $Z_3 = V^d Z_1 Z_2$, then we can write,

$$\left.\begin{aligned} Z_3^{L_x} &= (V^d Z_1 Z_2)^{L_x} \\ Z_3^{L_y} &= (V^d Z_1 Z_2)^{L_y} \end{aligned}\right\}$$

Based on that, equation C-I.1 can be written as,

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3' V^{dL_x - 2}}{V^{dL_x} Z_1^{L_x} Z_2^{L_x}} \qquad \text{C-I.3}$$

Substitute for X_3 from C-I.3 in C-I.2 to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Z_3^{L_x}(UZ_2^{L_x} X_1 - Y_1 V Z_2^{L_y}) - X_3' V^{dL_x - 2} U Z_2^{L_x} Z_1^{L_x}}{V(Z_1^{L_y} Z_2^{L_y}) Z_3^{L_x}}$$

Since $Z_3^{L_x} = V^{dL_x} Z_1^{L_x} Z_2^{L_x}$, $Z_3^{L_x}$ can be taken as a common factor in the numerator and canceled with $Z_3^{L_x}$ in the denominator to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{(UZ_2^{L_x} X_1 - Y_1 V Z_2^{L_y}) - X_3' V^{-2} U}{V(Z_1^{L_y} Z_2^{L_y})}$$

Multiply the right hand side by V^2 / V^2,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{V^2(UZ_2^{L_x} X_1 - Y_1 V Z_2^{L_y}) - X_3' U}{V^3(Z_1^{L_y} Z_2^{L_y})}$$

Rearrange the numerator of the above equation to exploit the previously computed terms,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{U(Z_2^{L_x} X_1 V^2 - X_3') - Y_1 V^3 Z_2^{L_y}}{V^3(Z_1^{L_y} Z_2^{L_y})}$$

Let $Y_3' = U(Z_2^{L_x} X_1 V^2 - X_3') - Y_1 V^3 Z_2^{L_y}$, then the above equation can be written as,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Y_3' V^{dL_y - 3}}{V^{dL_y}(Z_1^{L_y} Z_2^{L_y})} \qquad \text{C-I.4}$$

From equations C-I.3 and C-I.4, we get the following set of **optimized dynamic addition formulas**: note that $-V^2V_1 - V^2V_2 = -V^3 - 2V^2V_2$

$$\left.\begin{aligned}
&X_3 = X_3^{'}V^{dL_x-2}\\
&Y_3 = Y_3^{'}V^{dL_y-3}\\
&Z_3 = V^dT\\
&Z_3^{L_x} = (V^dT)^{L_x}\\
&Z_3^{L_y} = (V^dT)^{L_y}\\
&\textit{where},\quad U_1 = Y_2Z_1^{L_y},\quad U_2 = Y_1Z_2^{L_y},\quad U = U_1 - U_2,\\
&\qquad V_1 = X_2Z_1^{L_x},\quad V_2 = X_1Z_2^{L_x},\quad V = V_1 - V_2,\\
&\qquad T = Z_1Z_2,\quad T_1 = VT^{L_y-L_x}\\
&\qquad X_3^{'} = U^2T^{3L_x-2L_y} - V^3 - 2V^2V_2,\\
&\qquad Y_3^{'} = U(V^2V_2 - X_3^{'}) - U_2V^3
\end{aligned}\right\} \qquad \text{C-I.5}$$

Appendix D-I: Derivation of DPC Optimized Doubling Formula for *E/GF(p)*

From equation B-I.1 in appendix B-I, we have:

$$\frac{X_3}{Z_3^{L_x}} = \frac{(WZ_1^{L_y})^2 - 8X_1Z_1^{3L_x}Y_1^2}{(2Z_1^{2L_x}Y_1)^2}$$

Take $Z_1^{2L_x}$ as a common factor in the numerator and cancel it with $Z_1^{2L_x}$ in the denominator.

$$\frac{X_3}{Z_3^{L_x}} = \frac{\left((W^2Z_1^{2L_y-2L_x}) - 4(2Z_1^{L_x}Y_1)Y_1X_1\right)}{(2Z_1^{L_x}Y_1)^2}$$

Let $S = 2Z_1^{L_x}Y_1$ and $X_3^{'} = (W^2Z_1^{2L_y-2L_x}) - 4SY_1X_1)$, then we can write,

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3^{'}}{S^2} \qquad \text{D-I.1}$$

From equation B-I.3 in appendix B-I, we have:

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Z_3^{L_x}(WX_1Z_1^{2L_y} - 2Z_1^{2L_x}Y_1Z_1^{L_x}Y_1) - X_3WZ_1^{2L_y}Z_1^{L_x}}{2Z_1^{2L_x}Y_1Z_1^{L_x}Z_1^{L_y}Z_3^{L_x}} \qquad \text{D-I.2}$$

Let $Z_3 = S^d$, then we can write,

$$\left.\begin{aligned} Z_3^{L_x} &= S^{dL_x} \\ Z_3^{L_y} &= S^{dL_y} \end{aligned}\right\}$$

Therefore, D-I.1 can be written as,

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3' S^{dL_x-2}}{S^{dL_x}} \qquad \text{D-I.3}$$

Substitute for $X_3 = X_3' S^{dL_x-2}$, obtained in D-I.3, in D-I.2 to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Z_3^{L_x}(WX_1Z_1^{2L_y} - 2Z_1^{2L_x}Y_1Z_1^{L_x}Y_1) - X_3'S^{dL_x-2}WZ_1^{2L_y}Z_1^{L_x}}{2Z_1^{2L_x}Y_1Z_1^{L_x}Z_1^{L_y}Z_3^{L_x}}$$

Take $Z_3^{L_x}$ as a common factor in the numerator and cancel it with $Z_3^{L_x}$ in the denominator. Note that $Z_3^{L_x} = S^{dL_x}$,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{(WX_1Z_1^{2L_y} - 2Z_1^{2L_x}Y_1Z_1^{L_x}Y_1) - X_3'S^{-2}WZ_1^{2L_y}Z_1^{L_x}}{2Z_1^{2L_x}Y_1Z_1^{L_x}Z_1^{L_y}}$$

Multiply the right hand side by S^2/S^2 to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{S^2(WX_1Z_1^{2L_y} - 2Z_1^{2L_x}Y_1Z_1^{L_x}Y_1) - X_3'WZ_1^{2L_y}Z_1^{L_x}}{S^2 2Z_1^{2L_x}Y_1Z_1^{L_x}Z_1^{L_y}}$$

Take $Z_1^{2L_x}Z_1^{L_y}$ as a common factor in the numerator and cancel it with $Z_1^{2L_x}Z_1^{L_y}$ in the denominator (note that $S = 2Z_1^{L_x}Y_1$),

$$\frac{Y_3}{Z_3^{L_y}} = \frac{4Y_1^2(WX_1Z_1^{L_y} - 2Z_1^{2L_x-L_y}Y_1Z_1^{L_x}Y_1) - X_3'WZ_1^{L_y-L_x}}{S^2 2Y_1Z_1^{L_x}}$$

$$= \frac{2SY_1(X_1WZ_1^{L_y-L_x} - 2Y_1^2Z_1^{2L_x-L_y}) - X_3'WZ_1^{L_y-L_x}}{S^3}$$

Let $T = WZ_1^{L_y-L_x}$. Then rearrange the numerator of the above equation to exploit the previously computed terms.

$$\frac{Y_3}{Z_3^{L_y}} = \frac{T(2SY_1X_1 - X_3') - 4(SY_1)Y_1^2Z_1^{2L_x-L_y}}{S^3}$$

Let $Y_3' = T(2SY_1X_1 - X_3') - 4(SY_1)Y_1^2Z_1^{2L_x-L_y}$, then the above equation can be written as,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Y_3'S^{dL_y-3}}{S^{dL_y}} \qquad \text{D-I.4}$$

From equations D-I.3 and D-I.4, we get the following **dynamic optimized doubling formulas**:

$$\left.\begin{aligned} &X_3 = X_3'S^{dL_x-2} \\ &Y_3 = Y_3'S^{dL_y-3} \\ &Z_3 = S^d \\ &Z_3^{L_x} = (S^d)^{L_x} \\ &Z_3^{L_y} = (S^d)^{L_y} \\ &\textit{Where}, \quad W = 3X_1^2 + aZ_1^{2L_x}, \quad S = 2Z_1^{L_x}Y_1, \\ &\qquad T = WZ_1^{L_y-L_x}, \quad T_1 = SY_1, \quad T_2 = 2T_1X_1, \\ &\qquad X_3' = T^2 - 2T_2 \\ &\qquad Y_3' = T(T_2 - X_3') - 4T_1Y_1^2Z_1^{2L_x-L_y} \end{aligned}\right\} \qquad \text{D-I.5}$$

Appendix A-II: Derivation of DPC General Addition Formula for *E/GF(2^m)*

Transformation functions 7.1 are used to get the dynamic projective coordinates $(X_3, Y_3, Z_3^{L_x}, Z_3^{L_y})$ of the point *R* according to addition formula 3.7 (section 3.4 in chapter

3). The following subsections present the derivation of dynamic projective addition formulas.

A-II.1 Derivation of projective *x-coordinate,* X_3.

Let $P=(X_1,Y_1,Z_1^{L_x},Z_1^{L_y})$, $Q=(X_2,Y_2,Z_2^{L_x},Z_2^{L_y})$ and $R=(X_3,Y_3,Z_3^{L_x},Z_3^{L_y})$. Then the projective coordinate X_3 of the point $R = P + Q$ can be derived as follows:

By applying the transformation functions 7.1 to the affine *x-coordinate* equation, x_3, in 3.7, we get:

$$\frac{X_3}{Z_3^{L_x}}=\left(\frac{\frac{Y_2}{Z_2^{L_y}}+\frac{Y_1}{Z_1^{L_y}}}{\frac{X_2}{Z_2^{L_x}}+\frac{X_1}{Z_1^{L_x}}}\right)^2+\left(\frac{\frac{Y_2}{Z_2^{L_y}}+\frac{Y_1}{Z_1^{L_y}}}{\frac{X_2}{Z_2^{L_x}}+\frac{X_1}{Z_1^{L_x}}}\right)+\frac{X_1}{Z_1^{L_x}}+\frac{X_2}{Z_2^{L_x}}+a$$

$$=\left(\frac{(Y_2Z_1^{L_y}+Y_1Z_2^{L_y})(Z_1^{L_x}Z_2^{L_x})}{(X_2Z_1^{L_x}+X_1Z_2^{L_x})(Z_1^{L_y}Z_2^{L_y})}\right)^2+\left(\frac{(Y_2Z_1^{L_y}+Y_1Z_2^{L_y})(Z_1^{L_x}Z_2^{L_x})}{(X_2Z_1^{L_x}+X_1Z_2^{L_x})(Z_1^{L_y}Z_2^{L_y})}\right)+\frac{X_1Z_2^{L_x}+X_2Z_1^{L_x}}{Z_1^{L_x}Z_2^{L_x}}+a$$

Let $U=Y_2Z_1^{L_y}+Y_1Z_2^{L_y}$, $V=X_2Z_1^{L_x}+X_1Z_2^{L_x}$ then,

$$\frac{X_3}{Z_3^{L_x}}=\frac{U^2(Z_1^{L_x}Z_2^{L_x})^2}{V^2(Z_1^{L_y}Z_2^{L_y})^2}+\frac{U(Z_1^{L_x}Z_2^{L_x})}{V(Z_1^{L_y}Z_2^{L_y})}+\frac{V}{Z_1^{L_x}Z_2^{L_x}}+a$$

$$=\frac{U(Z_1^{L_x}Z_2^{L_x})^2\left(U(Z_1^{L_x}Z_2^{L_x})+V(Z_1^{L_y}Z_2^{L_y})\right)+V^2(Z_1^{L_y}Z_2^{L_y})^2\left(V+a(Z_1^{L_x}Z_2^{L_x})\right)}{V^2(Z_1^{L_y}Z_2^{L_y})^2(Z_1^{L_x}Z_2^{L_x})}$$

Let $R=VZ_1^{L_y}Z_2^{L_y}$ and $X_3^{'}=U(Z_1^{L_x}Z_2^{L_x})^2\left(U(Z_1^{L_x}Z_2^{L_x})+R\right)+R^2\left(V+a(Z_1^{L_x}Z_2^{L_x})\right)$, then

$$\frac{X_3}{Z_3^{L_x}}=\frac{X_3^{'}}{R^2(Z_1^{L_x}Z_2^{L_x})} \qquad \text{A-II.1}$$

A-II.2 Derivation of projective *y-coordinate,* Y_3.

By applying the transformation functions 7.1 to the affine *y-coordinate* equation, y_3, in 3.7, we get:

$$\frac{Y_3}{Z_3^{L_y}} = \left(\frac{\frac{Y_2}{Z_2^{L_y}} + \frac{Y_1}{Z_1^{L_y}}}{\frac{X_2}{Z_2^{L_x}} + \frac{X_1}{Z_1^{L_x}}} \right) \left(\frac{X_1}{Z_1^{L_x}} + \frac{X_3}{Z_3^{L_x}} \right) + \frac{X_3}{Z_3^{L_x}} + \frac{Y_1}{Z_1^{L_y}}$$

$$= \left(\frac{(Y_2 Z_1^{L_y} + Y_1 Z_2^{L_y})(Z_1^{L_x} Z_2^{L_x})}{(X_2 Z_1^{L_x} + X_1 Z_2^{L_x})(Z_1^{L_y} Z_2^{L_y})} \right) \left(\frac{X_1}{Z_1^{L_x}} + \frac{X_3}{Z_3^{L_x}} \right) + \frac{X_3}{Z_3^{L_x}} + \frac{Y_1}{Z_1^{L_y}}$$

$$= \left(\frac{U(Z_1^{L_x} Z_2^{L_x})}{V(Z_1^{L_y} Z_2^{L_y})} \right) \left(\frac{X_1}{Z_1^{L_x}} + \frac{X_3}{Z_3^{L_x}} \right) + \frac{X_3}{Z_3^{L_x}} + \frac{Y_1}{Z_1^{L_y}}$$

Unify denominators to get:

$$= \frac{UZ_2^{L_x} \left(X_1 Z_3^{L_x} + X_3 Z_1^{L_x} \right)}{VZ_1^{L_y} Z_2^{L_y} Z_3^{L_x}} + \frac{X_3}{Z_3^{L_x}} + \frac{Y_1}{Z_1^{L_y}}$$

$$= \frac{Z_3^{L_x} \left(UX_1 Z_2^{L_x} + VY_1 Z_2^{L_y} \right) + X_3 \left(UZ_1^{L_x} Z_2^{L_x} + VZ_1^{L_y} Z_2^{L_y} \right)}{VZ_1^{L_y} Z_2^{L_y} Z_3^{L_x}}$$

Finally,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Z_3^{L_x} \left(UX_1 Z_2^{L_x} + VY_1 Z_2^{L_y} \right) + X_3 \left(UZ_1^{L_x} Z_2^{L_x} + R \right)}{RZ_3^{L_x}} \qquad \text{A-II.2}$$

A-II.3 Choosing Common Z_3.

Let $Z_3 = R^d Z_1^{L_x} Z_2^{L_x}$, then we can write,

$$\left. \begin{aligned} Z_3^{L_x} &= (R^d Z_1^{L_x} Z_2^{L_x})^{L_x} \\ Z_3^{L_y} &= (R^d Z_1^{L_x} Z_2^{L_x})^{L_y} \end{aligned} \right\}$$

Based on that, equation A-II.1 can be written as,

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3' R^{dL_x-2}(Z_1^{L_x} Z_2^{L_x})^{L_x-1}}{R^{dL_x}(Z_1^{L_x} Z_2^{L_x})^{L_x}} \qquad \text{A-II.3}$$

Substitute for $X_3 = X_3' R^{dL_x-2}(Z_1^{L_x} Z_2^{L_x})^{L_x-1}$, obtained from A-II.3, in A-II.2 to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Z_3^{L_x}\left(UX_1 Z_2^{L_x} + VY_1 Z_2^{L_y}\right) + X_3' R^{dL_x-2}(Z_1^{L_x} Z_2^{L_x})^{L_x-1}\left(UZ_1^{L_x} Z_2^{L_x} + R\right)}{RZ_3^{L_x}}$$

Since $Z_3^{L_x} = (R^d Z_1^{L_x} Z_2^{L_x})^{L_x}$, $Z_3^{L_x}$ can be taken as a common factor in the numerator and canceled with $Z_3^{L_x}$ in the denominator to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{\left(UX_1 Z_2^{L_x} + VY_1 Z_2^{L_y}\right) + X_3' R^{-2}(Z_1^{L_x} Z_2^{L_x})^{-1}\left(UZ_1^{L_x} Z_2^{L_x} + R\right)}{R}$$

Multiply the right hand side by $R^2(Z_1^{L_x} Z_2^{L_x}) / R^2(Z_1^{L_x} Z_2^{L_x})$ to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{R^2(Z_1^{L_x} Z_2^{L_x})\left(UX_1 Z_2^{L_x} + VY_1 Z_2^{L_y}\right) + X_3'\left(UZ_1^{L_x} Z_2^{L_x} + R\right)}{R^3(Z_1^{L_x} Z_2^{L_x})}$$

According to the selection of $Z_3 = (R^d Z_1^{L_x} Z_2^{L_x})$ which result in $Z_3^{L_y} = (R^d Z_1^{L_x} Z_2^{L_x})^{L_y}$, the above equation can be written as,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{R^{dL_y-3}(Z_1^{L_x} Z_2^{L_x})^{L_y-1}\left(R^2(Z_1^{L_x} Z_2^{L_x})\left(UX_1 Z_2^{L_x} + VY_1 Z_2^{L_y}\right) + X_3'\left(UZ_1^{L_x} Z_2^{L_x} + R\right)\right)}{R^{dL_y}(Z_1^{L_x} Z_2^{L_x})^{L_y}}$$

Let $Y_3' = R^2(Z_1^{L_x} Z_2^{L_x})\left(UX_1 Z_2^{L_x} + VY_1 Z_2^{L_y}\right) + X_3'\left(UZ_1^{L_x} Z_2^{L_x} + R\right)$, then the above equation can be written as,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{R^{dL_y-3}(Z_1^{L_x} Z_2^{L_x})^{L_y-1} Y_3'}{R^{dL_y}(Z_1^{L_x} Z_2^{L_x})^{L_y}} \qquad \text{A-II.4}$$

From equations A-II.3 and A-II.4, we get the following **general dynamic addition formulas**:

$$
\left.
\begin{aligned}
&X_3 = X_3^{'} R^{dL_x-2} T^{L_x-1} \\
&Y_3 = Y_3^{'} R^{dL_y-3} T^{L_y-1} \\
&Z_3^{L_x} = (R^d T)^{L_x} \\
&Z_3^{L_y} = (R^d T)^{L_y} \\
&\textit{where}, \quad U_1 = Y_2 Z_1^{L_y}, \quad U_2 = Y_1 Z_2^{L_y}, \quad U = U_1 + U_2, \\
&\qquad V_1 = X_2 Z_1^{L_x}, \quad V_2 = X_1 Z_2^{L_x}, \quad V = V_1 + V_2, \\
&\qquad R = V Z_1^{L_y} Z_2^{L_y}, \quad T = Z_1^{L_x} Z_2^{L_x}, \quad T_1 = UT, \\
&\qquad X_3^{'} = T_1 T (T_1 + R) + R^2 (V + aT), \\
&\qquad Y_3^{'} = T_2 (UV_2 + VU_2) + X_3^{'} (T_1 + R)
\end{aligned}
\right\} \qquad \text{A-II.5}
$$

Appendix B-II: Derivation of DPC General Doubling Formula for *E/GF(2^m)*

Transformation functions 7.1 are used to get the dynamic projective coordinates $(X_3, Y_3, Z_3^{L_x}, Z_3^{L_y})$ of the point R according to doubling formula 3.8. The following subsections present the derivation of projective doubling formulas.

B-II.1 Derivation of projective *x-coordinate,* X_3.

Let $P = (X_1, Y_1, Z_1^{L_x}, Z_1^{L_y})$ and $R = (X_3, Y_3, Z_3^{L_x}, Z_3^{L_y})$. Then the projective coordinate X_3 of the point $R = 2P$ can be derived as follows:

By applying the transformation functions 7.1 to the affine *x-coordinate* equation, x_3, in 3.8, we get:

$$
\frac{X_3}{Z_3^{L_x}} = \left(\frac{\dfrac{X_1^2}{Z_1^{2L_x}} + \dfrac{Y_1}{Z_1^{L_y}}}{\dfrac{X_1}{Z_1^{L_x}}} \right)^2 + \left(\frac{\dfrac{X_1^2}{Z_1^{2L_x}} + \dfrac{Y_1}{Z_1^{L_y}}}{\dfrac{X_1}{Z_1^{L_x}}} \right) + a
$$

$$
= \frac{\left(X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x} \right)^2}{X_1^2 Z_1^{2L_x} Z_1^{2L_y}} + \frac{\left(X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x} \right)}{X_1 Z_1^{L_x} Z_1^{L_y}} + a
$$

Let $W = X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x}$ and $S = X_1 Z_1^{L_x} Z_1^{L_y}$, then we get:

$$\frac{X_3}{Z_3^{L_x}} = \frac{W(W+S) + aS^2}{S^2}$$

Let $X_3^{'} = W(W+S) + aS^2$, then

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3^{'}}{S^2} \qquad \text{B-II.1}$$

B-II.2 Derivation of projective *y-coordinate,* Y_3.

By applying the transformation functions 7.1 to the affine *y-coordinate* equation, y_3, in 3.8, we can get:

$$\frac{Y_3}{Z_3^{L_y}} = \left(\frac{\frac{X_1^2}{Z_1^{2L_x}} + \frac{Y_1}{Z_1^{L_y}}}{\frac{X_1}{Z_1^{L_x}}} \right) \left(\frac{X_1}{Z_1^{L_x}} + \frac{X_3}{Z_3^{L_x}} \right) + \frac{X_3}{Z_3^{L_x}} + \frac{Y_1}{Z_1^{L_y}}$$

$$= \left(\frac{W}{X_1 Z_1^{L_x} Z_1^{L_y}} \right) \left(\frac{X_1}{Z_1^{L_x}} + \frac{X_3}{Z_3^{L_x}} \right) + \frac{X_3}{Z_3^{L_x}} + \frac{Y_1}{Z_1^{L_y}}$$

Unify denominators to get:

$$\frac{Y_3}{Z_3^{L_y}} = \left(\frac{W\left(X_1 Z_3^{L_x} + X_3 Z_1^{L_x}\right)}{X_1 Z_1^{2L_x} Z_1^{L_y} Z_3^{L_x}} \right) + \frac{X_3}{Z_3^{L_x}} + \frac{Y_1}{Z_1^{L_y}}$$

$$= \frac{W\left(X_1 Z_3^{L_x} + X_3 Z_1^{L_x}\right) + X_3 X_1 Z_1^{2L_x} Z_1^{L_y} + Y_1 X_1 Z_1^{2L_x} Z_3^{L_x}}{X_1 Z_1^{2L_x} Z_1^{L_y} Z_3^{L_x}}$$

Finally,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{X_1 Z_3^{L_x}\left(W + Y_1 Z_1^{2L_x}\right) + X_3 Z_1^{L_x}(W+S)}{S Z_1^{L_x} Z_3^{L_x}} \qquad \text{B-II.2}$$

B-II.3 Choosing a common Z_3

Let $Z_3 = S^d$, then we can write,

$$\left.\begin{aligned} Z_3^{L_x} &= (S^d)^{L_x} \\ Z_3^{L_y} &= (S^d)^{L_y} \end{aligned}\right\}$$

Based on that, equation B-II.1 can be written as,

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3' S^{dL_x - 2}}{S^{dL_x}} \qquad \text{B-II.3}$$

Substitute for $X_3 = X_3' S^{dL_x - 2}$, taken from B-II.3, in B-II.2 to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{X_1 Z_3^{L_x}\left(W + Y_1 Z_1^{2L_x}\right) + X_3' S^{dL_x - 2} Z_1^{L_x}(W + S)}{S Z_1^{L_x} Z_3^{L_x}}$$

Since $Z_3^{L_x} = S^{dL_x}$, $Z_3^{L_x}$ can be taken as a common factor in the numerator and canceled with $Z_3^{L_x}$ in the denominator to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{X_1\left(W + Y_1 Z_1^{2L_x}\right) + X_3' S^{-2} Z_1^{L_x}(W + S)}{S Z_1^{L_x}}$$

Multiply the right hand side by S^2 / S^2 to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{S^2 X_1\left(W + Y_1 Z_1^{2L_x}\right) + X_3' Z_1^{L_x}(W + S)}{S^3 Z_1^{L_x}}$$

Take $Z_1^{L_x}$ as a common factor in the numerator and cancel it with $Z_1^{L_x}$ in the denominator. (we expand S^2 to $S(X_1 Z_1^{L_x} Z_1^{L_y}))$)

$$\frac{Y_3}{Z_3^{L_y}} = \frac{S(X_1^2 Z_1^{L_y})\left(W + Y_1 Z_1^{2L_x}\right) + X_3'(W + S)}{S^3}$$

Let $Y_3' = S(X_1^2 Z_1^{L_y})\left(W + Y_1 Z_1^{2L_x}\right) + X_3'(W + S)$, then the above equation can be written as,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Y_3'}{S^3}$$

Which can be written as,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Y_3' S^{dL_y - 3}}{S^{dL_y}} \qquad \text{B-II.4}$$

From equations B-II.3 and B-II.4, we get the following **dynamic general doubling formulas**:

$$\left.\begin{aligned} X_3 &= X_3' S^{dL_x - 2} \\ Y_3 &= Y_3' S^{dL_y - 3} \\ Z_3^{L_x} &= \left(S^d\right)^{L_x} \\ Z_3^{L_y} &= \left(S^d\right)^{L_y} \\ \textit{Where},\quad T_1 &= X_1^2 Z_1^{L_y}, \quad T_2 = Y_1 Z_1^{2L_x}, \quad W = T_1 + T_2, \quad S = X_1 Z_1^{L_x} Z_1^{L_y} \\ X_3' &= W(W+S) + aS^2 \\ Y_3' &= ST_1(W + T_2) + X_3'(W+S) \end{aligned}\right\} \qquad \text{B-II.5}$$

Appendix C-II: Derivation of DPC Optimized Addition Formula for *E/GF(2^m)*

From equation A-II.2 in appendix A-II, we have,

$$\frac{X_3}{Z_3^{L_x}} = \frac{U(Z_1^{L_x} Z_2^{L_x})^2 \left(U(Z_1^{L_x} Z_2^{L_x}) + V(Z_1^{L_y} Z_2^{L_y})\right) + V^2 (Z_1^{L_y} Z_2^{L_y})^2 \left(V + a(Z_1^{L_x} Z_2^{L_x})\right)}{V^2 (Z_1^{L_y} Z_2^{L_y})^2 (Z_1^{L_x} Z_2^{L_x})}$$

Take $(Z_1^{L_y} Z_2^{L_y})^2$ as a common factor from the numerator and simplify,

$$\frac{X_3}{Z_3^{L_x}} = \frac{U(Z_1 Z_2)^{2L_x - 2L_y} \left(U(Z_1^{L_x} Z_2^{L_x}) + V(Z_1^{L_y} Z_2^{L_y})\right) + V^2 \left(V + a(Z_1^{L_x} Z_2^{L_x})\right)}{V^2 (Z_1^{L_x} Z_2^{L_x})}$$

Further simplification yields,

$$\frac{X_3}{Z_3^{L_x}} = \frac{U(Z_1Z_2)^{3L_x-2L_y}\left(U+V(Z_1Z_2)^{L_y-L_x}\right)+aV^2Z_1^{L_x}Z_2^{L_x}+V^3}{V^2(Z_1^{L_x}Z_2^{L_x})}$$

Let $X_3^{'} = U(Z_1Z_2)^{3L_x-2L_y}\left(U+V(Z_1Z_2)^{L_y-L_x}\right)+aV^2Z_1^{L_x}Z_2^{L_x}+V^3$, then the above equation can be written as,

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3^{'}}{V^2(Z_1^{L_x}Z_2^{L_x})} \qquad \text{C-II.1}$$

From equation A-II.2 in appendix A-II, we have,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Z_3^{L_x}\left(UX_1Z_2^{L_x}+VY_1Z_2^{L_y}\right)+X_3\left(UZ_1^{L_x}Z_2^{L_x}+VZ_1^{L_y}Z_2^{L_y}\right)}{VZ_1^{L_y}Z_2^{L_y}Z_3^{L_x}} \qquad \text{C-II.2}$$

Let $Z_3 = V^dZ_1Z_2$, then we can write,

$$\left.\begin{aligned} Z_3^{L_x} &= (V^dZ_1Z_2)^{L_x} \\ Z_3^{L_y} &= (V^dZ_1Z_2)^{L_y} \end{aligned}\right\}$$

Based on that, equation C-II.1 can be written as,

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3^{'}V^{dL_x-2}}{V^{dL_x}Z_1^{L_x}Z_2^{L_x}} \qquad \text{C-II.3}$$

Substitute for X_3 from C-II.3 in C-II.2 to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Z_3^{L_x}\left(UX_1Z_2^{L_x}+VY_1Z_2^{L_y}\right)+X_3^{'}V^{dL_x-2}\left(UZ_1^{L_x}Z_2^{L_x}+VZ_1^{L_y}Z_2^{L_y}\right)}{VZ_1^{L_y}Z_2^{L_y}Z_3^{L_x}}$$

Since $Z_3^{L_x} = V^{dL_x}Z_1^{L_x}Z_2^{L_x}$, $Z_3^{L_x}$ can be taken as a common factor in the numerator and canceled with $Z_3^{L_x}$ in the denominator to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{\left(UX_1Z_2^{L_x}+VY_1Z_2^{L_y}\right)+X_3^{'}V^{-2}\left(U+V(Z_1Z_2)^{L_y-L_x}\right)}{VZ_1^{L_y}Z_2^{L_y}}$$

Multiply the right hand side by V^2/V^2,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{V^2\left(UX_1Z_2^{L_x} + VY_1Z_2^{L_y}\right) + X_3^{'}\left(U + V(Z_1Z_2)^{L_y - L_x}\right)}{V^3 Z_1^{L_y} Z_2^{L_y}}$$

Rearrange the numerator of the above equation to exploit the previously computed terms,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{U\left(V^2X_1Z_2^{L_x} + X_3^{'}\right) + V^3Y_1Z_2^{L_y} + X_3^{'}V(Z_1Z_2)^{L_y - L_x}}{V^3 Z_1^{L_y} Z_2^{L_y}}$$

Let $Y_3^{'} = U\left(V^2X_1Z_2^{L_x} + X_3^{'}\right) + V^3Y_1Z_2^{L_y} + X_3^{'}V(Z_1Z_2)^{L_y - L_x}$, then the above equation can be written as,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Y_3^{'}V^{dL_y - 3}}{V^{dL_y}(Z_1^{L_y}Z_2^{L_y})} \qquad \text{C-II.4}$$

From equations C-II.1 and C-II.4, we get the following **dynamic optimized addition formulas**:

$$\left.\begin{aligned}
&X_3 = X_3^{'}V^{dL_x - 2}\\
&Y_3 = Y_3^{'}V^{dL_y - 3}\\
&Z_3 = V^dT\\
&Z_3^{L_x} = (V^dT)^{L_x}\\
&Z_3^{L_y} = (V^dT)^{L_y}\\
&\textit{where},\quad U_1 = Y_2Z_1^{L_y},\quad U_2 = Y_1Z_2^{L_y},\quad U = U_1 + U_1,\\
&\qquad V_1 = X_2Z_1^{L_x},\quad V_2 = X_1Z_2^{L_x},\quad V = V_1 + V_1,\\
&\qquad T = Z_1Z_2,\quad T_1 = VT^{L_y - L_x}\\
&\qquad X_3^{'} = UT^{3L_x - 2L_y}\left(U + T_1\right) + aV^2T^{L_x} + V^3\\
&\qquad Y_3^{''} = U\left(V^2V_2 + X_3^{'}\right) + V^3U_2 + X_3^{'}T_1
\end{aligned}\right\} \qquad \text{C-II.5}$$

Appendix D-II: Derivation of DPC Optimized Doubling Formula for *E/GF(2^m)*

From equation B-II.2 in appendix B-II, we have:

$$\frac{X_3}{Z_3^{L_x}} = \frac{(X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x})\left(X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x} + X_1 Z_1^{L_x} Z_1^{L_y}\right) + a(X_1 Z_1^{L_x} Z_1^{L_y})^2}{(X_1 Z_1^{L_x} Z_1^{L_y})^2}$$

Take $Z_1^{2L_y}$ as a common factor in the numerator and cancel it with $Z_1^{2L_y}$ in the denominator.

$$\frac{X_3}{Z_3^{L_x}} = \frac{(X_1^2 + Y_1 Z_1^{2L_x - L_y})\left(X_1^2 + Y_1 Z_1^{2L_x - L_y} + X_1 Z_1^{L_x}\right) + a(X_1 Z_1^{L_x})^2}{(X_1 Z_1^{L_x})^2}$$

Let $S = X_1 Z_1^{L_x}$, $W = X_1^2 + Y_1 Z_1^{2L_x - L_y}$ and $X_3^{'}$ = numerator of the above equation, then we can write,

$X_3^{'} = W(W + S) + aS^2$ and,

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3^{'}}{S^2} \qquad \text{D-II.1}$$

From equation B-II.5 in appendix B-II, we have:

$$\frac{Y_3}{Z_3^{L_y}} = \frac{X_1 Z_3^{L_x}\left(X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x} + Y_1 Z_1^{2L_x}\right) + X_3 Z_1^{L_x}\left(X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x} + X_1 Z_1^{L_x} Z_1^{L_y}\right)}{X_1 Z_1^{L_x} Z_1^{L_y} Z_1^{L_x} Z_3^{L_x}} \qquad \text{D-II.2}$$

Let $Z_3 = S^d$, then we can write,

$$\left.\begin{aligned} Z_3^{L_x} &= (S^d)^{L_x} \\ Z_3^{L_y} &= (S^d)^{L_y} \end{aligned}\right\}$$

Based on that, equation D-II.1 can be written as,

$$\frac{X_3}{Z_3^{L_x}} = \frac{X_3^{'} S^{dL_x - 2}}{S^{dL_x}} \qquad \text{D-II.3}$$

Substitute for $X_3 = X_3' S^{dL_x - 2}$, taken from D-II.3, in D-II.2 to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{X_1 Z_3^{L_x}\left(X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x} + Y_1 Z_1^{2L_x}\right) + X_3' S^{dL_x - 2} Z_1^{L_x}\left(X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x} + X_1 Z_1^{L_x} Z_1^{L_y}\right)}{X_1 Z_1^{L_x} Z_1^{L_y} Z_1^{L_x} Z_3^{L_x}}$$

Since $Z_3^{L_x} = S^{dL_x}$, $Z_3^{L_x}$ can be taken as a common factor in the numerator and canceled with $Z_3^{L_x}$ in the denominator to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{X_1\left(X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x} + Y_1 Z_1^{2L_x}\right) + X_3' S^{-2} Z_1^{L_x}\left(X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x} + X_1 Z_1^{L_x} Z_1^{L_y}\right)}{X_1 Z_1^{L_x} Z_1^{L_y} Z_1^{L_x}}$$

Multiply the right hand side by S^2 / S^2 to get,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{S^2 X_1\left(X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x} + Y_1 Z_1^{2L_x}\right) + X_3' Z_1^{L_x}\left(X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x} + X_1 Z_1^{L_x} Z_1^{L_y}\right)}{S^2 X_1 Z_1^{L_x} Z_1^{L_y} Z_1^{L_x}}$$

Take $Z_1^{L_x}$ as a common factor in the numerator and cancel it with $Z_1^{L_x}$ in the denominator. (recall that $S = X_1 Z_1^{L_x}$)

$$\frac{Y_3}{Z_3^{L_y}} = \frac{S X_1^2\left(X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x} + Y_1 Z_1^{2L_x}\right) + X_3'\left(X_1^2 Z_1^{L_y} + Y_1 Z_1^{2L_x} + X_1 Z_1^{L_x} Z_1^{L_y}\right)}{S^2 X_1 Z_1^{L_x} Z_1^{L_y}}$$

Take $Z_1^{L_y}$ as a common factor in the numerator and cancel it with $Z_1^{L_y}$ in the denominator.

$$\frac{Y_3}{Z_3^{L_y}} = \frac{S X_1^2\left(X_1^2 + Y_1 Z_1^{2L_x - L_y} + Y_1 Z_1^{2L_x - L_y}\right) + X_3'\left(X_1^2 + Y_1 Z_1^{2L_x - L_y} + X_1 Z_1^{L_x}\right)}{S^3}$$

$$= \frac{S X_1^2\left(W + Y_1 Z_1^{2L_x - L_y}\right) + X_3'\left(W + S\right)}{S^3}$$

Let $Y_3' = S X_1^2\left(W + Y_1 Z_1^{2L_x - L_y}\right) + X_3'\left(W + S\right)$, then the above equation can be written as,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Y_3'}{S^3}$$

Which can be written as,

$$\frac{Y_3}{Z_3^{L_y}} = \frac{Y_3' S^{dL_y-3}}{S^{dL_y}} \qquad \text{D-II.4}$$

From equations D-II.3 and D-II.4, we get the following **dynamic optimized doubling formulas**:

$$\left.\begin{aligned} &X_3 = X_3' S^{dL_x-2} \\ &Y_3 = Y_3' S^{dL_y-3} \\ &Z_3 = S^d \\ &Z_3^{L_x} = \left(S^d\right)^{L_x} \\ &Z_3^{L_y} = \left(S^d\right)^{L_y} \\ &\textit{Where}, \quad T = Y_1 Z_1^{2L_x-L_y}, \quad W = X_1^2 + T, \quad S = X_1 Z_1^{L_x} \\ &\qquad X_3' = W(W+S) + aS^2 \\ &\qquad Y_3' = SX_1^2(W+T) + X_3'(W+S) \end{aligned}\right\} \qquad \text{D-II.5}$$

References

[1] N. Koblitz, “Elliptic curve cryptosystems”, Mathematics of Computation, 48 (1987), 203–209.

[2] V. S. Miller, “Use of elliptic curves in cryptography”, Advances in Cryptology Proceedings of Crypto’85, Lecture Notes in Computer Science, 218 (1986), Springer-Verlag, 417–426.

[3] T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms”, IEEE Trans. Inform. Theory, IT-31 (1985), 469–472.

[4] “Proposed federal information processing standard for digital signature standard (DSS)”, Federal Register, 56 No. 169, 30 Aug 1991, 42980–42982.

[5] R. Rivest, A. Shamir and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems”, Communications of the ACM, 21 No. 2 (1978), 120–126.

[6] R. J. McEliece, “Finite Fields for Computer Scientists and Engineers”, Kluwer Academic Publishers, 1987.

[7] D.E. Knuth, “The Art of Computer Programming, 2-semi-numerical Algorithms”, Addison-Wesly, 2nd edition, 1981.

[8] A. Menezes, P. van Oorschot and S. Vanstone, “handbook of Applied Cryptography”, CRC Press, 1997.

[9] I. Blake, G. Seroussi, and N. Smart, “Elliptic Curves in cryptography”, Cambridge University Press, 1999.

[10] K. ltoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara, “Fast implementation of public-key cryptography on a DSP TMS320C6201", In Proceedings of the First Workshop on Cryptographic Hardware and Embedded Systems (CHES'99), LNCS 1717, pp. 61-72, Springer-verlag, 1999.

[11] E. De Win, S. Mister, B. Prennel and M. Wiener, “on the performance of signature based on elliptic curves”. In Algorithmic Number Theory, Proceedings Third Intern. Symp., ANTS-III, LNCS 1423, pp. 252-266, Springer-verlag, 1998.

[12] T. Hasegawa, J. Nakajima and M. Matsui, “a practical implementation of elliptic curve cryptosystems over GF(p) on a 16-bit microcomputer", Public Key Cryptography Proceedings of PKC'98, LNCS 1431, pp. 182-194, Springer-verlag, 1998.

[13] A. Menezes, “Elliptic Curve Public Key Cryptosystems”, Kluwer Academic Publishers, 1993.

[14] N. Koblitz, “A Course in Number Theory and Cryptography”, 2nd edition, Springer- Verlag, 1994.

[15] SEC 1, “elliptic curve cryptography", Standards for Efficiency Cryptography Group, September, 1999. Working Draft. Available at http//:www.secg.org.

[16] S.C. Pohlig and M.E. Hellman, “An improved algorithm for computing logarithms over GF(p) and its cryptographic significance”, IEEE ransactions on Information Theory, 24, pp. 106-110, 1978.

[17] J. Pollard, “Toronto Carlo methods for index computation mod p", Mathematics of Computation, 32, pp. 918-924, 1978.

[18] J. Solinas, “An improved algorithm for arithmetic on a family of elliptic curves (revised)", Technical report CORR 99-06, Department of Combinatorics & Optimization, University of Waterloo, 1999. Available at http//:www.cacr.math.waterloo.ca.

[19] IEEE 141363, “standard specifications for public-key cryptography", ballot draft, 1999. Drafts available at http//:regrouped.ieee.org/groups/l363.

[20] J. Solinas, “Efficient arithmetic in Koblitz curves”, designs, codes and Cryptography, 19, pp. 195 – 249, 2000.

[21] C. H. Lim and P. J. Lee, “more flexible exponentiation with pre-computation", In Advances in Cryptography - CRYPTO '94, LNCS 839, pp. 95-107, Springer-verlag, 1994.

[22] Biljana Cubaleska, Andreas Rieke, and Thomas Hermann, “Improving and extending the Lim/Lee exponentiation algorithm", Proceeding of SAC'99, LNCS, 1999.

[23] Cohen, H., Miyaji, A. and Ono, T., “Efficient Elliptic Curve Exponentiation Using Mixed coordinates”, Advances in Cryptology-Asiacrypt’98, Lecture Notes in Computer Science, Vol. 1514. Springer-Verlag, (1998), pp. 50-65.

[24] H. Cohen, A. Miyaji and T. Ono, “Efficient elliptic curve exponentiation”, Advances in Cryptology Proceedings of ICICS’97, Lecture Notes in Computer Science, 1334 (1997), Springer-Verlag, pp. 282-290.

[25] D. V. Chudnovsky and G. V. Chudnovsky “Sequences of numbers generated by addition in formal groups and new primality and factorization tests” Advances in Applied Math., 7 (1986), pp. 385-434.

[26] Kocher, J. Jaffe and B. Jun “Differential Power Analysis”, Advances in Cryptology: Proceedings of CRYPTO '99, LNCS 1666, Springer-Verlag, (1999) pp. 388-397

[27] C. Kocher,“Timing attacks on Implementations of Diffi-Hellman, RSA, DSS, and other systems”, Crypto’96, LNCS 1109, pp. 104–113, Springer-Verlag, 1996.

[28] L. Goubin, “A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems”, PKC2003, Lecture Notes in Computer Science, 2567(2003), Springer-Verlag, 199-210.

[29] T. Akishita and T. Takagi “Zero-value Point Attacks on Elliptic Curve Cryptosystem”, ISC2003, Lecture Notes in Computer Science, 2851(2003), Springer-Verlag, 218–233.

[30] P.A. Fouque and F. Valette, “The doubling attack: why upwards is better than downwards" In Cryptographic Hardware and Embedded Systems: CHES '03, LNCS 2779, pp. 269-280, Springer-Verlag, 2003.

[31] Hideyo Mamiya, Atsuko Miyaji, Hiroaki Morimoto, “Efficient Countermeasures against RPA, DPA, and SPA” In Cryptographic Hardware and Embedded

Systems: CHES 2004: 6th International Workshop Cambridge, MA, USA, 2004 Proceedings. LNCS, Vol. 3156, pp. 343 – 356, Springer, 2004.

[32] J. Coron, “Resistance against differential power analysis for elliptic curve cryptosystem”, CHES ’99, Lecture Notes in Computer Science, 1717(1999), Springer- Verlag, 292–302.

[33] Izu, T., and Takagi, T., “A fast parallel elliptic curve multiplication resistant against side channel attacks” In Public Key Cryptography - PKC 2002 (2002), vol. 2274 of Lecture Notes in Computer Science, pp. 280 - 296.

[34] J. C. Ha and S. J. Moon, “Randomized signed-scalar multiplication of ECC to resist power attacks" In Cryptographic Hardware and Embedded Systems - CHES '02, LNCS 2523, pp. 551 - 563, Springer-Verlag, 2002.

[35] M. Joye and J. Quisquater, “Hessian elliptic curves and side-channel attacks," In Cryptographic Hardware and Embedded Systems - CHES '01, LNCS 2162, pp.402-410, Springer-Verlag, 2001.

[36] M. Joye and C. Tymen, “Protections against Differential Analysis for Elliptic Curve Cryptography," In Cryptographic Hardware and Embedded Systems - CHES '01, LNCS 2162, pp.377-390, Springer-Verlag, 2001.

[37] N. P. Smart, “An analysis Goubin's refined power analysis attack," Proc. of Cryptographic Hardware and Embedded Systems - CHES '03, LNCS 2779, pp. 281- 290, Springer-Verlag, 2003.

[38] T. Messerges, E. Dabbish, and R. Sloan, “Investigations of Power Analysis Attacks on Smartcards“, preprint, USENIX Workshop on Smartcard Technology, 1999.

[39] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka “Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA”, Cryptographic Hardware and Embedded Systems: Proceedings of CHES '2002, LNCS 2523, Springer-Verlag, (2002) pp. 129-143.

[40] D. May, H.L. Muller, and N.P. Smart, "Random Register Renaming to Foil PA", CHES 2001, LNCS 2162, pp. 28–38, Springer-Verlag, 2001.

[41] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka "A Practical Countermeasure against Address-Bit Differential Power Analysis", Cryptographic Hardware and Embedded Systems: Proceedings of CHES 2003, LNCS 2779, Springer-Verlag, (2003) pp. 382–396.

[42] Nigel Smart, Jacques Stern and David Naccache, "Projective Coordinates Leak", In Advances in Cryptology - EuroCrypt 2004, pages 257-267. Springer Verlag LNCS 3027, April 2004.

[43] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults. In Advances in Cryptology - Eurocrypt '97, volume 1233 of LNCS, pp. 37-51. Springer-Verlag, 1997.

[44] S. P. Scorabogatov and R. J. Anderson. *Optical fault induction attacks.* In B. S. Kaliski Jr., Ç. K. Koç and C. Paar, editors, Proceedings of 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), number 2523 of LNCS, pages 2-12, 2002, Springer-Verlag.

[45] I. Biehl, B. Meyer, and V. Muller, Differential fault attacks on elliptic curve *cryptosystems*, Advances in Cryptology - CRYPTO 2000 (M. Bellare, ed.), Lectures Notes in Computer Science (LNCS), vol. 1880, Springer-Verlag, 2000.

[46] Jean-Jacques Quisquater and David Samyde, *A new tool for non-intrusive analysis of smart cards based on electro-magnetic emissions: the SEMA and DEMA methods*, Eurocrypt rump session, 2000.

[47] Jean-Jacques Quisquater and David Samyde*, Electromagnetic analysis (EMA): measures and countermeasures for smart cards*, Smart cards programming and security (e-Smart 2001), Lectures Notes in Computer Science (LNCS), vol. 2140, Springer, 2001, pp. 200-210.

[48] K. Gandol, C. Mourtel, and F. Olivier, *Electromagnetic analysis: Concrete results*, Proc. of Cryptographic Hardware and Embedded Systems (CHES 2001)

(Cetin Kaya Koc, David Naccache, and Christof Paar, eds.), Lecture Notes in Computer Science, vol. 2162, Springer, 2001, pp. 251{261.

[49] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM Side-Channel(s). In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, page 29. Springer-Verlag, Berlin, 2003.

[50] *NSA tempest series*, Available at http://cryptome.org/#NSA--TS

[51] J.-J. Quisquater and D. Samyde, Automatic code recognition for smartcards using a Kohonen neural network, USENIX Association (ed.), Fifth Working Conference on Smart Card Re*search and Advanced Applications (CARDIS '02), 2002*.

[52] D. Hankerson et al, “Guide to Elliptic Curve Cryptography”, Springer-Verlag, 2004.

VITAE

- Al-Gahtani, Theeb Ayedh
- Completed Bachelor of Science (B.Sc) degree in Computer Engineering from King Saud University (KSU), Riyadh, Saudi Arabia in July 1991.
- Completed Master of Science (MS) degree in Computer Engineering from King Saud University (KSU), Riyadh, Saudi Arabia in June 1997.
- Completed Doctor of Philosophy (Ph.D.) degree in Computer Science and Engineering from King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia in May 2006.