2010-08-08

Efficient Side-Channel Aware Elliptic Curve Cryptosystems over Prime Fields

Deniz Karakoyunlu
Worcester Polytechnic Institute

EFFICIENT SIDE-CHANNEL AWARE ELLIPTIC CURVE CRYPTOSYSTEMS OVER PRIME FIELDS

A Dissertation
Submitted to the Faculty
of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Electrical and Computer Engineering

by

Deniz Karakoyunlu

August 2010

APPROVED:

Prof. Berk Sunar, Major Advisor	Prof. Xinming Huang
Prof. Wenjing Lou	Prof. Erkay Savas

Prof. Fred J. Looft, Department Head

To my family

Abstract

Elliptic Curve Cryptosystems (ECCs) are utilized as an alternative to traditional public-key cryptosystems, and are more suitable for resource limited environments due to smaller parameter size. In this dissertation we carry out a thorough investigation of side-channel attack aware ECC implementations over finite fields of prime characteristic including the recently introduced Edwards formulation of elliptic curves, which have built-in resiliency against simple side-channel attacks. We implement Joye's highly regular add-always scalar multiplication algorithm both with the Weierstrass and Edwards formulation of elliptic curves. We also propose a technique to apply non-adjacent form (NAF) scalar multiplication algorithm with side-channel security using the Edwards formulation. Our results show that the Edwards formulation allows increased area-time performance with projective coordinates. However, the Weierstrass formulation with affine coordinates results in the simplest architecture, and therefore has the best area-time performance as long as an efficient modular divider is available.

Acknowledgements

The work presented in this dissertation was supported in part by National Science Foundation through NSF Cybertrust Grant No. 0831416, NSF-ANI-Career Grant No. 0133297, and in part by Intel Corporation.

I would like to express my utmost gratitude to Professor Berk Sunar for his guidance and support not only in my research and studies, but also in all matters of life. He has been a great advisor to assist me in my career path and help me develop my professional skills, a committed and motivative teacher, a friend, and a role model.

I would also like commend the members of my thesis committee for their valuable assisitance on my dissertation. I would like to thank to Professor Xinming Huang and Professor Wenjing Lou for their guidance throughout my studies at WPI, Professor Erkay Savas for advising me during my undergraduate studies, and during his visit at WPI. Thanks are also due to Professor Fred J. Looft for his prompt response and actions as the department head to help me resolve conflicts and accomplish my objectives at WPI.

I would like to extend my appreciation to Professor Yusuf Leblebici for his consultancy to achieve my career endeavours; Dr. Frank K. Gurkaynak, Dr. Marcelo Kaihara and Professor Arien Lenstra for their support, assistance and inspiration during the implementation period of the work presented in this dissertation.

I am also indebted to my friends and colleagues Ghaith Hammouri, Erdinc Ozturk, Bradford Carleen, Sena Ada, Izzet Sengel, and especially Kahraman Daglar Akdemir and Ferit Ozan Akgul for their continuous support and friendship. My sincere thanks are peculiarly due to Aysegul Gorkem Yalcin for always being there for me.

Finally, I am so proud to be able to have the opportunity to express my respect and love to my family. My parents and my brother Cengiz have beared so many difficulties, and always provided their support and encouragement for me to receive my education and achive my goals.

Contents

Abbreviations and Acronyms **vii**

1 Introduction **1**

2 Background **5**

2.1 Weierstrass Formulation for Elliptic Curves 6

2.2 Edwards Formulation for Elliptic Curves 7

2.3 Projective Coordinate Systems 9

3 Side-Channel Information Leakage **12**

3.1 SCA Countermeasures 13

3.1.1 Countermeasures for Different Point Operations 15

3.1.2 Countermeasures for Unified Point Operations 16

3.2 DCA Countermeasures 18

4 Efficient Mapping to Hardware **20**

4.1 Modular Arithmetic Operations 22

4.1.1 Carry Propagation Free Addition with *SD2* Representation 22

4.1.2 Radix-4 Montgomery Multiplication 23

4.1.3 Extended Binary GCD Modular Division 25

4.1.4 Modular Addition and Subtraction 28

4.1.5 Combined Modular Arithmetic Units 28
4.2 Point Addition and Doubling Operations 30
4.2.1 Weierstrass Affine Point Addition and Doubling 33
4.2.2 Weierstrass Jacobian Point Addition and Doubling 37
4.2.3 Edwards Affine Point Addition and Doubling 43
4.2.4 Edwards Projective Point Addition and Doubling 47
4.2.5 Summary of Point Addition ad Doubling Operations 51
4.3 Point Multiplication Operations 51

5 Results and Performance Comparison 54

5.1 ECC Processor 57

6 Conclusion 61

List of Algorithms

1 Add-Always Scalar Multiplication Algorithm 17
2 Side-Channel Attack Aware NAF Scalar Multiplication Algorithm 18
3 Radix-4 Montgomery Multiplication Algorithm 23
4 Extended Binary GCD Modular Division Algorithm [49] 26
5 Modular Addition and Subtraction Algorithm 28

List of Figures

4.1 Addition of *SD2* Digits Using Carry-Save Adders 23

4.2 Radix-4 Montgomery Multiplier 24

4.3 Extended Binary GCD Modular Divider 27

4.4 Modular Adder & Subtractor 29

4.5 Block Diagram of Point Addition & Doubling Unit 32

4.6 Dataflow for Weierstrass Affine Point Addition 33

4.7 Dataflow for Weierstrass Affine Point Doubling 35

4.8 Dataflow for Weierstrass Jacobian Point Addition 37

4.9 Dataflow for Weierstrass Jacobian Point Doubling 40

4.10 Dataflow for Edwards Affine Unified Point Addition & Doubling 43

4.11 Dataflow for Edwards Affine Optimized Point Doubling 45

4.12 Dataflow for Edwards Projective Unified Point Addition & Doubling . . . 47

4.13 Dataflow for Edwards Projective Optimized Point Doubling 49

4.14 Block Diagram of Point Multiplication Units 53

5.1 Time-Area Space of Point Multiplication 55

5.2 Block Diagram of Elliptic Curve Processor 59

5.3 Elliptic Curve Processor Realized on a Single Chip 60

List of Tables

4.1 Clock Cycles for Modular Arithmetic Operations 30

4.2 Operation Order for Weierstrass Affine Point Addition 34

4.3 Operation Order for Weierstrass Affine Point Doubling 36

4.4 Operation Order for Weierstrass Jacobian Point Addition 38

4.5 Operation Order for Weierstrass Jacobian-Affine Point Addition 39

4.6 Operation Order for Weierstrass Jacobian Point Doubling 41

4.7 Operation Order for Edwards Affine Unified Point Addition & Doubling . 44

4.8 Operation Order for Edwards Affine Optimized Point Doubling 46

4.9 Operation Order for Edwards Projective Unified Point Addition & Doubling 48

4.10 Operation Order for Edwards Projective Optimized Point Doubling 50

4.11 Number of Modular Operations for Point Addition and Doubling Units . . 51

4.12 Number of Arithmetic Operations for Point Multiplication Units 52

5.1 Comparison of Synthesis Results for Point Multiplication [$GF(p)\ 192-bit$] 56

5.2 P&R Results for the ECC Processor 59

5.3 P&R Results for Point Multiplication 60

Abbreviations and Acronyms

ASIC	Application-Specific Integrated Circuit
CMOS	Complementary Metal-Oxide Semiconductor
DCA	Differential Side Channel Analysis
ECC	Elliptic Curve Cryptosystem
ECDLP	Elliptic Curve Discrete Logarithm Problem
FPGA	Field Programmable Gate Array
GCD	Greatest Common Divisor
IC	Integrated Circuit
LSD	Least Significant Digit
MAU	Modular Arithmetic Unit
MCML	Metal-Oxide Semiconductor Current-Mode Differential Logic
MMU	Modular Multiplication Unit
MOS	Metal-Oxide Semiconductor
NAF	Non-Adjacent Form
P&R	Placement and Routing
RBA	Redundant Binary Adder
RSA	Public-Key Cryptography Algorithm of Rivest, Shamir and Adleman
SCA	Simple Side Channel Analysis
SD2	Radix-2 Signed Digit

Chapter 1

Introduction

Modern society largely rely on digital information systems and information storage that depend on cryptographic services to function properly. A variety of cryptographic algorithms are used to implement common cryptographic services such as: confidentiality, integrity, authenticity, access control and non-repudiation. Providing suitable implementation of cryptographic algorithms both in hardware and in software has become an increasingly challenging task. There are two main forms of cryptographic algorithms. Private key algorithms assume that the secret key is (somehow) available to legitimate participants, while public key algorithms allow two (or more) communicating parties to negotiate a secret key on demand. Traditionally, public key cryptographic algorithms are known to have higher computation demands, which reduce their throughput and make them difficult to implement in hardware. However, due to the key distribution problem with private-key algorithms, there is an increasing trend of implementing public-key algorithms in hardware.

In the mid-eighties Neal Koblitz [1] and Victor Miller [2] independently proposed using elliptic curves for public key cryptosystems. Since then, ECC has been intensively studied, and became popular among other common public-key cryptosystems such as

RSA, Diffie-Hellman and ElGamal. In [3], Lenstra and Verheul reported that ECC using a 130-bit key offers comparable security as RSA with a key length of 1024 bits. The shorter parameter size makes ECC especially attractive for embedded applications. However, such devices are more prone to side-channel attacks, since the attacker can procure, isolate, and test such a system without being detected [4–8]. Therefore security against side-channel attacks is considered to be vital for ECC deployed in embedded systems, even though it leads to degradation in performance. Several techniques were proposed for efficient and side-channel attack aware hardware implementation of ECC [9–12]. Unfortunately, these techniques use either specialized fields or specifically chosen elliptic curves. On the other hand, more generic side-channel attack aware implementations involve more complicated equations [13], demand more hardware [14], or leave the system vulnerable to other types of attacks [15–17]. Hence, providing a high performance non-specialized implementation, while retaining a degree of side-channel resiliency remains a challenge.

In 2007, Edwards proposed a novel formulation of elliptic curves and associated point arithmetic operations defined over all non-binary fields [18]. Bernstein and Lange analyzed and compared the complexity (in number of elementary field operations) of basic group operations for different forms of elliptic curves in various coordinate systems [19]. They suggest that the Edwards elliptic curve formulation has superior performance than the fastest known ECC algorithms. Binary Edwards curves also exist [20], but they are not in the scope of this work.

Contributions and Outline

This dissertation presents a comprehensive overview and comparison of parameter agnostic hardware implementations of ECC over finite fields of prime characteristics. In par-

ticular we present optimized hardware realizations of ECC in Weierstrass and Edwards formulations using affine and projective coordinates. We compare these implementations in terms of their area and throughput performance. We also realize them in a ECC processor both with CMOS technology, and power balanced MOS Current-Mode Differential Logic (MCML) technology [21] that provides resiliency against differential side-channel analysis (DCA).

Furthermore, we introduce techniques for improving the performance at various implementation levels without undermining side-channel awareness. In most ASIC arithmetic units, carry chains cause bottlenecks. Our systematic use of redundant digits for all modular arithmetic operations is a significant advantage for reaching higher operating frequencies, therefore we are setting ASIC speed records for prime-field ECC. We implement Marc Joye's recently introduced highly regular Add-Always scalar multiplication algorithm, which is proven to be secure against SCA-type attacks and safe-error attacks [22]. Finally, we introduce a side-channel aware version of NAF scalar multiplication algorithm for Edwards formulation in Algorithm 2.

The organization of this dissertation is as follows. Chapter 2 provides a preliminary introduction to ECC, defines the main parameters, and introduces the new Edwards formulation for ECC. Chapter 3 investigates design of ECC building blocks with side-channel attack precautions. The details for efficient mapping of elliptic curve cryptosystems to hardware are explained in Chapter 4. In Chapter 5, the implementation results are presented. Finally Chapter 6 concludes the dissertation.

Publications relevant to this dissertation:

- D. Karakoyunlu, F. K. Gurkaynak, B. Sunar, Y. Leblebici, "Efficient and Side Channel Aware Implementations of ECC over Prime Fields", IET Information Security, Volume 4, Issue 1, Pages 30-43, 2010.

- S. K. Yoo, D. Karakoyunlu, B. Birand, B. Sunar, ”Improving the Robustness of Ring Oscillator TRNGs”, ACM Transactions on Reconfigurable Technology and Systems, Volume 3, No 2, Article 9, 2010.

- D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, B. Sunar, ”Trojan Detection using IC Fingerprinting”, Proceedings of the 2007 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 2007.

Chapter 2

Background

In this chapter, we briefly present the ECC formulations over finite fields of prime characteristics. We first describe the elliptic curve discrete logarithm problem, which assures the computational security of elliptic curve cryptosystems. Then in the first section, we present the Weierstrass formulation for elliptic curves, and provide the equations for point addition and point doubling on Weierstrass elliptic curves. In the next section, we present the Edwards formulation for elliptic curves, and provide the equations for point addition and point doubling on Edwards elliptic curves. Finally, the third section introduces the projective coordinates, and provides the point addition and doubling equations with projective coordinates both on Weierstrass and Edwards elliptic curves. The reader is referred to [23], for a more detailed treatment of ECC.

In order to construct a cryptographic system, we first need to define a suitable elliptic curve E defined over a prime field $\mathbb{F}_p$ [24]. A cyclic subgroup of $E(\mathbb{F}_p)$ can be generated by selecting a point P of order n, and computing its multiples:

$$\langle P \rangle = \{\infty, P, 2P, 3P, \ldots, (\#n - 1)P\}$$

The elliptic curve discrete logarithm problem (ECDLP) is defined as determining the

value $k \in [1, \#n - 1]$, given a point $P \in E(\mathbb{F}_p)$ of order $\#n$, and a point $Q = kP \in \langle P \rangle$. ECDLP is the underlying number theoretical problem used by ECC. In the cryptosystem, the private key is obtained by selecting an integer k randomly from the interval $[1, \#n-1]$. The corresponding public key will be $Q = kP$, and needs to be calculated by scalar point multiplication.

2.1 Weierstrass Formulation for Elliptic Curves

An elliptic curve E defined over a prime field $\mathbb{F}_p$ (with $p > 3$) can be written in the simplified Weierstrass form as:

$$E(\mathbb{F}_p) \; : \; y^2 = x^3 + ax + b \tag{2.1}$$

where $a, b \in F_p$, and the discriminant of the curve $\Delta = -16(4a^3 + 27b^2) \neq 0$. A point addition operation is defined as adding two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ in $E(\mathbb{F}_p)$ resulting in a third point $P + Q = (x_3, y_3)$ in $E(\mathbb{F}_p)$ with the point at ∞ serving as identity element ($P + \infty = P$). Assuming that $P \neq \pm Q$, the point $P + Q = (x_3, y_3)$ can be calculated as:

$$\begin{aligned} x_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \pmod p \\ y_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1 \pmod p \end{aligned} \tag{2.2}$$

For $P = Q$ the operation is called doubling, and the calculation of $2P = (x_3, y_3)$ is slightly different:

$$
\begin{aligned}
x_3 &= \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \pmod{p} \\
y_3 &= \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1 \pmod{p}
\end{aligned} \tag{2.3}
$$

Finally, if $P = -Q$ the operation results in point at infinity, and it should be handled separately.

2.2 Edwards Formulation for Elliptic Curves

In [18], Edwards showed that elliptic curves over a prime field $\mathbb{F}_p$ (with $p > 3$) in the normal form:

$$
E(\mathbb{F}_p) \; : \; x^2 + y^2 = c^2(1 + dx^2y^2) \tag{2.4}
$$

are bi-rationally equivalent to Weierstrass elliptic curves, and can be efficiently transformed from the short Weierstrass form given in Equation (2.1). The parameter c can be chosen as 1 without loss of generality. Therefore, it will be assumed to be 1 in subsequent chapters. Bernstein and Lange introduced explicit equations for performing the transformation of the ECC coordinates from Weierstrass to Edwards as well as for performing the group operations on an Edwards curve [19]. The most attractive property of the Edwards formulation is that the same point addition operation can be used even if the two points on the curve are equal:

$$
\begin{aligned}
x_3 &= \frac{x_1 y_2 + y_1 x_2}{1 + d x_1 x_2 y_1 y_2} \pmod{p} \\
y_3 &= \frac{y_1 y_2 - x_1 x_2}{1 - d x_1 x_2 y_1 y_2} \pmod{p}
\end{aligned}
\tag{2.5}
$$

Whereas, in the Weierstrass elliptic curve formulation a separate doubling operation as shown in Equation (2.3) is needed when $P = Q$, and special handling of point at infinity is needed when $P = -Q$. Since only a single type of operation is used, it is reasonable to expect a higher performance from side-channel attack aware ECC implementations using the Edwards formulation when compared to those using the Weierstrass formulation. In addition, in the Edwards formulation, there is no special point at ∞, removing another special case that has to be handled by implementations. The Edwards doubling formulation can also be further simplified by using the Edwards elliptic curve definition and rewriting dx^2y^2 as $x^2 + y^2 - 1$ as suggested by Marc Joye to Bernstein et al. in [19]. This optimization makes the point addition and doubling asymmetric, taking away the side-channel resiliency advantage of unified addition and doubling operations. Nevertheless, Edwards formulation with optimized doubling operations may be utilized with a side-channel aware multiplication algorithm as in the case of Weierstrass formulation:

$$
\begin{aligned}
x_3 &= \frac{2 x_1 y_1}{x_1^2 + y_1^2} \pmod{p} \\
y_3 &= \frac{x_1^2 - y_1^2}{x_1^2 + y_1^2 - 2} \pmod{p}
\end{aligned}
\tag{2.6}
$$

2.3 Projective Coordinate Systems

ECC implementations may be viewed at several layers. At the point level the main operation is the scalar-point multiplication, which is realized with multiple point additions and point doubling operations. Each point addition and doubling involves a number of elementary modular arithmetic operations. Modular addition and subtraction are relatively straightforward to implement. Modular multiplication is a reasonably costly operation. At the arithmetic level the implementation of the modular inversion is the most costly operation . The high cost of modular inversion has motivated the investigation of alternative coordinate representations, which avoid the inversion operation at a cost of increased number of field multiplications and additions. The classical formulation where a point P on an elliptic curve E is represented by a pair of elements (x, y) is known as the affine coordinate representation. The affine coordinates can be transformed into projective coordinates that use three elements to represent a point (X, Y, Z), allowing the numerator and the denominator to be calculated separately.

A number of projective coordinate transformations have been proposed in the literature: homogeneous projective, Jacobian, Chudnovsky Jacobian [25], Lopez-Dahab [26], and mixed coordinates [27]. Homogeneous projective coordinates are rarely used in Weierstrass formulation, since the number of multiplications required in exchange for avoiding the inversion is too high. However, Jacobian projective coordinates turn out to be more efficient and most commonly applied either as is, or in a mixed form with affine coordinates. On the other hand, due to the balanced form of equations, homogeneous projective coordinate work well on Edwards elliptic curves.

A Weierstrass elliptic curve defined in Equation (2.1) is converted to Jacobian coordinates as follows:

$$E(\mathbb{F}_p) \; : \; Y^2 = X^3 + aXZ^4 + bZ^6$$

where $X = xZ^2$, $Y = yZ^3$. Then the point addition (Equation 2.7) and doubling (Equation 2.8) formulations with Jacobian coordinates become [25]:

$$\begin{aligned} X_3 &= (Y_2Z_1^3 - Y_1Z_2^3)^2 - (X_2Z_1^2 - X_1Z_2^2)^2(X_2Z_1^2 + X_1Z_2^2) \pmod p \\ 2Y_3 &= (Y_2Z_1^3 - Y_1Z_2^3)[(X_2Z_1^2 - X_1Z_2^2)^2(X_2Z_1^2 + X_1Z_2^2) - 2X_3] \\ &\quad -(X_2Z_1^2 - X_1Z_2^2)^3(Y_2Z_1^3 + Y_1Z_2^3) \pmod p \\ Z_3 &= (X_2Z_1^2 - X_1Z_2^2)Z_1Z_2 \pmod p \end{aligned} \tag{2.7}$$

$$\begin{aligned} X_3 &= (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2 \pmod p \\ Y_3 &= (3X_1^2 + aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4 \pmod p \\ Z_3 &= 2Y_1Z_1 \pmod p \end{aligned} \tag{2.8}$$

The addition formulation can be optimized by removing Z_2 values, if one of the points is affine (i.e. $Z_2 = 1$), resulting in so-called mixed point addition. An Edwards elliptic curve defined in Equation (2.4) is converted to homogeneous projective coordinates as follows:

$$E(\mathbb{F}_p) \,:\, X^2 + Y^2 = Z^4 + dX^2Y^2$$

where $X = xZ$, $Y = yZ$. The following formulas compute the unified point addition and doubling (Equation 2.9), and optimized doubling (Equation 2.10) operations with projective coordinates [19]. Similar to Jacobian coordinates, addition can be optimized in the case of mixed coordinates.

$$
\begin{aligned}
X_3 &= Z_1 Z_2 (X_1 Y_2 + Y_1 X_2)(Z_1^2 Z_2^2 - d X_1 X_2 Y_1 Y_2) \pmod p \\
Y_3 &= Z_1 Z_2 (Y_1 Y_2 - X_1 X_2)(Z_1^2 Z_2^2 + d X_1 X_2 Y_1 Y_2) \pmod p \\
Z_3 &= (Z_1^2 Z_2^2 - d X_1 X_2 Y_1 Y_2)(Z_1^2 Z_2^2 + d X_1 X_2 Y_1 Y_2) \pmod p
\end{aligned}
\tag{2.9}
$$

$$
\begin{aligned}
X_3 &= 2 X_1 Y_1 (X_1^2 + Y_1^2 - 2 Z_1^2) \pmod p \\
Y_3 &= (X_1^2 - Y_1^2)(X_1^2 + Y_1^2) \pmod p \\
Z_3 &= (X_1^2 + Y_1^2)(X_1^2 + Y_1^2 - 2 Z_1^2) \pmod p
\end{aligned}
\tag{2.10}
$$

Chapter 3

Side-Channel Information Leakage

In this chapter, we explain the sources of side-channel information leakage, classify the side-channel attacks, and provide algorithmic and implementation countermeasures. In the first section, we point out how data dependency and conditional executions play a factor in side-channel information leakage. We discuss the methods to uniformly order the distinguishable point operations for achieving a scalar point multiplication that withstands the simple side-channel attacks. We introduce a secure version of NAF algorithm to work with unified point addition and doubling operations of Edwards elliptic curves. The NAF algorithm improves the run time of scalar point multiplication by 25%, while our contribution allows it to retain resiliency against simple side-channel attacks. In the second section, we talk about the countermeasures against differential side channel attacks.

The security of an algorithm is measured in terms of the effort required by the attacker to extract the secret information. Good cryptographic algorithms are based on number theoretic problems that have a well studied computational complexity, hence the effort to break the algorithm is well-known. Algorithm parameters are chosen so that practical attacks are rendered infeasible. However, once an algorithm is implemented in hardware or in software, the implementation acquires physical properties such as power consumption,

electromagnetic radiation, surface temperature, or time required to complete an operation. All these properties that can be observed to vary while cryptograpic operations are processed, are side channel information sources, which can potentially be used by an adversary to reveal parts of the secret key. Side-channel attacks can be classified into simple side-channel attacks (SCA), which directly interpret data characteristics that are visible in a single or a few measurement traces, and differential side-channel attacks (DCA), which interpret the side-channel differences of correlated measurements. Side-channel attacks can be further enhanced by applying statistical methods over a template of measurement traces [28]. While it seems impossible to foresee all possible side-channel attacks that might emerge in the future, we believe that cryptographic architectures should be designed to withstand side-channel leakage. We call such architectures side-channel aware.

3.1 SCA Countermeasures

Side-channel awareness starts with preventing SCA, which requires avoiding conditional executions and data-dependent run times in all levels of an implementation. An elliptic curve cryptosystem is based on multiplying a point on the elliptic curve with an n-bit scalar. The scalar point multiplication is realized by point addition and doubling operations, which involve a number of elementary modular arithmetic operations. The first step of side-channel awareness is developing constant run-time modular arithmetic operations. This requires removing data dependent optimizations, and achieving constant run-time operations regardless of the inputs. In Section 4.1, we present our implementations of modular mutliplication, division, addition and subraction operations.

The next step is to design side-channel aware point addition and doubling units that realize the formulations in Equations (2.2), (2.3), (2.5), (2.6), (2.7), (2.8), (2.9), and (2.10). The side-channel aware modular arithmetic units allow each point operation to

be constant-time regardless of the input point coordinates. However, the different formulations of Weierstrass point addition and doubling operations make the side channel characteristics of point addition and doubling distinguishable, which requires the point multiplication algorithm to uniformly order the additions and doublings in order to achieve side-channel awareness (i.e. regular point multiplication algorithm). This requirement also applies to optimized doublings for Edwards formulation, since the optimized point doubling is also different from point addition. On the other hand, Edwards formulation allows using a unified point operation both for addition and doubling. In this case, the point multiplication algorithm does not need to be regular, and therefore a faster irregular multiplication algorithm can be utilized. The hardware implementation details of point addition and doubling units are further investigated in Section 4.2.

The so-called binary multiplication methods provide a systematic way of ordering the addition and doubling operations. In a typical binary multiplication scheme, as the bits of the multiplicand are processed sequentially, a point doubling is performed for each bit, and a point addition is performed if the current bit is equal to one. Hence, the run time of the binary multiplication scheme depends on the number of non-zero bits of the multiplicand. On average, for a multiplicand of bit length n, the point multiplication requires n doubling operations and $\frac{n}{2}$ point additions. A more advanced binary multiplication method requires the scalar multiplicand to be recorded into the non-adjacent signed digit form (NAF) [29]. A NAF binary number will not have two consecutive non-zero digits (1 or -1 in signed digit form), reducing the number of point additions to less than $\frac{n}{2}$ throughout a n-bit scalar point multiplication ($\frac{n}{3}$ point additions on average). While the number of doublings remains constant, the NAF method leads to a modest linear improvement in the number of point additions.

Both the standard binary multiplication scheme and the NAF scheme conditionally perform a point addition (or subtraction) driven by the binary digit values of the secret

multiplicand. Side-channel characteristics of distinguishable point addition and doubling operations can be observed to vary while a point multiplication is carried out, which can potentially be used by an adversary to reveal parts of the secret key [30, 31]. Moreover, even if the point addition and doubling operations are indistinguishable as in unified Edwards formulation, the total run time of the point multiplication still depends on the number of non-zero binary digits of the multiplicand, since point addition is only carried out when a digit is non-zero. In this case, an attacker observing the run time, could determine the Hamming weight of the multiplicand, reducing the possible solution space significantly. In the following subsections, we present appropriate point multiplication methods for different and unified point addition and doubling operations.

3.1.1 Countermeasures for Different Point Operations

When the point addition and doubling operations are different, the only way to make a point multiplication side-channel attack aware is to use a uniform sequence of point operations that do not depend on the value of the multiplicand. A method proposed by Moller performs point multiplication with fixed pattern of doublings and additions with less than $2n$ point operations in total [17], but it involves a fixed look-up table that makes the system susceptible to statistical attacks described in [32]. Recognizing this problem, he proposes a new method to avoid a fixed table [14], employing a randomized initialization stage to achieve resistance against side-channel attacks. However, when a random number generator is not incorporated, one has to use a regular multiplication algorithm that involves one point addition and one point doubling for each binary digit of the multiplicand to avoid revealing the order and number of the non-zero digits.

One solution to achieve a regular multiplication algorithm is to introduce point addition operations when the binary digit is zero [15, 16], or inserting dummy atomic operations to achieve side-channel atomicity [33]. However, this is not always a trivial task.

If the operations are dummy, they are vulnerable to fault insertion attacks, where the attacker deliberately introduces a fault during an operation and monitors the output for a change. If the correct output is produced in the presence of faults, the attacker will be able to conclude that the operation, where the fault was introduced, was a dummy operation [34, 35].

The so-called Montgomery binary ladder [36] protects against SCA and fault insertion attacks, since it is highly regular and does not involve dummy operations. Recent studies has shown that processing the bits of multiplicand from left-to-right, as in Montgomery ladder, are also vulnerable to certain attacks [37, 38]. In 2007, Joye introduced the add-always [1] binary scalar multiplication algorithm [22]. This new algorithm (Algorithm 1) is highly regular, processed from right-to-left, and it requires no precomputation or prior recoding. Add-always multiplication algorithm always requires n point doublings and n point additions regardless of the value of the scalar multiplicand, and two temporary registers are needed to store the results of each iteration. We have utilized the Add-always algorithm in our implementations where point addition and doubling operations are different. It should be noted that the standard left-to-right algorithm with dummy operations allows accumulating the multiplication result in only one register, and using mixed-coordinates since the coordinates of the input point is kept intact (Z-coordinate of the input point will be 1). Nevertheless, dummy operations and left-to-right processing should be avoided, due to the vulnerabilities described above.

3.1.2 Countermeasures for Unified Point Operations

In the case of distinguishable point addition and doubling operations, a side-channel attack aware point multiplication requires using a regular point multiplication algorithm that

[1] Also referred as: always add-and-double algorithm.

Algorithm 1 Add-Always Scalar Multiplication Algorithm

Inputs: $P \in F_p$ and $k = (k_{n-1}, \ldots, k_0)_2 \in N$
Output: $Q = kP \in F_p$

1: $R_0 := 0$; $R_1 := P$;
2: **for** $j = 0$ **to** $n - 1$ **do**
3: $b := 1 - k_j$; $R_b := 2R_b$;
4: $R_b := R_b + R_{k_j}$;
5: **end for**
6: **return** R_0

consists of n doubling operations and n point additions for a multiplicand of bit length n. On the other hand, the Edwards formulation allows unified point addition and doubling without requiring specialized elliptic curves, or any randomization or initialization stage. When the point addition and doubling operations are unified, even if the scalar-point multiplication algorithm is irregular it will not cause simple side-channel leakage as long as the total number of operations is constant. As it is mentioned in the beginning of this section, the NAF point multiplication algorithm always requires fewer than $\frac{n}{2}$ point additions. By carrying out necessary number of extra operations after finishing the point multiplication, the total run-time could be set to the worst case in order to prevent the dependency on the multiplicand value. However, if these extra operations do not update the value of the result, the system will be vulnerable to fault-insertion attacks as explained in Subsection 3.1.1. In Algorithm 2, we propose a method that computes the extra operations at the end of a NAF multiplication, where the additional operations do affect the computed result. Therefore, the algorithm is also robust against fault insertion attacks. The first 7 lines of the algorithm compute the NAF point multiplication, with the result stored in R_0. Note that the addition and subtraction operations in lines 4 and 5 are virtually the same operations on elliptic curves. Moreover, our implementation uses radix-2 signed-digit (*SD2*) redundant representation, which makes it trivial to achieve indistinguishable addition and subtraction operations. All we need to is to swap the wiring of *SD2* representation of the

second operand digits in the case of point subtraction. In line 8, the number of necessary extra operations is calculated and stored in r, and the register R_0 is updated by the sum of R_0 and R_1. After this addition, the result can be expressed as: $result = R_0 - R_1$. Throughout the *for-loop* in lines 9-12, both R_0 and R_1 are continuously updated for $\frac{r}{2}$ iterations, so that $2\left\lfloor\frac{r}{2}\right\rfloor$ extra operations are carried out, while $result = R_0 - R_1$ still holds. Finally in line 12, the result is recomputed by the subtraction: $R_0 - R_1$. The addition in line 13 is conditionally performed in order to achieve $r + 2$ extra point additions in total regardless of r being odd or even. Hence, the computations will end after $\frac{3n}{2} + 2$ unified point operations.

Algorithm 2 Side-Channel Attack Aware NAF Scalar Multiplication Algorithm

Inputs: $P \in F_p$ and $k = (k_{n-1}, \ldots, k_0)_2 \in N$
Output: $Q = kP \in F_p$

1: $R_0 := 0;\ \ R_1 := P;\ \ a := 0;\ \ r := 0;$
2: **for** $j = 0$ **to** $n - 1$ **do**
3: Recode k_j on the fly into non-adjacent signed-digit form with Reitwiener's method [29].
4: **if** $(k_j = \ \ 1)$ **then** $R_0 := R_0 + R_1;\ \ a := a + 1;$ **end if**
5: **if** $(k_j = -1)$ **then** $R_0 := R_0 - R_1;\ \ a := a + 1;$ **end if**
6: $R_1 := R_1 + R_1;$
7: **end for**
8: $R_0 := R_0 + R_1;\ \ r := \frac{n}{2} - a;$
9: **for** $j = 0$ **to** $\frac{r}{2}$ **do**
10: $R_0 := R_0 + R_1;\ \ R_1 := R_1 + R_1;$
11: **end for**
12: $R_0 := R_0 - R_1;$
13: **if** r **is** odd **then** $R_1 := R_1 + R_1;$ **end if**
14: **return** R_0

3.2 DCA Countermeasures

Differential side-channel analysis allows more powerful attacks that succeed even in the presence of SCA countermeasures [39]. Comprehensive information about performing

differential side-channel analysis can be found in [40]. Several classes of countermeasures were proposed against DCA, e.g. using noise generators to confuse attackers by adding random noise to the power signature [41], feeding idle datapath units with random data to provide a more uniform data profile [42], using masking techniques [43, 44], and finally using power balanced IC libraries that have data independent power consumption characteristics [45–47]. For enhanced robustness against side-channel attacks, the design may be synthesized with such precautions at the circuit level. We synthesized our design both with standard CMOS technology, and with the power balanced MOS Current-Mode Differential Logic (MCML) technology [21].

At the algorithm level, DCA resiliency can be achieved by multiplying the point by a random number prior to each point addition or doubling with projective coordinates. In Section 2.3, we have stated that homogeneous projective coordinates (xZ, yZ, Z) are more suitable for the Edwards formulation, whereas the Jacobian coordinates, i.e. (xZ^2, yZ^3, Z) are more suitable for the Weierstrass formulation. Prior to each point operation, randomization can be carried out by replacing the point coordinates (X, Y, Z) with $(\lambda X : \lambda Y : \lambda Z)$ in the case of homogeneous projective coordinates, and with $(\lambda^2 X : \lambda^3 Y : \lambda Z)$ in the case of the Jacobian coordinates, where $\lambda \neq 0$ is a random number [15]. Hence, if projective randomization is utilized, the Edwards formulation has further performance benefit of requiring fewer field multiplications, since there is no need for computing the square and cube of the random number prior to each point addition or doubling. We do not apply projective randomization, since we do not want to incorporate a random number generator in our hardware implementation.

Chapter 4

Efficient Mapping to Hardware

This chapter provides the details for efficient mapping of elliptic curve cryptosystems to hardware. The mapping process involves a bottom-up methodology. We first design the modular arithmetic units. Later, we design elliptic curve point addition and doubling units utilizing the modular arithmetic units. Then, we design elliptic curve point multiplication units on top of the point addition and doubling units. The organization of this chapter is as follows:

We first present methods for efficient modular arithmetic in Section 4.1. Our goal is to achieve the lowest possible area-time product by careful selection and implementation of modular arithmetic algorithms, while making sure that the arithmetic operations always have constant run-time independent of the data being processed. Since we are dealing with large operand sizes, our first goal is to reduce the carry propagation in additions. Our choice of using redundant binary adder, which utilizes carry save adders with operands in radix-2 signed digit (SD2) representation, allows us to completely avoid the carry propagation. Therefore, we are able to achieve single clock cycle addition regardless of the size or value of the operands. Hence, both the side-channel leakage is avoided, and a very fast addition operation is achieved at the cost of doubling the area in comparison with the area

of a ripple-carry adder. For multiplication, we employ radix-4 Montgomery multiplication that processes 2 digits in each iteration, and completes in $\frac{n}{2} + 2$ iterations regardless of the length n of the operands. For modular division, we employ two different methods. The first division method computes the division with exponentiation using Fermat's theorem: $Z^{-1} = Z^{p-2} \pmod{p}$, where $gcd(Z, p) = 1$ [48]. In this method, the division is very costly in terms of time ($\frac{3n^2}{4} + 3n$ iterations on average), but there is no additional area cost. The second division method computes the division using extended binary GCD algorithm, and completes in only $2n+4$ iterations at the cost of 50% increase in area. Our implementation of modular division with binary GCD algorithm continues to iterate until the control register is fully processed as suggested in [49], hence it has constant run-time regardless of the input data. For modular addition and subtraction, we modify the modular addition method described in [50] to work with the redundant binary adders, and to have compatible operand range with the modular multiplication and division algorithms.

In Section 4.2, we present the implementation details of ECC point addition and doubling operations for Weierstrass elliptic curves and Edwards elliptic curves both with affine coordinates and projective coordinates. The point addition and doubling equations were presented in Chapter 2. In this chapter, we map these equations to hardware, which requires careful scheduling of modular arithmetic operations in order to complete the point operation in least possible number of iterations and with smallest possible number of temporary storage registers.

Section 4.3 gives the details of ECC point multiplication operations. Based on the side channel leakage properties of binary multiplication methods that were examined in Chapter 3, we apply the add-always algorithm for all different ECC configurations (Weierstrass affine point multiplication, Weierstrass Jacobian point multiplication, Edwards affine point multiplication with unified point operations, Edwards affine point multiplication with optimized point doublings, Edwards projective point multiplication with

unified point operations, Edwards projective point multiplication with optimized point doublings), and we apply our enhancement to NAF algorithm for Edwards affine and projective point multiplications with unified point operations.

4.1 Modular Arithmetic Operations

Modular arithmetic operations are the core operations of ECC. In order to improve the performance of the overall system, it is crucial to optimize the modular arithmetic operations. Moreover, in order to avoid side-channel information leakage, operation run times should not be data dependent. As the first step of ECC implementation, we have designed the following modular arithmetic components.

4.1.1 Carry Propagation Free Addition with *SD2* Representation

Addition is the primary building block in implementing arithmetic operations. If addition is slow or area-expensive, all other operations suffer from this. In order to achieve parallel addition of two n-digit redundant binary numbers in constant time without carry propagation, we used the radix-2 signed digit (SD2) representation that uses the digit set $\{$-1,0,1$\}$ [51], and carry propagation free addition as proposed in [52]. Figure 4.1 shows addition of 2 consecutive SD2 digits using carry save adders. It can be observed that, signals do not propagate through more than 2 full adders. The n-digit redundant binary adders (RBAs) are realized by cascading 4-to-2 signed-digit carry-save adders as presented in [53], which allowed us to keep the critical delay path of computing n-digit addition within the delay of two full-adders. The RBA is used as the primary building block in the implementation of the modular arithmetic operations.

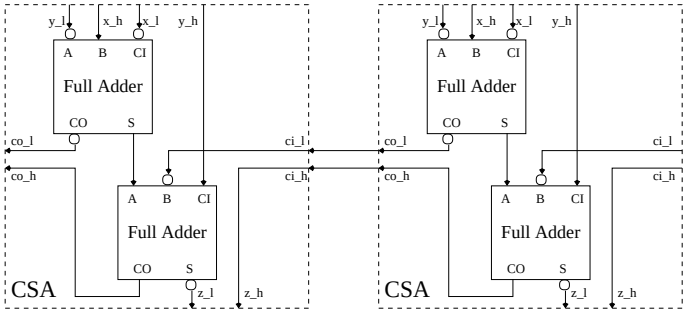


Figure 4.1: Addition of *SD2* Digits Using Carry-Save Adders

4.1.2 Radix-4 Montgomery Multiplication

A constant run-time radix-2 Montgomery modular multiplier that uses redundant representation is presented in [49]. We have modified this algorithm to perform multiplication in radix-4, reducing the run time by a factor of 2. This multiplier works by computing 5 steps for each radix-4 digit as presented in Algorithm 3, where LSD stands for least significant digit.

Algorithm 3 Radix-4 Montgomery Multiplication Algorithm

Inputs: X:= Multiplier, Y:= Multiplicand, M:= Modulus
Output: Z:= Result

Z:$=0$;
for $i=1$ **to** $\frac{n}{2}$
Step 1: a:= LSD(X), X$\gg 2$;
Step 2: P:= a$*$Y; (where: a $\in \{-2,-1,0,1,2\}$)
Step 3: Z:= Z$+$P;
Step 4: Z:= Z$+$q$*$M; (where: q $\in \{-2,-1,0,1,2\}$, so that LSD(Z)$=0$)
Step 5: Z$\gg 2$;
end for
return Z;

Step 2 requires only a single or double left shift of *SD2* digits, whereas Step 3 and Step 4 require n-digit redundant binary addition operations. The modular multiplier com-

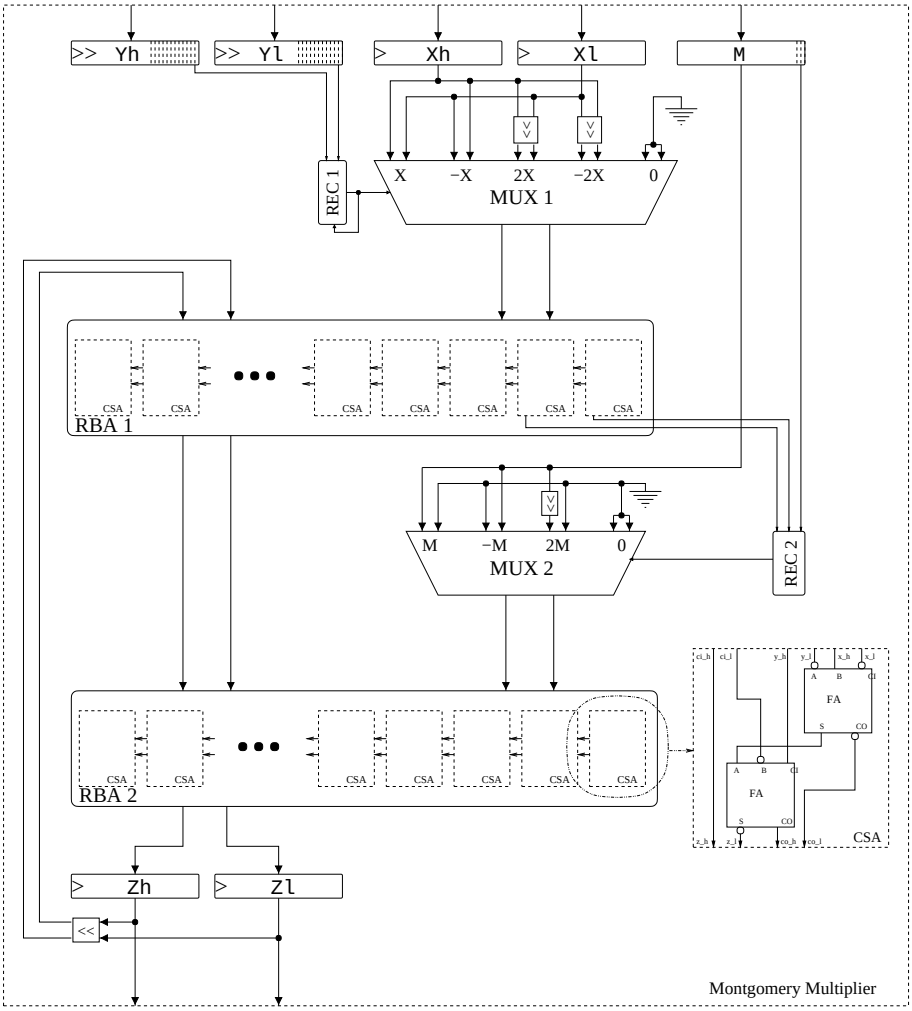


Figure 4.2: Radix-4 Montgomery Multiplier

pletes the n-bit multiplication in $\frac{n}{2} + 1$ iterations through 2 RBA stages. Obviously, the multiplication result will be in Montgomery residue form (divided by 2^{n+2}). A side-effect of using a radix-4 multiplier is that the range of operand values has increased from ($-M$, M) of original algorithm in [49] to ($-2M$, $2M$), where M is the prime modulus. Figure 4.2 shows the block diagram of the radix-4 Montgomery multiplier.

4.1.3 Extended Binary GCD Modular Division

Modular division is the most costly operation in ECC operations, which is usually avoided by using projective coordinates to trade several additional multiplications with division at every point addition cycle. If an efficient division unit could be implemented, the additional complexity incurred due to projective coordinates can be avoided. In order to achieve a high-performance divider, we implemented the modular division presented in Algorithm 4. This algorithm computes the GCD of the divisor and the prime modulus, which is equal to 1. Meanwhile, the same operations are applied to the dividend in parallel with a modulus reduction after each iteration. When the algorithm terminates by computing the GCD of the divisor and prime modulus as 1, the same operations applied to the dividend effectively computes the quotient of the modular division. The binary GCD algorithm is further optimized by observing the facts that the prime modulus is always an odd number; and when both numbers are odd, either their sum or their difference is a multiple of 4. Hence it reduces to following two cases, where Y is the divisor and M is the prime modulus:

```
If Y is even, M is odd:  GCD(Y,M) := GCD(Y/2,M)
If Y is odd,  M is odd:  GCD(Y,M) := GCD([Y±M]/4,M)
```

Algorithm 4 Extended Binary GCD Modular Division Algorithm [49]

Inputs: X:= Dividend, Y:= Divisor, M:= Modulus
Output: Z:= Result

p:= n; d:= 0; Z:= 0;
while p$\neq 0$ **do**
  while Y **is** even **do**
    Y:=Y$/2$; X:=X$/2\ mod$ M;
    p:=p-1; d:=d-1;
  end while
  if d< 0 **then**
    swap(Y,M); swap(X,Z); d:= $-$d;
  end if
  Y:=(Y+k$*$M)$/4$; X:=(X+k$*$Z)$/4\ mod$ M; (where: k $\in \{-1, 1\}$)
  p:=p-1; d:=d-1;
end while
if M$= -1$ **then** Z:=M$-$Z; **end if**
return Z;

The modular divider completes n-bit division in only $2n + 4$ iterations in ($-2M$, $2M$) range. Constant run-time for side-channel awareness is achieved by continuing the iterations until the control register fully processed as suggested in [49]. For each iteration, an adder is required for computing the GCD of the divisor and the prime modulus, and a modular adder is required for applying the same operation to the dividend in parallel together with modulus reduction. Therefore, a total of 3 RBA stages are necessary (1 RBA for regular addition and 2 RBAs for modular addition). It should also be noted that the arithmetic operations are carried out in the Montgomery residue format, and division does not preserve the Montgomery residue form of the operands. Therefore, a division should be followed by a multiplication to transform the result back into Montgomery residue form, which increases the effective division time to $\frac{5n}{2} + 6$ clock cycles. Figure 4.3 shows the block diagram of the extended binary GCD modular divider.

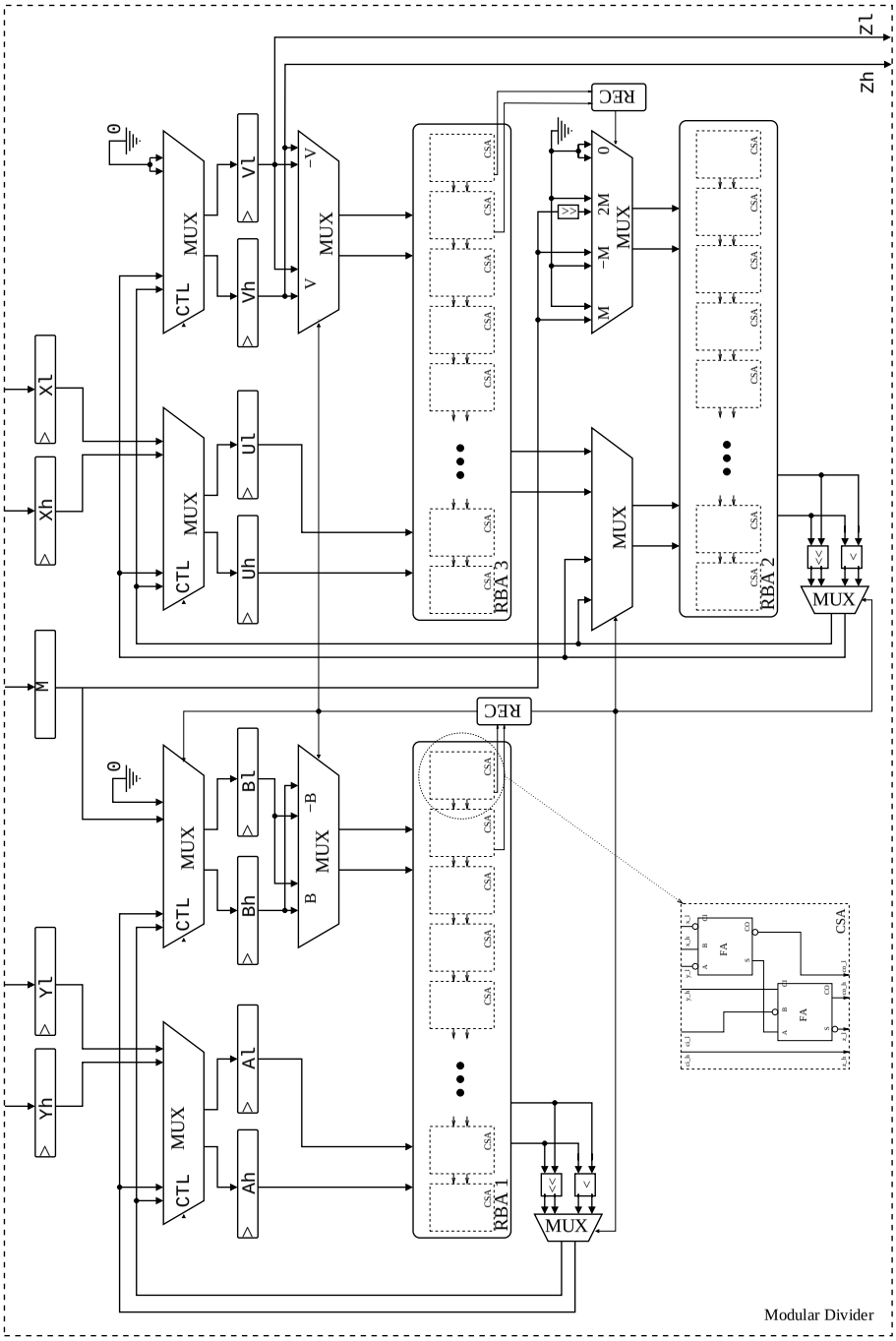


Figure 4.3: Extended Binary GCD Modular Divider

4.1.4 Modular Addition and Subtraction

The modular addition method described in [50] allows computing the n-bit modular addition or subtraction via a regular addition followed by a modular correction step that depends on checking only the most significant 3 digits of the intermediate result. Hence, modular addition and subtraction can be computed in a single iteration through 2 RBA stages. This method was also modified to work for $(-2M, 2M)$ range instead of $(-M, M)$ range of the original algorithm, in order to achieve consistency with the multiplication and division. The $3M$ value used in the algorithm is calculated by putting $4M$ in the low register, and M in the high register of *SD2* representation. Figure 4.4 shows the block diagram of the modular adder & subtractor.

Algorithm 5 Modular Addition and Subtraction Algorithm

Inputs: X:= 1^{st} Term, Y:= 2^{nd} Term, M:= Modulus
Output: Z:= Result

Step 1: $T := X \mp Y$;
Step 2: **if** $([T_{n+1}T_nT_{n-1}] \geq 4)$ **then** $Z := T - 3M$;
elseif $(3 \geq [T_{n+1}T_nT_{n-1}] \geq 2)$ **then** $Z := T - 2M$;
elseif $(1 \geq [T_{n+1}T_nT_{n-1}] \geq -1)$ **then** $Z := T$;
elseif $(-2 \geq [T_{n+1}T_nT_{n-1}] \geq -3)$ **then** $Z := T + 2M$;
elseif $(-4 \geq [T_{n+1}T_nT_{n-1}])$ **then** $Z := T + 3M$; **endif**

4.1.5 Combined Modular Arithmetic Units

The modular arithmetic units described above (Figures 4.2, 4.3, and 4.4) are all based on RBAs, and therefore efficient resource sharing is possible. We have implemented two different arithmetic units. The first arithmetic unit (mmu) is capable of modular multiplication, addition and subtraction; and it is intended to be used in projective point operations. The second unit (mau) is additionally capable of modular division; and it is intended to be used in affine point operations. The first arithmetic unit (mmu) requires 2 RBA stages,

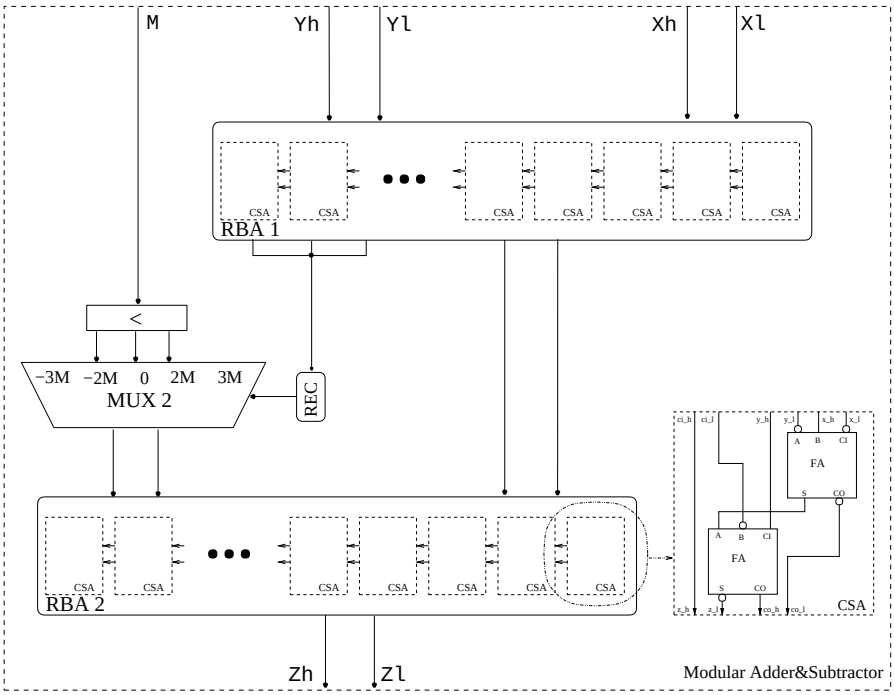


Figure 4.4: Modular Adder & Subtractor

whereas the second unit (mau) requires 3 RBA stages in order to accommodate for division.

When the point multiplication operation is carried out with projective coordinates, a final division is necessary to have the resulting point in affine coordinates. We realized this operation by taking the modular inverse of the Z-coordinate using multiplications according to Fermat's theorem: $Z^{-1} = Z^{p-2} \pmod{p}$, if $gcd(Z, p) = 1$ [48]. Although this inversion takes much longer ($\frac{3n^2}{4} + 3n$ on average) than the extended binary GCD division algorithm, it is carried out only once at the end of a point multiplication. Therefore, the performance gain in terms of area is more than the performance loss in terms of time, and it turns out to be more area-time efficient. The number of clock cycles for different modular operations and the hardware resources in each modular arithmetic unit are shown in Table 4.1.

Unit	Multiplication	Addition & Subtraction	Division	Resources
mmu	$\frac{n}{2}+2$	1	$\frac{3n^2}{4}+3n$	2 RBAs
mau	$\frac{n}{2}+2$	1	$\frac{5n}{2}+6$	3 RBAs

Table 4.1: Clock Cycles for Modular Arithmetic Operations

4.2 Point Addition and Doubling Operations

The next step in the design process is implementing elliptic curve point addition and doubling units. Point addition and doubling can be carried out either with affine coordinates or projective coordinates. Operations with projective coordinates do not involve divisions at the cost of a number of extra multiplications; therefore, they can be implemented without a modular divider. However, more storage space is needed due to the increased complexity of the operations, and an additional coordinate for representing a point.

We have designed four different point addition and doubling units (Weierstrass affine, Weierstrass projective, Edwards affine, Edwards projective), which implement the addition and doubling formulations in Equations (2.2), (2.3), (2.5), (2.6), (2.7), (2.8), (2.9), (2.10). In addition to point operations, each unit is also capable of necessary initial and final transformations. The initial transform computes the Montgomery residue forms of the point coordinates. The final transform computes the inverse Montgomery transformation and projective-to-affine transformation for projective coordinates. We also take advantage of the homogeneity in Edwards projective operations by avoiding Montgomery transformations. This is possible since the Montgomery modular multiplications in non-residue form do not affect the $\frac{X}{Z}$ and $\frac{Y}{Z}$ ratios at the end of a point addition or doubling.

For Weierstrass affine unit, we cannot utilize more than one arithmetic unit, due to the data dependence in point operations. For the others, having multiple units in parallel is possible. The single unit case always has the best area-time product, since all parallel units may not be utilized with the same type of operations in all stages of the dataflow.

Having a single arithmetic unit is also preferable due to the limited area resources of ECC applications. The Weierstrass implementations require special handling of point at infinity. During each Weierstrass point operation, a check for whether the resulting point will be the point at infinity is performed as well. This check is carried out offline (off the critical path delay) through a $\frac{n}{4}$-bit comparator without stalling the point operation. Selecting the comparator size as $\frac{n}{4}$-bit also allows keeping the area overhead low. For bit values less than 256, the comparator will require 3-levels of $(1 + 4 + 16 = 21)$ 4-input XOR gates.

Each unit is implemented with a datapath that consists of an appropriate modular arithmetic unit, a set of input selection multiplexers and temporary registers, and a control unit as shown in Figure 4.5. The control units for each point operation are designed with finite state machine strategy, where each state corresponds to an arithmetic operation through register-to-register dataflow. For each point operation, the arithmetic operations are scheduled to require minimum number of temporary storage registers after a careful data dependence analysis. The following subsections detail the implementation process of point addition and doubling units, and summarize the time and area results for each unit.

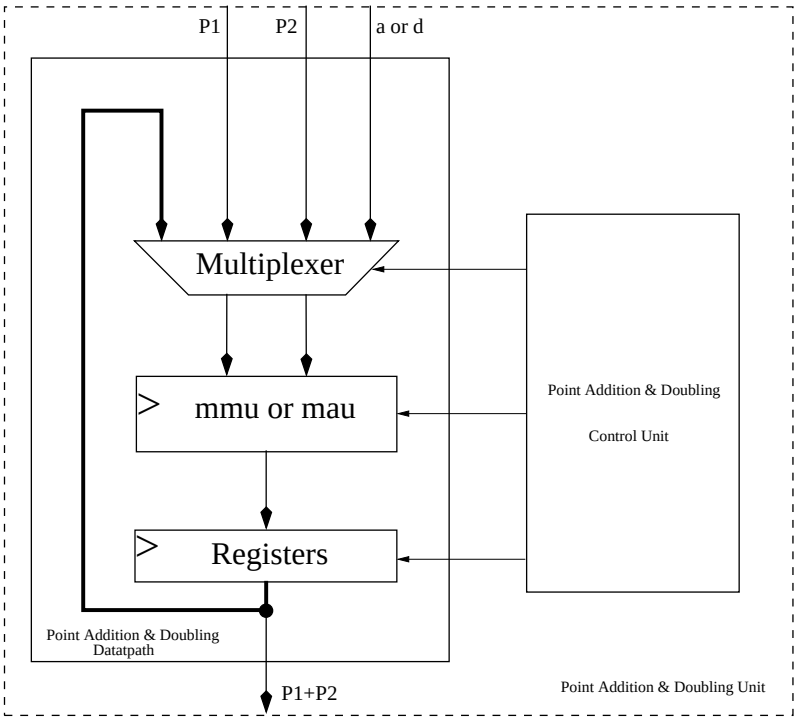


Figure 4.5: Block Diagram of Point Addition & Doubling Unit

4.2.1 Weierstrass Affine Point Addition and Doubling

Point Addition

$$\tilde{x}_3 = [(\frac{\tilde{y}_2-\tilde{y}_1}{\tilde{x}_2-\tilde{x}_1})^2 - \tilde{x}_1 - \tilde{x}_2] \; mod \; M$$

$$\tilde{y}_3 = [(\frac{\tilde{y}_2-\tilde{y}_1}{\tilde{x}_2-\tilde{x}_1})(\tilde{x}_1 - \tilde{x}_3) - \tilde{y}_1] \; mod \; M$$

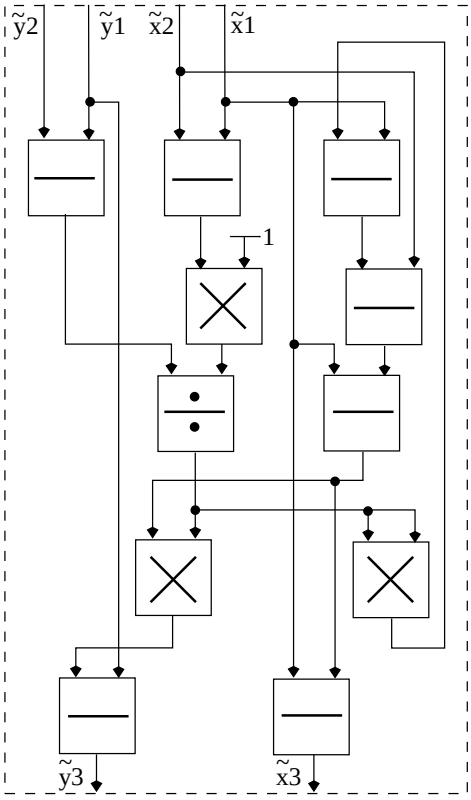


Figure 4.6: Dataflow for Weierstrass Affine Point Addition

RESULT	OP	CYCLES
0	SET	1
$\tilde{y}_2 - \tilde{y}_1$	SUB	1
$\tilde{x}_2 - \tilde{x}_1$	SUB	1
$x_2 - x_1$	MUL	$\frac{n}{2} + 2$
$\frac{\tilde{y}_2 - \tilde{y}_1}{\tilde{x}_2 - \tilde{x}_1}$	DIV	$\frac{5n}{2} + 6$
$(\frac{\tilde{y}_2 - \tilde{y}_1}{\tilde{x}_2 - \tilde{x}_1})^2$	MUL	$\frac{n}{2} + 2$
$(\frac{\tilde{y}_2 - \tilde{y}_1}{\tilde{x}_2 - \tilde{x}_1})^2 - \tilde{x}_1$	SUB	1
$\tilde{x}_3$	SUB	1
$\tilde{x}_1 - \tilde{x}_3$	SUB	1
$(\frac{\tilde{y}_2 - \tilde{y}_1}{\tilde{x}_2 - \tilde{x}_1})(\tilde{x}_1 - \tilde{x}_3)$	MUL	$\frac{n}{2} + 2$
$\tilde{y}_3$	SUB	1
$\tilde{x}_3$	SUB	1

Table 4.2: Operation Order for Weierstrass Affine Point Addition

Point Doubling

$$\tilde{x}_3 = [(\frac{3\tilde{x}_1^2+\tilde{a}}{2\tilde{y}_1})^2 - 2\tilde{x}_1] \; mod \; M$$

$$\tilde{y}_3 = [(\frac{3\tilde{x}_1^2+\tilde{a}}{2\tilde{y}_1})(\tilde{x}_1 - \tilde{x}_3) - \tilde{y}_1] \; mod \; M$$

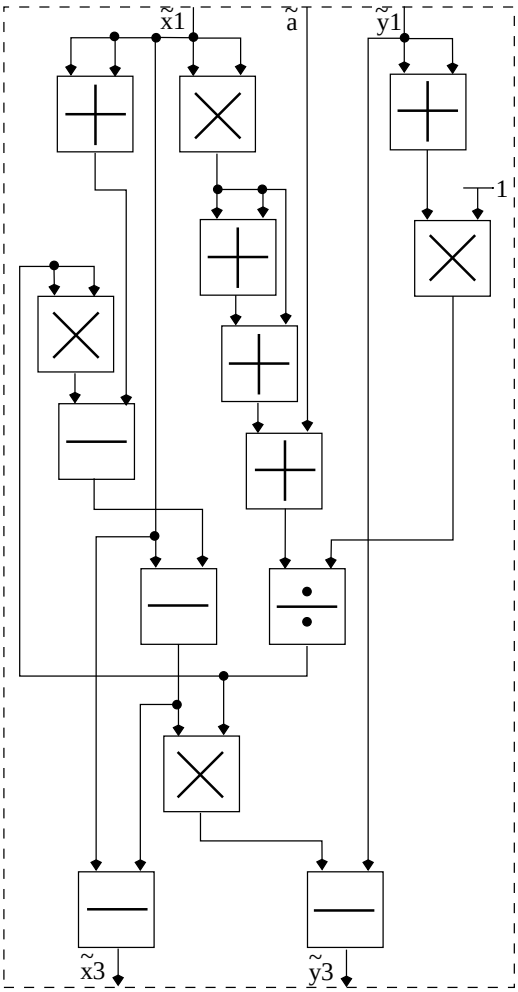


Figure 4.7: Dataflow for Weierstrass Affine Point Doubling

RESULT	OP	CYCLES
0	SET	1
$\tilde{x}_1^2$	MUL	$\frac{n}{2}+2$
$2\tilde{x}_1^2$	ADD	1
$3\tilde{x}_1^2$	ADD	1
$3\tilde{x}_1^2+\tilde{a}$	ADD	1
$2\tilde{y}_1$	ADD	1
$2y_1$	MUL	$\frac{n}{2}+2$
$\frac{3\tilde{x}_1^2+\tilde{a}}{2\tilde{y}_1}$	DIV	$\frac{5n}{2}+6$
$(\frac{3\tilde{x}_1^2+\tilde{a}}{2\tilde{y}_1})^2$	MUL	$\frac{n}{2}+2$
$(\frac{3\tilde{x}_1^2+\tilde{a}}{2\tilde{y}_1})^2-\tilde{x}_1$	SUB	1
$\tilde{x}_3$	SUB	1
$\tilde{x}_1-\tilde{x}_3$	SUB	1
$(\frac{3\tilde{x}_1^2+\tilde{a}}{2\tilde{y}_1})(\tilde{x}_1-\tilde{x}_3)$	MUL	$\frac{n}{2}+2$
$\tilde{y}_3$	SUB	1
$\tilde{x}_3$	SUB	1

Table 4.3: Operation Order for Weierstrass Affine Point Doubling

Resource Usage and Operation Counts of Weierstrass Affine Unit

Resources:	2 mux, 1 mau, 1 reg
Initial Montgomery Transform	2 MUL
Addition Time with Affine Coordinates	1 DIV, 3 MUL, 7 ADD/SUB
Doubling Time with Affine Coordinates	1 DIV, 4 MUL, 9 ADD/SUB
Final Inverse Montgomery Transform	2 MUL

4.2.2 Weierstrass Jacobian Point Addition and Doubling

Point Addition

$$\tilde{X}_3 = (\tilde{Y}_2\tilde{Z}_1^3 - \tilde{Y}_1\tilde{Z}_2^3)^2 - (\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1\tilde{Z}_2^2)^2(\tilde{X}_2\tilde{Z}_1^2 + \tilde{X}_1\tilde{Z}_2^2)\ mod\ M$$

$$\tilde{Y}_3 = \frac{(\tilde{Y}_2\tilde{Z}_1^3 - \tilde{Y}_1\tilde{Z}_2^3)[(\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1\tilde{Z}_2^2)^2(\tilde{X}_2\tilde{Z}_1^2 + \tilde{X}_1\tilde{Z}_2^2) - 2\tilde{X}_3] - (\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1\tilde{Z}_2^2)^3(\tilde{Y}_2\tilde{Z}_1^3 + \tilde{Y}_1\tilde{Z}_2^3)}{2}\ mod\ M$$

$$\tilde{Z}_3 = (\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1\tilde{Z}_2^2)\tilde{Z}_1\tilde{Z}_2\ mod\ M$$

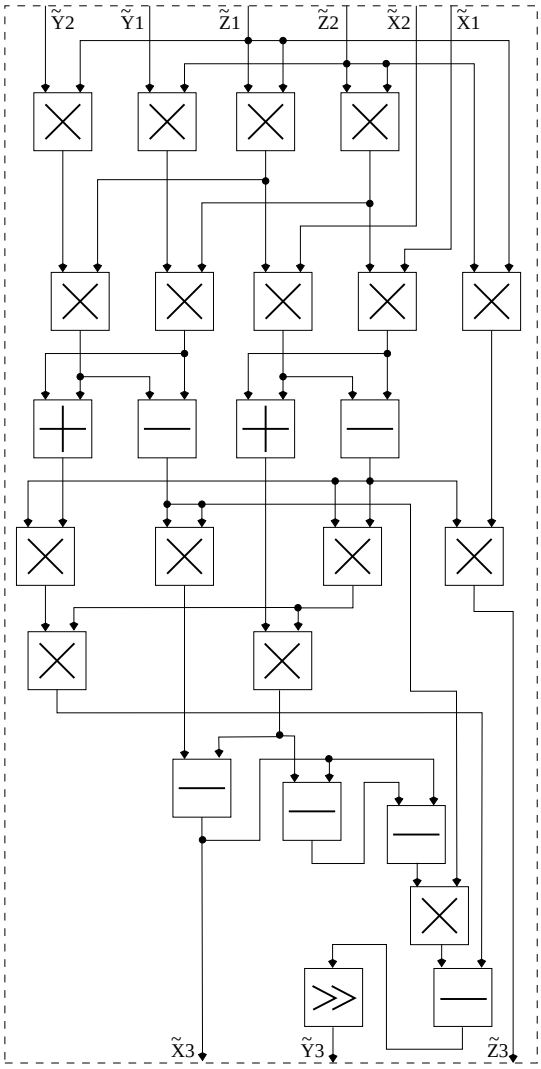


Figure 4.8: Dataflow for Weierstrass Jacobian Point Addition

RESULT	OP	CYCLES
0	SET	1
$\tilde{Z_1^2}$	MUL	$\frac{n}{2}+2$
$\tilde{Y_2}\tilde{Z_1}$	MUL	$\frac{n}{2}+2$
$\tilde{Y_2}\tilde{Z_1^3}$	MUL	$\frac{n}{2}+2$
$\tilde{X_2}\tilde{Z_1^2}$	MUL	$\frac{n}{2}+2$
$\tilde{Z_2^2}$	MUL	$\frac{n}{2}+2$
$\tilde{Y_1}\tilde{Z_2}$	MUL	$\frac{n}{2}+2$
$\tilde{Y_1}\tilde{Z_2^3}$	MUL	$\frac{n}{2}+2$
$\tilde{X_1}\tilde{Z_2^2}$	MUL	$\frac{n}{2}+2$
$\tilde{Z_1}\tilde{Z_2}R$	MUL	$\frac{n}{2}+2$
$(\tilde{X_2}\tilde{Z_1^2}-\tilde{X_1}\tilde{Z_2^2})$	SUB	1
$(\tilde{X_2}\tilde{Z_1^2}+\tilde{X_1}\tilde{Z_2^2})$	ADD	1
$(\tilde{Y_2}\tilde{Z_1^3}-\tilde{Y_1}\tilde{Z_2^3})$	SUB	1
$(\tilde{Y_2}\tilde{Z_1^3}+\tilde{Y_1}\tilde{Z_2^3})$	ADD	1
$\tilde{Z_3}$	MUL	$\frac{n}{2}+2$
$(\tilde{X_2}\tilde{Z_1^2}-\tilde{X_1}\tilde{Z_2^2})^2$	MUL	$\frac{n}{2}+2$
$(\tilde{X_2}\tilde{Z_1^2}-\tilde{X_1}\tilde{Z_2^2})^2(\tilde{X_2}\tilde{Z_1^2}+\tilde{X_1}\tilde{Z_2^2})$	MUL	$\frac{n}{2}+2$
$(\tilde{X_2}\tilde{Z_1^2}-\tilde{X_1}\tilde{Z_2^2})^2(\tilde{Y_2}\tilde{Z_1^3}+\tilde{Y_1}\tilde{Z_2^3})$	MUL	$\frac{n}{2}+2$
$(\tilde{X_2}\tilde{Z_1^2}-\tilde{X_1}\tilde{Z_2^2})^3(\tilde{Y_2}\tilde{Z_1^3}+\tilde{Y_1}\tilde{Z_2^3})$	MUL	$\frac{n}{2}+2$
$(\tilde{Y_2}\tilde{Z_1^3}-\tilde{Y_1}\tilde{Z_2^3})^2$	MUL	$\frac{n}{2}+2$
$\tilde{X_3}$	SUB	1
$(\tilde{X_2}\tilde{Z_1^2}-\tilde{X_1}\tilde{Z_2^2})^2(\tilde{X_2}\tilde{Z_1^2}+\tilde{X_1}\tilde{Z_2^2})-\tilde{X_3}$	SUB	1
$(\tilde{X_2}\tilde{Z_1^2}-\tilde{X_1}\tilde{Z_2^2})^2(\tilde{X_2}\tilde{Z_1^2}+\tilde{X_1}\tilde{Z_2^2})-2\tilde{X_3}$	SUB	1
$(\tilde{Y_2}\tilde{Z_1^3}-\tilde{Y_1}\tilde{Z_2^3})[(\tilde{X_2}\tilde{Z_1^2}-\tilde{X_1}\tilde{Z_2^2})^2(\tilde{X_2}\tilde{Z_1^2}+\tilde{X_1}\tilde{Z_2^2})-2\tilde{X_3}]$	MUL	$\frac{n}{2}+2$
$2\tilde{Y_3}$	SUB	1
$\tilde{Y_3}$	SHIFT	1

Table 4.4: Operation Order for Weierstrass Jacobian Point Addition

Mixed Point Addition (Z_2 = 1)

$\tilde{X}_3 = (\tilde{Y}_2\tilde{Z}_1^3 - \tilde{Y}_1)^2 - (\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1)^2(\tilde{X}_2\tilde{Z}_1^2 + \tilde{X}_1)\ mod\ M$

$\tilde{Y}_3 = \frac{(\tilde{Y}_2\tilde{Z}_1^3 - \tilde{Y}_1)[(\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1)^2(\tilde{X}_2\tilde{Z}_1^2 + \tilde{X}_1) - 2\tilde{X}_3] - (\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1)^3(\tilde{Y}_2\tilde{Z}_1^3 + \tilde{Y}_1)}{2}\ mod\ M$

$\tilde{Z}_3 = (\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1)\tilde{Z}_1\ mod\ M$

RESULT	OP	CYCLES
0	SET	1
$\tilde{Z}_1^2$	MUL	$\frac{n}{2} + 2$
$\tilde{Y}_2\tilde{Z}_1$	MUL	$\frac{n}{2} + 2$
$\tilde{Y}_2\tilde{Z}_1^3$	MUL	$\frac{n}{2} + 2$
$\tilde{X}_1\tilde{Z}_2^2$	MUL	$\frac{n}{2} + 2$
$(\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1)$	SUB	1
$(\tilde{X}_2\tilde{Z}_1^2 + \tilde{X}_1)$	ADD	1
$(\tilde{Y}_2\tilde{Z}_1^3 - \tilde{Y}_1)$	SUB	1
$(\tilde{Y}_2\tilde{Z}_1^3 + \tilde{Y}_1)$	ADD	1
$\tilde{Z}_3$	MUL	$\frac{n}{2} + 2$
$(\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1)^2$	MUL	$\frac{n}{2} + 2$
$(\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1)^2(\tilde{X}_2\tilde{Z}_1^2 + \tilde{X}_1)$	MUL	$\frac{n}{2} + 2$
$(\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1)^2(\tilde{Y}_2\tilde{Z}_1^3 + \tilde{Y}_1)$	MUL	$\frac{n}{2} + 2$
$(\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1)^3(\tilde{Y}_2\tilde{Z}_1^3 + \tilde{Y}_1)$	MUL	$\frac{n}{2} + 2$
$(\tilde{Y}_2\tilde{Z}_1^3 - \tilde{Y}_1)^2$	MUL	$\frac{n}{2} + 2$
$\tilde{X}_3$	SUB	1
$(\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1)^2(\tilde{X}_2\tilde{Z}_1^2 + \tilde{X}_1) - \tilde{X}_3$	SUB	1
$(\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1)^2(\tilde{X}_2\tilde{Z}_1^2 + \tilde{X}_1) - 2\tilde{X}_3$	SUB	1
$(\tilde{Y}_2\tilde{Z}_1^3 - \tilde{Y}_1)[(\tilde{X}_2\tilde{Z}_1^2 - \tilde{X}_1)^2(\tilde{X}_2\tilde{Z}_1^2 + \tilde{X}_1) - 2\tilde{X}_3]$	MUL	$\frac{n}{2} + 2$
$2\tilde{Y}_3$	SUB	1
$\tilde{Y}_3$	SHIFT	1

Table 4.5: Operation Order for Weierstrass Jacobian-Affine Point Addition

Point Doubling

$$\tilde{X}_3 = (3\tilde{X}_1^2 + a\tilde{Z}_1^4)^2 - 8\tilde{X}_1\tilde{Y}_1^2 \; mod \; M$$

$$\tilde{Y}_3 = (3\tilde{X}_1^2 + a\tilde{Z}_1^4)(4\tilde{X}_1\tilde{Y}_1^2 - \tilde{X}_3) - 8\tilde{Y}_1^4 \; mod \; M$$

$$\tilde{Z}_3 = 2\tilde{Y}_1\tilde{Z}_1 \; mod \; M$$

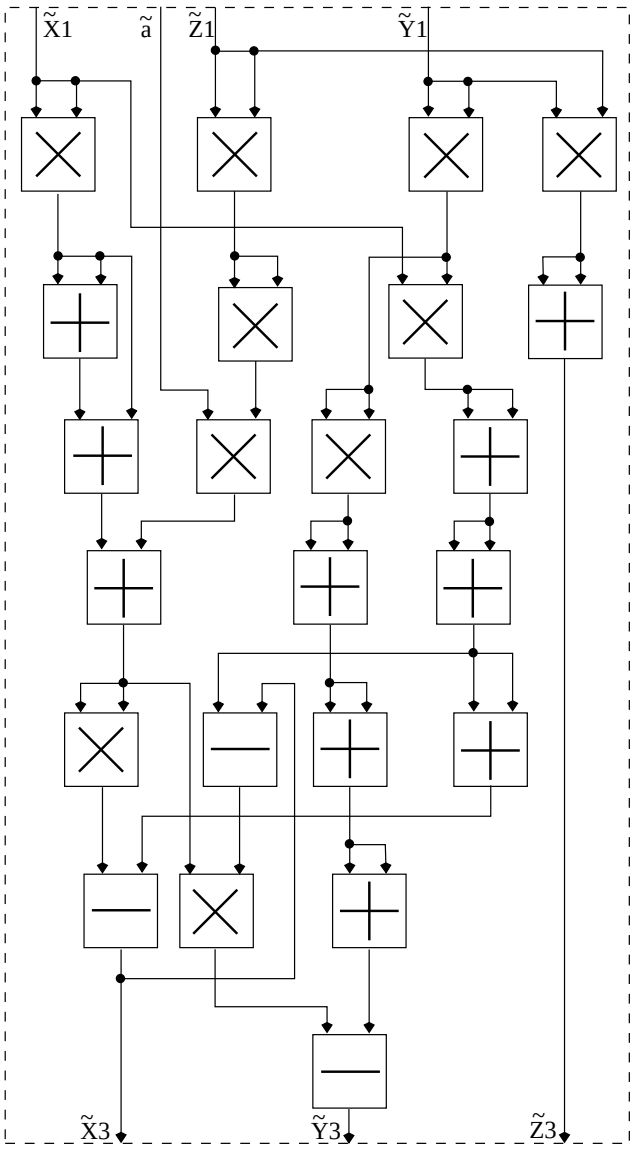


Figure 4.9: Dataflow for Weierstrass Jacobian Point Doubling

RESULT	OP	CYCLES
0	SET	1
$\tilde{Y}_1^2$	MUL	$\frac{n}{2}+2$
$\tilde{Z}_1^2$	MUL	$\frac{n}{2}+2$
$\tilde{Z}_1^4$	MUL	$\frac{n}{2}+2$
$\tilde{a}\tilde{Z}_1^4$	MUL	$\frac{n}{2}+2$
$\tilde{X}_1^2$	MUL	$\frac{n}{2}+2$
$2\tilde{X}_1^2$	ADD	1
$3\tilde{X}_1^2$	ADD	1
$(3\tilde{X}_1^2+\tilde{a}\tilde{Z}_1^4)$	ADD	1
$(3\tilde{X}_1^2+\tilde{a}\tilde{Z}_1^4)^2$	MUL	$\frac{n}{2}+2$
$\tilde{Y}_1\tilde{Z}_1$	MUL	$\frac{n}{2}+2$
$\tilde{Z}_3$	ADD	1
$\tilde{X}_1\tilde{Y}_1^2$	MUL	$\frac{n}{2}+2$
$2\tilde{X}_1\tilde{Y}_1^2$	ADD	1
$4\tilde{X}_1\tilde{Y}_1^2$	ADD	1
$8\tilde{X}_1\tilde{Y}_1^2$	ADD	1
$\tilde{X}_3$	SUB	1
$(4\tilde{X}_1\tilde{Y}_1^2-\tilde{X}_3)$	SUB	1
$(3\tilde{X}_1^2+\tilde{a}\tilde{Z}_1^4)(4\tilde{X}_1\tilde{Y}_1^2-\tilde{X}_3)R$	MUL	$\frac{n}{2}+2$
$\tilde{Y}_1^4$	MUL	$\frac{n}{2}+2$
$2\tilde{Y}_1^4$	ADD	1
$4\tilde{Y}_1^4$	ADD	1
$8\tilde{Y}_1^4$	ADD	1
$\tilde{Y}_3$	SUB	1

Table 4.6: Operation Order for Weierstrass Jacobian Point Doubling

Final Inversion:

$$x = \tilde{X} \cdot (\tilde{Z})^{2(M-2)} \ mod \ M$$

$$y = \tilde{Y} \cdot (\tilde{Z})^{3(M-2)} \ mod \ M$$

Resource Usage and Operation Counts of Weierstrass Jacobian Unit

Resources:	2 mux, 1 mmu, 5 reg
Initial Montgomery Transform	2 MUL, 2 ADD
Addition Time with Jacobian Coordinates	16 MUL, 9 ADD/SUB
Addition Time with Jacobian-Affine Coordinates	11 MUL, 9 ADD/SUB
Doubling Time with Jacobian Coordinates	10 MUL, 13 ADD/SUB
Final Inversion	$\frac{3n}{2} + 3$ MUL (Average)

4.2.3 Edwards Affine Point Addition and Doubling

Unified Point Addition & Doubling

$$\tilde{x}_3 = \frac{(\tilde{x}_1\tilde{y}_2+\tilde{y}_1\tilde{x}_2)}{1+d\tilde{x}_1\tilde{x}_2\tilde{y}_1\tilde{y}_2} \; mod \; M$$

$$\tilde{y}_3 = \frac{(\tilde{y}_1\tilde{y}_2-\tilde{x}_1\tilde{x}_2)}{1-d\tilde{x}_1\tilde{x}_2\tilde{y}_1\tilde{y}_2} \; mod \; M$$

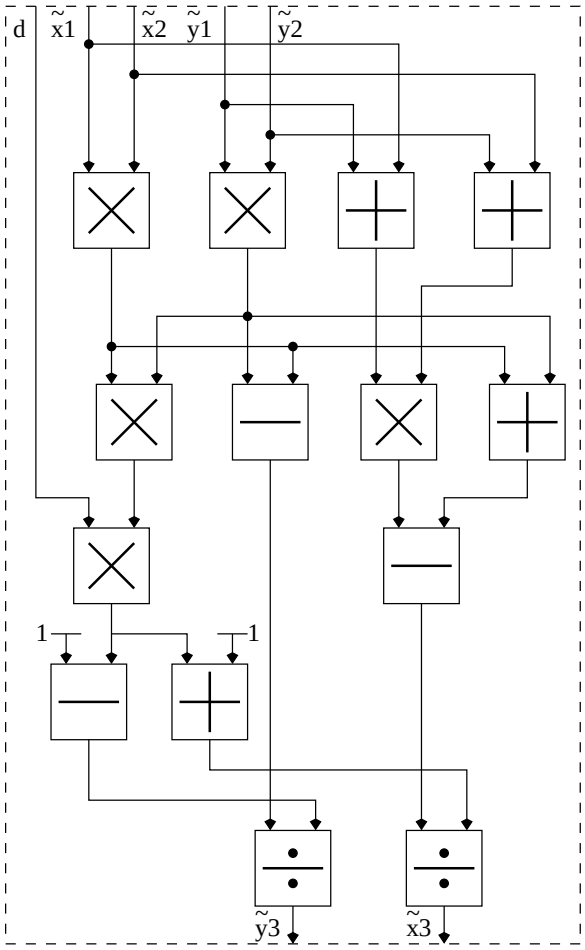


Figure 4.10: Dataflow for Edwards Affine Unified Point Addition & Doubling

RESULT	OP	CYCLES
0	SET	1
$\tilde{y}_1\tilde{y}_2$	MUL	$\frac{n}{2}+2$
$\tilde{x}_1\tilde{x}_2$	MUL	$\frac{n}{2}+2$
$(\tilde{x}_1+\tilde{y}_1)$	ADD	1
$(\tilde{x}_2+\tilde{y}_2)$	ADD	1
$(\tilde{x}_1+\tilde{y}_1)(\tilde{x}_2+\tilde{y}_2)$	MUL	$\frac{n}{2}+2$
$[(\tilde{x}_1+\tilde{y}_1)(\tilde{x}_2+\tilde{y}_2)-\tilde{x}_1\tilde{x}_2]$	SUB	1
$(\tilde{x}_1\tilde{y}_2+\tilde{y}_1\tilde{x}_2)$	SUB	1
$(\tilde{y}_1\tilde{y}_2-\tilde{x}_1\tilde{x}_2)$	SUB	1
$\tilde{x}_1\tilde{x}_2\tilde{y}_1\tilde{y}_2$	MUL	$\frac{n}{2}+2$
$dx_1x_2y_1y_2$	MUL	$\frac{n}{2}+2$
$1-dx_1x_2y_1y_2$	SUB	1
$1+dx_1x_2y_1y_2$	ADD	1
$\tilde{y}_3$	DIV	$\frac{5n}{2}+6$
$\tilde{x}_3$	DIV	$\frac{5n}{2}+6$

Table 4.7: Operation Order for Edwards Affine Unified Point Addition & Doubling

Optimized Point Doubling

$$\tilde{x}_3 = \frac{2\tilde{x}_1\tilde{y}_1}{\tilde{x}_1^2+\tilde{y}_1^2} \; mod \; M$$
$$\tilde{y}_3 = \frac{\tilde{x}_1^2-\tilde{y}_1^2}{\tilde{x}_1^2+\tilde{y}_1^2-2} \; mod \; M$$

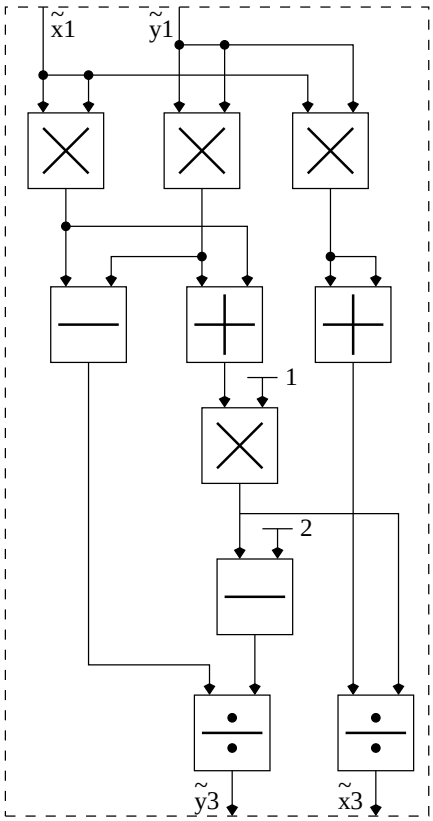


Figure 4.11: Dataflow for Edwards Affine Optimized Point Doubling

RESULT	OP	CYCLES
0	SET	1
$\tilde{y}_1^2$	MUL	$\frac{n}{2}+2$
$\tilde{x}_1^2$	MUL	$\frac{n}{2}+2$
$\tilde{x}_1\tilde{y}_1$	MUL	$\frac{n}{2}+2$
$2\tilde{x}_1\tilde{y}_1$	ADD	1
$(\tilde{x}_1^2-\tilde{y}_1^2)$	SUB	1
$(\tilde{x}_1^2+\tilde{y}_1^2)$	ADD	1
$(x_1^2+y_1^2)$	MUL	$\frac{n}{2}+2$
$(x_1^2+y_1^2-2)$	SUB	1
$\tilde{y}_3$	DIV	$\frac{5n}{2}+6$
$\tilde{x}_3$	DIV	$\frac{5n}{2}+6$

Table 4.8: Operation Order for Edwards Affine Optimized Point Doubling

Resource Usage and Operation Counts of Edwards Affine Unit

Resources:	2 mux, 1 mau, 3 reg
Initial Montgomery Transform	2 MUL
Unified Addition Time with Affine Coordinates	2 DIV, 5 MUL, 7 ADD/SUB
Optimized Doubling Time with Affine Coordinates	2 DIV, 4 MUL, 4 ADD/SUB
Final Inverse Montgomery Transform	2 MUL

4.2.4 Edwards Projective Point Addition and Doubling

Unified Point Addition/Doubling

$$X_3 = Z_1 Z_2 (X_1 Y_2 + Y_1 X_2)(Z_1^2 Z_2^2 - d X_1 X_2 Y_1 Y_2) \; mod \; M$$

$$Y_3 = Z_1 Z_2 (Y_1 Y_2 - X_1 X_2)(Z_1^2 Z_2^2 + d X_1 X_2 Y_1 Y_2) \; mod \; M$$

$$Z_3 = (Z_1^2 Z_2^2 - d X_1 X_2 Y_1 Y_2)(Z_1^2 Z_2^2 + d X_1 X_2 Y_1 Y_2) \; mod \; M$$

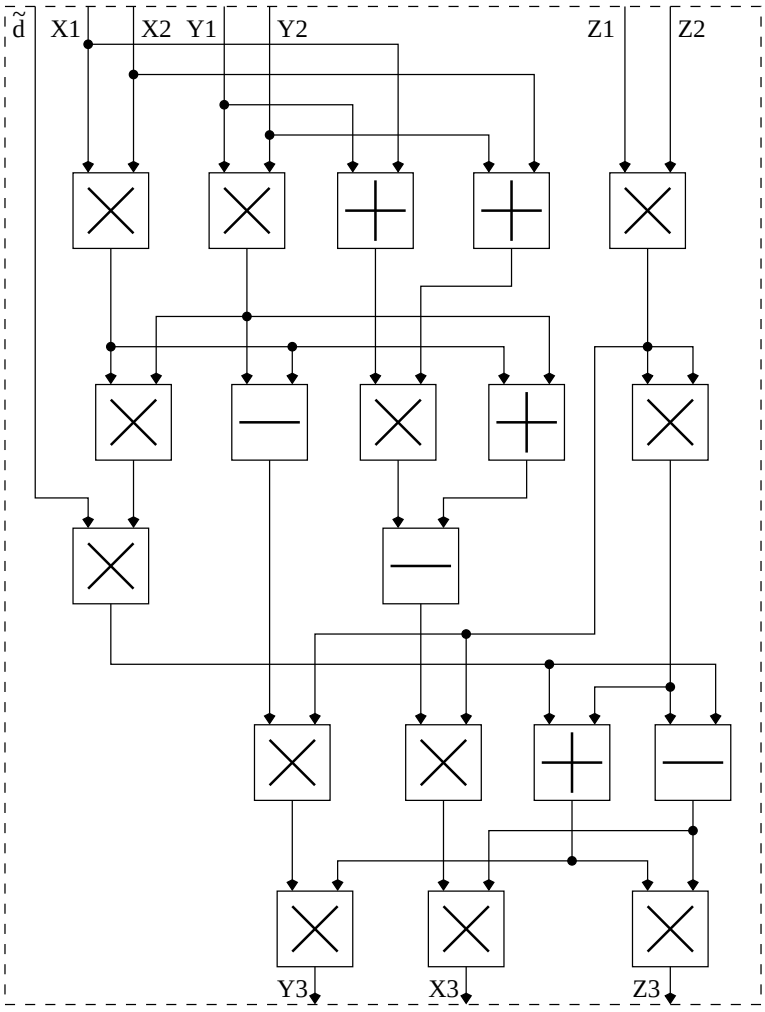


Figure 4.12: Dataflow for Edwards Projective Unified Point Addition & Doubling

RESULT	OP	CYCLES
0	SET	1
Z_1Z_2R	MUL	$\frac{n}{2}+2$
Y_1Y_2R	MUL	$\frac{n}{2}+2$
X_1X_2R	MUL	$\frac{n}{2}+2$
$(X_1+Y_1)R$	ADD	1
$(X_2+Y_2)R$	ADD	1
$(X_1+Y_1)(X_2+Y_2)R$	MUL	$\frac{n}{2}+2$
$(X_1Y_2+Y_1X_2+Y_1Y_2)R$	SUB	1
$(X_1Y_2+Y_1X_2)R$	SUB	1
$Z_1Z_2(X_1Y_2+Y_1X_2)R$	MUL	$\frac{n}{2}+2$
$(Y_1Y_2-X_1X_2)R$	SUB	1
$Z_1Z_2(Y_1Y_2-X_1X_2)R$	MUL	$\frac{n}{2}+2$
$X_1X_2Y_1Y_2R$	MUL	$\frac{n}{2}+2$
$dX_1X_2Y_1Y_2R$	MUL	$\frac{n}{2}+2$
$(Z_1Z_2)^2R$	MUL	$\frac{n}{2}+2$
$[(Z_1Z_2)^2+dX_1X_2Y_1Y_2]R$	ADD	1
$[(Z_1Z_2)^2-dX_1X_2Y_1Y_2]R$	SUB	1
Y_3R	MUL	$\frac{n}{2}+2$
Z_3R	MUL	$\frac{n}{2}+2$
X_3R	MUL	$\frac{n}{2}+2$

Table 4.9: Operation Order for Edwards Projective Unified Point Addition & Doubling

Optimized Point Doubling

$$X_3 = 2X_1Y_1(X_1^2 + Y_1^2 - 2Z_1^2)\ mod\ M$$

$$Y_3 = (X_1^2 - Y_1^2)(X_1^2 + Y_1^2)\ mod\ M$$

$$Z_3 = (X_1^2 + Y_1^2)(X_1^2 + Y_1^2 - 2Z_1^2)\ mod\ M$$

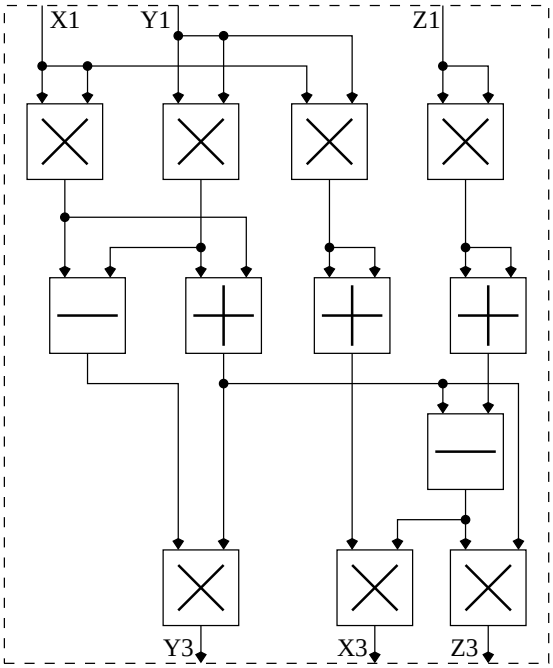


Figure 4.13: Dataflow for Edwards Projective Optimized Point Doubling

RESULT	OP	CYCLES
0	SET	1
Z_1^2R	MUL	$\frac{n}{2}+2$
$2Z_1^2R$	ADD	1
Y_1^2R	MUL	$\frac{n}{2}+2$
X_1^2R	MUL	$\frac{n}{2}+2$
X_1Y_1R	MUL	$\frac{n}{2}+2$
$2X_1Y_1R$	ADD	1
$(X_1^2-Y_1^2)R$	SUB	1
$(X_1^2+Y_1^2)R$	ADD	1
$(X_1^2+Y_1^2-2Z_1^2)R$	SUB	1
Z_3R	MUL	$\frac{n}{2}+2$
Y_3R	MUL	$\frac{n}{2}+2$
X_3R	MUL	$\frac{n}{2}+2$

Table 4.10: Operation Order for Edwards Projective Optimized Point Doubling

Final Inversion:

$x = X \cdot Z^{M-2}\ mod\ M$

$y = Y \cdot Z^{M-2}\ mod\ M$

Resource Usage and Operation Counts of Edwards Projective Unit

Resources:	2 mux, 1 mmu, 4 reg
Initial Montgomery Transform	NOP
Unified Addition Time with Projective Coordinates	12 MUL, 7 ADD/SUB
Optimized Doubling Time with Projective Coordinates	7 MUL, 5 ADD/SUB
Final Inversion	$\frac{3n}{2}+1$ MUL (Average)

Unit	Weierstrass Affine	Weierstrass Jacobian	Edwards Affine	Edwards Projective
Resources	1 mau, 1 reg	1 mmu, 5 regs	1 mau, 3 regs	1 mmu, 4 regs
Addition	1**CD** + 2**M** + 7**A**	16**M** + 9**A**	2**CD** + 5**M** + 7**A**	12**M** + 7**A**
Mixed Addition	Not Applicable	11**M** + 9**A**	Not Applicable	11**M** + 7**A**
Doubling	1**CD** + 3**M** + 9**A**	10**M** + 13**A**	2**CD** + 3**M** + 4**A**	7**M** + 5**A**
Initial Transform	2**M**	2**M** + 2**A**	2**M**	No Operation
Final Transform	2**M**	1**ED** + 4**M**	2**M**	1**ED** + 2**M**

(**M:** Multiplication, **A:** Addition & Subtraction, **CD:** Cheap Division, **ED:** Expensive Division)

Table 4.11: Number of Modular Operations for Point Addition and Doubling Units

4.2.5 Summary of Point Addition ad Doubling Operations

Table 4.11 shows the hardware resources, and the cost of point operations in terms of modular arithmetic operations for each unit. Looking at the operation counts, we observe that operations with projective coordinates have better performance in the Edwards formulation; whereas, operations with affine coordinates have better performance in the Weierstrass formulation. Meanwhile, the comparison between Weierstrass affine and Edwards projective operations depend on the performance ratio of division and multiplication. In terms of area, the projective units have the advantage of using smaller arithmetic units. However, they need more registers due to the increased complexity of the point operations with projective coordinates, and additional storage requirement for Z-coordinates.

4.3 Point Multiplication Operations

After the implementation of point addition and doubling units, the final step is to implement multiplication of a point on the elliptic curve with a scalar in order to realize the elliptic curve cryptosystem. We have introduced the binary multiplication methods, and examined their side channel leakage properties in Chapter 3. To have a complete design

space, we have implemented the add-always algorithm for all different ECC configurations. In addition, the secure NAF algorithm presented in Algorithm 2 was implemented for the Edwards formulation with unified addition and doubling operations. Table 4.12 shows the run times of the multiplication algorithms with the modular arithmetic operation costs of point additions and doublings taken from Table 4.11. Figure 4.14 shows the generic block diagram of a point multiplication unit. Each individual system consists of optimized datapath and control units to implement the point multiplication operations.

Algorithm:	Add-Always	Secure NAF
Number of Operations:	n additions, n doublings	$3n/2$ unified additions
Weierstrass Affine	$2n$**CD** + $(5n+4)$**M** + $16n$**A**	Not Safe
Weierstrass Jacobian	1**ED** + $(26n+6)$**M** + $(22n+2)$**A**	Not Safe
Edwards Affine with Unified Doublings	$4n$**CD** + $(10n+4)$**M** + $14n$**A**	$3n$**CD** + $(\frac{15n}{2}+4)$**M** + $\frac{21n}{2}$**A**
Edwards Affine with Optimized Doublings	$4n$**CD** + $(8n+4)$**M** + $11n$**A**	Not Safe
Edwards Projective with Unified Doublings	1**ED** + $(24n+2)$**M** + $14n$**A**	1**ED** + $(18n+2)$**M** + $\frac{21n}{2}$**A**
Edwards Projective with Optimized Doublings	1**ED** + $(19n+2)$**M** + $12n$**A**	Not Safe

(**M:** Multiplication, **A:** Addition & Subtraction, **CD:** Cheap Division, **ED:** Expensive Division)

Table 4.12: Number of Arithmetic Operations for Point Multiplication Units

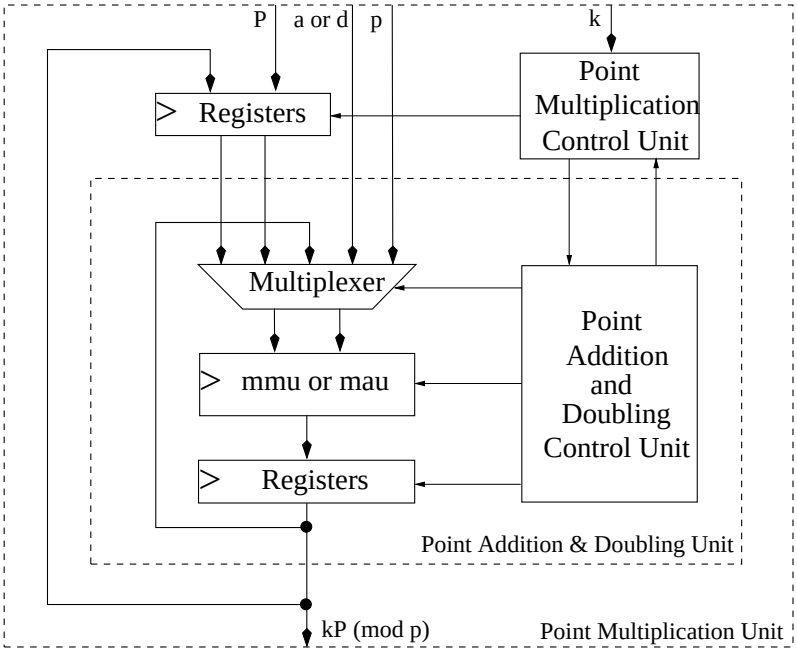


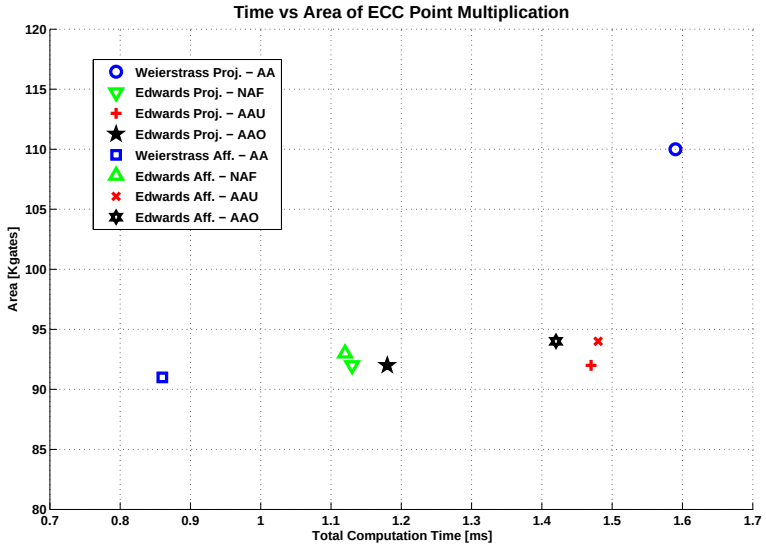
Figure 4.14: Block Diagram of Point Multiplication Units

Chapter 5

Results and Performance Comparison

In this chapter, we analyze the hardware implementation results, compare the time and area performance of our 8 different ECC configurations, and compare our results with other hardware implementations in the literature. We start by presenting the synthesis results in terms of maximum frequency, computation time, area, and time-area product. In Section 5.1, we explain the procedure of combining all ECC configurations and realizing them as an ECC processor on a single chip. Then, we present the placement and routing results of the ECC processor both with $0.18\ \mu m$ CMOS technology and $0.18\ \mu m$ MOS Current-Mode Differential Logic (MCML) technology [21]. Finally we list the maximum frequency, computation time and area values of our two best ECC implementations realized with $0.18\ \mu m$ CMOS technology for bit lengths of $n = 160$ and $n = 192$, together with 4 other ECC implementations realized on FPGA.

Most ECC hardware implementations in the literature have been realized over binary fields. There are only a small number of hardware implementations targeting prime fields, mostly implemented in FPGA. We found two comparable ASIC implementations that were recently published. Both implementations were realized using $0.13\ \mu m$ CMOS technology for bit length of $n = 192$ with no wireload model included. Moreover, they do



AA: Add-Always, AAO: Add-Always with Optimized Doublings, AAU: Add-Always with Unified Doublings

Figure 5.1: Time-Area Space of Point Multiplication

not address side-channel security. To be able to make fair comparison with our work, we synthesized our design for bit length of $n = 192$. Our implementations were synthesized using UMC $0.18\ \mu m$ CMOS technology with a target clock of $3\ ns$, and with no wireload model.

The overall gate count percentages of the arithmetic units for each ECC point multiplication system are provided in Table 5.1. Since the number of flip-flops in state machine registers are much less than storage registers, the area percentage of control units are negligible in size. Therefore, the remaining area percentage is allocated mostly for the storage registers. Figure 5.1 plots the time-area space of point multiplication. Both Table 5.1 and Figure 5.1 show that the Weierstrass affine system has the best area-time performance among our implementations, whereas the Weierstrass projective system has the worst area-time performance. In comparison to other implementations, 7 out of our 8 implementations have less area, and 4 out of our 8 implementations have better timing, although we use a slower technology, and address side-channel security as well.

ECC System	Multiplication Algorithm	Platform	Max f	Time	Area	Area× Time
Weierstrass Affine	Add-Always	$0.18\ \mu m$ CMOS	333.3 MHz	0.86 ms	$91K$ gates (mau: 55.0%)	78.26 gates×s
Weierstrass Projective	Add-Always	$0.18\ \mu m$ CMOS	333.3 MHz	1.59 ms	$110K$ gates (mmu: 28.4%)	174.90 gates×s
Edwards Affine	Add-Always with Unif. Doublings	$0.18\ \mu m$ CMOS	333.3 MHz	1.48 ms	$94K$ gates (mau: 53.9%)	139.12 gates×s
Edwards Affine	Add-Always with Opt. Doublings	$0.18\ \mu m$ CMOS	333.3 MHz	1.42 ms	$94K$ gates (mau: 53.9%)	133.48 gates×s
Edwards Affine	Secure NAF	$0.18\ \mu m$ CMOS	333.3 MHz	1.12 ms	$93K$ gates (mau: 54.5%)	104.16 gates×s
Edwards Projective	Add-Always with Unif. Doublings	$0.18\ \mu m$ CMOS	333.3 MHz	1.47 ms	$92K$ gates (mmu: 34.3%)	135.24 gates×s
Edwards Projective	Add-Always with Opt. Doublings	$0.18\ \mu m$ CMOS	333.3 MHz	1.18 ms	$92K$ gates (mmu: 34.3%)	108.56 gates×s
Edwards Projective	Secure NAF	$0.18\ \mu m$ CMOS	333.3 MHz	1.13 ms	$92K$ gates (mmu: 34.3%)	103.96 gates×s
Satoh et al. 2003 [54]	NAF	$0.13\ \mu m$ CMOS	137.7 MHz	1.44 ms	$110K$ gates	158.40 gates×s
Sozzani et al. 2005 [55]	Add-Double	$0.13\ \mu m$ CMOS	294.0 MHz	1.33 ms	$108K$ gates	143.64 gates×s

Table 5.1: Comparison of Synthesis Results for Point Multiplication [$GF(p)\ 192 - bit$]

5.1 ECC Processor

To be able to make a performance comparison among our 8 individual implementations as well as with other ASIC implementations, we used post-synthesis results. Our next step is to realize all implementations on a single chip. In this case, it does not make sense to implement very similar but locally optimized units as seperate entities. We combined all our implementations as an ECC processor that is capable of all different versions of point multiplication algorithms, and realized it on a single chip. The point addition and doubling units, detailed in Section 4.2, all have similar datapaths. But the datapaths are optimized together with the addition & doubling control unit to obtain the best performance out of an application, as if it would be used for the cryptosystem. We tried to achive least possible register usage and combinational logic complexity for each different point addition and doubling unit. For example Weierstrass affine point addition and doubling operations need only a single temporary register; whereas, Weierstrass Jacobian point addition and doubling operations need 5 temporary registers and a larger multiplexer to be able to select form each register. Similarly, the point multiplication units are also optimized for each specific case. For example 2 coordinates are used with affine operations; whereas, 3 coordinates are used projective operations. We merged all entities on a single datapath, going through the following steps:

1. The first step was to merge all 4 different point addition and doubling units (Weierstrass affine, Weierstrass Jacobian, Edwards affine, and Edwards projective) on a single datapath. Among these units, Weierstrass Jacobian needs the most number of registers and a multiplexer capable of selecting from each register. Therefore, we used the datapath of Weierstrass Jacobian unit, although not all registers are necessarily needed by other addition and doubling units.

2. As the second step, we replaced the mmu (modular multiplication, addition and subtraction unit) in the projective implementations with mau (modular multiplication, division, addition and subtraction unit), although these implementations never use the division operation in mau.

3. After having the same multiplexer, arithmetic unit and register number in all point addition and doubling units, as the third step, we clearly seperated the state machines and datapaths in each unit. Therefore, we were able to have a single datapath unit, and 4 different point addition and doubling control units.

4. Next, we merged the datapath parts of the 2 different multiplication algorithms (add-always, secure NAF) in a single datapath. We also merged the multiplication control units in a single state machine.

5. Finally, we added serial input reading and output writing functionality in this combined multiplication state machine, so that it can interface with the limited number of IO pads of the chip.

Eventually all components are put together as shown in Figure 5.2, and the ECC processor was realized both with $0.18\ \mu m$ CMOS technology, and $0.18\ \mu m$ MOS Current-Mode Differential Logic (MCML) technology [21] [1] as shown in Figure 5.3. Table 5.2 presents the time and area values after placement and routing. Finally, we are also presenting the P&R results of our 2 best implementations for bit lengths of $n = 160$ and $n = 192$, together with several FPGA implementation results in Table 5.3.

[1] MCML technology has a power balanced IC library, and provides DCA resiliency as mentioned in Section 3.2.

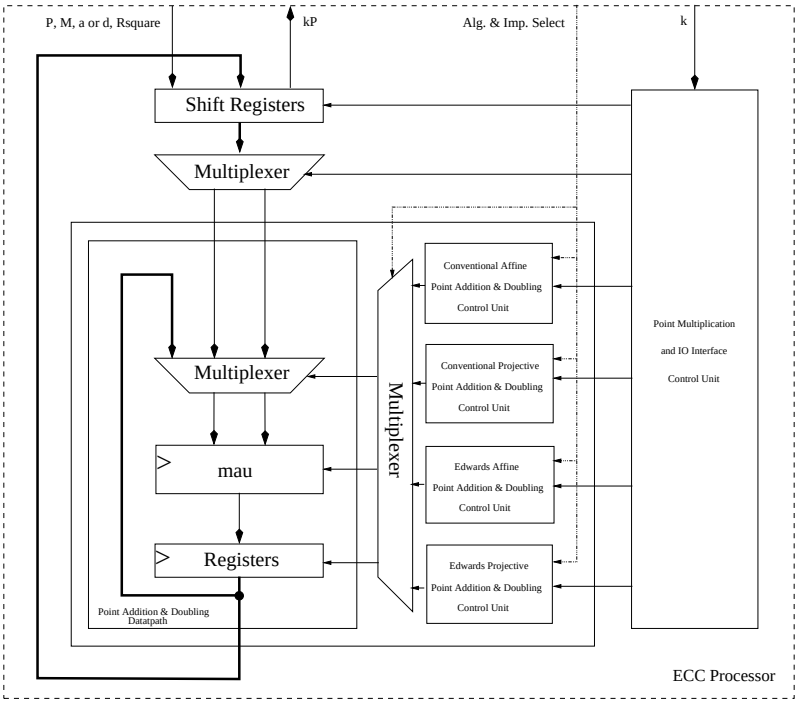


Figure 5.2: Block Diagram of Elliptic Curve Processor

Technology	Critical Path Delay	Cell Area
UMC CMOS 0.18 μm	6.8 ns	1.29 mm^2
LAUREL MCML 0.18 μm	15 ns	6.99 mm^2

Table 5.2: P&R Results for the ECC Processor

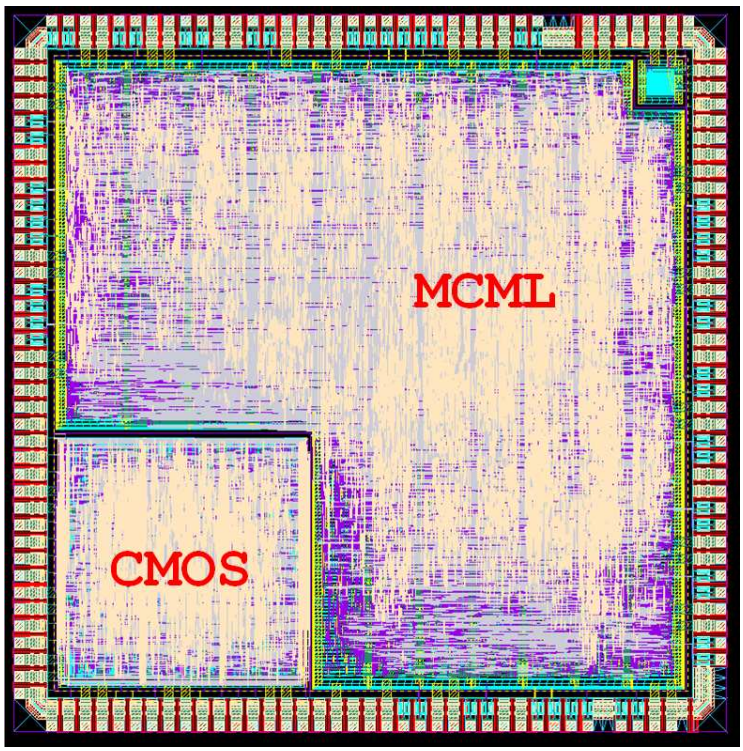


Figure 5.3: Elliptic Curve Processor Realized on a Single Chip

ECC System	GF(p) Field	Platform	Max f	Time	Area
Weierstrass Affine	160	0.18 μm CMOS	147 MHz	1.37 ms	86K gates
Edwards Projective	160	0.18 μm CMOS	147 MHz	1.80 ms	86K gates
Weierstrass Affine	192	0.18 μm CMOS	147 MHz	1.95 ms	100K gates
Edwards Projective	192	0.18 μm CMOS	147 MHz	2.56 ms	101K gates
Ors et al. 2003 [56]	160	XILINX XCV1000E-8	91.3 MHz	14.4 ms	115.5K gates
Mentens et al. 2007 [57]	160	XILINX XC3S5000-5	66 MHz	26.8 ms	4826 slices, 66 RAMs, 66 mults
Mcivor et al. 2006 [58]	256	XILINX XC2VP125-7	34.46 MHz	3.86 ms	15755 slices, 256 mults
Sakiyama et al. 2007 [59]	256	XILINX XC3S5000-5	40 MHz	17.7 ms	27597 slices

Table 5.3: P&R Results for Point Multiplication

Chapter 6

Conclusion

In this dissertation, we addressed the design and implementation of high performance non-specialized elliptic curve cryptosystems over finite fields of prime characteristics, while retaining side-channel awareness. Our investigation also included the recently introduced Edwards elliptic curves with built-in resiliency against simple side-channel attacks.

We presented methods to improve the performance of ECC building blocks with side-channel attack precautions, and explained the details for efficient mapping to hardware. Many optimizations had been previously proposed for the Weierstrass formulation, aimed at reducing the cost of point addition and doubling operations, as well as decreasing the number of point additions and doublings. Unfortunately, most of these optimizations made the systems prone to side-channel attacks. We avoided reducing the number of point additions, and discussed methods to uniformly order the distinguishable point operations of Weierstrass elliptic curves for achieving a scalar point multiplication resistant against simple side-channel attacks. We also introduced our secure version of NAF multiplication algorithm to work with unified point addition and doubling operations of Edwards elliptic curves.

We implemented 8 different ECC point multiplication systems, and synthesized each configuration with UMC $0.18\ \mu m$ CMOS technology:

1. Weierstrass affine using add-always algorithm,
2. Weierstrass projective using add-always algorithm,
3. Edwards affine with unified doublings using add-always algorithm,
4. Edwards affine with optimized doublings using add-always algorithm,
5. Edwards affine with unified doublings using secure NAF algorithm,
6. Edwards projective with unified doublings using add-always algorithm,
7. Edwards projective with optimized doublings using add-always algorithm,
8. Edwards projective with unified doublings using secure NAF algorithm.

Then, we combined all ECC configurations as an ECC processor, and realized it on a single chip with $0.18\ \mu m$ CMOS technology, and also with $0.18\ \mu m$ MOS Current-Mode Differential Logic (MCML) technology [21] to provide resiliency against differential side channel attacks.

Edwards elliptic curves were shown to be faster than previous elliptic curve formulations [19] in addition to their advantages for side-channel security. However, due to more complicated point operations of Edwards affine formulation, the performance benefits were only applicable for projective coordinates. Therefore, with the availability of an efficient divider, Weierstrass affine formulation still offered the best performance. Using projective coordinates had both positive and negative effects on the total cell area. Avoiding the implementation of a divider allowed reducing the area. However, more complicated point addition and doubling operations required using more temporary registers. Moreover, using 3 instead of 2 coordinates to represent a point increased the area to store

the temporary points during point multiplication. The adverse effects of all these additional register requirements were exacerbated when each binary digit was stored in 2 bits due to our choice of using the redundant binary representation to implement fast modular operations. Finally, with the use of efficient resource sharing between divider and multiplier, the area cost of implementing the divider was reduced. Indeed, we observed that the area cost of additional register usage canceled out the area gain of not implementing a divider. Thus, the areas of affine and projective systems had almost same values that were in the range 91K-94K, except the Weierstrass Projective implementation. The complicated formulations of Weierstrass Projective system consumed an area of 110K even without a divider. Having lost the advantage of area reduction, the only attraction to use projective coordinates would have been timing optimization by avoiding divisions. However, in our implementation the division/multiplication timing ratio was only 4, and was not enough to make the projective operations significantly faster. Therefore, we obtained the time costs presented in Table 5.1 and Figure 5.1.

We showed that, in an efficient hardware implementation, Weierstrass formulation with affine coordinates offered the best performance due to its simplicity, and Edwards superseded Weierstrass formulation only when projective coordinates were used. The fact that affine systems would offer better performance than projective systems was also pointed out in [60–62], which demonstrated that it would be profitable to investigate efficient inversion architectures to be able to use the simpler formulations with affine coordinates.

Bibliography

[1] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987.

[2] V. Miller, "Use of Elliptic Curves in Cryptography," *Advances in Cryptology (CRYPTO 1985), Santa Barbara, CA, USA*, vol. 218, pp. 417–426, 1985.

[3] A. Lenstra and E. Verheul, "Selecting Cryptographic Key Sizes," *Journal of Cryptology*, vol. 14, pp. 255–293, 2001.

[4] J. Quisquater and D. Samyde, "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards," *Smart Card Programming and Security, International Conference on Research in Smart Cards (E-smart 2001), Cannes, France*, vol. 2140, pp. 200–210, 2001.

[5] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic Analysis: Concrete Results," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), Paris, France*, vol. 2162, pp. 251–261, 2001.

[6] J. Lee, S. Jung, and J. Lim, "Detecting Trapdoors in Smart Cards Using Timing and Power Analysis," *IFIP TC6/WG 6.1 International Conference: Testing of Communicating Systems (TestCom 2005), Montreal, Canada*, vol. 3502, pp. 275–288, 2005.

[7] R. Anderson and M. Kuhn, "Tamper Resistance–a Cautionary Note," *The Second USENIX Workshop on Electronic Commerce, Oakland, CA, USA*, vol. 2, pp. 1–11, 1996.

[8] R. Anderson and M. Kuhn, "Low Cost Attacks on Tamper Resistant Devices," *International Workshop on Security Protocols, Paris, France*, vol. 1361, pp. 125–136, 1997.

[9] P. Liardet and N. Smart, "Preventing SPA/DPA in ECC Systems Using the Jacobi Form," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), Paris, France*, vol. 2162, pp. 391–401, 2001.

[10] M. Joye and J. Quisquater, "Hessian Elliptic Curves and Side-Channel Attacks," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), Paris, France*, vol. 2162, pp. 402–410, 2001.

[11] K. Okeya and K. Sakurai, "Power Analysis Breaks Elliptic Curve Cryptosystems even Secure against the Timing Attack," *International Conference in Cryptology in India (INDOCRYPT 2000), Calcutta, India*, vol. 1977, pp. 178–190, 2000.

[12] M. Joye and C. Tymen, "Protections Against Differential Analysis for Elliptic Curve Cryptography," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), Paris, France*, vol. 2162, pp. 377–390, 2001.

[13] E. Brier and M. Joye, "Weierstrass Elliptic Curves and Side-Channel Attacks," *International Workshop on Practice and Theory in Public Key Cryptosystems (PKC 2002), Paris, France*, vol. 2274, pp. 335–345, 2002.

[14] B. Möller, "Parallelizable Elliptic Curve Point Multiplication Method with Resistance against Side-Channel Attacks," *International Conference on Information Security (ISC 2002), Sao Paulo, Brazil*, vol. 2433, pp. 402–413, 2002.

[15] J. Coron, "Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 1999), Worcester, MA, USA*, vol. 1717, pp. 292–302, 1999.

[16] T. Izu and T. Takagi, "A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks," *International Workshop on Practice and Theory in Public Key Cryptosystems (PKC 2002), Paris, France*, vol. 2274, pp. 280–296, 2002.

[17] B. Möller, "Securing Elliptic Curve Point Multiplication against Side-Channel Attacks," *International Conference on Information Security (ISC 2001), Malaga, Spain*, vol. 2200, pp. 324–334, 2001.

[18] H. Edwards, "A Normal Form for Elliptic Curves," *Bulletin of the American Mathematical Society*, vol. 44, no. 3, pp. 393–422, 2007.

[19] D. Bernstein and T. Lange, "Faster Addition and Doubling on Elliptic Curves," *International Conference on the Theory and Applications of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT 2007), Kuching, Malaysia*, vol. 4833, pp. 29–50, 2007.

[20] D. Bernstein, T. Lange, and R. Farashahi, "Binary Edwards Curves," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2008), Washington, D.C., USA*, vol. 5154, pp. 244–265, 2008.

[21] S. Badel, *MOS Current-Mode Logic Standard Cells for High-Speed Low-Noise Applications*. PhD thesis, Swiss Federal Institute of Technology, Lausanne (EPFL), 2008.

[22] M. Joye, "Highly Regular Right-to-Left Algorithms for Scalar Multiplication," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007), Vienna, Austria*, vol. 4727, pp. 135–147, 2007.

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

[24] A. Lenstra, "Cryptosystems with Elliptic Curves Chosen by Users," Sept. 3 2002. US Patent 6,446,205.

[25] D. Chudnovsky and G. Chudnovsky, "Sequences of Numbers Generated by Addition in Formal Groups and New Primality and Factorization Tests," *Advances in Applied Mathematics*, vol. 7, no. 4, pp. 385–434, 1986.

[26] J. Lopez and R. Dahab, "Fast Multiplication on Elliptic Curves over GF (2^m) without Precomputation," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 1999), Worcester, MA, USA*, vol. 1717, pp. 316–327, 1999.

[27] H. Cohen, A. Miyaji, and T. Ono, "Efficient Elliptic Curve Exponentiation Using Mixed Coordinates," *International Conference on the Theory and Applications of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT 1998), Beijing, China*, vol. 1514, pp. 51–65, 1998.

[28] S. Chari, J. Rao, and P. Rohatgi, "Template Attacks," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002), Redwood Shores, CA, USA*, vol. 2523, pp. 13–28, 2002.

[29] G. Reitwiesner, "Binary Arithmetic," *Advances in Computers*, vol. 1, pp. 231–308, 1960.

[30] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," *International Cryptology Conference: Advances in Cryptology (CRYPTO 1996), Santa Barbara, CA, USA*, vol. 1109, pp. 104–113, 1996.

[31] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *International Cryptology Conference: Advances in Cryptology (CRYPTO 1999), Santa Barbara, CA, USA*, vol. 1666, pp. 388–397, 1999.

[32] W. Schindler, "A Combined Timing and Power Attack," *International Workshop on Practice and Theory in Public Key Cryptosystems (PKC 2002), Paris, France*, vol. 2274, pp. 263–279, 2002.

[33] B. Chevallier-Mames, M. Ciet, and M. Joye, "Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity," *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 760–768, 2004.

[34] I. Biehl, B. Meyer, and V. Muller, "Differential Fault Attacks on Elliptic Curve Cryptosystems," *International Cryptology Conference: Advances in Cryptology (CRYPTO 2000), Santa Barbara, CA, USA*, vol. 1880, pp. 131–146, 2000.

[35] S. Yen and M. Joye, "Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis," *IEEE Transactions on Computers*, vol. 49, no. 9, pp. 967–970, 2000.

[36] M. Joye and S. Yen, "The Montgomery Powering Ladder," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002), Redwood Shores, CA, USA*, vol. 2523, pp. 291–302, 2002.

[37] P. Fouque and F. Valette, "The Doubling Attack Why Upwards is better than Downwards," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003), Cologne, Germany*, vol. 2779, pp. 269–280, 2003.

[38] C. Walter, "Sliding Windows Succumbs to Big Mac Attack," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), Paris, France*, vol. 2162, pp. 286–299, 2001.

[39] C. Clavier, J. Coron, and N. Dabbous, "Differential Power Analysis in the Presence of Hardware Countermeasures," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000), Worcester, MA, USA*, vol. 1965, pp. 252–263, 2000.

[40] M. Aigner and E. Oswald, "Power Analysis Tutorial," tech. rep., Graz University of Technology (TU Graz), 2000.

[41] N. Pramstaller, F. Gurkaynak, S. Haene, H. Kaeslin, N. Felber, and W. Fichtner, "Towards an AES Crypto-chip Resistant to Differential Power Analysis," *European Solid-State Circuits Conference (ESSCIRC 2004), Leuven, Belgium*, pp. 307–310, 2004.

[42] F. Gürkaynak, S. Oetiker, H. Kaeslin, N. Felber, and W. Fichtner, "Improving DPA Security by Using Globally-Asynchronous Locally-Synchronous Systems," *European Solid-State Circuits Conference (ESSCIRC 2005), Grenoble, France*, pp. 407–410, 2005.

[43] T. Messerges, "Securing the AES Finalists Against Power Analysis Attacks," *International Workshop on Fast Software Encryption (FSE 2000), New York, NY, USA*, vol. 1978, pp. 150–164, 2001.

[44] E. Oswald, S. Mangard, and N. Pramstaller, "Secure and Efficient Masking of AES-A Mission Impossible," tech. rep., Technical Report IAIK-TR 2003/11/1, 2004. http://eprint.iacr.org/.

[45] K. Tiri, M. Akmal, and I. Verbauwhede, "A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withstand Differential Power Analysis on Smart Cards," *European Solid-State Circuits Conference (ESSCIRC 2002), Florence, Italy*, pp. 403–406, 2002.

[46] Z. Toprak and Y. Leblebici, "Low-Power Current Mode Logic for Improved DPA-Resistance in Embedded Systems," *International Symposium on Circuits and Systems (ISCAS 2005), Kobe, Japan*, vol. 2, pp. 1059–1062, 2005.

[47] F. Regazzoni, S. Badel, T. Eisenbarth, J. Grobschadl, A. Poschmann, Z. Toprak, M. Macchetti, L. Pozzi, C. Paar, Y. Leblebici, *et al.*, "A Simulation-Based Methodology for Evaluating the DPA-Resistance of Cryptographic Functional Units with Application to CMOS and MCML Technologies," *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS 2007), Samos, Greece*, pp. 209–214, 2007.

[48] S. Ors, L. Batina, B. Preneel, and J. Vandewalle, "Hardware Implementation of an Elliptic Curve Processor over GF (p)," *14th IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP 2003), The Hague, The Netherlands*, vol. 0, pp. 433–443, 2003.

[49] M. Kaihara and N. Takagi, "A Hardware Algorithm for Modular Multiplication/Division," *IEEE Transactions on Computers*, vol. 54, no. 1, pp. 12–21, 2005.

[50] N. Takagi and S. Yajima, "Modular Multiplication Hardware Algorithms with a Redundant Representation and their Application to RSA Cryptosystem," *IEEE Transactions on Computers*, vol. 41, no. 7, pp. 887–891, 1992.

[51] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Transactions on Electronic Computers*, vol. 10, no. 3, pp. 389–400, 1961.

[52] N. Takagi, H. Yasuura, and S. Yajima, "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree," *IEEE Transactions on Computers*, vol. 34, no. 9, pp. 789–796, 1985.

[53] P. Kornerup, "Reviewing 4-to-2 Adders for Multi-Operand Addition," *The Journal of VLSI Signal Processing*, vol. 40, no. 1, pp. 143–152, 2005.

[54] A. Satoh and K. Takano, "A Scalable Dual-Field Elliptic Curve Cryptographic Processor," *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 449–460, 2003.

[55] F. Sozzani, G. Bertoni, S. Turcato, and L. Breveglieri, "A Parallelized Design for an Elliptic Curve Cryptosystem Coprocessor," *International Symposium on Information Technology: Coding and Computing (ITCC 2005), Las Vegas, Nevada, USA*, vol. 1, pp. 626–630, 2005.

[56] S. Ors, L. Batina, B. Preneel, and J. Vandewalle, "Hardware implementation of an elliptic curve processor over gf(p)," *IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP 2003), The Hague, The Netherlands*, pp. 433–443, 2003.

[57] N. Mentens, K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "A side-channel attack resistant programmable pkc coprocessor for embedded applications," *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS 2007), Samos, Greece*, pp. 194–200, 2007.

[58] C. McIvor, M. McLoone, and J. McCanny, "Hardware elliptic curve cryptographic processor over gf (p)," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 9, pp. 1946–1957, 2006.

[59] K. Sakiyama, N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede, "Reconfigurable modular arithmetic logic unit supporting high-performance rsa and ecc over gf(p)," *International Journal of Electronics*, vol. 94, no. 5, pp. 501–514, 2007.

[60] E. Savas and Ç. Koc, "Architectures for Unified Field Inversion with Applications in Elliptic Curve Cryptography," *9th International Conference on Electronics, Circuits and Systems (ICECS 2002), Dubrovnik, Croatia*, vol. 3, pp. 1155–1158, 2002.

[61] E. Ozturk, B. Sunar, and E. Savas, "Low-Power Elliptic Curve Cryptography Using Scaled Modular Arithmetic," *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004), Boston, MA, USA*, vol. 3156, pp. 92–106, 2004.

[62] E. Savas, M. Naseer, A. Gutub, and Ç. Koc, "Efficient Unified Montgomery Inversion with Multibit Shifting," *IEE Proceedings - Computers and Digital Techniques*, vol. 152, no. 4, pp. 489–498, 2005.