2017•2018

Faculteit Industriële ingenieurswetenschappen

master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Design of a fully balanced ASIC coprocessor implementing complete addition formulas on Weierstrass elliptic curves

PROMOTOR :

De heer Jo VLIEGEN

Prof. dr. ir. Nele MENTENS

PROMOTOR :

Prof. Batina LEJLA

Niels Pirotte

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

2017•2018
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Design of a fully balanced ASIC coprocessor implementing complete addition formulas on Weierstrass elliptic curves

PROMOTOR :
De heer Jo VLIEGEN
Prof. dr. ir. Nele MENTENS

PROMOTOR :
Prof. Batina LEJLA

Niels Pirotte
Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

PREFACE

With this thesis, I finally conclude my journey as an engineer in training and before cutting to the chase, I would like to take this opportunity to thank everybody involved in the process of writing this text. Infinitely important, was the support of my family, who motivated me not only during this quest, but long before. To my dear friends, thank you for your significant part in this achievement and in my life, because without your positive influence and sometimes distractions, none of this was possible. Also, I want to thank my fellow students, with whom I shared the joy and challenge of the process of learning. Of course, my gratitude goes to all mentors, teachers and professors I had the pleasure of working with during this Bachelor's and Master's program. They did an excellent job of providing my fellow students and me with the necessary knowledge and skills to reach our professional goals. Last but not least, special thanks goes to my supervisors: Nele Mentens, Jo Vliegen and Lejla Batina. Who guided and counseled me into the right direction. Who always made time for my questions and who substantially elevated the quality of this work.

I also was incredibly thankful to receive a topic within the field of cryptography, because of my shared interest in IT, security and mathematics. For me, this setting presented the perfect opportunity to combine all my newly acquired skills into one challenging project and I am very delighted about the things I learned during the previous year. Therefore, I hope you enjoy reading my findings as much as I enjoyed writing them down.

Hasselt, 05.06.2018

Niels Pirotte

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND SYMBOLS

A_{Mont}	Montgomery representation of field element A
$\mathbb{F}_p$	Finite field with p elements
$(G, \circ)$	Group with corresponding group operation $\circ$
$GF(p)$	Galois field with p elements
mP	Point multiplication on an Elliptic Curve
$\mathcal{O}$	Unity element for Elliptic Curve groups
$P + Q$	Point addition of two points of an Elliptic Curve
R	Montgomery parameter
$\mathbb{R}$	The set of real numbers
Z_p^*	Multiplicative group with $p - 1$ elements
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
CDHP	Computational Diffie-Hellman Problem
COSIC	Computer Security and Industrial Cryptography group
CSA	Carry Save Adder
DHKE	Diffie-Hellman Key Exchange
DPA	Differential Power analysis
DSA	Digital Signature Algorithm
DSG	Digital Security Group
DSS	Digital Signature Standard
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman scheme
ECDLP	Elliptic Curve Discrete Logarithm Problem
ES&S	Embedded Systems & Security group
FPGA	Field-programmable gate array
FSM	Finite State Machine
GE	Gate Equivalents
LSB	Least Significant Bit
MM	Montgomery Multiplication
MMM	Montgomery Modular Multiplication algorithm
MALU	Modular ALU
MMALU	Montgomery Multiplier ALU
MSB	Most Significant Bit
MWR2MM	Multiple Word Radix-2 Montgomery Multiplication
NIST	National Institute of Standards and Technology
PKC	Public Key Cryptography
RCA	Ripple Carry Adder
RSA	Rivest-Shamir-Adleman
SCA	Side-channel analysis
SPA	Simple Power analysis

TLS	Transport Layer Security
VHDL	(Very High Speed Integrated Circuit) Hardware Description Language

ABSTRACT

This thesis discusses the first design of an ASIC coprocessor for Elliptic Curve Cryptography (ECC) using the complete addition law of Renes et al. The main reason for using the complete addition law is the reduced vulnerability to side-channel analysis (SCA) attacks, since point addition and point doubling can be performed with the same addition formulas and all inputs are valid, so there is no need for conditional statements handling special cases such as the point at infinity. The proposed hardware architecture is optimized for area efficiency, targeting applications such as smart cards and RFID tags. A bottom-up design approach is used, minimizing the total implementation area by optimizations in each abstraction layer. The design implements a Montgomery Multiplier ALU (MMALU) with built-in adder functionality and is developed in both a full-word version and a scalable version. The two approaches are compared in size and speed. Additionally, an exploration is done on the design parameters of the MMALU and the scheduling of the modular operations in order to minimize the size of the register file. For point multiplication, a Montgomery ladder is implemented with the option of randomizing the execution order of the point operations as a countermeasure against SCA attacks. The post-synthesis implementation results are generated using the open source NanGate 45nm library.

ABSTRACT

Deze masterthesis bespreekt een eerste implementatie van een ASIC coprocessor voor Elliptische Kromme Cryptografie (ECC), welke gebruikmaakt van de complete formules voor puntoptelling van Renes et al. Het belangrijkste voordeel van deze formules is de verhoogde resistentie tegen nevenkanaal aanvallen, omdat er geen onderscheid moet worden gemaakt tussen een puntoptelling en puntverdubbeling. Dit is niet mogelijk met de traditionele aanpak, waar bijvoorbeeld een optelling met het punt op oneindig apart moet worden afgehandeld. De voorgestelde architectuur is geoptimaliseerd naar oppervlakte en richt zich op applicaties zoals RFID tags en smartcards. Een bottom-up design methode is gebruikt, waarbij de oppervlakte van elke abstractielaag is geminimaliseerd. Het ontwerp implementeert een Montgomery Multiplier ALU (MMALU) met ingebouwde opteller functionaliteit. Er is zowel een full-word als een schaalbare versie van deze MMALU ontwikkeld en beide methodes worden met elkaar vergeleken in grootte en snelheid. Ook is er onderzoek gedaan naar de ontwerpparameters van de MMALU en de planning van de modulaire bewerkingen, zodat de register file grootte kon worden geminimaliseerd. Voor puntvermenigvuldiging is een Montgomery ladder geïmplementeerd met de optie voor randomisatie van de volgorde van puntbewerkingen. De post-synthese resultaten zijn bekomen m.b.v. de open source NanGate 45nm bibliotheek.

1. INTRODUCTION

The amount of embedded systems used in everyday life grows rapidly in devices like mobile cell phones, RFID tags and IoT devices. These applications introduce new challenges concerning the protection of data and communication. First of all, they have design constraints due to the very limited implementation surface and energy budget. The second challenge is the physical accessibility of the devices to potential malicious users, which makes the devices vulnerable to side-channel analysis attacks (SCA) [2]. The goal of a SCA attack is not to break the cryptographic algorithms mathematically, but to extract secret information from the physical implementation of the algorithms. For example, the processed information leaks through the power consumption, the timing behavior and the electromagnetic radiation of the device.

When embedded devices need public-key cryptography, Elliptic Curve Cryptography (ECC) [3] [4] is preferred, because of smaller key sizes compared to e.g. RSA [5]. This leads to a reduction of the power consumption and the computational resources. ECC was introduced in 1985 and 1987 independently by Victor Miller [3] and Neal Koblitz [4], respectively. They proposed the use of a group of points on an elliptic curve (EC) to create discrete-log based cryptosystems and defined the addition law for the resulting group structure. This addition law results in a different set of equations for a pair of identical points and for a pair of different points in the group. These equations are called point doubling and point addition, respectively. The most straightforward way of implementing an EC point multiplication, i.e. the basic operation in an EC-based cryptosystem, is through iterative conditional branching of point doublings and point additions. This is disadvantageous for SCA resistance, because the conditional branching reveals information on the executed operation through side channels, as shown by Örs et al. on an Field-Programmable Gate Array (FPGA) implementation [6].

In [7], Bernstein and Lange show the benefits of complete addition formulas, which use the same set of equations for point doubling and point addition. This leads to constant-time and exception-free implementations, mitigating the behavioral effects of branching. While the authors of [7] present their formulas for specific types of curves over binary extension fields, namely Edwards curves, Renes et al. are the first to propose complete addition formulas for the broadly used Weierstrass curves over prime fields. The authors of [8] also notice that in some cases, performance loss will be unnoticeable in comparison to traditional incomplete formulas.

1.1 Contribution

A first ASIC implementation of the complete formulas introduced in [8] is realized. In combination with the implementation of the Montgomery ladder algorithm for point multiplication, this results in an inherently balanced implementation (without dummy operations) of ECC over Weierstrass curves. The design is optimized for area in the interest of lightweight embedded applications in three ways. (1) A full-word and a scalable Montgomery multiplier (MM) with integrated adder functionality are designed, eliminating the need for separate modular addition hardware. Both approaches are compared in speed and implementation surface. (2) A careful exploration of the design parameters and bounds of the Montgomery multiplier is done, in order to minimize the operation count in the point addition formula. (3) The size of the register file is minimized by optimizing the number of registers, by reducing the number of writable registers (through the intelligent use of shift registers), and by optimizing the size of the input multiplexers (by limiting left-operand and right-operand accessibility). Important to note is that a completely balanced operation of the coprocessor mainly protects against simple power analysis (SPA) attacks. Therefore, the randomization of the point operations is integrated as well as a countermeasure against differential power analysis (DPA) attacks.

1.2 Related Work

The first work to introduce a fully balanced ECC implementation was by Batina et al. [9]. The authors modified the non-complete addition formulas over binary extension fields in order to make them balanced. For point multiplication, the Montgomery ladder algorithm was used. The implementation was implemented on an FPGA and the resistance against an SPA attack was evaluated. Examples of elliptic curves with complete addition laws are Edwards curves [7], twisted Edwards curves [10] and twisted Hessian curves [11]. They all operate over binary extension fields. The work of Renes et al. [8] was the first to propose complete addition formulas on Weierstrass curves over prime fields.

In [12], Massolino et al. present the first FPGA implementation of the formulas in [8]. The result is a competitive design emphasizing parallelization possibilities. The design executes a number of field operations simultaneously by using two up to six processors, which increases throughput, but also silicon area. In [13], Chmielewski et al. implement and evaluate three FPGA implementations of the formulas in [8]: a non-protected architecture and two architectures protected by randomization countermeasures for DPA protection. The design develloped in this thesis realizes the first ASIC implementation of the complete formulas of Renes et al., optimized towards minimal silicon area.

1.3 Collaboration

This research is conducted in cooperation with the Digital Security Group (DSG), the Embedded Systems & Security (ES&S) group, and the Computer Security and Industrial Cryptography (COSIC) group. DSG is a research group at Radboud University, specialized in a broad range of topics in computer security and therefore also cryptography. Research topics include applied security protocols, secure hardware implementations for smartcards and RFID, and the security and correctness of software [14]. ES&S is a research group at KU Leuven campus Diepenbeek, working on embedded security, with a focus on configurable hardware and security in constrained environments. Furthermore, COSIC is a research group of the department of electrical engineering (ESAT) at KU Leuven, focusing on a variety of applications concerning cryptography, including the development of security architectures for communication systems and the building of security mechanisms for embedded systems [15].

1.4 Organization

Chapter 2 provides the necessary preliminaries and is divided into two sections. First, Sect. 2.1 will introduce the reader into the world of cryptography. Next, Sect. 2.2 will conclude this chapter with an introduction to ECC and an overview of commonly used design techniques for ECC hardware architectures. In Chapter 3, the design choices and the experimental setup are discussed. Sect. 3.1 will start with an elaboration on the design parameters of the MMM algorithm. Next, the MM design, i.e. for both the full-word as the scalable approach, is discussed in Sect. 3.2. The next sections examine point addition and point multiplication, in that order. Finally, Chapter 4 and Chapter 5 will elaborate on the results and formulate a conclusion respectively.

2. PRELIMINARIES

2.1 Cryptography

A significant part of digital communication takes place via open networks, e.g., the Internet. In an open communication network not only the source and destination have access to a message, but also other users can access data sent over a channel. However, applications handling sensitive information (e.g. transactions and commercial activities) require accessibility of data for only a select group of users. Cryptography provides methods for protecting such applications as it studies secure communication.

Figure 2.1 represents a generalized model to study the security of a communication channel. Herein, a message is sent over a channel from a source to a destination, while a malicious user tries to extract data from the message. To aid comprehension, the source and destination will be personified by Alice and Bob respectively, throughout the remainder of this text. At the same time, the eavesdropper will be represented by Mal.

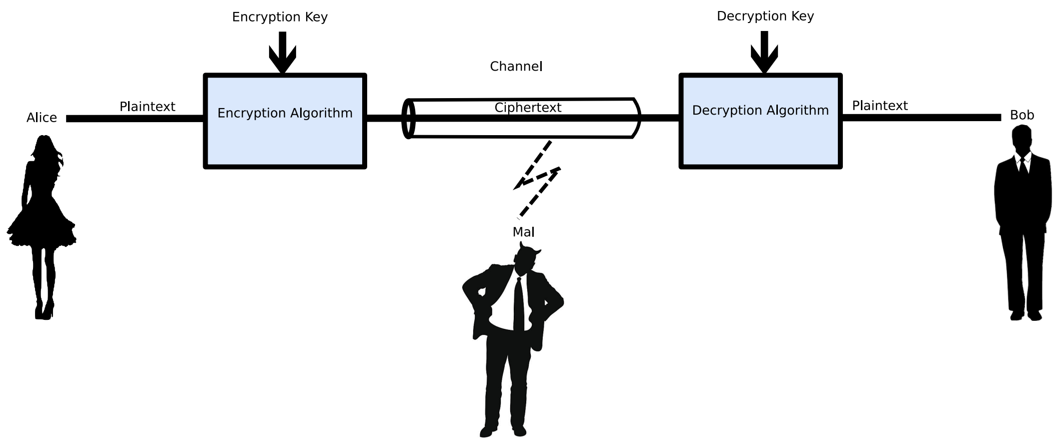


Figure 2.1 *General model for studying the security of a channel*

It is assumed that Mal is always monitoring the data on the channel. Additionally, he can use three different tools to try to extract data. (1) He is a legitimate user of the network and can therefore communicate with any other user in the network. (2) He can intercept messages and optionally prevent them from reaching their destination. (3) He can impersonate any other user in the network and send or receive messages in their name. The model described above is also known as the Dolev-Yao threat model [16, sect. 2.3].

Securing a communication channel considering the Dolev-Yao threat is not an easy task. Also, it is important to note that complete protection of all aspects of the channel is often unnecessary and would go at the expense of software and hardware resources. Therefore, it is usually preferred to only protect certain features of the system's communication process, which is a simpler approach than the defend-everything approach. The required security features are called security objectives and the most commonly required security objectives are listed below [17, sect. 10.1]:

Confidentiality shields information for all but authorized parties.

Data integrity ensures the messages are not altered during transmission.

Data-origin authentication ensures the receiver that the message is sent by the intended sender.

Data authentication encompasses both data integrity and data-origin authentication.

Non-repudiation ensures that the sender cannot deny the creation of the message.

Security schemes, i.e. key exchange, encryption, digital signatures, etc., provide protection to one or more security objectives mentioned above. Furthermore, all techniques used to build these security systems can be divided in three categories: hash functions, private key cryptography and public key cryptography. This thesis examines ECC, which is a subdomain of public key cryptography, as shown in Figure 2.2. A further elaboration on public key cryptography is given in Sect. 2.1.1 and Chapter 2.2 gives an introduction to ECC.

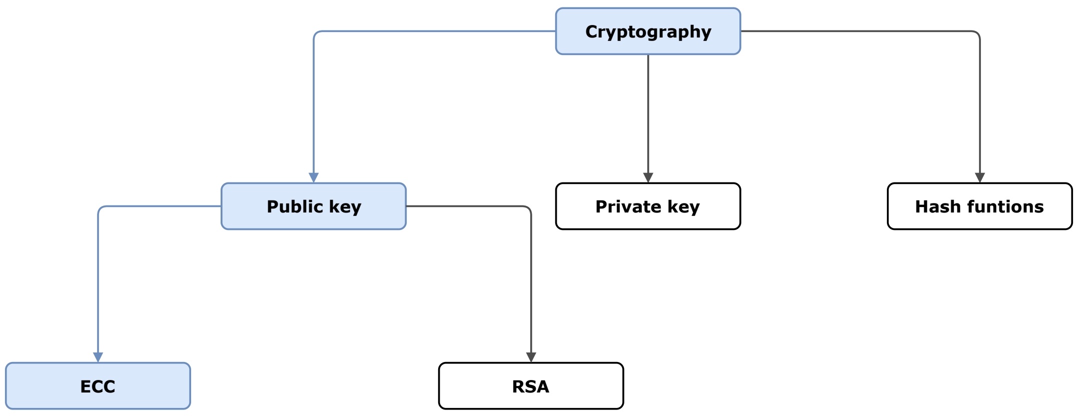


***Figure** 2.2 Situation of ECC in the domain of cryptography*

2.1.1 Public Key Cryptography

Public key cryptography (PKC) was introduced in 1976 by Whitfield Diffie and Martin Hellman in [1]. They proposed a key exchange protocol for establishing a secret key over an insecure channel, called Diffie-Hellman Key Exchange (DHKE). The key feature of DHKE is the computational Diffie-Hellman problem (CDHP), stating that it is hard to obtain g^{ab} in a large prime field $GF(p)$ given the prime p and $g, g^a, g^b \in Z_p^{*}$[1] [16, sect. 8.3]. For this reason, g^{ab} becomes a shared secret between Alice and Bob when both exchange g^a and g^b respectively, while keeping a and b secret. In other words, Diffie and Hellman accomplished secure sharing of the secret field element g^{ab}, without using a prearranged secret. This ability to provide security without prearranged covert communication is the reason for existence of PKC. The DHKE protocol is summarized in Figure 2.3 for clarification.

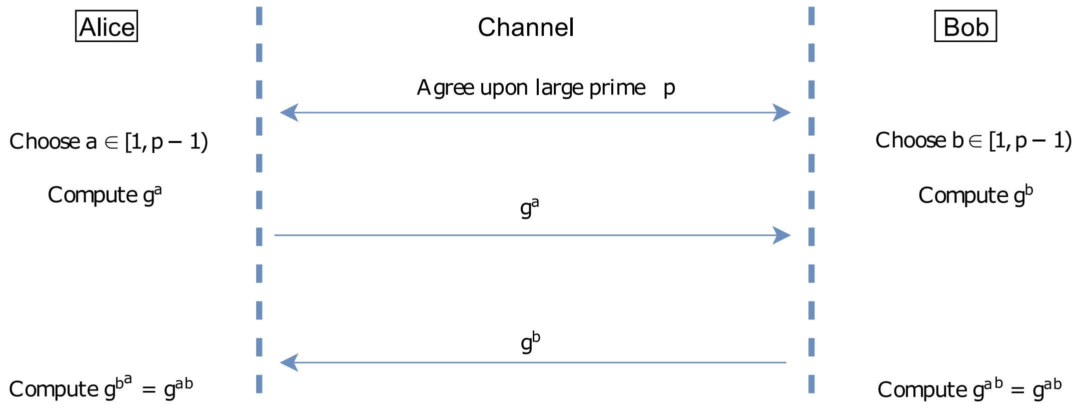


***Figure** 2.3 DHKE as originally proposed by Whitfield Diffie and Martin Hellman in [1]*

What distinguishes private key cryptography from public key cryptography is the fact that the former only uses a single key, while the latter uses a pair of keys. The two keys for PKC are called the public key and the private key. In DHKE, a and b are the private keys for Alice and Bob respectively. g^a and g^b are their corresponding public keys. It is important to note that Mal also has access to the public keys, but because of the CDHP he cannot reconstruct the secret g^{ab}. In general, The public and private keys are a unique set of numerics solving a difficult mathematical equation. As a result, when inserting the public key in the equation, only the corresponding private key can solve the equation, and vice versa. Therefore, the effectiveness of the PKC algorithm is dependent on the difficulty of the underlying mathematical problem, such as the CDHP for DHKE, to protect information.

The difficulty of this mathematical problem is expressed with the level of security of the PKC algorithm, defined as the integer n such that the best known attack on the algorithm requires 2^n steps and is expressed in bits. Proportional with n are the number of bits of the private key, i.e. the key that remains hidden from the public, which determines the key space. The key space is the set of all valid private keys. In [17, chap. 6] an estimation of the level of security of different PKC schemes is given for different key lengths. For long term security(2) a level of security of 128 bits is required. This corresponds to a key size of 3072 bits for DHKE and RSA(3).

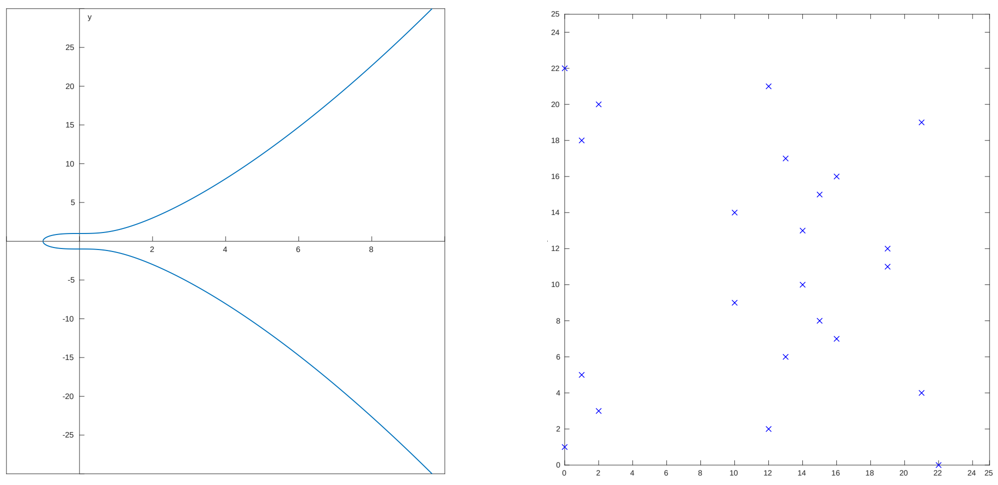
(1) Multiplicative group of order $p - 1$. See Appendix A for more information on group theory.
(2) Several decades in absence of quantum computers.
(3) Dependent on the integer factorization problem.

2.2 Elliptic Curve Cryptography (ECC)

ECC is based on an algebraic group structure[(4)] on an EC. An EC is a set of points (x, y), which are solutions of a polynomial equation defined over a Galois field of the following form[(5)]:

$$y^2 = x^3 + ax + b \tag{2.1}$$

Prime fields, i.e. $GF(p)$, and binary extension fields, i.e. $GF(2^n)$, are the most commonly used finite fields in the context of ECC. In this thesis, only prime fields are considered and therefore a and b in Eq. (2.1) are constants in $GF(p)$ and p is a large prime. To illustrate, Figure 2.4(a) depicts an example of an EC when plotted over $\mathbb{R}^2$. However the actual points in $\mathbb{F}_{23}^2$ included in the corresponding EC group structure are shown in Figure 2.4(b).



(a) Plotted over $\mathbb{R}$ (b) Plotted over $\mathbb{F}_{23}$

Figure 2.4 *Example of an elliptic curve* ($y^2 = x^3 + 1$)

Next, an additive Abelian group can be obtained by defining an addition law $(+)$ on an EC that is nonsingular, requiring $4a^3 + 27b^2 \neq 0 \mod p$. The elements in the group are the points on the EC, along with an additional point called the point at infinity. The point at infinity is denoted by $\mathcal{O}$ and can be expressed as $\mathcal{O} = (x, \infty)$. The obtained Abelian group can be represented as follows:

$$\big(\{(x, y) \mid x, y \in GF(p) \text{ satisfying Eq. (2.1)}\} \cup \mathcal{O}, +\big) \tag{2.2}$$

The addition law can be implemented using the following operations modulo p: addition, subtraction, multiplication and inversion. In hardware, inversions modulo p are very costly operations. Nevertheless, they can be avoided when embedding

(4) See Appendix A for more information on group structures.

(5) This holds for short Weierstrass elliptic curves, which are isomorphic with every possible elliptic curve.

the elliptic curve in the projective plane, i.e. $\mathbb{P}^2(GF(p))$. In order to do so, every point (x, y) is mapped to $(x : y : 1)$ and the point at infinity is mapped to $(0 : 1 : 0)$. The projections $(x : y : z)$ and $(\lambda x : \lambda y : \lambda z)$ are equal, and thus $(\frac{x}{z} : \frac{y}{z} : 1)$ equals $(x : y : z)$. When embedding the EC in the projective space, Eq. (2.1) becomes

$$y^2z = x^3 + axz^2 + bz^3. \tag{2.3}$$

A series of point additions is called a point multiplication or scalar multiplication and is defined as follows:

$$mP = \underbrace{P + P \cdots + P}_{\text{m times}}, \tag{2.4}$$

with P an element of an EC group structure and m a positive integer. For negative m, Eq. (2.4) becomes $m(-P)$. Point multiplication is used in ECC schemes, because the security relies on the elliptic curve discrete logarithm problem (ECDLP). The ECDLP states that it is unfeasible to calculate m, such that $Q = mP$, when the points P and Q are known. Only a 256-bit private key is required for a 128 bits level of security, which is significantly smaller than the 3072 bits necessary in the case of DH or RSA.

A commonly used technique to design implementations of ECC schemes, is a bottom-up design approach, wherein each abstraction level uses the operations of the abstraction level below. Figure 2.5 depicts the different abstraction levels for ECC applications. The remainder of this section discusses these different abstraction levels up to the point multiplication and introduces the algorithms that are used in the design.

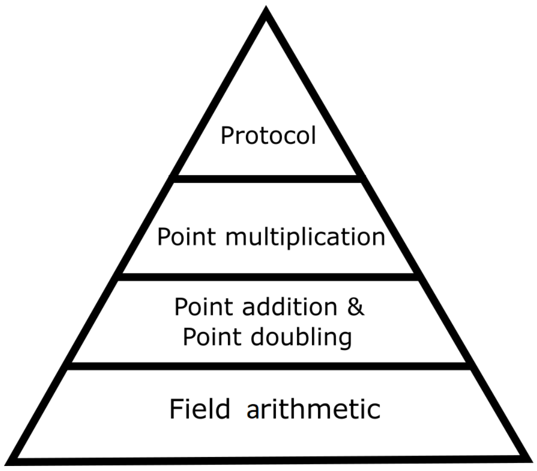


Figure 2.5 *Abstraction levels in the design of an ECC coprocessor*

2.2.1 Field Arithmetic

At the lowest level are the field arithmetic operations. When using the projective representation of the EC, only modular addition, modular subtraction and modular multiplication suffice to implement the addition laws. Modular multiplication modulo p is significantly more complex than modular addition and subtraction, due to the required trial divisions. Therefore, a variety of techniques were designed to speed up or scale down this operation in hardware. For modular multiplication, Montgomery multiplication is a popular technique with a small chip area, a low power consumption and a high throughput in mind. The Montgomery Modular Multiplication (MMM) algorithm was introduced by Peter Montgomery in 1985 [18]. In 1999, Colin Walter suggested an improvement, which made the final reduction at the end of the original algorithm unnecessary by introducing modified input bounds as well as a lower bound for the Montgomery parameter [19]. This improvement is advantageous for SCA resistance since it leads to a time-constant implementation. The Montgomery multiplication algorithm, which is used to design the full-word MMALU, is given in Algorithm 1.

Algorithm 1: Montgomery Modular Multiplication [19]

```
Input  : $A = (a_{n-1}, \dots, a_0)_r, B = (b_{n-1}, \dots, b_0)_r, p =$
         $(p_{n-1}, \dots, p_0)_r, R$ (with $RR^{-1} = 1 \mod p$)
Output: $S = \mathrm{MMM}(A, B) = ABR^{-1} \mod p$
1 $S := 0$;
2 for $i \leftarrow 0$ to $n-1$ do
3     $q_i := (s_0 + a_i b_0)(-p_0^{-1}) \mod r$;
4     $S := (S + a_i B + q_i p) >> r$;
5 end
6 Return $S$;
```

In Algorithm 1, a_i stands for the i-th digit of the word A. When A is an n-digit number expressed in base r, A can be written as $A = \sum_{i=0}^{n-1} a_i r^i$. B and p are represented in a similar way.

The power of the Montgomery algorithm is the ability to calculate a modular operation by replacing costly trial divisions by a prime with divisions by a power of 2. The latter comes for free in hardware, by implementing logical shift operations. MMM, working on the input operands A and B, computes $ABR^{-1} \mod p$, where p is the modulus and R^{-1} is an element of the finite field satisfying $RR^{-1} - pp' = 1$ or identically $RR^{-1} = 1 \mod p$(6)(7). R is a power of two, i.e. r^n, such that $R = r^n > p$; R is also called the Montgomery parameter. The algorithm becomes useful when the operands are first transformed to Montgomery representation, mapping A and B to respectively $A_{Mont} = AR \mod p$ and $B_{Mont} = BR \mod p$. This way, Montgomery multiplication can be used to calculate modular multiplications, because the algorithm ensures $S_{Mont} = SR \mod p = \mathrm{MMM}(A_{Mont}, B_{Mont})$, with $S = AB \mod p$. At the end of the calculations in the Montgomery domain, the result needs to be converted back. As a consequence, Montgomery multiplication requires two additional steps. However, this overhead becomes negligible when a large series

(6)Bézout's identity ensures the existence of R^{-1} [18] [19].

(7)R^{-1} and p' can be calculated using the extended Euclidean algorithm.

of consecutive operations is performed, which is the case in the context of scalar multiplication.

2.2.2 Point Addition & Point Doubling

The next abstraction layer involves the point operations, namely point addition and point doubling. As mentioned in Chapter 1, point doubling and point addition use a different set of equations in most addition laws. When implemented in hardware or software, this leads to different execution times and power traces for both, inevitably leaking information on the scalar m in the computation of the point multiplication mP. Evidently, this side-channel leakage needs to be avoided to avert weakening the implemented ECC scheme.

For this reason, the design realized in Chapter 3 implements the complete addition law proposed in [8], which utilizes the same addition formulas for point addition and point doubling. More specifically, Algorithm 7 in [8] is used, since it has a minimal number of operations and no input restrictions. This algorithm targets short Weierstrass curves with $a = 0$, i.e. j-invariant 0 curves. These curves are also used in practice, such as the *secp256k1* curve used in the Bitcoin Protocol [20].

The concerned formulas of Algorithm 7 in [8], giving new point coordinates $(X_3 : Y_3 : Z_3)$ dependent on input points $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$, are

$$\begin{aligned}
X_3 =& (X_1Y_2 + X_2Y_1)(Y_1Y_2 - 3bZ_1Z_2) \\
&- 3b(Y_1Z_2 + Y_2Z_1)(X_1Z_2 + X_2Z_1), \\
Y_3 =& (Y_1Y_2 + 3bZ_1Z_2)(Y_1Y_2 - 3bZ_1Z_2) \\
&+ 9bX_1X_2(X_1Z_2 + X_2Z_1), \\
Z_3 =& (Y_1Z_2 + Y_2Z_1)(Y_1Y_2 + 3bZ_1Z_2) \\
&+ 3X_1X_2(X_1Y_2 + X_2Y_1).
\end{aligned} \tag{2.5}$$

2.2.3 Point Multiplication

The most straightforward algorithm for point multiplication is the double-and-add algorithm as shown in Algorithm 2. This method uses a left-to-right bit scan of the multiplier while performing consecutive point additions and point doublings. To overcome the side-channel leakage caused by the conditional branches in the algorithm, a better solution is to use the Montgomery ladder algorithm for point multiplication. The Montgomery ladder, as presented by Peter Montgomery in [21], is shown in Algorithm 3. In contrast with the double-and-add method, each iteration executes the same operations, namely one point addition and one point doubling.

Algorithm 2: Double-and-add algorithm for point multiplication

```
Input  : $P$ ,
         $m = (m_{t-1}, m_{t-2}, \ldots, m_0)_2$ with $m_{t-1} = 1$
Output: $R = mP$
$R := P$;
for $i \leftarrow t-2$ to $0$ do
    $R := double(R)$;
    if $m_i = 1$ then
        $R := add(R, P)$;
    end
end
Return $R$;
```

Algorithm 3: Montgomery ladder [21] for point multiplication

```
Input  : $P$, $m = (m_{t-1}, \ldots, m_0)_2$ with $m_{t-1} = 1$
Output: $R = mP$
$R_0 := P$;
$R_1 := 2P$;
for $i \leftarrow t-2$ to $0$ do
    if $m_i = 1$ then
        $R_0 := R_0 + R_1$;
        $R_1 := 2R_1$;
    else
        $R_1 := R_0 + R_1$;
        $R_0 := 2R_0$;
    end
    {$R_1 - R_0$ remains invariant}
end
$R := R_0$;
Return $R$;
```

When using the Montgomery ladder in Algorithm 3, first a point addition and then a point doubling is performed, independent of m_i. In [22], the authors introduce additional randomization by using a random bit which decides on the execution order of both operations. This order introduces an additional uncertainty independent of the key, further complicating SCA attacks. The resulting Montgomery ladder is shown in Algorithm 4. With a minor improvement (explained in Sect. 3.4), this algorithm is applied in the ASIC design realized in Chapter 3.

2.2.4 ECC Applications

The uppermost abstraction layer in Figure 2.5 implements protocols that use the EC point multiplication to construct secure communication services. To show an example of how ECC can be used, this section sums up the ECC schemes implemented

Algorithm 4: Montgomery ladder for point multiplication with random order execution [22]

Input : P, $m = (m_{t-1}, \dots, m_0)_2$ with $m_{t-1} = 1$, random bits $r_{t-2}, \dots, r_0$
Output: $R = mP$

```
$R_0 := P$;
$R_1 := 2P$;
for $i \leftarrow t-2$ to $0$ do
    if $m_i = 1$ then
        if $r_i = 0$ then
            $T_0 := R_0 + R_1$ ; $T_1 := 2R_1$;
            $R_0 := T_0$ ; $R_1 := T_1$;
        else
            $T_1 := 2R_1$ ; $T_0 := R_0 + R_1$;
            $R_0 := T_0$ ; $R_1 := T_1$;
        end
    else
        if $r_i = 0$ then
            $T_1 := R_0 + R_1$ ; $T_0 := 2R_0$;
            $R_0 := T_0$ ; $R_1 := T_1$;
        else
            $T_0 := 2R_0$ ; $T_1 := R_0 + R_1$;
            $R_0 := T_0$ ; $R_1 := T_1$;
        end
    end
    {$R_1 - R_0$ remains invariant}
end
$R := R_0$;
Return $R$;
```

in OpenSSL, i.e. an open-source cryptographic library for Transport Layer Security (TLS). ECC is most commonly used for key exchange and for creating digital signatures. However, other usages include encryption and pseudorandom number generators.

The first ECC based scheme implemented in OpenSSL is Elliptic Curve Diffie-Hellman (ECDH) key exchange. ECDH is the elliptic curve variant of the DHKE protocol and Figure 2.6 gives an overview of the functional principle. Alice and Bob can obtain a shared secret key, namely the EC point $(kl)G$. Note that in ECDH, exponentiation in F_p^* is swapped for point multiplication in the EC group.

Next, OpenSSL includes the Elliptic Curve Digital Signature Algorithm (ECDSA). A digital signature provides for three security objectives as discussed in Sect. 2.1, i.e. message authentication, integrity and non-repudiation. They are used as proof that a sender saw and/or created the signed data and were given legal binding in the United Nations International Trade Law in July 2001. This means a digital

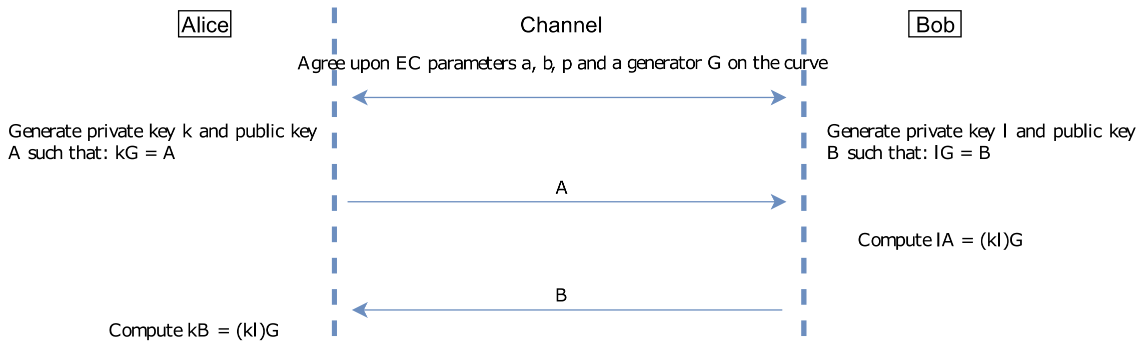

Alice
Channel
Bob
Agree upon EC parameters a, b, p and a generator G on the curve
Generate private key k and public key A such that: kG = A
Generate private key l and public key B such that: lG = B
A
Compute lA = (kl)G
B
Compute kB = (kl)G

Figure 2.6 *The ECDH protocol*

signature has as much legal value as a written signature. The ECDSA scheme is a variant of the digital signature algorithm (DSA) as proposed in 1991 by the National Institute of Standards and Technology (NIST). A more thorough description of the ECDSA can be found in the Digital Signature Standard (DSS) [23].

3. DESIGN

This chapter elaborates on the functionality and design choices of the ASIC implementation in a bottom-up way, following the abstraction layers in Figure 2.5. After giving an overview of the design parameters of full-word Montgomery multipliers, the Montgomery Modular ALU (MMALU) is discussed, which enables modular multiplication, addition and subtraction. This includes both a full-word as a scalable implementation of the MMALU. Hereafter, the design choices for the control logic, implementing the complete point addition formulas in Eq. (2.5), are explained. Finally, the Montgomery ladder implementation for point multiplication is discussed.

3.1 Design Parameters of the MMM

This paragraph discusses the influence of certain design parameters on the function of Algorithm 1, inspired by the doctoral thesis of Lejla Batina [24]. Especially important is the relation between the input bounds and the Montgomery parameter R. This relationship influences speed and area, because it is directly linked to the required number of iterations of the algorithm and the size of the input and output registers. To understand the influence of these parameters, the operation of the algorithm is explained below. For simplicity, it is assumed that all data is expressed in binary form and therefore the base r is 2.

Working principle

The correctness of the MMM algorithm depends on two statements, both easily verified via induction. The first statement,

$$0 \leq S < p + B, \tag{3.1}$$

holds during each operation of the for loop in Algorithm 1 and ensures the output remains bounded. After n iterations, $S = ABR^{-1} \mod p$ with $R = r^n$. This is verifiable with the help of the second statement, in which $Q = (q_{n-1}, \ldots, q_0)$:

$$RS = AB + Qp \Rightarrow S = ABR^{-1} \mod p \tag{3.2}$$

Divisibility by 2

In Algorithm 1, addition with $q_i p$ always ensures an outcome divisible by 2 in the last step of the for loop. In other words, the least significant bit (LSB) of the outcome on line 4 is 0. This is easily validated by replacing q_i with the expression on line 3 in Algorithm 1. As a result, the LSB of S becomes:

$$\begin{aligned} & s_0 + a_i b_0 + q_i p_0 \\ = \; & s_0 + a_i b_0 + \big((s_0 + a_i b_0)(-p_0^{-1})\big) p_0 \\ = \; & s_0 + a_i b_0 + (s_0 + a_i b_0)(-1) \\ = \; & 0 \end{aligned}$$

Upper Bound on Q

Q cannot be bigger than $R - 1$, because Q has maximum n bits and therefore has a maximum value of $r^n - 1 = R - 1$.

Lower Bound on the Inputs

A design parameter with significant influence on the implementation is the bound on the inputs; it directly determines the size of the inputs, the output and the intermediate registers in the implementation. Let us assume that the inputs are bounded by a multiple k of the modulus p:

$$A, B < kp \overset{\text{Eq. 3.1}}{\Rightarrow} S < p + kp = (k+1)p$$

Thus, when the inputs are smaller than k times the modulus p, the input registers need to store $\lceil log_r(k) \rceil$ more bits than the number of bits needed to represent p. Further, the intermediate result needs $\lceil log_r(k+1) \rceil$ bits more than p.

Lower Bound on the Montgomery Parameter

The Montgomery parameter ($R = r^n$) determines the number of iterations (n) to obtain the result. It is assumed that the lower bound on the Montgomery parameter is a multiple l of p.

Lower Bound on the Output

The relation between the Montgomery parameter and the input bound has an important effect on the output of the MMM. When assuming that both the inputs and the Montgomery parameter are bounded by a multiple ($< kp$ and $> lp$, respectively)

Parameter	**Bound**	**# bits**
Prime (p)	-	$\boldsymbol{x}$
Montgomery parameter ($R = r^n$)	$> \boldsymbol{lp}$	$n+1$
Inputs	$< \boldsymbol{kp}$	$x + \lceil log_2(k) \rceil$
# iterations	$= n$	$\lceil log_2(n) \rceil$
Intermediate result (after shift)	$< (k+1)p$	$x + \lceil log_2(k+1) \rceil$
Intermediate result (before shift)	$< 2(k+1)p$	$x + \lceil log_2(k+1) \rceil + 1$
Output	$< \big(\frac{k^2}{l} + 1\big)p$	$x + \lceil log_2\big(\frac{k^2}{l} + 1\big) \rceil$

Table 3.1 *Influence of the boundaries of the MMM design parameters on register sizes*

of p, S becomes bounded by:

$$\begin{aligned} & A, B < kp \text{ and } R > lp \\ \overset{\text{Eq. 3.2}}{\Rightarrow} \; & S = \frac{AB}{R} + \frac{Qp}{R} \\ \Rightarrow \; & S < \frac{k^2p^2}{R} + \frac{(R-1)p}{R} \\ \Rightarrow \; & S < \frac{k^2p^2}{lp} + \frac{(R-1)p}{R} \\ \Rightarrow \; & S < \frac{k^2}{l}p + p - \frac{p}{R} \\ \Rightarrow \; & S < \big(\frac{k^2}{l} + 1\big)p \end{aligned}$$

As an example, assume that the input bounds are $A, B < 2p$ and the Montgomery parameter is larger than $4p$. Then the output remains smaller than $2p$, in accordance with the original result in the paper of Walter, enabling MMM without final subtraction [19].

Summary

Table 3.1 gives an overview of the design parameters of the MMM for a prime of x bits. By carefully selecting the lower bound of the Montgomery parameter (lp) and the upper bound of the inputs (kp), the number of iterations (n) and the bound on the output and the intermediate values can be determined. These bounds determine the register sizes in the design and are therefore important design parameters.

	Upper limit	**Number of bits**
Prime	p	k
Montgomery parameter	$2^{(k+4)}(> 16p)$	$k + 5$
Inputs	$4p$	$k + 2$
# iterations	$k + 4$	$\lceil log_2(k + 4)\rceil$
Intermediate result (after shift)	$5p$	$k + 3$
Intermediate result (before shift)	$10p$	$k + 4$
Output	$2p$	$k + 1$

Table 3.2 *Necessary recourses of the MMALU, with corresponding upper bounds for* $R > 16p$

3.2 Montgomery Modular ALU

Most literature on Montgomery multipliers in hardware is focused on fast implementations of the MMM algorithm, often at the expense of area efficiency. For these fast implementations, systolic arrays (e.g. [25]), pipelining (e.g. [26]) and high-radix approaches (e.g. [27]) are very popular. However, this paper focuses on low silicon area implementations. As a result, a full-word implementation of the Montgomery multiplication algorithm without final subtraction [19], inspired by the Modular ALU (MALU) design of Sakiyama et al. [28], was chosen.

3.2.1 Full-Word MMALU

As discussed in Sect. 3.1, the MMM has a bounded output dependent on the upper bound of the inputs and the lower bound of the Montgomery parameter R. When the inputs of the MMM are limited to numbers smaller than $4p$, the intermediate results of Algorithm 1 are numbers smaller than $5p$ (after the shift operation). If the Montgomery parameter is chosen to be larger than $16p$, the output is bounded by $2p$. Table 3.2 summarizes the previous reasoning.

The functionality of the MMALU can be selected using two control signals, *cmd* and *sub*. This can also be seen in Figure 3.1. When *cmd* is set to 0, a MMM is performed on the input operands. When operating in MMM mode, the *sub* control signal selects either the MMM of the two input operands, i.e. when *sub* equals 0, or otherwise the MMM of one input operand with the second input operand equal to 1. The second feature can be used to scale down the input operand to ensure an outcome smaller than or equal to p, because

$$\begin{aligned}
& A < 4p \text{ and } B = 1 \text{ and } R > 16p \\
\overset{\text{Eq. 3.2}}{\Rightarrow} \; & RS = AB + Qp \\
\Rightarrow \; & RS < 4p \cdot 1 + (R-1)p \\
\Rightarrow \; & RS < (R+3)p \\
\Rightarrow \; & S < p + \frac{3p}{16p} \\
\Rightarrow \; & S \leq p
\end{aligned}$$

Normally, the MMM of an input operand and 1 transforms the input from the Montgomery representation back to the original domain. However, when performed on all coordinates of a point on the EC, this operation only results in a scaling. Remember that the complete formulas to be implemented have inputs and an output in the projective space. Consequently, The formulas yield the same output for scalar multiples of either of the input points, i.e. the output remains unaffected when loading a set of inputs (X, Y, Z) or $(\lambda X, \lambda Y, \lambda Z)$ for either input point. It should also be noted that this attribute renders transformation of the input coordinates, i.e. $\{X_1, Y_1, Z_1, X_2, Y_2, Z_2\}$, to their Montgomery representation and transformation of the output coordinates back to the original domain unnecessary. This follows directly from $(X : Y : Z) = (RX : RY : RZ)$.

Figure 3.2 gives a conceptual view of the MMALU when $cmd = 0$. This is a straightforward implementation of Algorithm 1 with two ripple carry adders (RCA).

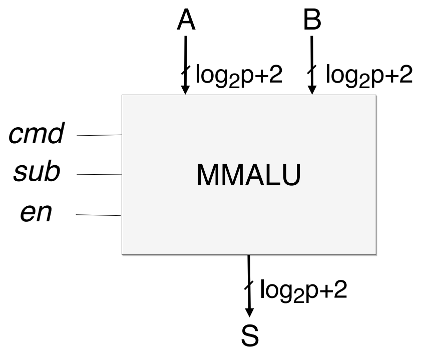


Figure 3.1 *Interface of the MMALU functional block*

Figure 3.3 shows the functionality of the MMALU when $cmd = 1$. In this mode, the integrated adder functionality is selected. As a result, two field element inputs can be added or subtracted depending on the value of *sub*. When *sub* equals 0, the adder adds two $2p$ inputs to an output of maximum $4p$. Additionally, subtraction is implemented when *sub* equals 1. The same output range is available, due to the addition of $2p$ with use of the second RCA, after subtracting with the first RCA. Without this extra addition of $2p$, the output would be larger than $-2p$ and smaller than $2p$.

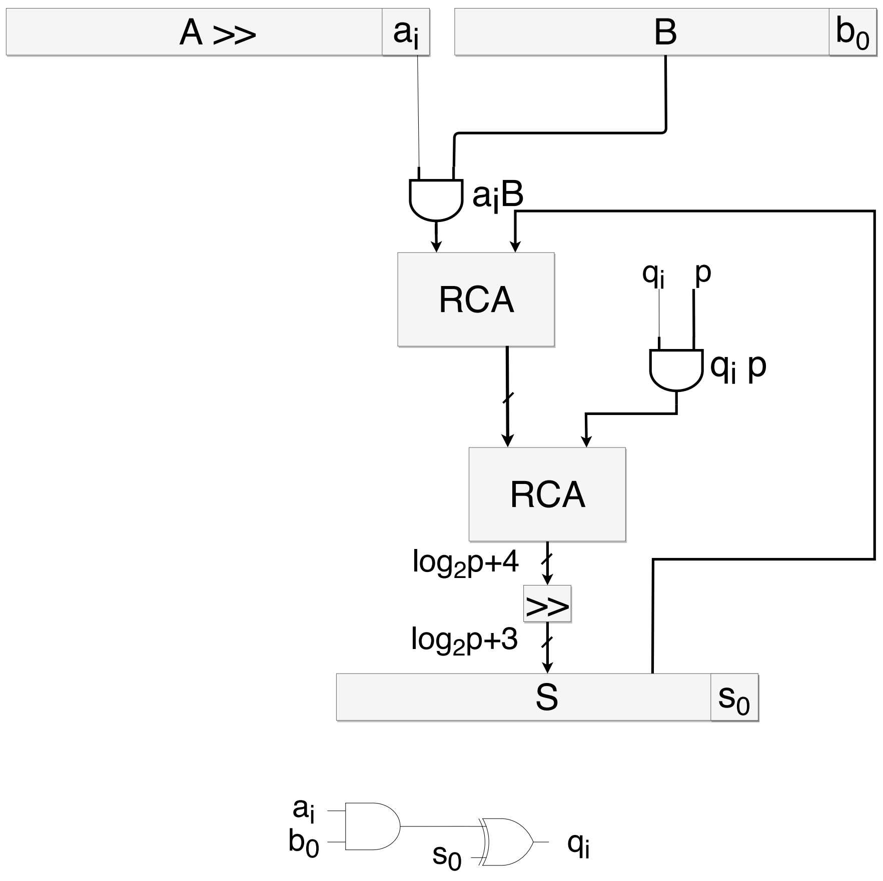


Figure 3.2 *Architecture of MMALU when* $cmd = 0$

In summary, the MMALU can perform addition and subtraction without dedicated modular addition hardware, which ensures outputs between 0 and $4p$ when handling inputs smaller than $2p$. Therefore, an output of the MMALU in MMM mode, which takes $4p$ inputs and returns a $2p$ output, can be used as input of a subsequent addition. More generally, a series of additions, subtractions and multiplications becomes possible, when the input bounds of each operation are respected. This also means that a series of subsequent additions is not always possible without intermediate scaling or MMM, which can increase the number of operations of the respective algorithm. Nevertheless, other properties of the MMM are useful for performing such a series of operations. For example, three outputs of Montgomery multiplications of values smaller than $2p$ can be added resulting in an output smaller than $4p$. This is because an MMM with two $2p$ input values results in the following output bound:

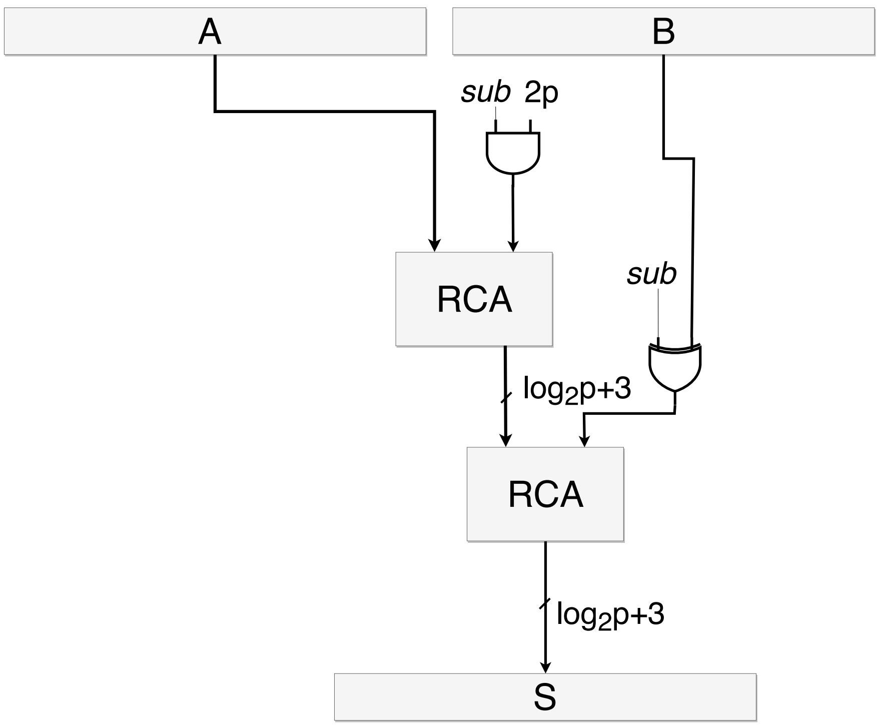


Figure 3.3 *Architecture of MMALU when cmd = 1*

$$
\begin{aligned}
& A, B < 2p \text{ and } R > 16p \\
\overset{\text{Eq. 3.2}}{\Rightarrow}\ & S = \frac{AB}{R} + \frac{Qp}{R} \\
\Rightarrow\ & S < \frac{4 \cdot p^2}{R} + \frac{(R-1)p}{R} \\
\Rightarrow\ & S < \frac{4 \cdot p^2}{16 \cdot p} + \frac{(R-1)p}{R} \\
\Rightarrow\ & S < (\frac{1}{4} + 1)p - \frac{p}{R} \\
\Rightarrow\ & S < \frac{5}{4}p
\end{aligned}
$$

Table 3.3 specifies all functions of the MMALU. All calculations must respect these upper bounds on the input, otherwise a correct operation of the MMALU is not guaranteed. As a consequence of the optimizations made, not all algorithms will be directly compatible with this design, i.e. without appending scaling operations.

3.2.2 Scalable MMALU

When using the previous full-word design, the data path size and thus the implementation surface, increases with increasing key size. In 1999, Tenca and Koç proposed a scalable hardware architecture in [29], which can generate multi-precision results independent of the data path precision. They proposed to split the multiplicand into

Functionality	**Input**	**Output**
add/subtract	$2 \times 2p$	$1 \times 4p$
add/subtract	$3 \times \frac{5}{4}p$	$1 \times 4p$
add	$2 \times p$	$1 \times 2p$
multiply	$2 \times 4p$	$1 \times 2p$
multiply	$2 \times 2p$	$1 \times \frac{5}{4}p$
scale	$1 \times 4p$	$1 \times p$

Table 3.3 *Bounds of the inputs and outputs of the MMALU*

words in order to reduce the data path size. This resulted in the algorithm called the Multiple Word Radix-2 Montgomery Multiplication algorithm (MWR2MM), shown in Algorithm 5 below. To study the influence of the scalability in Algorithm 5, a second MMALU design inspired by the MWR2MM algorithm was designed.

Algorithm 5: *MWR2MM(A, B)*

```
Input  : A , B
Output: S
1  $S := 0$;
2  for $i \leftarrow 0$ to $n-1$ do
3      $q_i := (S_0^{(0)} + a_i B_0^{(0)})(-p^{-1})_0^{(0)} \mod r$;
4      $(C, S^{(0)}) := a_i \times B^{(0)} + S^{(0)} + q_i \times p^{(0)}$;
5      for $j \leftarrow 1$ to $e-1$ do
6          $(C, S^{(j)}) := C + a_i \times B^{(j)} + S^{(j)} + q_i \times p^{(j)}$;
7          $S^{(j-1)} := \left(S_0^{(j)}, S_{w-1\ldots1}^{(j-1)}\right)$;
8      end
9      $S^{(e-1)} := \left(C, S_{w-1\ldots1}^{(e-1)}\right)$;
10 end
11 Return $S$;
```

It is important to note that the observations made in Sect. 3.1 are still applicable. Also, the MWR2MM algorithm can be parallelized by implementing pipelines, although these pipelines, while speeding up calculations, are not preferable for designs requiring minimal silicon space.

The main principle of Algorithm 5 is the same as in Algorithm 1, with exception of the inner loop to enable word by word scanning of the multiplicand, comprised of e words of w bits, i.e. $e = \lceil \frac{m}{w} \rceil$ with m the number of bits of the intermediate result S. A subscript is used to distinguish between the different bits of a word and a superscript to indicate different words of a number. For example, $B_k^{(j)}$ represents the k-th bit of the j-th word of a number. The carry register in the inner loop needs to be at least two bits, to ensure containment during the entire iteration. This is a result of the following inequality:

$$\begin{aligned} & 3(2^w - 1) + C_{max} \leq C_{max}2^w + 2^w - 1 \\ \Rightarrow \ & C_{max} \geq 2 \end{aligned}$$

This inequality states that during the next iterations, the sum of three maximum words plus the maximum carry needs to be contained in the vector of the maximum carry concatenated with a one digit word.

The scalable MMALU is implemented with the same interface as in Figure 3.1. However, carry save adders (CSA) are used to successfully pass on the carry to subsequent iterations, while scanning the inner loop. A 2-bit register is used to store the carry for the next iteration. Additionally, the adders for Montgomery multiplication are reused for addition and subtraction as was the case for the full-word design. For simplicity, a data path size of a single bit is chosen. The resulting architecture is shown in Figure 3.4 for Montgomery multiplication and scaling, i.e. when *cmd* $= 0$, and in Figure 3.5 for addition and subtraction, i.e. when *cmd* $= 1$.

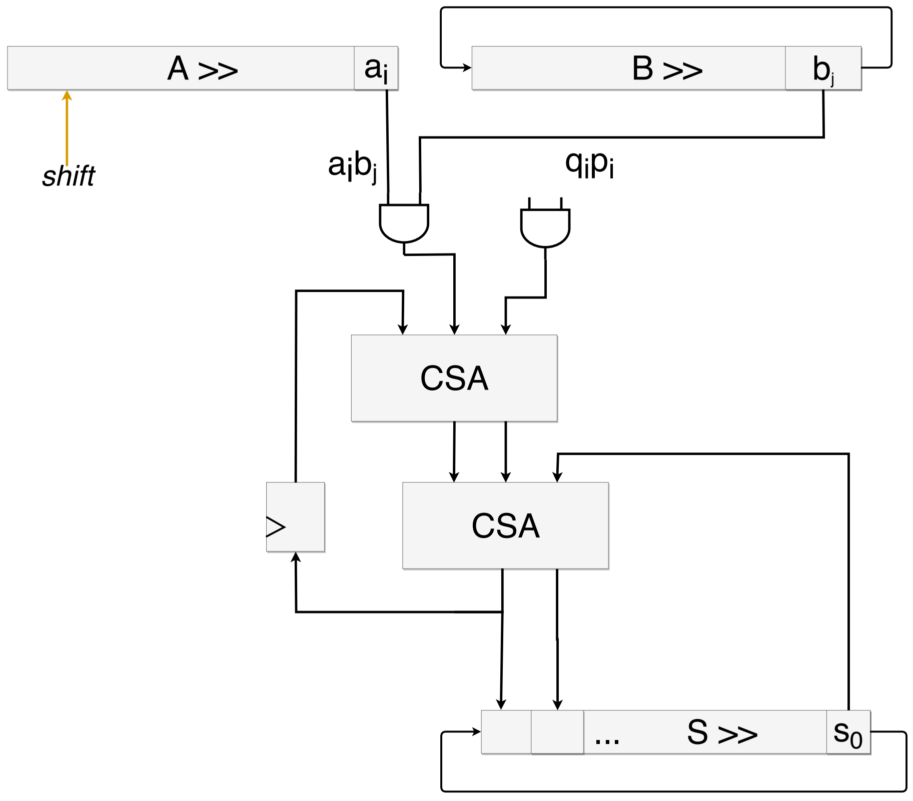


Figure 3.4 *Architecture of the scaled MMALU when cmd* $= 0$

Notice that for register B and S rotating registers were used, i.e. instead of shifting out the LSB, it becomes the MSB during the next iteration. During the innerloop in the MWR2MM algorithm, the rotating registers keep shifting, while register A only shifts during the next iteration in the outerloop, i.e. when $shift = 1$. As a consequence, the control logic becomes more extensive. Also, the signal q_i, which

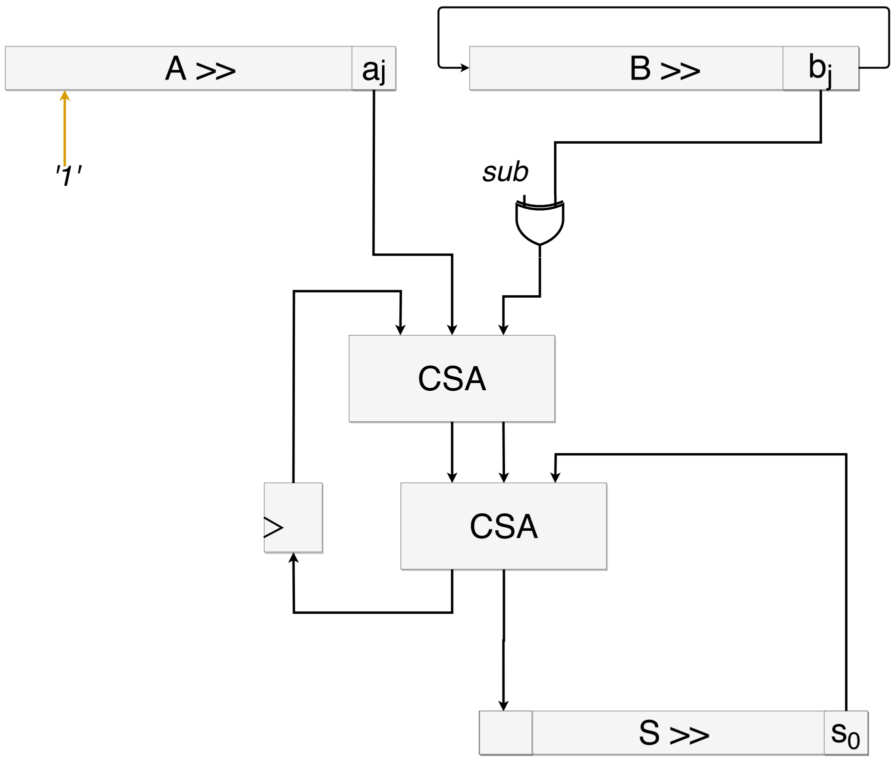


Figure 3.5 *Architecture of the scaled MMALU when cmd* $= 1$

needs to remain constant while iterating over the innerloop, is stored in a flip flop and updated only on a shift pulse. Finally, it should be noted that when subtracting, the output register S is initialized to $2p$ to ensure an output in the range between 0 and $4p$.

3.3 Point Addition

Moving one abstraction level up in Figure 2.5, the next step is to implement the complete point addition using Algorithm 7 in [8], given in Eq. (2.5), using the previously designed MMALU. An overview of common architectures for ECC processor designs is given in [30]. A finite state machine (FSM) is implemented to integrate the rules for point addition in combination with a register file to store the intermediate results.

In [8], the authors already present an implementation algorithm for the proposed addition law in the form of consecutive field operations. Algorithm 4 is a modified version of this sequence of operations. These optimizations were performed with use of the register lifetime graphs presented in Appendix B. First, the register lifetime of the original algorithm is given. Next, the second graph rearranges the operations, taking into account the restrictions given in Table 3.3 and the optimizations explained in the previous paragraph. This results in a minimization of the number of registers and the size of the multiplexers.

(1)Register data of t_1 is shifted into t_0 ($t_0 \leftarrow t_1$).

Algorithm 4: Complete, projective point addition for prime order j-invariant 0 short Weierstrass curves $E/F_p : y^2 = x^3 + b$

```
Require: $P = (X_1 : Y_1 : Z_1)$, $Q = (X_2 : Y_2 : Z_2)$ on $E : Y^2Z = X^3 + bZ^3$ and $b3 = 3 \cdot b$
Ensure: $(X_3 : Y_3 : Z_3) = P + Q$

1. $t_1 \leftarrow X_1 \cdot X_2$
2. $t_1 \leftarrow Y_1 \cdot Y_2$ $^{(1)}$
3. $t_2 \leftarrow Z_1 \cdot Z_2$
4. $t_3 \leftarrow X_1 + Y_1$
5. $t_4 \leftarrow X_2 + Y_2$
6. $t_3 \leftarrow t_3 \cdot t_4$
7. $t_4 \leftarrow t_0 + t_1$
8. $t_3 \leftarrow t_3 - t_4$
9. $t_4 \leftarrow Y_1 + Z_1$
10. $Y_2 \leftarrow Y_2 + Z_2$
11. $Y_1 \leftarrow t_4 \cdot Y_2$
12. $Y_2 \leftarrow t_1 + t_2$
13. $t_4 \leftarrow Y_1 - Y_2$
14. $Y_2 \leftarrow X_1 + Z_1$
15. $Z_1 \leftarrow X_2 + Z_2$
16. $Z_1 \leftarrow Y_2 \cdot Z_1$
17. $Y_1 \leftarrow t_0 + t_2$
18. $Y_1 \leftarrow Z_1 - Y_1$
19. $X_1 \leftarrow t_0 + t_0$
20. $t_1 \leftarrow X_1 + t_0^{(1)}$
21. $Y_2 \leftarrow b3 \cdot t_2$
22. $Z_1 \leftarrow t_0 + Y_2$
23. $t_1 \leftarrow t_0 - Y_2^{(1)}$
24. $Y_1 \leftarrow b3 \cdot Y_1$
25. $X_1 \leftarrow t_4 \cdot Y_1$
26. $Y_2 \leftarrow t_3 \cdot t_1$
27. $X_1 \leftarrow Y_2 - X_1$
28. $Y_1 \leftarrow Y_1 \cdot t_0$
29. $Y_2 \leftarrow t_1 \cdot Z_1$
30. $Y_1 \leftarrow Y_2 + Y_1$
31. $t_1 \leftarrow t_0 \cdot t_3^{(1)}$
32. $Z_1 \leftarrow Z_1 \cdot t_4$
33. $Z_1 \leftarrow Z_1 + t_1$
34. $X_1 \leftarrow scale(X_1)$
35. $Y_1 \leftarrow scale(Y_1)$
36. $Z_1 \leftarrow scale(Z_1)$
```

Originally, algorithm 7 in [8] presumes 14 registers, i.e. 6 input, 3 output and 5 temporary registers. The attained design utilizes only 11 registers, reusing 3 input registers (X_1, Y_1 and Z_1) as output registers. In addition, the algorithm ensures only 3 address bits to select the left and right operand for the MMALU and to select the write address. This is accomplished by organizing the formulas in Algorithm 4 such that maximum 8 registers need to be accessible as left or right operand. In order to reduce the number of writable registers, register t_0 is not directly writable; instead, when writing to t_1, the value of t_1 is shifted to t_0. This is incorporated in Algorithm 4, which writes to t_0 and t_1 alternately. Figure 3.6 gives the resulting architecture implementing the point addition.

3.4 Point Multiplication

The next design stage is implementing point multiplication with a Montgomery ladder as described in Algorithm 3. Important to note is that this algorithm assumes the MSB of m is not 0, therefore cutting the key space in half and making brute force attacks more feasible. This assumption was made to avoid addition with the point at infinity, which was not possible while using the typical addition formulas. However, this is not the case for their complete counterparts. Therefore, the algorithm can be altered, restoring all possibilities in the key space. In order to do so, R_0 and R_1 are initialized to $\mathcal{O}$ and P respectively and the loop must iterate through all bits of m, i.e. from $t - 1$ to 0.

It should be noted that by using the Montgomery ladder instead of the double-and-add method, a possible speedup and reduction of resources is not feasible. The

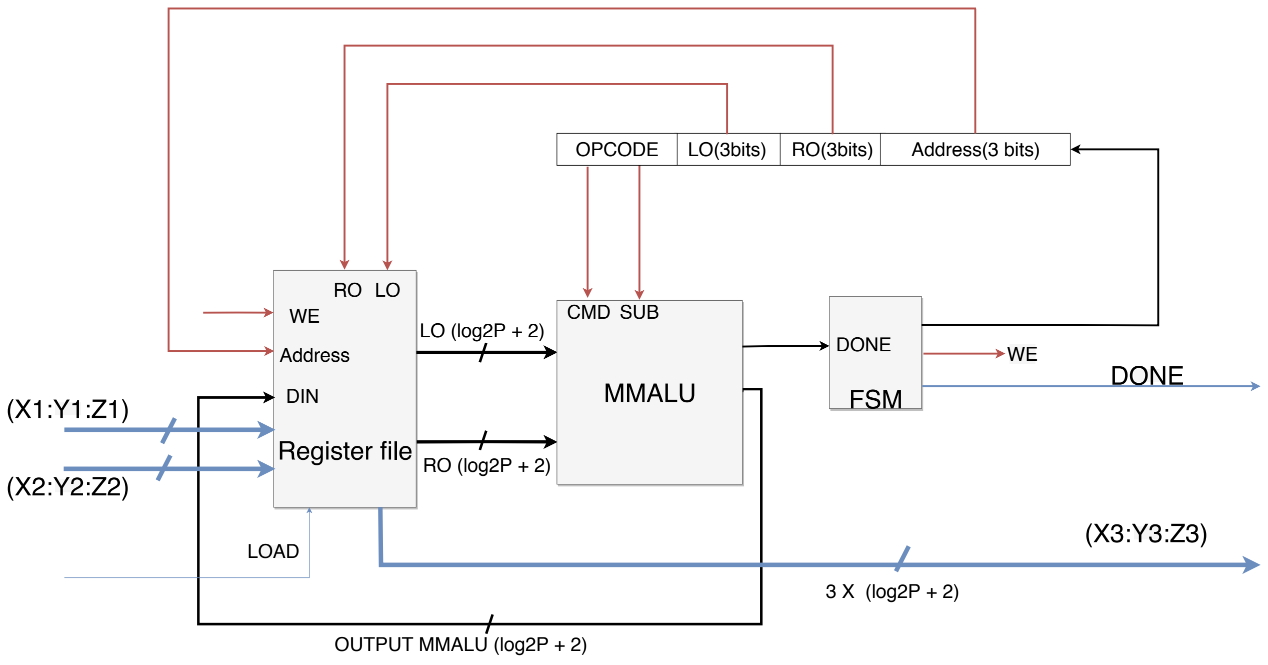


Figure 3.6 *Architecture of the point addition module*

reason is that the output of a point addition is directly written in the input registers. In other words, the output of a point addition will automatically be the input for the next point addition without the need for additional loading. In the double-and-add method, one input remains unmodified. Consequently, only one input should be reloaded during the next iteration. However, the benefits of a balanced ladder method outweigh the lost speed gain and resource reduction, in spite of the missed opportunity.

4. RESULTS

All blocks in the design were implemented in VHDL and simulated with ModelSim PE[1]. The ASIC area, expressed in gate equivalents (GE), and the maximum clock frequency are calculated by Synopsis Design Compiler 2016 using the open source NANGATE45 library. For testing the correctness of operation, test vectors are constructed in Magma[2] [31]. The testbench also implements the *secp160k1* and *secp256k1* curves used in the Bitcoin protocol.

For the implementation without randomization in the Montgomery ladder, Table 4.1 shows the utilized silicon area in kilo gate equivalents and the maximum clock frequency of the design with respect to the size of the prime. For an easy comparison in future designs, the silicon area is given with and without the inclusion of the register file. Also, for the *secp160k1* and the *secp256k1* curves, which are described in [32], the duration of a single scalar multiplication is simulated at maximum frequency. The results show that the size of the prime and the area of the implementation are linearly correlated. Each increase of 32 bits for the prime, results in an increase in implementation size of approximately 6.5 kGE. In contrast, the maximum clock frequency decreases with an increasing prime size. This decline is slower than linear decrease. For completeness, Table 4.2 gives the results for an implementation with the randomization of the order of the point operations. Note that the speed is unaffected. However, the area increases due to the extra temporary registers in the Montgomery ladder.

Finally, Figure 4.1 shows a comparison between the full-word MMALU and the scalable MMALU. The scalable MMALU with a data path size of 1 bit is significantly smaller than the full-word approach and the gain in implementation surface increases with increasing key length. However, also the duration of a single point multiplication increases. The scalable approach reaches a speed of 188.60 ms and 760.83 ms per multiplication for the *secp160k1* and the *secp256k1* respectively. This is significantly slower than the previously obtained 5.52 ms and 23.06 ms per multiplication for the full-word MMALU.

[1] Appendix C explains how to reconstruct the test results.

[2] Appendix D contains the Magma script used for validation.

# bits	Area (kGE)	Area w/ reg. file (kGE)	max. Freq. (MHz)	Scalar mult. (ms)
64	17.77	10.03	333.33	-
96	26.30	14.77	250.00	-
128	34.12	18.64	166.67	-
160	42.48	23.22	166.67	5.52 (secp160k1)
192	51.02	27.96	142.86	-
224	59.12	32.45	111.11	-
256	66.51	36.06	100.00	23.06 (secp256k1)

Table 4.1 *Results generated using Design Compiler 2016 with the NANGATE45 library without randomization of operations in the Montgomery ladder*

# bits	Area (kGE)	Area w/ reg. file (kGE)	max. Freq. (MHz)	Scalar mult. (ms)
64	21.58	13.84	333.33	-
96	31.91	20.38	250.00	-
128	41.51	26.03	166.67	-
160	51.08	31.81	166.67	5.52 (secp160k1)
192	61.35	38.29	142.86	-
224	71.59	44.92	111.11	-
256	81.89	51.45	100.00	23.06 (secp256k1)

Table 4.2 *Results generated using Design Compiler 2016 with the NANGATE45 library with randomization of operations in the Montgomery ladder*

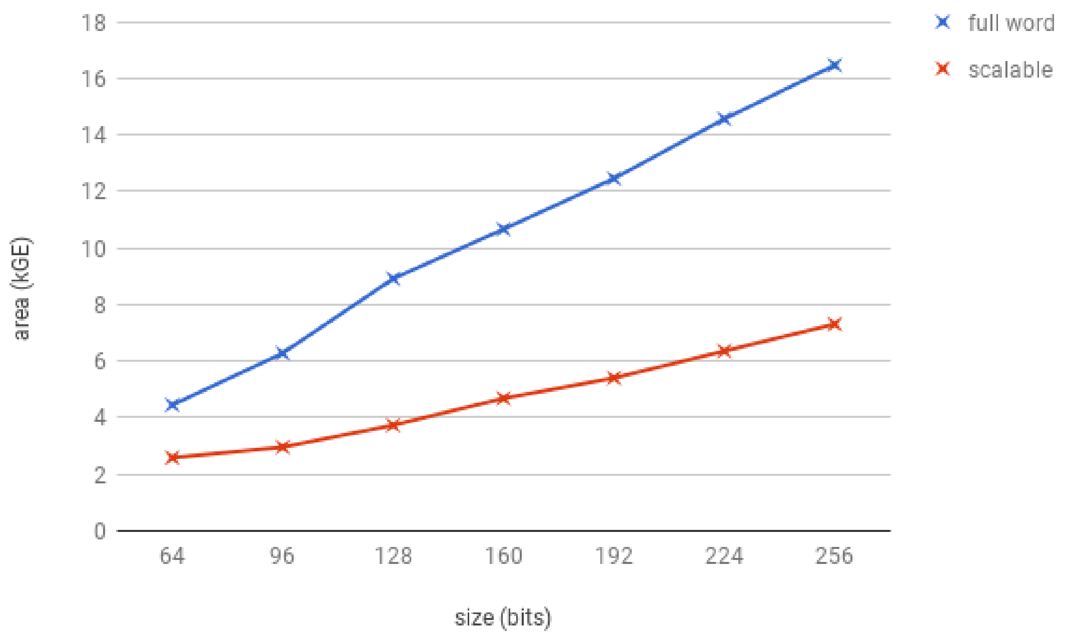


Figure 4.1 *Comparison in silicon area between the full-word MMALU and the scalable MMALU with 1-bit word size*

5. CONCLUSION

This work presents the first ASIC coprocessor design implementing ECC with complete formulas in $GF(p)$. The design consists of a fully balanced implementation, targeting protection against SPA attacks. As a first step towards DPA protection, the randomization of the execution order of the point operations was incorporated in the Montgomery ladder. Additionally, the design was optimized for minimal silicon area by (1) using complete formulas for short Weierstrass curves with $a = 0$, i.e. j-invariant 0 curves; (2) performing data path optimizations in the Montgomery Modular ALU with integrated adder functionality; (3) minimizing the size of the register file by exploring the design parameters of the MMALU and intelligent scheduling of the modular operations. Additionally, the design is compatible with both a full-word as scalable MMALU with 1-bit data path. Scalable approaches use less silicon area at the expense of speed. The silicon area, the maximum operating frequency and the scalar multiplication execution time are evaluated using Synopsys Design Compiler 2016.

Future work includes a side-channel analysis of the proposed architecture. It is expected that the design will be resistant against SPA attacks and some DPA attacks. More countermeasures need to implemented, however, to provide a fully protected implementation.

BIBLIOGRAPHY

[1] W. Diffie and M. Hellman, "New directions in cryptography," *Information Theory, IEEE Transactions on*, vol. 22, no. 6, pp. 644–654, November 1976.

[2] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology – Proceedings of CRYPTO*, ser. Lecture Notes in Computer Science, M. Wiener, Ed., no. 1666. Springer-Verlag, 1999, pp. 388–397.

[3] V. Miller, "Uses of elliptic curves in cryptography," in *Advances in Cryptology – Proceedings of CRYPTO*, ser. Lecture Notes in Computer Science, H. C. Williams, Ed., no. 218. Springer-Verlag, 1985, pp. 417–426.

[4] N. Koblitz, "Elliptic curve cryptosystem," *Mathematics of Computation*, vol. 48, pp. 203–209, 1987.

[5] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[6] S. B. Örs, E. Oswald, and B. Preneel, "Power-analysis attacks on an FPGA – first experimental results," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. Lecture Notes in Computer Science, C. D. Walter, Ç. K. Koç, and C. Paar, Eds., no. 2779. Springer, 2003, pp. 35–50.

[7] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, ser. Lecture Notes in Computer Science, K. Kurosawa, Ed., no. 4833. Springer, 2007, pp. 29–50.

[8] J. Renes, C. Costello, and L. Batina, "Complete addition formulas for prime order elliptic curves," in *35th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, ser. Lecture Notes in Computer Science, M. Fischlin and J.-S. Coron, Eds., no. 9665. Springer, 2016, pp. 403–428.

[9] L. Batina, N. Mentens, B. Preneel, and I. Verbauwhede, "Balanced point operations for side-channel protection of elliptic curve cryptography," *IEE Proceedings-Information Security*, vol. 152, no. 1, pp. 57–65, 2005.

[10] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted Edwards curves," in *International Conference on Cryptology in Africa (AFRICACRYPT)*, ser. Lecture Notes in Computer Science, S. Vaudenay, Ed., no. 5023. Springer, 2008, pp. 389–405.

[11] D. J. Bernstein, C. Chuengsatiansup, D. Kohel, and T. Lange, "Twisted hessian curves," in *International Conference on Cryptology and Information Security in Latin America (LATINCRYPT)*, ser. Lecture Notes in Computer Science, K. Lauter and F. Rodríguez-Henríquez, Eds., no. 9230. Springer, 2015, pp. 269–294.

[12] P. M. C. Massolino, J. Renes, and L. Batina, "Implementing complete formulas on Weierstrass curves in hardware," in *International Conference on Security, Privacy, and Applied Cryptography Engineering (SPACE)*, ser. Lecture Notes in Computer Science, C. Carlet, M. A. Hasan, and V. Saraswat, Eds., no. 10076. Springer, 2016, pp. 89–108.

[13] Ł. Chmielewski, P. M. C. Massolino, J. Vliegen, L. Batina, and N. Mentens, "Completing the complete ECC formulae with countermeasures," *Journal of Low Power Electronics and Applications*, vol. 7, no. 1, 2017.

[14] (2017) Faculty of science, digital security. [Online]. Available: http://www.ru.nl/ds/

[15] (2016) Research group COSIC, KU LEUVEN. [Online]. Available: https://www.esat.kuleuven.be/cosic/

[16] W. Mao, *Modern cryptography : theory and practice*, 2nd ed., ser. Hewlett-Packard professional books. Upper Saddle River: Prentice Hall, 2004.

[17] C. Paar and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[18] P. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[19] C. Walter, "Montgomery exponentiation needs no final subtractions," *Electronics Letters*, vol. 35, no. 21, pp. 1831–1832, 1999.

[20] "Bitcoin," https://github.com/bitcoin/bitcoin/tree/master/src/secp256k1, 2017.

[21] P. L. Montgomery, "Speeding the pollard and elliptic curve methods of factorization," *Mathematics of computation*, vol. 48, no. 177, pp. 243–264, 1987.

[22] L. Batina, J. Hogenboom, N. Mentens, J. Moelans, and J. Vliegen, "Side-channel evaluation of FPGA implementations of binary Edwards curves," in *17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS).* IEEE Publishing, 2010, pp. 1248–1251.

[23] NIST, "FIPS PUB 186-3 : Digital signature standard (DSS)," 2009. [Online]. Available: https://csrc.nist.gov/csrc/media/publications/fips/186/3/archive/2009-06-25/documents/fips_186-3.pdf

[24] L. Batina, “Arithmetic and architectures for secure hardware implementations of public-key cryptography,” PhD thesis, 2005.

[25] S. B. Örs, L. Batina, B. Preneel, and J. Vandewalle, “Hardware implementation of a Montgomery modular multiplier in a systolic array,” in *International Parallel and Distributed Processing Symposium.* IEEE, 2003.

[26] N. Mentens, K. Sakiyama, B. Preneel, and I. Verbauwhede, “Efficient pipelining for modular multiplication architectures in prime fields,” in *17th ACM Great Lakes symposium on VLSI.* ACM, 2007, pp. 534–539.

[27] F. Bernard, “Scalable hardware implementing high-radix Montgomery multiplication algorithm,” *Journal of Systems Architecture*, vol. 53, no. 2, pp. 117–126, 2007.

[28] K. Sakiyama, L. Batina, N. Mentens, B. Preneel, and I. Verbauwhede, “Small-footprint ALU for public-key processors for pervasive security,” in *Workshop on RFID Security*, vol. 12, 2006.

[29] A. Tenca and C. Koc, “A scalable architecture for montgomery multiplication,” *Cryptographic Hardware And Embedded Systems*, vol. 1717, pp. 94–108, 1999.

[30] H. Marzouqi, M. Al-Qutayri, and K. Salah, “Review of elliptic curve cryptography processor designs,” *Microprocessors and Microsystems*, vol. 39, no. 2, pp. 97–112, 2015.

[31] “Magma computational algebra system,” 2018. [Online]. Available: http://magma.maths.usyd.edu.au/magma/

[32] Certicom Corp., “SEC 2: Recommended elliptic curve domain parameters,” 2000. [Online]. Available: http://www.secg.org/SEC2-Ver-1.0.pdf

APPENDIX A. MATHEMATICAL BACKGROUND

A set is a collection of elements and the field of Algebra studies the underlying relations between elements in a set. In the process of doing so, it defines structures in sets and studies the underlying properties of these structures. When a set can be abstracted as a certain structure, all properties of this structure are applicable to the set. The remainder of this appendix elaborates on groups and fields.

Group

Definition. $(G, \circ)$ is called a group, when G is a non-empty set and $\circ : G \times G \mapsto G$ is an operation defined on the set with the following properties:

1. $\forall a, b \in G : a \circ b \in G$ (Closure Axiom)
2. $\forall a, b, c \in G : (a \circ b) \circ c = a \circ (b \circ c)$ (Associativity Axiom)
3. $\exists! e \in G : \forall a \in G : e \circ a = a = a \circ e$ (Identity Axiom)
4. $\forall a \in G : \exists a^{-1} \in G : a \circ a^{-1} = e = a^{-1} \circ a$ (Inverse Axiom)

An Abelian or commutative group is a group with a fifth property:

5. $\forall a, b \in G : a \circ b = b \circ a$ (Commutative Axiom)

In the case of Abelian groups, the operation is often denoted with $+$. Also, the inverse element of an element a becomes $-a$ and the unity element is denoted with 0.

Definition. A subgroup of a group $(G, \cdot)$ is a non-empty subset H of G, such that $(H, \cdot)$ is also a group.

Examples.

$(\mathbb{R}, +)$ The set of real numbers under addition.

$(\mathbb{Z}, +)$ The set of integers under addition.

$(\mathbb{Z}_n, +)$ The finite set of integers $\{0 \ldots n-1\}$ under addition.

$(\mathbb{Z}_p^*, \cdot)$ The finite set of integers $\{1 \ldots p-1\}$ under multiplication, with p prime.

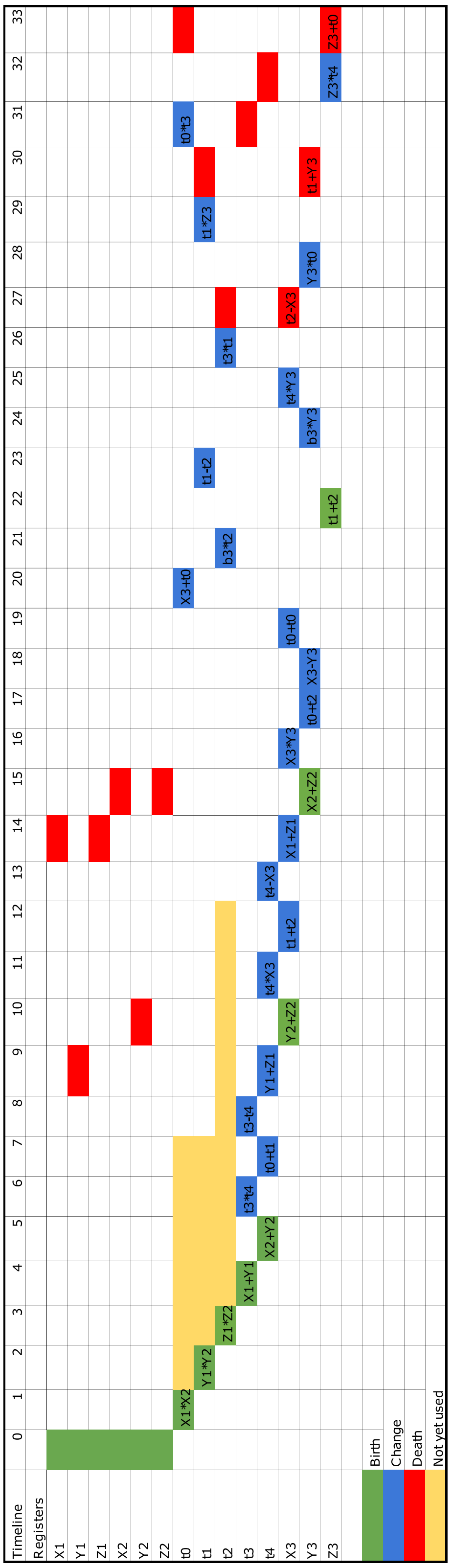
Field

A field is a Abelian group F with a second operation $\cdot : F \times F \mapsto F$ that meets the following statements:

1. $(F \backslash \{0\}, \cdot)$ is an Abelian group with the identity element denoted by 1
2. $\forall a, b, c \in F : a \cdot (b + c) = a \cdot b + a \cdot c$ (Distributive Axiom)

Remark. A field with a finite number of elements is called a finite field or Galois field. The Galois field of order p, i.e. with p number of elements, is denoted by $\mathbb{F}_p$ or $GF(p)$.

APPENDIX B. REGISTER LIFETIME OF ALGORITHM 7



Timeline	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Registers																																		
X1	🟩														🟥																			
Y1	🟩									🟥																								
Z1	🟩														🟥																			
X2	🟩															🟥																		
Y2	🟩										🟥																							
Z2	🟩															🟥																		
t0		X1*X2	🟨	🟨	🟨	🟨	🟨	🟨													X3+t0											t0*t3		🟥
t1			Y1*Y2	🟨	🟨	🟨	🟨	🟨																t1-t2						t1*Z3	🟥			
t2				Z1*Z2	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨									b3*t2					t3*t1	🟥						
t3					X1+Y1		t3*t4		t3-t4																							🟥		
t4						X2+Y2		t0+t1		Y1+Z1		t4*X3		t4-X3																			🟥	
X3											Y2+Z2		t1+t2		X1+Z1		X3*Y3			t0+t0						t4*Y3		t2-X3						
Y3																X2+Z2		t0+t2	X3-Y3						b3*Y3				Y3*t0		t1+Y3			
Z3																							t1+t2										Z3*t4	Z3+t0
🟩	Birth																																	
🟦	Change																																	
🟥	Death																																	
🟨	Not yet used																																	

Register lifetime of Algorithm 7 in [13]

Timeline	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Registers																																		
X1	🟩																			t0+t0						t4*Y1		Y2-X1						
Y1	🟩											t4*Y2						t0+t2	Z1-Y1						b3*Y1				Y1*t0		Y2+Y1			
Z1	🟩															X2+Z2	Y2*Z1						t0+Y2										Z1*t4	Z1+t1
X2	🟩															🟥																		
Y2	🟩										Y2+Z2		t1+t2		X1+Z1							b3*t2					t3*t1	🟥		t1*Z1				
Z2	🟩																																	
t0		X1*X2	🟨	🟨	🟨	🟨	🟨	🟨													X1+t0											t0*t3		🟥
t1			Y1*Y2	🟨	🟨	🟨	🟨	🟨																t0-Y2							🟥			
t2				Z1*Z2	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨									🟥												
t3					X1+Y1		t3*t4		t3-t4																							🟥		
t4						X2+Y2		t0+t1		Y1+Z1				Y1-Y2																			🟥	
🟩	Birth																																	
🟦	Change																																	
🟥	Death																																	
🟨	Not yet used																																	

Register lifetime of Algorithm 7 in [13] after rescheduling

APPENDIX C. OPEN SOURCE VHDL

This appendix explains how to use the design VHDL code to reconstruct the results. All VHDL code can be found in the Github repository: `https://github.com/NielsPirotte/MasterThesis_Niels_Pirotte`.

The repository has the following directory structure:

```
MasterThesis_Niels_Pirotte/
  ├─ magma_sim/
  ├─ MM_design1/
  ├─ MM_design2/
  ├─ PA_design1/
  ├─ PA_design2/
  ├─ PM_design1/
  └─ PM_design2/
```

magma_sim Validation scripts for Magma to check the correct operation of the design.

MM_design1 First implementation of Montgomery Multiplication with two 4P inputs and a 2P output. Also two 2P inputs can be added and subtracted resulting in a 4P output.

MM_design2 Implementation according to the paper of Koç: "A scalable Architecture for Montgomery Multiplier".
Word size equals 1.

PA_design1 Implementation of Point Addition with complete formulae according to: "Complete addition formulas for prime order elliptic curves" p13 algorithm 8.

Remark: The implementation is working properly, but keep in mind that x2 and y2 need to be converted to the Montgomery domain. They are the only inputs that need to be converted.

PA_design2 Implementation of Point Addition with complete formulas according to: "Complete addition formulas for prime order elliptic curves" p12 algorithm 7. Uses the same number of registers as design 1.

PM_design1 Implementation of point multiplication using the principle of the Montgomery Ladder.

PM_design2 Same as PM_design1, but with randomization of operations.

How To

1. Customize constants.vhd
2. Load all necessary[1] VHDL files in ModelSim PE (or another HDL simulator)
3. Customize the inputs in the test bench file (tb_*Top_Level_Entity*)

constants.vhd In the constants.vhd file, the number of bits of the prime ($log2primeM$), the prime ($primeM$) and $B3 = 3 \cdot b$ need to be specified, as can be seen in Program 1 below. With these parameters the elliptic curve becomes uniquely defined. Important to note is that the $B3$ constant needs to be transformed into the Montgomery domain ($B3 \cdot R$).

[1] When simulating point addition, a valid MMALU architecture needs to be loaded. Likewise, point multiplication relies on a valid implementation of point addition.

```
--------------------------------------------------------------------
-- Author: Niels Pirotte
--
-- Project Name: Masterthesis Niels Pirotte
-- Package Name: constants
-- Description: Definition and settings of parameters ECC coprocessor
-- Information:
-- This implementation provides a ECC (Elliptic Curve Crypto)
-- hardware implementation for an ASIC
-- We use the complete addition formulas for prime order
-- elliptic curves as described by: Complete Addition Formulas
-- for Prime Order Elliptic Curves
-- by Joost Renes, Craig Costello, and Lejla Batina
-- The goal is an implementation optimizing the area of the ASIC
--------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

-- To determine number of bits for an integer
use IEEE.math_real."ceil";
use IEEE.math_real."log2";

package constants is
-- Parameters
--
-- Number of bits of the prime
--constant log2primeM: integer := 3;
constant log2primeM: integer := 256;

-- Therefore the inputs of the MMALU are < 2M
-- Number of bits of scanning counter for the 2M inputs
constant e: integer := integer(ceil(log2(Real(log2primeM+4))));

--constant primeM: std_logic_vector(log2primeM-1 downto 0) :=
"111"; -- 7
--For the secp256k1 curve
constant primeM: std_logic_vector(log2primeM-1 downto 0) :=
x"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F";

--EC (Elliptic Curve)
--defining the EC
--needs to be in Montgomery representation => 3*b*R

--i.e. R = 128 = 2 mod 7 and b = 1 => 3*2*2 = 12 mod 7 = 5 mod 7
-->It is crucial b3 is given in Montgomery coordinates!!!
--constant B3: std_logic_vector(log2primeM-1 downto 0) := "101";
constant B3: std_logic_vector(log2primeM-1 downto 0) :=
x"0000000000000000000000000000000000000000000000000000015000050250";
end constants;
```

Program 1 *constants.vhd*

APPENDIX D. MAGMA VALIDATION SCRIPT

```
//This is a magma script
//secp256k1 (http://www.secg.org/SEC2-Ver-1.0.pdf)
n:= 780799;

//Define prime
p := 2^256 - 2^32 - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1;
p;

field := GF(p);

b := 7;
b3 := 3*b;

//Curve polynomial
curve_poly<x> := Polynomial(field, [b, 0, 0, 1]);
curve_poly;

//Define Elliptic Curve
E<x, y, z> := EllipticCurve(curve_poly);
E;

#(field);
#(E);

//Get a points on the elliptic curve
//Points(E);
X := E!
[55066263022277343669578718895168534326250603453777594
175500187360389116729240,
326705100207588169780830851305070431844712733806592432
7593890433575733748242 4,
1];
printf "X = "; X;

cartesian := CartesianPower(E, 2);
cartesian;

// Define Addition law
add_law := map<cartesian -> E | P :->
E![
(P[1][1]*P[2][2]+P[2][1]*P[1][2])*(P[1][2]*P[2][2]-
3*b*P[1][3]*P[2][3])-3*b*(P[1][2]*P[2][3]+P[2][2]*P[1][3])
*(P[1][1]*P[2][3]+P[2][1]*P[1][3]),
(P[1][2]*P[2][2]+3*b*P[1][3]*P[2][3])*(P[1][2]*P[2][2]-
3*b*P[1][3]*P[2][3])+9*b*P[1][1]*P[2][1]*
(P[1][1]*P[2][3]+P[2][1]*P[1][3]),
(P[1][2]*P[2][3]+P[2][2]*P[1][3])*(P[1][2]*P[2][2]
3*b*P[1][3]*P[2][3])+3*P[1][1]*P[2][1]*
(P[1][1]*P[2][2]+P[2][1]*P[1][2])
]>;
```

```
//Point multiplication
pm := function(m, P)
   Out := P;
   S := [x : x in [1..m-1]];
   if not (P in E) then
        return false;
   end if;
   for i in S do
     In := cartesian!<P, Out>;
     Out := In @ add_law;
   end for;

   return Out;
end function;

printf "n = "; n;

res := pm(n, X);
printf "nX = "; res;

testres := E!
[78615513602287977103249641354618743671933021561241
4034950761232341923158212 6,
15981915174009075968799798789644747178575518956998 6
887558313962362031917425 61,
95215699043371475437109552690683404534696503558034 6
88976019490288295699161980];

assert testres eq res;
printf "testres = "; testres;
```

Program 2 *Magma script to validate correct operation of the design applied to the secp256k1 curve.*