Elliptic-Curve Cryptography Implementation for Passive RFID Tags

Thomas Lenz
t.kern@student.tugraz.at

Institute for Applied Information
Processing and Communications (IAIK)
Graz University of Technology
Inffeldgasse 16a
8010 Graz, Austria

A Master Thesis

Supervisor: Dipl-Ing. Dr. techn. Michael Hutter
Supervisor: Dipl-Ing. Dr. techn. Martin Feldhofer

Assessor: Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Karl Christian Posch

September, 2011

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am 13.09.2011 Thomas Lenz

Acknowledgements

At first, I would like to thank my supervisors Michael Hutter and Martin Feldhofer for advising this master thesis. Many thanks also go to the Institute for Applied Information Processing and Communications (IAIK).

A special thank goes to my wife Andrea for supporting me in various ways during the years of study. Furthermore, I thank my family for supporting me during the study and especially my brother Wolfgang for proof-reading this work.

Abstract

Radio-frequency identification (RFID) technology becomes more and more interesting for safety-related applications. Such applications, like e-passports or access control require a secure authentication process to check if the RFID tag and the corresponding product is valid or counterfeit. The elliptic-curve digital signature algorithm (ECDSA) uses asymmetric cryptography to generate a digital signature which can be used for authentication. An advantage of elliptic-curve cryptography (ECC) is the short key length in comparison to other public-key systems like RSA. Therefore, ECC is well suitable for RFID applications.

In this thesis, we design an elliptic-curve processor with full ECDSA functionality. This ECDSA implementation is based on the recommended $\mathbb{F}_{P160}$ Standards for Efficient Cryptography Group (SECG) elliptic curve $SECP160_{r1}$, which has only 160 bits in contrast to the smallest recommended National Institute of Standards and Technology (NIST) elliptic curve with 192 bits. This 160-bit elliptic curve is well suitable for a low-area implementation. In order to fulfill the fierce constraints with respect to power consumption and chip area, we use a 16-bit datapath. Therefore, an implementation of all required algorithms on word level is necessary. Additionally, we implement a new field-multiplication algorithm with implicit fast reduction modulo the special prime $\mathbb{F}_{P160}$. Furthermore, this is basically the first ECDSA implementation in hardware, which is optimized for low power and low area and which uses the prime $SECP160_{r1}$ from SECG.

This hardware module consists of four submodules. A memory module, which comprised a latch based 90×16-bit dual-port RAM, a 83×16-bit ROM, and a 10×16-bit EEPROM. The main part of the arithmetic-logic unit is a 16×16-bit multiply-accumulate unit, which can calculate a 16×16-bit multiplication and a 36-bit addition within one clock cycle. We use a hybrid control-unit architecture, which means that our controller consists of a finite-state machine and also micro-coded parts. An AMBA interface is used for interconnection with other hardware modules.

The ECDSA module has been synthesized by using digital standard-cells from c35b4 CMOS libraries published by Austriamicrosystem AG. Our module requires a chip area of 18 315 GE and can generate an ECDSA digital signature within 510 831 clock cycles. The power consumption has been ascertained by using a near-spice simulator at which one signature-generation operation requires 1 108.0 μW@1 MHz with a supply voltage of 3.5 V.

Keywords: ECDSA, electronic signature, RFID, elliptic-curve cryptography, prime-field arithmetic, low-power processor, low-area processor, SECP160.

Kurzfassung

Radiofrequenz-Identifikationstechnologie (RFID) wird auch für sicherheitsrelevante Anwendungen immer interessanter. Solche Anwendungen, wie elektronische Reisepässe oder Zugangskontrollsysteme, benötigen ein sicheres Authentifizierungsverfahren um überprüfen zu können ob ein RFID-Transponder und das dazugehörige Produkt gültig oder gefälscht sind. Der Algorithmus zur Erstellung digitaler Unterschriften mithilfe elliptischer Kurven (ECDSA) verwendet asymmetrische Kryptografie um eine digitale Unterschrift, welche zur Authentifizierung verwendet werden kann, zu erstellen. Ein Vorteil der elliptischen Kurven Kryptografie (ECC) ist die kürzere Schlüssellänge im Vergleich zu anderen Verschlüsselungssystemen mit öffentlichem Schlüssel, wie RSA. Daher ist ECC für den Einsatz in RFID Anwendungen gut geeignet.

In dieser Masterarbeit designen wir einen Prozessor für elliptische Kurven mit vollständiger ECDSA funktionalität. Diese ECDSA-Implementierung beruht auf der von der Gruppe für Standards für effiziente Kryptografie (SECG) empfohlenen elliptischen Kurve $SECP160_{r1}$, welche nur 160 bits im Gegensatz zur kürzesten empfohlenen elliptischen Kurve des Nationalen Instituts für Standards und Technologie (NIST) mit 192 bits hat. Diese elliptische Kurve mit 160 bits ist gut für eine flächensparende Implementierung verwendbar. Um die anspruchsvollen Auflagen, bezüglich Leistungsbedarf und Chip Fläche erfüllen zu können, verwendeten wir einen 16-bit Datenpfad. Somit ist eine Implementierung aller benötigen Algorithmen auf Wortebene nötig. Zusätzlich implementierten wir einen neuen Feld-Multiplikationsalgorithmus mit eingeschlossener schneller Reduktion modulo der speziellen Primzahl $\mathbb{F}_{P160}$. Darüber hinaus ist das die erste ECDSA-Implementierung in Hardware, welche auf einen geringen Leistungsbedarf und einen geringen Flächenbedarf optimiert ist und die die Primzahl $SECP160_{r1}$ von SECG verwendet.

Diese Hardwarebaugruppe besteht aus vier Untermodulen. Einer Speicherbaugruppe, welche ein auf Latch basierendes 90×16-bit RAM mit zwei Anschlüssen, einen 83×16-bit ROM und einen 10×16-bit EEPROM beinhaltet. Das gröte Element im Rechenwerk ist eine 16×16-bit Multipliziereinheit, welche eine 16×16-bit Multiplikation und eine 36-bit Addition in einem Taktzyklus rechnen kann. Wir verwenden eine Kontrolleinheit mit einer gemischten Architektur, welche sowohl aus einem endlichen Zustandsautomat als auch aus einem speicherprogrammierten Teile besteht. Als Verbindung zu anderen Baugruppen wird eine AMBA-Schnittstelle eingesetzt.

Die ECDSA-Baugruppe wurde unter Verwendung von digitalen Standardzellen aus der CMOS Bibliothek c35b4, herausgegeben durch die Austriamicrosystem AG, aufgebaut. Diese Baugruppe benötigt eine Modulfläche von 18 315 GEs und kann eine ECDSA digitale Unterschrift in 510 831 Taktzyklen erzeugen. Der Leistungsbedarf wurde mit Hilfe eines spice-nahen Simulators ermittelt, wobei ein Arbeitsvorgang zur Unterschriftenerzeugung 1 108.0 μW@1 MHz bei einer Versorgungsspannung von 3.5 V benötigt.

Stichwörter: ECDSA, elektronische Signatur, RFID, elliptische Kurven Kryptografie, Primkörper Arithmetik, leistungssparend, flächensparend, Prozessor, SECP160.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

This thesis presents the design and implementation of an elliptic-curve processor which can generate an elliptic-curve digital signature. In the first section of this chapter, we give a motivation for this work. In the second section, we present our contribution to this topic and give an outline of the following chapters.

1.1 Motivation

Radio-Frequency Identification (RFID) technologies become interesting for more and more areas in our daily life. They are not only used for anti-theft protection systems and inbound and outbound logistic systems but also for door-locking applications, micro-payment systems, or anti-counterfeiting purposes. An RFID system mostly consist of an RFID transponder (a tiny microchip attached to an antenna) and a reader. Usually a so called passive RFID transponder is used for door-locking applications and micro-payment systems, which means that only an electromagnetic field is used to supply the RFID transponder with power and to communicate with it. This electromagnetic field is generated by the reader and only RFID transponders which are in the reading range of the reader can be used. The reading range varies from a few millimeters up to three meters for passive transponders but the power which is available on the RFID transponder decreases sharply with distance to the reader. Consequently, an RFID transponder should need less power as possible. Most RFID applications, like for anti-counterfeiting purposes, require a high number of transponders. Therefore, transponders should be as cheap as possible to be applicable in practice. The microchip on a transponder should meet low-area requirements, because the price of an RFID transponder corresponds directly with the required area of the microchip.

If we use RFID technology for security-related applications, such as door-locking applications, micro-payment systems, or anti-counterfeiting purposes, then a cryptographic secure authentication protocol is required. That can be done by using symmetric or asymmetric cryptography. Symmetric cryptography means that all members of the authentication protocol (prover and verifier) use the same cryptographic key and they can use them for signing or verifying operations. In contrast to symmetric cryptography, asymmetric cryptography uses an associated key pair which consist of a private key and a public key. The public key can be used by everyone for verification purposes, but only the prover knows his own private key and can use it for signing operations. Therefore, asymmetric cryptography is a good choice for an authentication system which is used by a lot of

members, because the key distribution and the key management is easier. But asymmetric cryptography is much more intricate than symmetric cryptography, because the algorithms which are used for asymmetric cryptography are mostly more complex. There exist many cryptographic principles, which use asymmetric cryptography, but currently only two are mostly used. These two are RSA and elliptic-curve cryptography (ECC). An advantage of ECC is the shorter key length in comparison to RSA. Elliptic-curve cryptography with a key length of 160 bits is as secure as RSA with a key length of 1024 bits, cf. [46]. A second advantage of ECC is that a more efficient hardware implementation in comparison to established RSA implementations are possible. Therefore, elliptic-curve cryptography is a good choice for an authentication system in combination with RFID technology.

This thesis focuses on an authentication and confidentiality system which requires low power and low area to fulfill RFID-system requirements. So far, the field of research in case of elliptic-curve cryptography, which is optimized for low-power and low-area implementations, goes into two directions. The first one implements only basic elliptic-curve operations with arithmetic in a binary finite field, like $\mathbb{F}_{2^{163}}$, but they did not implement a digital signature-protocol. Some examples of such implementations are given by Hein et al. [25] and Lee et al. [38], which implemented an identification scheme. The second direction goes towards a full ECDSA implementation by using elliptic-curve cryptography with arithmetic in a prime finite field. Such implementations are shown by Wolkerstorfer et al. [56], Auer [4], Hutter et al. [26], and Wenger et al. [54]. They all used the prime field $\mathbb{F}_{P192}$, which is the smallest recommended prime standardized by the National Institute of Standards and Technology (NIST) [45]. This prime has a special characteristic which allows to implement a fast reduction algorithm, the so-called NIST reduction [45].

In our work, we use an elliptic curve which is defined over the prime field $\mathbb{F}_{P160}$. This elliptic curve is standardized by the Standards for Efficient Cryptography Group (SECG) [48] and has only 160 bits, instead of 192 bit of the smallest NIST elliptic curve. Therefore, this prime from SECG should be better for a low-area optimized implementation and we think the security level of 80 bits should be reasonable for passive RFID systems. At first, it is necessary to evaluate the prime field $\mathbb{F}_{P160}$, because there exist no fast reduction algorithm in the literature. But the prime $\mathbb{F}_{P160}$ belongs to the family of generalized Mersenne numbers and Jerome Solinas [53] proposed a technique to find a fast reduction algorithm for this group of primes. Afterward, we implement the full ECDSA digital signature algorithm, which uses the elliptic curve from SECG. Table 1.1 gives a brief overview of our implementation results. At last, we compare our implementation results with existing results to find advantages or disadvantages of this elliptic-curve implementation. This work is the first one which uses the elliptic curve $\mathbb{F}_{P160}$, for a low-area and low-power optimized hardware implementation.

Table 1.1: Summarization of our results $V_{DD} = 3.3$V.

Technology	**Area**		**Runtime**	**Power**	$\mathbf{f_{max}}$
	[μm^2]	[#GE]	[cycles]	[μW]	[MHz]
ECDSA 0.35 μm AMS	1 007 334	18 315	510 831	1 108.0@1 MHz	70

1.2 Outline

Chapter 2 gives a brief introduction into RFID. There is a short description about construction formats, energy supply, and reading ranges of RFID systems.

Chapter 3 presents the elliptic-curve digital signature algorithm (ECDSA) and gives some introduction about key generation. Additionaly, we present two other identification schemes and discuss some advantages and disadvantages of these three protocols.

The following Chapter 4 gives some information about finite fields. Here, we discuss different algorithms for basic arithmetical operations in finite fields and we present a fast multiplication algorithm for the special prime $\mathbb{F}_{P160}$. The central part of Chapter 4 gives information about elliptic curves and arithmetic on elliptic curves. It presents all used algorithms for elliptic-curve arithmetic and discusses some improvements which we have done. Chapter 4 is closed by a short introduction into the secure hash standard (SHA-1), because SHA-1 is defined as hash function for the ECDSA digital-signature algorithm.

In Chapter 5, we give a detailed description of the implementation process and the used architecture. First, there is an overview of the user interface and the I/O specification of the ECDSA processor. Afterwards, we discuss different memory architectures and present the datapath, the main components of the arithmetic-logic unit (ALU) and the control machine. Also some information about side-channel attacks and methods for low-power optimization are given. Chapter 5 is closed by presenting the results of our ECDSA module implementation and we compare our implementation with other related work.

Chapter 6 concludes this thesis and gives some aspects for optimizations and future work.

Chapter 2

Radio-Frequency Identification

Radio-Frequency Identification (RFID) becomes interesting for more and more areas like automatic-identification techniques, inbound and outbound logistics, product tracking or authentication techniques. The market for RFID technology becomes bigger and bigger every year and this technology finds its way into a lot of new areas. This chapter gives you a briefly introduction into the RFID technology and different RFID systems.

2.1 Introduction

RFID is a general term for all HF-frequency based techniques to transfer data and power contactlessly between a reader and a tag. If we use RFID technology for door-locking applications or micro-payment applications, a secure authentication process is required. An RFID system consist of two parts, first the RFID reader and the application and second one or more RFID tags. Figure 2.1 shows a schematic of a basic RFID system. The aim of

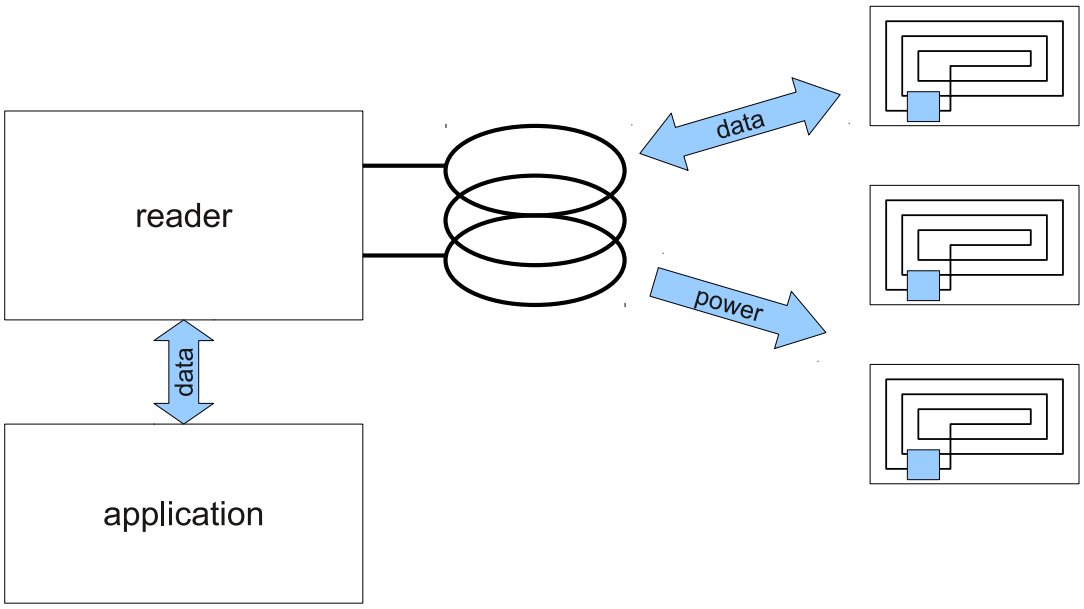

Figure 2.1: Schematic of a basic RFID system.

the reader is to supply the RFID tags with power and to communicate with them. There exist different transmission techniques for communication with one or more tags or to supply them with energy. The second part of the RFID system is the tag or transponder, which is the mobile part of the RFID system. In the following sections, we give more information about RFID tags, construction formats, energy supply, and communication ranges and frequencies. For more information on this topic we refer to [18].

2.2 RFID Tag

Figure 2.2 is from [18] and gives you an overview of an RFID transponder layout. This transponder could be placed on an object which should be identified. The construction format of the housing can be chosen depending on the application. We can separate an RFID tag into three big parts. The coupling element or antenna, the housing, and the microchip. Figure 2.3 shows the block diagram of an RFID chip. This RFID chip can be also divided into two parts, an RF front-end module and a module which include the functionality of the RFID tag, including cryptographic modules like ECDSA. The RF front-end module consists of two sub-modules, an analog front-end and a digital front-end. The analog front-end includes the High-Frequency Modulator (HF-Modulator) with control functionality, a clock-generation unit and power-supply system which attend all other parts of the transponder with energy. All operations for communication, like the header control, the data extraction, the return-link modulation and the encoder are parts of the digital front-end sub-module. Therefore, the RF front-end module includes all functionalities to receive and transmit data over a radio channel. The functionality of the RFID tag comprises the right block in Figure 2.3. It can be a simple memory, if the transponder has only storage functionality, it can be a complete microprocessor, which can generate a full ECDSA digital signature (like it is in the case of many e-passport applications).

2.2.1 Construction Formats

The design or construction format of a transponder depends mainly on the antenna. There exist many different construction formats, today. The following enumeration presents the most popular designs and gives a short description of different application areas. The

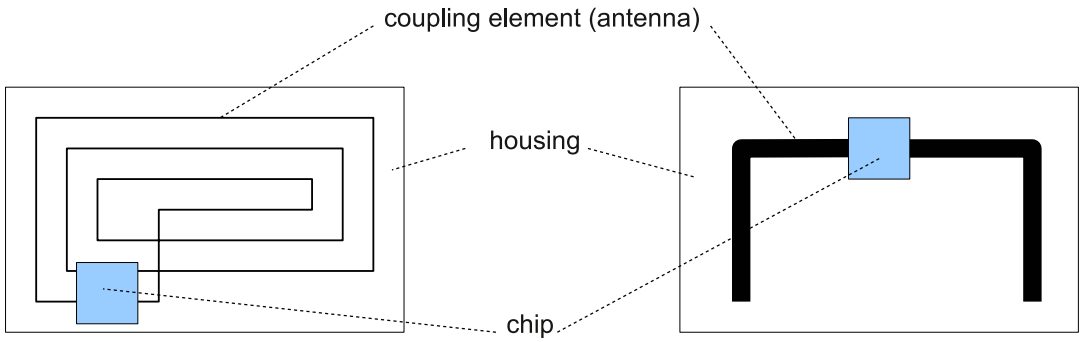

Figure 2.2: Basic layout of an RFID transponder [18].

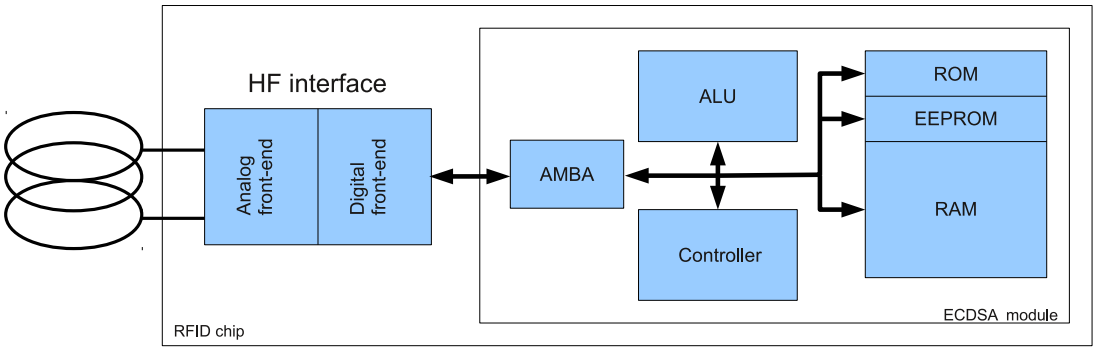

Figure 2.3: Block diagram of an RFID chip.

information about construction formats are from [16, 18, 35].

- **Glass Housing**: The main application area of glass transponders are the identification of animals. These glass tubes have a length of just 12 mm to 22 mm and get injected under the skin. Figure 2.4 shows a glass transponder. The coil and the ferrit rod are parts of the antenna. To ensure a smooth supply current, a chip capacitor is used and the functionality is implement in the chip.

- **Keys and Key Fobs**: Here, the transponder is used for door-locking applications or micro-payment applications. It is integrated into a mechanical key or a key fob. These transponders mostly have to fulfill very high security requirements, if they are part of an office-access system, for example. Figure 2.5 shows one example of a key fob transponder taken from [18]

- **Contactless Smart-Cards:** has the same format as smart cards with galvanic connections or credit cards. They are also used for door-locking applications or micro-payment applications. The advantage of this construction format is the size, because they can use a bigger antenna which is better for an inductively coupled system.

- **Smart Labels:** are paper-thin transponders which are applied to a plastic foil of just 0.1 mm thickness. The plastic foil is adhere to a layer of paper in den most cases. This construction is thin and flexible and so it could be attached to a luggage, a book, or all other types of products as a self-adhesive label. Figure 2.7 shows an RFID

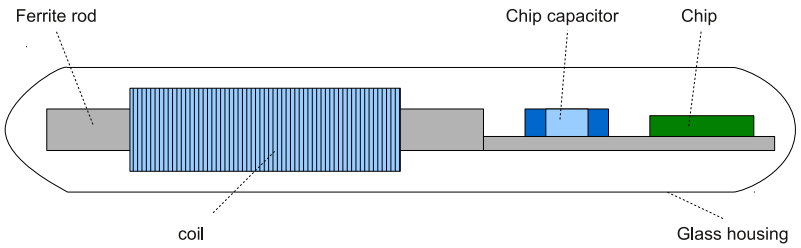

Figure 2.4: Layout of a glass transponder [18].

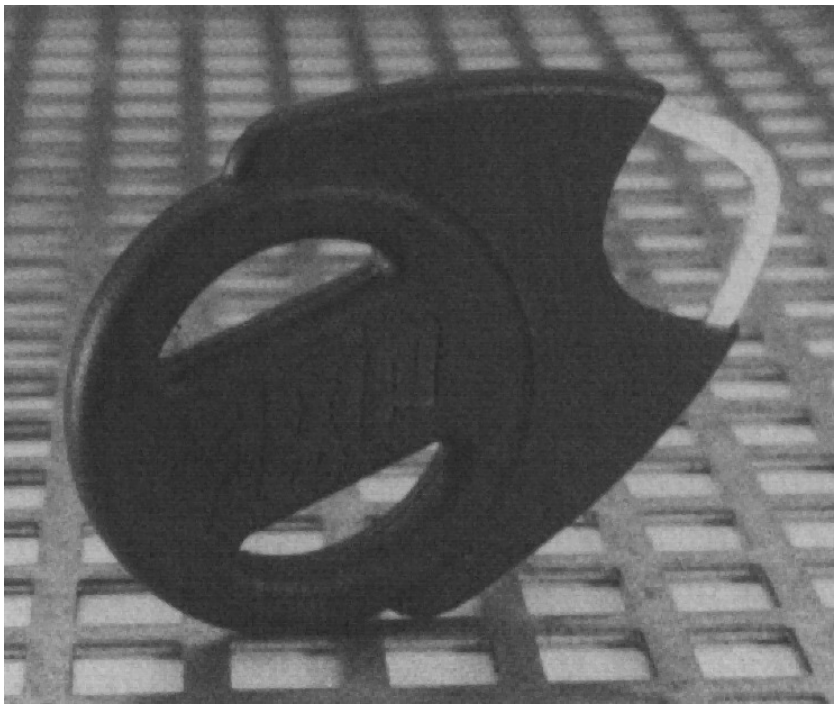

Figure 2.5: Example of a key-fob transponder [18].

chip with antenna on a plastic foil and Figure 2.6 shows a smart-label transponder for luggage tracing and identification (both taken from [18]).

2.2.2 Energy Supply

Another possibility to classify transponders is their energy supply. There exist passive transponders and active transponders. A passive transponder has no energy-supply unit on the tag. Consequently, the power which is required by the RFID chip is provided by the reader in type of a magnetic field or an electromagnetic field. If the transponder is not in the reading range of a reader, no power supply is available. Figure 2.8 shows the

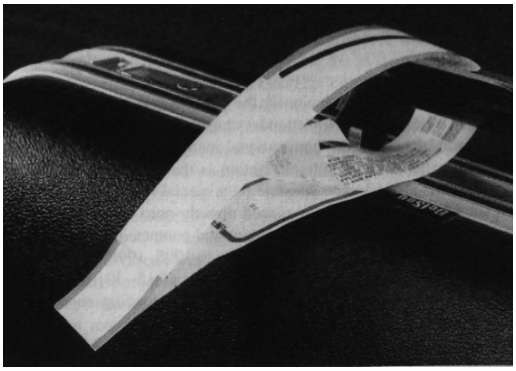

Figure 2.6: A smart-label on a luggage [18].

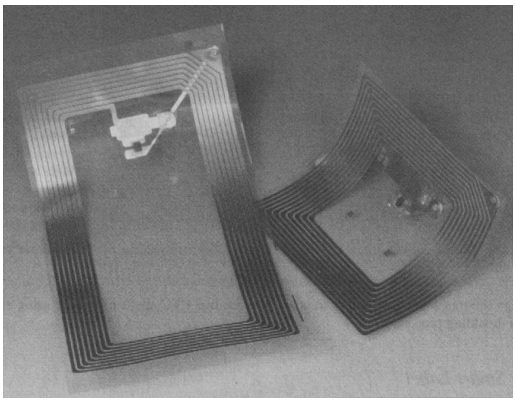

Figure 2.7: A smart-label transponder [18].

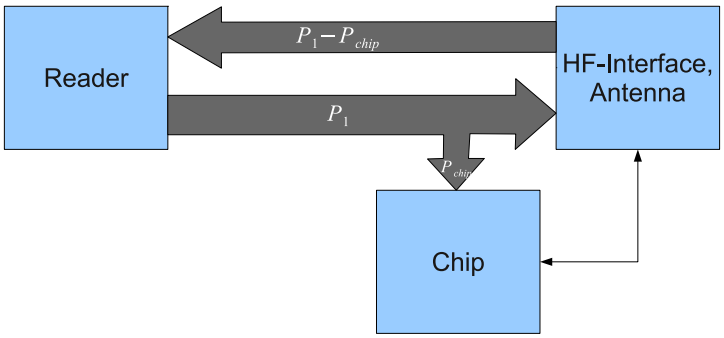

Figure 2.8: Block diagram of a passive transponder [18].

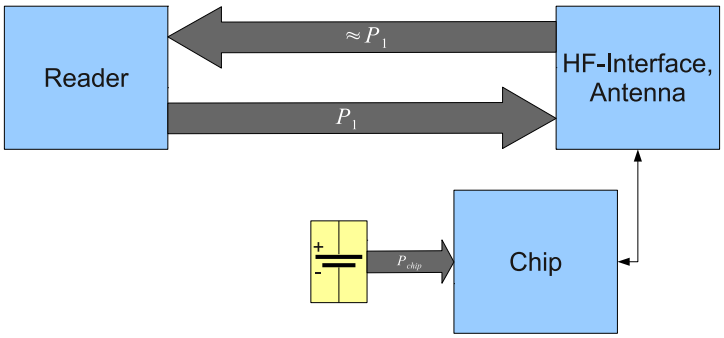

Figure 2.9: Block diagram of an active transponder [18].

energy flow of a passive transponder. The flow P_1 in Figure 2.8 means the electromagnetic energy which is provided by the reader. The transponder needs the energy P_{chip} to supply all modules, like a cryptographic chip and the RF front-end module with power. Only the energy difference $P_1 - P_{chip}$ can be used to transmit data back from the transponder to the reader. In contrast to the passive transponder, an active transponder uses a battery to supply the module with power. Figure 2.9 shows the energy flow of an active transponder. Here, the full energy P_1 is used to transmit data back from the tag to the reader. Therefore, the electromagnetic field of the reader can be much weaker, because it is only used for communication. But both types of transponders can not generate a high-frequency signal from its own. They have to modulate the reader field to transmit data from the RFID tag to the RFID reader.

2.2.3 Frequency and Reading Range

The reading range of an RFID system goes from a few centimeter up to 10 meter. Depending on the reading range, different frequencies are used. In the following, we give a short description of different reading ranges and frequencies [16, 18, 49].

- **Closed-Coupling Systems** are RFID systems with a range up to one centimeter and frequencies up to 30 MHz. They use electrical fields or magnetic fields to communicate between reader and tag. The small distance between reader and tag make a high-energy transfer possible. Closed-Coupling systems are mostly used for

high-security applications.

- **Remote-Coupling Systems** have a reading reading range up to one meter. They use inductive coupling for communication and energy transport and the mostly used frequencies are 135 kHz and 13.56 MHz. Remote-Coupling systems are the most widely used RFID systems, used for example for anti-theft protection, time recording, or access control.

- **Long-Range Systems** have a reading range up to 3 meters for passive transponders. If active transponders are used, a reading range up to 10 meters is possible. They use frequencies in the range of Ultra-High Frequency (UHF), like 868 MHz, or microwaves like 2.45 GHz or 5.8 GHz. These systems are mostly used for product-tracing systems and stock-management system.

Figure 2.10 shows the frequency spectrum of RFID applications. The following enumeration presents the four most used frequency groups.

- **Low Frequency (LF):** LF means frequencies about 100 kHz to 135 kHz. This frequencies are for large ranges and low-cost transponders. An advantage is the low-power consumption due to the lower clock cycle and the low-absorption rate in nonmetallic materials and water. Such transponders are used for animal identification, as example.

- **High Frequency (HF):** The HF frequency 13.56 MHz is mostly used for high-speed and high-end applications or medium-speed and low-end applications. An advantage is that this frequency can be used worldwide as an Industrial, Scientific and Medical (ISM) frequency. The high clock frequency allow complex applications which requires a lot of clock cycles and also data-transmission rates between 106 kbit/s and 848 kbit/s are possible.

- **Ultra-High Frequency (UHF):** These frequencies can not be used worldwide, because several countries use different frequencies, like 868 MHz in Europe, or 915 MHz in the USA. UHF frequencies can only be used for short-range devices, because buildings and other obstacles evoke a strong dampening and high reflection of this frequencies.

- **Microwave (MW):** These two frequency ranges (2.45 GHz and 5.8 GHz) are also be used by amateur radio and radio-location services. The behavior respective dampening and the reflection are the same as for UHF frequencies. Typical applications for these frequencies are movement sensors, telemetry transmitters, or wireless-networking systems.

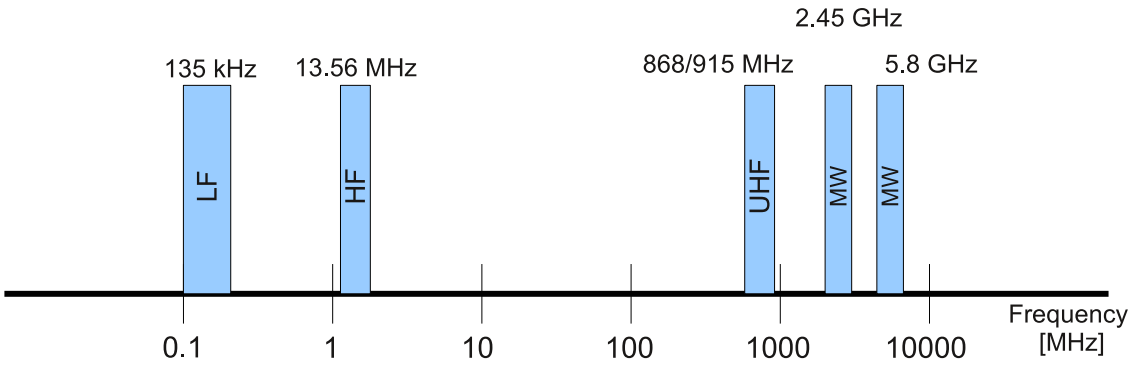

Figure 2.10: Frequency spectrum of RFID systems.

Chapter 3

Authentication Protocols

Today, RFID tags are increasingly used for product identification in place of bar codes. These tags can also be used for product tracking and anti-counterfeiting purposes. This chapter deals with RFID tags for anti-counterfeiting and presents three different authentication protocols. First, we describe the ECDSA protocol implemented in this thesis. Second, we describe two other protocols, which are also based on the Elliptic-Curve Discrete Logarithm Problem (ECDLP). Both protocols are based on identification schemes. We will discuss advantages and disadvantages of these three protocols.

3.1 Introduction

Authentication protocols are used to prove the identity of an user, or in our case an RFID tagged product. An authentication protocol which is used for anti-counterfeiting purposes must have some fundamental characteristics, cf. [37]. The following enumeration shows some important characteristics:

- **Scalability:** means, that the computational workload of the verifier should not increase linearly with the number of RFID tags. Because a modern RFID system has to support many RFID tags.
- **Anti-cloning** means, that an attacker can not impersonate a tag. It should not be possible to make a clone of an RFID tag. It is important to implement some countermeasures which prevents the extraction of secret information.
- **Replay attack:** means, that it should not be possible to use a message or some parts of a message to fulfill a valid authentication process, without the private key is known. A countermeasure to prevent replay attacks is the usage of session tokens. These are one-time tokens and they have to been generated randomly at every protocol run. The ECDSA identification protocol, which we use for our RFID tag, fulfill this requirement.
- **Privacy or anonymity:** means, that an RFID system should be secure against tracking attacks. It is a privacy problem, if some part of a message, which is transmitted from tag to the reader is fixed or predictable. An attacker could use this message to track a tag, and hence its owner too.
- **Availability:** means that an RFID system has to guarantee a minimal continuity of service. There exist some attacks, like jamming attacks or denial-of-service attacks,

which prevent the communication between reader and tag. Therefore, it is important to implement some countermeasures against these types of attacks.

- **Side-channel resistance:** means a requirement to prevent the extraction of secret information. Side-channel attacks are one frequently used type of attack to extract secret information from an RFID tag and consequently to impersonate it. Therefore, it is important to use algorithms which include some countermeasures against side-channel attacks.

In generally, attacks could be divide into active attacks and passive attacks. Passive attacks means that the attacker can only observe the communication between reader and tag, but it could not influence the communication. In an active attack, like replay attack, or denial-of-service attack, the attacker can influence the communication between reader and tag.

In the next sections, we will discuss three protocols. The ECDSA algorithm, Schnorr's identification scheme, and Okamoto's identification scheme. These three algorithms differ in the message exchange between reader and an RFID tag. They also have a different computational complexity.

3.2 Elliptic-Curve Digital Signature Algorithm (ECDSA)

The ECDSA digital signature algorithm was proposed by the American National Standards Institute (ANSI) in the year 1998, cf. [3], as a new digital-signature algorithm for the financial service industry. They defined SHA-1 as a hash function which should be used. They also provided some criteria for key generation and secure usage of the ECDSA digital signature algorithm. The ECDSA authentication protocol can be used to prove the identity of the signer (identification scheme) or authenticate a message (signature scheme).

The ECDSA digital signature algorithm uses asymmetric cryptography to prove the assurance of the signer. Algorithm 3.1 shows this algorithm, which was proposed by ANSI [3]. The first step is to generate a cryptographic secure random number, which is called the ephemeral key k. At next, a scalar multiplication is performed. This second step is

Algorithm 3.1 ECDSA digital signature generation, cf. [3].

Require: domain parameter $D = (q, FR, a, b, G, n, h)$
Require: private key d of an associated key pair (d, Q)
Require: message m
Ensure: signature values (r, s)
1: select a random k, with $1 \leq k \leq n - 1$
2: $(x_1, y_1) = k \cdot G$, convert x_1 to an integer $\bar{x_1}$
3: $r = \bar{x_1} \ mod \ n$
4: $k^{-1} \ mod \ n$
5: $e = SHA - 1(m)$, where e is an integer
6: $s = k^{-1} \cdot (e + d \cdot r) \ mod \ n$
7: **if** $(r = 0) \vee (s = 0)$ **then**
8: goto step 1
9: **end if**
10: **return** (r, s)

Algorithm 3.2 Key pair generation, cf. [24].

Require: domain parameter $D = (q, FR, a, b, G, n, h)$
Ensure: public key Q, private key d
select a random d, with $1 \leq d \leq n - 1$
$Q = d \cdot G$
return (d, Q)

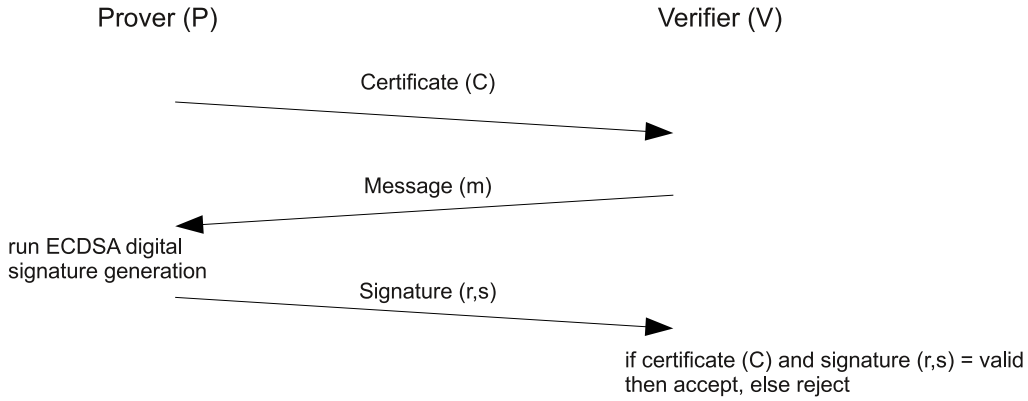

Figure 3.1: Authentication protocol with ECDSA.

the most time intensive part of the whole ECDSA signature-generation process. SHA-1 is used as a hash function to generate a fingerprint of the message m. The long-time public-private key pair is only used in step six of Algorithm 3.1. An algorithm to generate an associated key pair is shown in Algorithm 3.2, cf [3, 24]. This algorithm uses the same domain parameter as Algorithm 3.1 and d means the private key. Figure 3.1 shows the authentication protocol message exchange which uses ECDSA according to the ISO 9798-3 standard. In the first step, the prover or in our case the RFID tag transmits a certificate to the verifier, which includes the public key Q. This certificate is signed by a trusted third party and could get validated by the verifier. If the certificate is valid, then the verifier transmits a message m to the prover. The prover executes the ECDSA digital signature algorithm and sign the message m. In the third step, the prover transmits the signature values r and s back to the verifier. Now, the verifier uses Algorithm 3.3 to verify the signature values. If the signature is valid, then the authentication process is finish and the prover is accept, else the prover is reject.

3.3 Schnorr's Identification Scheme

In 1991, Claus Peter Schnorr, [52] proposed an other authentication protocol, which is based on an identification scheme. Figure 3.2 shows the message exchange of Schnorr's identification scheme. One execution of this protocol requires a set of parameters and arrangements. We present this parameters in the following enumeration:

- Domain parameters of the elliptic curve $D = (q, FR, a, b, G, n, h)$, where q means the finite field, a and b are parameters which characterize the elliptic curve, G is the base point, and n is called the order of the elliptic curve. This parameters have to

Algorithm 3.3 ECDSA digital signature verification, cf. [24].

Require: domain parameter $D = (q, FR, a, b, G, n, h)$
Require: public key Q of an associated key pair (d, Q)
Require: message m, signature values (r, s)
Ensure: valid or rejected signature
  $e = SHA - 1(m)$, where e is an integer
  $w = s^{-1}\ mod\ n$
  $u_1 = e \cdot w\ mod\ n$, $u_2 = r \cdot w\ mod\ n$
  $X = u_1 \cdot P + u_2 \cdot Q$
  if $X = \infty$ **then**
    return reject signature
  end if
  $v = \bar{x_1}\ mod\ n$, with $\bar{x_1}$ is x_1 converted to an integer
  if $v \neq r$ **then**
    return reject signature
  else
    return accept signature
  end if

be fixed in case of identification.

- Similar to ECDSA, the prover requires a private secret a, such that $Z = -a \cdot G$, where G is the base point and Z is the public part of the key pair, similar to ECDSA.

The prover generates, in the first step a secure random number r, which is similar to the ephemeral key in ECDSA. Afterwards, the prover calculates a new point X with $X = r \cdot G$ and transmits the coordinates of this point to the verifier. In the second step, the verifier generates a random value e and transmits it to the prover. Now the prover calculates y, where $y = a \cdot e + r\ mod\ n$ and transmits this value back to the verifier. At last, the verifier can check the identification process by calculating $y \cdot P + e \cdot Z = X$. If $y \cdot P + e \cdot Z$ equals to X then accept the prover, else reject it. From a security point of view, there are the same critical operations as already described for ECDSA. The two critical operations are the point multiplication $r \cdot G$ and the multiplication $a \cdot e\ mod\ n$. In order to prevent attacks

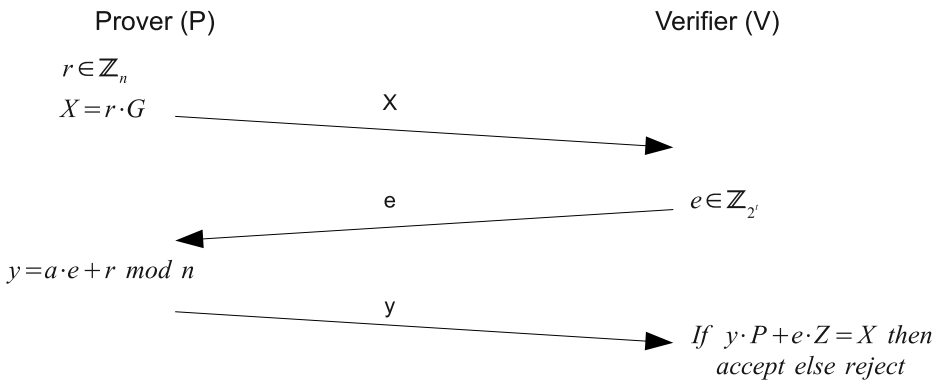

Figure 3.2: Schnorr's identification scheme [34].

on this operations, secure algorithms can be used, like the Montgomery ladder algorithm or the Montgomery multiplication algorithm.

3.4 Okamoto's Identification Scheme

In 1993, Tatsuaki Okomoto, [47] proposed an identification scheme which has same advantages as Schnorr's identification scheme. This identification scheme is also resistant against active attacks and concurrent attacks, cf. [5, 34]. Figure 3.3 shows the message exchange of Okamoto's identification scheme. One execution of this protocol requires a set of parameters and arrangements. We present this parameters in the following enumeration:

- Domain parameters of the elliptic curve $D = (q, FR, a, b, G_1, G_2, n, h)$, where q means the finite field, a and b are parameters which characterize the elliptic curve, G_1 and G_2 are two base points and n is called the order of the elliptic curve. This parameters have to be fix in case of authentication.
- This protocol uses a private key pair (s_1, s_2), which consists of two secure random numbers. The corresponding public key Z can be calculated with $Z = -s_1 \cdot G_1 - s_2 \cdot G_2$

At first, the prover generates two secure random numbers r_1 and r_2, which have the same functionality as the ephemeral key in case of ECDSA. Afterwards, the prover calculates $X = r_1 \cdot G_1 + r_2 \cdot G_2$ and transmits the result X to the verifier. In the next step, the verifier generates also a secure random number e and transmit this value to the prover. Now, the prover calculates two values y_1 and y_2, where $y_i = r_i + e \cdot s_i \ mod \ n$ and $i \in \{1, 2\}$, and transmit both values to the verifier. The verifier can use all this information to identify the prover. If $y_1 \cdot G_1 + y_2 \cdot G_2 + e \cdot Z = X$, then accept the prover, else reject it. In contrast to ECDSA authentication protocol and Schnorr's identification scheme, Okamoto's identification scheme requires two point-multiplications instead of one point multiplication. Consequently, Okamoto's protocol requires much more clock cycles for one protocol exchange, because the point multiplication is the most time intensive part. From a security point of view, this protocol has the same critical operations such as we described for the Schnorr's identification scheme and the ECDSA authentication protocol.

If only identification is required, the advantage of Schnorr's and Okamoto's identification scheme is, that they require no hash function. But this fact has only a small

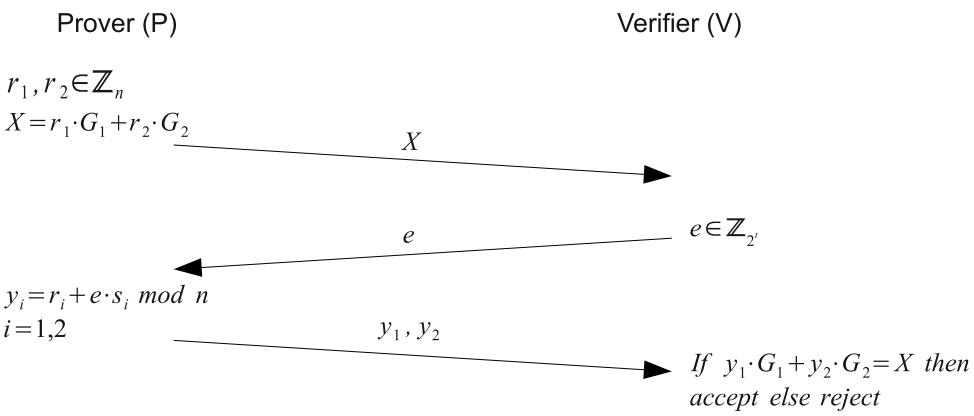

Figure 3.3: Okamoto's identification scheme [34].

impact to the size of the module area, if a prime field is used. Because almost all functionality and also the memory requirements are determined by the scalar multiplication algorithm, which is equal for all three protocols. With the exception, that Okamoto's protocol requires much more clock cycles.

Chapter 4

Mathematical Basics for ECDSA

Modern cryptographic systems use mathematical principles to ensure their safety. All algorithms are used there, defined in terms of arithmetic operations. This chapter describes all algorithms and mathematical basics which are required to generate an ECDSA digital signature. The fist section contains information about finite-field arithmetics. Finite-field arithmetic is the basic for all other algorithms which are required for ECDSA. The second part deals specifically with the arithmetic in elliptic curves. Operations in elliptic curves are time intensive and so it is important to find fast algorithms. The ECDSA algorithm needs also a hash function and the third section gives an overview of SHA-1.

4.1 Finite-Field Arithmetic

Fields are abstractions of familiar number systems such as the rational numbers $\mathbb{Q}$ or the real numbers $\mathbb{R}$. A finite field $\mathbb{F}$ consists of a finite set of objects called field elements. We can define two operations, addition (denoted by $+$) and multiplication (denoted by $\cdot$) between this field elements and this operations must possess certain properties, cf. [24].

1. $(\mathbb{F}, +)$ is an abelian group with additive identity denoted by 0
2. $(\mathbb{F}, \cdot)$ is an abelian group with multiplicative identity denoted by 1
3. The distributive law holds: $(a + b) \cdot c = a \cdot c + b \cdot c$ for all $a, b, c \in \mathbb{F}$

All other operations on the finite field F_p can be defined with this two operations. Subtraction can be defined in terms of addition and division can be defined in terms of multiplications. For two field elements $a, b \in \mathbb{F}$ the subtractions $(a - b)$ can be written as $a + (-b)$, with $-b$ which is called the unique element in $\mathbb{F}$ so that $b + (-b) = 0$. The division $\frac{a}{b}$ can be written as $a \cdot b^{-1}$, with $b \neq 0$ and b^{-1} which is called the unique element in $\mathbb{F}$ such that $b \cdot b^{-1} = 1$.

The element $-b$ is called the negative element of b and the element b^{-1} is called the inverse element of b in $\mathbb{F}$.

The number of elements in a field $\mathbb{F}_q$ is called the *order* of the field. A finite field $\mathbb{F}$ with order q only exists if q is a prime power $q = p^m$. In this definition for q, p, the prime number is called the characteristic of $\mathbb{F}$ and m is a positive integer in $\mathbb{Z}^+$. A finite field $\mathbb{F}$ with q elements is denoted with $\mathbb{F}_q$. Furthermore, there is for each q precisely one finite field $\mathbb{F}_q$. This means that except a different labeling of the field element, two fields of order q are structurally the same.

Depending on m, we can distinguish between two types of finite fields $\mathbb{F}$. If $m = 1$, the field $\mathbb{F}$ is called a *prime field* $\mathbb{F}_p$, otherwise $\mathbb{F}$ is called an *extension field*. A frequently used type of *extension field* are *binary fields* $\mathbb{F}_{2^m}$. The choices of p and m can have a dramatic influence on the performance of algorithms over finite fields. There exist generic algorithms for arithmetic in arbitrary finite fields and specialized algorithms with a articulate better performance in finite fields with a particular form, cf. [24]

Binary Fields $\mathbb{F}_{2^m}$

A characteristic 2 finite field $\mathbb{F}_{2^m}$ contains 2^m elements. There exist many different ways to represent elements of $\mathbb{F}_{2^m}$. One way is to use the polynomial basis representation. Here the elements of $\mathbb{F}_{2^m}$ are the binary polynomials whose coefficients are in den field $\mathbb{F}_2 = (0, 1)$. The degree of the polynomial is most $m - 1$.

$$\mathbb{F}_{2^m} = \left\{ a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \cdots + a_2z^2 + a_0 \ : \ a_i \in 0, 1 \right\} \tag{4.1}$$

For any m exists an irreducible binary polynomial $f(z)$ which can be found efficiently. Irreducible of $f(z)$ means that m cannot be factored as a product of binary polynomials each of degree less than m.

Addition and multiplication operations in $\mathbb{F}_{2^m}$ can be calculated efficiently by using standard algorithms for ordinary integer and polynomial arithmetic. This two operations in binary fields $\mathbb{F}_{2^m}$ are defined as follow, see [8]:

- Addition: We define two elements $a = a_{m-1}z^{m-1}+\cdots+a_1z^1+a_0$ and $b = b_{m-1}z^{m-1}+\cdots+b_1z^1+b_0 \ \in \mathbb{F}_{2^m}$. The element r is the sum from $a+b$ in $\mathbb{F}_{2^m}$ where r is described as $r_{m-1}z^{m-1} + \cdots + r_1z^1 + r_0$ with $r_i \equiv a_i + b_i \ mod \ 2$.
- Multiplication: We define two elements $a = a_{m-1}z^{m-1} + \cdots + a_1z^1 + a_0$ and $b = b_{m-1}z^{m-1} + \cdots + b_1z^1 + b_0 \ \in \mathbb{F}_{2^m}$. The element s is the product from $a \cdot b$ in $\mathbb{F}_{2^m}$, where s is described as $s_{m-1}z^{m-1} + \cdots + s_1z^1 + s_0$ when the polynomial $a \cdot b$ is divided by $f(z)$ and all coefficient arithmetics performed modulo 2.

That was only a brief description of binary fields $\mathbb{F}_{2^m}$, because this work deals with elliptic curves over prime fields $\mathbb{F}_p$. More information about binary fields can be found in standard literature, e.g. [12, 24].

Prime Fields $\mathbb{F}_p$

For each odd prime, there exist a prime field $\mathbb{F}_p$ which contains p elements. There is only one prime field for each prime, but there exist many different ways to represent these elements. In this thesis the elements of $\mathbb{F}_p$ will be represented by a set of integers:

$$\{0, 1, \cdots, p - 1\} \tag{4.2}$$

In an abstract form addition and multiplication in prime fields can defined as follows, see [8]:

- Addition: We define two elements $a, b \in \mathbb{F}_p$. The sum r from $a + b$ in $\mathbb{F}_p$ is the remainder when the integer $a + b$ is divided by p. This operation is known as addition modulo p and is written $a + b \equiv r \ mod \ p$.

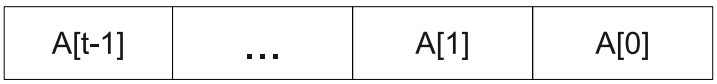

Figure 4.1: Representation of $a \in \mathbb{F}_p$ as an array A.

Algorithm 4.1 Addition of multi-word integers.

Require: $a, b \in \left[0, 2^{W \cdot t}\right)$ and $\epsilon^{'} \in \{0, 1\}$
Ensure: $(\epsilon,\ s)$, where $s \in \left[0, 2^{W \cdot t}\right)$ and $\epsilon \in \{0, 1\}$
  $(\epsilon,\ S[0]) \leftarrow A[0] + B[0] + \epsilon^{'}$
  for $i = 1$ to $t - 1$ **do**
    $(\epsilon,\ S[i]) \leftarrow A[i] + B[i] + \epsilon$
  end for
  return $(\epsilon,\ s)$

- Multiplication: We define two elements $a, b \in \mathbb{F}_p$. The product s from $a \cdot b$ in $\mathbb{F}_p$ is the remainder when the integer $a \cdot b$ is divided by p. This operation is known as multiplication modulo p and written $a \cdot b \equiv s \ mod \ p$.

The bit length of p is m where $m = \lceil \log_2 p \rceil$. In real implementations, integers with m bits cannot be used directly because the word size W of the architecture is often smaller then m. For this reason integers must be split and stored in an array $A = (A[0], A[1], \cdots A[t-1])$ of t W-bit words. The word-length t is defined as $t = \left\lceil \frac{m}{W} \right\rceil$. An example of an array A is shown in Figure 4.1, where the rightmost bit of $A[0]$ is the least significant bit. The bit representation, shown in 4.1 is used in all algorithms described in Section 4 and Section 5.

Algorithms for arbitrary primes p and special primes recommended by SECG [48] are presented in the subsequent Sections 4.1.1 to 4.1.4.

4.1.1 Addition and Subtraction

Algorithms for field addition and field subtraction can be separated in algorithms for multi-word integers and an optionally reduction step modulo p. The basic operation is the addition of two single word integers $a, b \in \left[0, 2^W\right)$ and a carry bit $\epsilon^{'} \in \{0, 1\}$. We define this addition result $(\epsilon,\ s)$ with the sum s and the carry bit ϵ as follows, cf. [24]:

$$\begin{aligned} w &= a + b + \epsilon^{'} \\ s &\leftarrow w \ mod \ 2^W \\ \epsilon &\leftarrow 0 \ if \ w \in \left[0, 2^W\right), \ otherwise \ \epsilon \leftarrow 1 \end{aligned} \tag{4.3}$$

Algorithm 4.1 is from [24] and illustrates a generic version of a multi-word integer addition. Arithmetic in a finite field $\mathbb{F}_p$ needs also an optional reduction step modulo p. The complete algorithms for addition and subtraction are shown in Algorithm 4.2 and Algorithm 4.3. There, $\neg a$ means the binary-inverse integer from a. If $s \geq p$, a final reduction operation after the last operation is needed. This versions are used in the ECDSA module with a word size of 16 bits.

Algorithm 4.2 Addition in $\mathbb{F}_p$.

Require: $a, b \in [0, 2^{W \cdot t})$
Ensure: $s = (a + b)\ mod(p)$
  Use Algorithm 4.1 to calculate $(\epsilon,\ s)$, with a, b and $\epsilon^{'} = 0$
  if $\epsilon = 1$ **then**
    Use Algorithm 4.1 to calculate $(\epsilon,\ s)$, with $s, \neg p$ and $\epsilon^{'} = 0$
  end if
  return (s)

Algorithm 4.3 Subtraction in $\mathbb{F}_p$.

Require: $a, b \in [0, 2^{W \cdot t})$
Ensure: $s = (a - b)\ mod(p)$
  Use Algorithm 4.1 to calculate $(\epsilon,\ s)$, with $a, \neg b$ and $\epsilon^{'} = 1$
  if $\epsilon = 0$ **then**
    Use Algorithm 4.1 to calculate $(\epsilon,\ s)$, with s, p and $\epsilon^{'} = 0$
  end if
  return (s)

4.1.2 Multiplication

Field multiplication in $\mathbb{F}_p$ can also accomplished in two steps. The first step is to multiply a and b, which are elements of $\mathbb{F}_p$ as integers. Afterwards, the result can be reduced modulo p as the second step. Algorithm 4.4 and Algorithm 4.5 are two generic versions of elementary integer-multiplication routines. These algorithms create a $2 \times W \times t$-bit quantity obtained by concatenation of $W \times t$-bit words a and b. This could be a problem in products with constraint devices like RFID tags or smartcards. Another problem is that reduction modulo p needs a division operation which is a complex operation. For these applications, algorithms with implicit reduction are the better choice. Therefore, we can use the Montgomery multiplication algorithm cf. Section 4.1.2 or an advanced version of Algorithm 4.5 with implicit reduction. This sophisticated algorithm for field multiplication is shown in Section 4.1.2 and uses a technique called fast reduction or NIST reduction which is described in Section 4.1.3.

There exist some algorithms for field squaring of $a \in \mathbb{F}_p$ which reduce the number of required single-precision multiplications by roughly the half. An improved version of Algorithm 4.5 to calculate the square of a is shown in [24]. An own squaring algorithm has been renounced since this ECDSA module is optimized for area.

Integer Multiplication

The following two algorithms calculate the $2 \times W$-bit multiplication result from $a, b \in [0, W \cdot t]$. Both algorithms necessitate a runtime of $\mathbb{O}(t^2)$ for multiplication of two $W \times t$-bit integers. There exist some other algorithms like *Karatsuba-Ofman* multiplication with a complexity of $\mathbb{O}(t^{log_2 3})$. But these algorithms are rather interesting for implementation in software or hardware implementations with binary fields $\mathbb{F}_{2^m}$. Algorithms which use *Karatsuba-Ofman* multiplication not further mentioned here. For some more information see [10, 17, 24, 40].

Algorithm 4.4 uses the so called inner product $(C\,[i + j] + A[i] \cdot B[j] + U)$ to calculate

Algorithm 4.4 Multiplication in $\mathbb{F}_p$ - operand-scanning form.

Require: $a, b \in [0, p-1)$
Ensure: $s = a \cdot b$
  Set $C[i] \leftarrow 0$ for $0 \leq i \leq t-1$
  for $i = 0$ to $t-1$ **do**
    $U \leftarrow 0$
    for $j = 0$ to $t-1$ **do**
      $(U\ V) \leftarrow C\,[i+j] + A[i] \cdot B[j] + U$
      $C\,[i+j] \leftarrow V$
    end for
    $C\,[i+t] \leftarrow U$
  end for
  return (c)

Algorithm 4.5 Multiplication in $\mathbb{F}_p$ - product-scanning form.

Require: $a, b \in [0, p-1)$
Ensure: $s = a \cdot b$
  $R_0 \leftarrow 0,\ R_1 \leftarrow 0,\ R_2 \leftarrow 0$
  for $k = 0$ to $2t-2$ **do**
    for For each element of $\{(i,j) | i + j = k,\ 0 \leq i, j \leq t-1\}$ **do**
      $(U\ V) \leftarrow A[i] \cdot B[j]$
      $(\epsilon, R_0) \leftarrow R_0 + V$
      $(\epsilon, R_1) \leftarrow R_1 + U + \epsilon$
      $R_2 \leftarrow R_2 + \epsilon$
    end for
    $C[k] \leftarrow R_0,\ R_0 \leftarrow R_1,\ R_1 \leftarrow R_2$
  end for
  $C[2t-1] \leftarrow R[0]$
  return (c)

the result and Algorithm 4.5 is arranged that the product is calculated from left to right. But both algorithms are from [24] and create a result with a size of $2 \times W$ bits.

Montgomery Multiplication Algorithm

Peter Montgomery proposed in 1985 [43] an efficient algorithm for modular multiplication. The Montgomery algorithm is a fast algorithm to compute $a \cdot b\ mod\ p$, if the modulo p has no particular form. This method to multiply modulo p avoids division by p for the reduction. The idea is to convert the reduction modulo p to a reduction modulo R where division is easier. It is useful to select R as a power of 2 because in this case a complex division can be replaced by a simple shift right operation. If we choose R, it is important that R is relatively prime to p ($gcd(n, R) = 1$).

An integer $a\ mod\ p$ can be represented in the Montgomery domain as $[a]_R = a \cdot R\ mod\ p$. This transformation is agreeable with an addition

$$[a+b]_R \equiv (a+b) \cdot R \equiv a \cdot R + b \cdot R \equiv [a]_R + [b]_R \tag{4.4}$$

Algorithm 4.6 Montgomery multiplication algorithm.

Require: $p \in \mathbb{P},\ a, b \in [0, p-1)\,,\ R = 2^n,\ R \cdot R^{-1} - p \cdot p^{'} = 1$
Ensure: $a \cdot b \cdot R^{-1}\ mod\ p$
  $T \leftarrow a \cdot b$
  $S \leftarrow T \cdot p^{'}\ mod\ R$
  $U \leftarrow (T + S \cdot p)\,/R$
  if $U \geq M$ **then**
    return $U - M$
  else
    return U
  end if

but not agreeable with a multiplication

$$[a \cdot b]_R = [a]_R \cdot [b]_R \cdot R^{-1} \tag{4.5}$$

The Montgomery multiplication algorithm, see Algorithm 4.6 transforms two numbers $a, b \in [0, p-1]$ implicit to the Montgomery domain. In this case, we can define the Montgomery multiplication $a * b$ as follow:

$$[c]_R = a * b = a \cdot b \cdot R^{-1}\ mod\ p \tag{4.6}$$

In order to get the desired result $c = a \cdot b\ mod\ p$, some pre or post computations are required to remove the factor R^{-1}.

- **Pre computation:** To remove the factor R^{-1} from the output, one input operant should get converted into the Montgomery domain. The Montgomery multiplication algorithm can be used itself, if one multiplication operant is chosen as a constant factor R^2.

$$[a]_R = a * R^2 = a \cdot R^2 \cdot R^{-1}\ mod\ p = a \cdot R\ mod\ p \tag{4.7}$$
$$c = [a]_R * b = [a]_R \cdot b \cdot R^{-1}\ mod\ p = a \cdot b\ mod\ p \tag{4.8}$$

- **Post computation:** When both input operants of the Montgomery multiplication algorithm are in Montgomery domain format the multiplication result is going to be $a \cdot b \cdot R\ mod\ p$. To remove this extra factor R in the result, performing one more Montgomery multiplication by a constant factor 1 is necessary.

$$c = [c]_R * 1 = [c]_R \cdot R^{-1}\ mod\ p = a \cdot b \cdot R \cdot R^{-1}\ mod\ p = a \cdot b\ mod\ p \tag{4.9}$$

Algorithm 4.6 shows the original version of the Montgomery multiplication algorithm which was published by Peter Montgomery in 1985, see [43]. In most projects this original version is not implemented. There is a variety of other implementations to perform Montgomery multiplication, which need a different number of basic integer operations like addition, multiplication or shift operations. Zhang Jia-hong et al. [59] proposed in 2009 a Montgomery multiplication algorithm which only needs $2 \cdot t^2 + t + 2$ clock cycles for one multiplication. This published version requires $3 \times W$-bit registers and is currently the fastest implementation of Montgomery multiplication, cf. [59]. In this work, we use the algorithm proposed by Zhang Jia-hong, with some modifications. The multiplication

Table 4.1: Time requirements of Montgomery multiplication methods.

Method	Multiplications	Additional Operations
SOS	$2 \cdot t^2 + t$	$4 \cdot t^2 + 4 \cdot t + 2$
CIOS	$2 \cdot t^2 + t$	$4 \cdot t^2 + 4 \cdot t + 2$
FIOS	$2 \cdot t^2 + t$	$5 \cdot t^2 + 3 \cdot t + 2$
FIPS	$2 \cdot t^2 + t$	$6 \cdot t^2 + 2 \cdot t + 2$
CIHS	$2 \cdot t^2 + t$	$4 \cdot t^2 + 4 \cdot t + 2$
Zhang Jia-hong	$2 \cdot t^2 + t$	2
This work	$\mathbf{2 \cdot t^2 + t}$	$\mathbf{2 \cdot t}$

unit, which is used for multiplication with implicit fast reduction, has a similar design than the presented by Jong Zhang Jia-hong but our design required only $2 \times W + \frac{W}{2}$-bit registers. It was possible to map the original design to our hardware architecture and we include some countermeasures against SPA and DPA attacks. The modified Montgomery multiplication algorithm, which is described in Algorithm 4.7, is only slightly slower and needs $2 \cdot t^2 + 3 \cdot t$ clock cycles for one multiplication, but it is optimized for area and reusing operations from multiplication with implicit fast reduction. Table 4.1 constitutes an overview of different Montgomery multiplication algorithms, cf. [33].

Multiplication with implicit fast Reduction

A second possibility for multiplication with implicit reduction is a combination of a generic multiplication algorithm in product scan form, see Algorithm 4.5 and an implicite fast reduction, see Section 4.1.3. But this method only works with special prime numbers, like NIST primes or other special primes. The described algorithm uses the prime from the SECP160r1 elliptic curve. For more information about this fast reduction look at Section 4.1.3.

Multiplication with fast reduction uses as origin the multiplication algorithm in product-scanning form with a word size of 16 bits. An advantage of this algorithm is that the current word $C[i]$ of the multiplication result is finished before the next word is calculated. The algorithm for fast reduction which is described in Algorithm 4.17 applies the reduction modulo p_{160r1}. This algorithm uses the lower and higher 160 bits of the multiplication result in an interleaved form. At first, the lower 160 bits of the result $c = (C[9], \ldots, C[1], C[0])$ is calculated and stored. In the next step the element $C[10]$ of the higher 160 bits is calculated. This element is added at two different places to the lower result c. The addition of one element is done interleaved by storing the carry bit if one is generated. For that purpose some additional memory is required to save two separate carry bits. Then the next element $C[12]$ is calculated and added interleaved. The last to elements $C[18]$ and $C[19]$ have to be added on three different places to c. All addition operations are processed modulo p_{160r1}, therefore maybe an additional reduction step is needed. The number of clock cycles to multiply two integers modulo p_{160r1} is $t^2 + 5t$ in the worst case. Hence this algorithm is four times faster than the Montgomery multiplication algorithm described in Algorithm 4.7. Because the Montgomery multiplication algorithm must be used twice. One time for multiplication of the integers a and b and a second time to transform the result from the Montgomery domain to the integer domain. Algorithm 4.8 describes the multiplication with implicit fast reduction which is used for field operations modulo p_{160r1}.

Algorithm 4.7 Our improved Montgomery multiplication algorithm (16-bit word size).

Require: $p \in \mathbb{P},\ a, b \in [0, p-1)\,,\ R = W^t,\ p^{'} = -p^{-1}\ mod\ W$
Require: x, y with 176 bits, u and d with 176 bits, reg with 36 bits
Ensure: $a \cdot b \cdot R^{-1}\ mod\ p$

```
reg ← x[0] · y[0]
u[0] ← reg[0] · p'
reg ← reg + u[0] · p[0]
reg[0] ← reg[1], reg[1] ← reg[2], reg[2] ← 0
for j = 1 to j < 11 do
  for i = 1 to i ≤ j do
    reg ← reg + u[i − 1] · p[j + 1 − i]
  end for
  for i = j to i ≥ 0 do
    reg ← reg + x[i] · y[j − i]
  end for
  u[j] ← reg[0] · p'
  reg ← reg + u[j] · p[0]
  reg[0] ← reg[1], reg[1] ← reg[2], reg[2] ← 0
end for
for j = 9 to 0 do
  for i = 0 to i ≤ j do
    reg ← reg + u[10 − j + i] · p[10 − i]
  end for
  for i = j to i ≥ 0 do
    reg ← reg + x[10 − j + i] · y[10 − i]
  end for
  d[9 − j] ← reg[0], reg[0] ← reg[1], reg[1] ← reg[2], reg[2] ← 0
end for
u ← d − p
if reg[1] > 0 then
  return (u)
else
  return (d)
end if
```

4.1.3 Reduction

Reduction $a\ mod\ p$ is an important part of modular arithmetic, because it can be a time and memory expensive operation. In particular, if the prime p has no special form then the reduction is as expensive as a multiplication operation. Field multiplications and reductions are used in many cryptographic algorithms. Therefore, it is important to select moduli with a special form, so that reduction gets faster. This section only presents the reduction method of Barrett and the fast reduction for special primes, because the modulus p of the field $\mathbb{F}_p$ on which the elliptic curve is defined has a special form. Only the modulus involved to the signature-generation operation has a general form, but therefore the Montgomery multiplication algorithm is used to multiply.

Algorithm 4.8 Multiplication with implicit fast reduction modulo p_{160r1}.

Require: $a, b \in [0, p_{160r1} - 1)$
Ensure: $c = a \cdot b \ mod \ p_{160r1}$
$R_0 \leftarrow 0,\ R_1 \leftarrow 0,\ R_2 \leftarrow 0$
for $i = 0$ to $i < 10$ **do**
  for $j = 0$ to $j \leq i$ **do**
    $\{R_2, R_1, R_0\} \leftarrow \{R_2, R_1, R_0\} + A[i] \cdot B[i - j]$
  end for
  $C[i] \leftarrow R_0,\ R_0 \leftarrow R_1,\ R_1 \leftarrow R_2$
end for
$\epsilon_1 \leftarrow 0$ and $\epsilon_2 \leftarrow 0$
for $i = 0$ to $i < 9$ **do**
  for $j = i + 1$ to $j \leq 9$ **do**
    $\{R_2, R_1, R_0\} \leftarrow \{R_2, R_1, R_0\} + A[j] \cdot B[10 + i - j]$
  end for
  $(\epsilon_1, C[i]) = C[i] + R_0 + \epsilon_1$
  $\epsilon_s \leftarrow \epsilon_2$
  if $i < 8$ **then**
    $(\epsilon_2, C[i + 1]) = C[i + 1] + ((R_0 << 15) \wedge 0\text{x}8000)$
    $(\epsilon_2, C[i + 2]) = C[i + 2] + ((R_0 >> 1) \wedge 0\text{x}7FFF) + (\epsilon_2 \vee \epsilon_s)$
    $R_0 \leftarrow R_1,\ R_1 \leftarrow R_2$
  else
    $(\epsilon_1, C[i]) = C[i] + R_0 + \epsilon_1$
    $(\epsilon_2, C[i + 1]) = C[i + 1] + ((R_0 << 15) \wedge 0\text{x}8000)$
  end if
end for
if $\epsilon_1 = 1$ **then**
  Use Algorithm 4.1 to calculate $(\epsilon_1,\ c)$, with $c, not(p)$ and $\epsilon^{'} = 0$
end if
if $(\epsilon_2 \vee \epsilon_s) = 1$ **then**
  Use Algorithm 4.1 to calculate $(\epsilon_1,\ c)$, with $c, not(p)$ and $\epsilon^{'} = 0$
end if
$(\epsilon_1, C[0]) = C[0] + (((R_0 >> 1) \wedge 0\text{x}7FFF) \vee ((R_1 << 15) \wedge 0\text{x}8000))$
$(\epsilon_1, C[1]) = C[1] + (((R_1 >> 1) \wedge 0\text{x}7FFF) \vee ((R_0 << 14) \wedge 0\text{x}8000)) + \epsilon_1$
$(\epsilon_1, C[2]) = C[2] + (((R_0 >> 2) \wedge 0\text{x}3FFF) \vee ((R_1 << 14) \wedge 0\text{x}C000)) + \epsilon_1$
$(\epsilon_1, C[3]) = C[3] + (((R_1 >> 2) \wedge 0\text{x}3FFF)) + \epsilon_1$
for $i = 4$ to $i < 10$ **do**
  $(\epsilon_1, C[i]) = C[i] + \epsilon_1$
end for
if $\epsilon_1 = 1$ **then**
  Use Algorithm 4.1 to calculate $(\epsilon_1,\ c)$, with $c, not(p)$ and $\epsilon^{'} = 0$
end if

Barrett Reduction

On basic method for reduction, which does not exploit a special modulus p, is the Barrett reduction algorithm. This algorithm finds for two positive integers a and p an integer $c = a \ mod \ p$. The basic idea of Barret reduction is to calculate the quotient $\frac{a}{p}$ without

Algorithm 4.9 Barrett reduction algorithm.

Require: $p,\ b \geq 3,\ k = \lfloor log_b\ p \rfloor + 1,\ 0 \leq a < b^{2k},\ and\ \mu = \lfloor b^{2k}/p \rfloor$
Ensure: $a\ mod\ p$
  $\hat{q} \leftarrow \left\lfloor \lfloor z/b^{k-1} \rfloor \cdot \mu / b^{k+1} \right\rfloor$
  $r \leftarrow \left(a\ mod\ b^{k+1}\right) - \left(\hat{q} \cdot p\ mod\ b^{k+1}\right)$
  if $r < 0$ **then**
    $r \leftarrow r + b^{k+1}$
  end if
  while $r \geq p$ **do**
    $r \leftarrow r - p$
  end while
  return (r)

a division by p, which is a time expensive operation. For this purpose, Barret uses a pre-calculated value μ, which only depends on p. Therefore, only two multiplications, four shift-right operations and some addition operations are required to calculate $a\ mod\ p$, cf. [6, 24, 30] Algorithm 4.9 is from [24] and demonstrates one possible implementation of the Barrett-reduction method.

Fast Reduction for special Primes

A very well example for algorithms which uses fast reduction for special primes, was shown by the NIST in FIPS 186-3 [45]. The technique which uses special NIST primes is called NIST reduction. This reduction method can be expanded to other primes like the primes in the SEC2 standard for elliptic-curve domain parameters [48].

The mathematical background for *NIST* reduction is the advantage of special moduli, called Mersenne numbers $m = 2^k - 1$. However Mersenne numbers are rarely primes and only these particular primes are cryptographically useful. Mersenne numbers can be expanded to a bigger family of integers, known as pseudo Mersenne numbers. Pseudo Mersenne numbers are described by Richard Crandall, see [14]. He defined those as $2^k - c$, when c is a small integer. Modular reduction by pseudo Mersenne numbers are very efficiently done using a few constant multiplications by the small integer c. In 1999, Jerome Solinas [53] described a bigger family of numbers called the generalized Mersenne numbers. Solinas show also a method to calculate the reduction $a\ mod\ p$ which only a few integer additions and subtractions. Therefor, he uses a special form of the modulus $p = f(t)$, where t is a power of 2, which he called generalized Mersenne numbers. All five NIST primes and the primes in SEC2 standard are based on generalized Mersenne numbers, cf. [11]

The mathematical description in this section based on the work of Jerome Solinas, cf. [53, 58]. The generalized Mersenne numbers, described by Solinas has the form $m = 2^d - c_1 2^{d-1} - \ldots - c_d$, with integers $c_i \in \{-1, 0, 1\}$. If we work in $\mathbb{Z}_p$ and make a reduction step after each multiplication, the number do reduce can not be larger than $(p-1)\cdot(p-1) < p^2$. An integer $a \in [0, (p-1)\cdot(p-1)]$ can be written as

$$a = \sum_{j=0}^{2t-1} A[j] \cdot 2^{jW} \tag{4.10}$$

where W is the word size and t is the number of words from integer a. The aim is to find

some B_i that are linear combinations of A_j such that $a \ mod \ p$ is congruent to their sum and difference. This requirement can be described in a mathematical form as follow:

$$a = \sum_{j=0}^{2t-1} A[j] \cdot 2^{jW} \equiv \sum_{i=0}^{t-1} B[i] \cdot 2^{iW} \ mod \ p \tag{4.11}$$

Solinas presents an excellent method to find B_i for arbitrary generalized Mersenne numbers. In the first step it is necessary to delineate p as a polynomial in 2^W. The general form of this polynomial is:

$$p = f(2^{tW}) = 2^{tW} - c_1 \cdot 2^{(t-1)W} - \ldots - c_t \quad c_t \in \{-1, 0, 1\} \tag{4.12}$$

From Equation 4.11, there will be in general $2 \cdot t$ A_j and t B_i elements. An integer a can be described as two vector multiplication operations and one addition operation.

$$a = (A_0 \ldots A_{t-1}) \cdot \begin{pmatrix} 1 \\ \vdots \\ 2^{(t-1)W} \end{pmatrix} + (A_t \ldots A_{2t-1}) \cdot \begin{pmatrix} 2^{tW} \\ \vdots \\ 2^{(2t-1)W} \end{pmatrix} \tag{4.13}$$

The next step is to compute the reduction mod p from $\begin{pmatrix} 2^{tW} \\ \vdots \\ 2^{(2t-1)W} \end{pmatrix}$ and write it as a matrix vector multiplication.

$$\begin{pmatrix} 2^{tW} \\ \vdots \\ 2^{(2t-1)W} \end{pmatrix} \equiv X \cdot \begin{pmatrix} 1 \\ \vdots \\ 2^{(t-1)W} \end{pmatrix} \ mod \ p \tag{4.14}$$

Now Equation 4.14 can be used to substitute into Equation 4.13. The result, described in Equation 4.15, is the finished rule for fast reduction modulo p. Matrix X includes the information, how the higher bits of a number must be used to reduce a number with only addition and subtraction operations.

$$a \equiv \left(A_0, \ldots, A_{(t-1)W}\right) + \left(A_{tW}, \ldots, A_{(2t-1)W}\right) \cdot X \cdot \begin{pmatrix} 1 \\ \vdots \\ 2^{(t-1)W} \end{pmatrix} \tag{4.15}$$

Solinas described in his paper also another way to determine the reduction matrix X. Therefore he uses a LFSR to describe the reduction role. For more details about this second method see [53].

Reduction with this method has great advantages if the exponents of the polynomial p are multiple of the word size W and $W = 2^k$ with $k \in \mathbb{Z}_+$. In this case, only addition and subtraction operations are needed. If the exponents of the prime in polynomial form, cf. Equation 4.12 are not a multiple of W, the reduction becomes more complicated, because bit shifts are required. The *NIST* primes, defined in *FIPS186-3* [45], use this advantages. In this thesis, primes from *SECP160* standard [48] are used and those have not this simple form. The following enumeration shows findings for fast reduction with primes from *SECP160k1* and *SECP160r1*. The prime from *SECP160r2* is equal to the prime in the *SECP160k1* definition. In Equations 4.16 and Equation 4.17, which show the

primes from *SECP160*, are some exponents of the polynom primes. Therefore, the only possible word size is $W = 1$ and the square matrix X from Equation 4.14 has 160 rows and columns. Hence, only the finished reduction direction is illustrated for *SECP160k1* and *SECP160r1*. In the subsequent consideration, symbols and operations described in Appendix A.2 are used.

- **SECP160k1:** The prime of the elliptic curve underlying field is described in polynomial form, see Equation 4.16. With the mathematical theory, described by Solinas, we created an algorithm for fast reduction, see Algorithm 4.10. This algorithm needs 74 additions, 8 rotate-right operation and 64 shift operations. The reduction methode described by Solinas is not efficient for the prime p_{160k1}. With a combination from Solinas reduction theorem and Crandall reduction theorem there exist some possible improvements. The prime p_{160k1} can retyped to $p_{160k1} = 2^{160} - 2^{32} - c$, where $c = 2^{14} - 2^{12} - 2^9 - 2^8 - 2^7 - 2^3 - 2^2 - 1$. But those need some additional multiplications with c instead of shift and addition operations. In comparison to the other prime from *SECP160r1*, the improved reduction is still more difficult.

$$p_{160k1} = 2^{160} - 2^{32} - 2^{14} - 2^{12} - 2^9 - 2^8 - 2^7 - 2^3 - 2^2 - 1 \tag{4.16}$$

- **SECP160r1:** The prime of the elliptic curve underlying field is described in polynomial form, see Equation 4.17. An algorithm to reduce a number modulo this prime is shown in Algorithm 4.11. This algorithm needs 3 additions, one rotate-right operation and one shift operations. A fast reduction algorithm can be implemented efficiently for this prime. It is also possible to use this algorithm to create a multiplication method with implicit fast reduction. A cogitable implementation is described in Section 4.1.2. For this reason the *SECP160r1* elliptic curve with prime p_{160r1} is selected.

$$p_{160r1} = 2^{160} - 2^{31} - 1 \tag{4.17}$$

4.1.4 Inversion

Modular-inverse arithmetic is an essential operation in public-key cryptography. In this work, the modular inverse of an element $a \in \mathbb{F}_p$ is required twice. One time to convert an elliptic-curve point from projective coordinates to affine coordinates and a second time to generate the ECDSA digital signature. The standard modular inverse can be defined as follow: Assume a is a nonzero element in $\mathbb{F}_p$. An integer x is called the modulo inverse of a, if and only if $a \cdot x \equiv 1 \ mod \ p$, where $x \in \mathbb{F}_p$. The inverse element is denoted with a^{-1} and there exist some algorithms to compute this. The following subsections present two different algorithms to solve this problem.

Extended Euclidean Algorithm

The classical algorithm to compute the inverse element in $\mathbb{F}_p$ is the extended Euclidean algorithm for integers. Let a and b be integers, both are not 0. The greatest common divisor *gcd* of a and b, denoted $gcd(a, b)$, is the largest integer d that divides both a and b. To calculate the *gcd* of positive integers a and b where $b \geq a$, we can use the classical

Algorithm 4.10 Fast reduction modulo $p_{160k1} = 2^{160} - 2^{32} - 2^{14} - 2^{12} - 2^9 - 2^8 - 2^7 - 2^3 - 2^2 - 1$.

Require: An integer $c = (c_{319}, c_{318}, \ldots, c_1, c_0)$ in base 2 with $0 \leq c < p_{160k1}^2$
Ensure: $a \ mod \ p_{160k1}$
Define s_i as 160 bit integers
$s_1 = (c_{159}, c_{158}, \ldots, c_1, c_0)$ $s_2 = (c_{319}, c_{318}, \ldots, c_{161}, c_{160})$
$s_3 = ROTL_2(s_2)$ $s_4 = ROTL_3(s_2)$ $s_5 = ROTL_7(s_2)$ $s_6 = ROTL_8(s_2)$
$s_7 = ROTL_9(s_2)$ $s_8 = ROTL_{12}(s_2)$ $s_9 = ROTL_2(s_{14})$ $s_{10} = ROTL_2(s_2)$
$s_{11} = (s_2 >> 126) \wedge \neg 0x2$ $s_{12} = (s_2 >> 125) \wedge \neg 0x3$ $s_{13} = (s_2 >> 121) \wedge \neg 0x7$
$s_{14} = (s_2 >> 120) \wedge \neg 0x8$ $s_{15} = (s_2 >> 119) \wedge \neg 0x9$ $s_{16} = (s_2 >> 116) \wedge \neg 0xC$
$s_{17} = (s_2 >> 114) \wedge \neg 0xE$ $s_{18} = (s_2 >> 96) \wedge \neg 0x20$
$s_{19} = (s_2 >> 144) \wedge \neg 0x2$ $s_{20} = (s_2 >> 143) \wedge \neg 0x3$ $s_{21} = (s_2 >> 139) \wedge \neg 0x7$
$s_{22} = (s_2 >> 138) \wedge \neg 0x8$ $s_{23} = (s_2 >> 137) \wedge \neg 0x9$ $s_{24} = (s_2 >> 134) \wedge \neg 0xC$
$s_{25} = (s_2 >> 132) \wedge \neg 0xE$ $s_{26} = (s_2 >> 112) \wedge \neg 0x20$
$s_{27} = (s_2 >> 146) \wedge \neg 0x2$ $s_{28} = (s_2 >> 145) \wedge \neg 0x3$ $s_{29} = (s_2 >> 141) \wedge \neg 0x7$
$s_{30} = (s_2 >> 140) \wedge \neg 0x8$ $s_{31} = (s_2 >> 139) \wedge \neg 0x9$ $s_{32} = (s_2 >> 136) \wedge \neg 0xC$
$s_{33} = (s_2 >> 134) \wedge \neg 0xE$ $s_{34} = (s_2 >> 116) \wedge \neg 0x20$
$s_{35} = (s_2 >> 149) \wedge \neg 0x2$ $s_{36} = (s_2 >> 148) \wedge \neg 0x3$ $s_{37} = (s_2 >> 144) \wedge \neg 0x7$
$s_{38} = (s_2 >> 143) \wedge \neg 0x8$ $s_{39} = (s_2 >> 142) \wedge \neg 0x9$ $s_{40} = (s_2 >> 139) \wedge \neg 0xC$
$s_{41} = (s_2 >> 137) \wedge \neg 0xE$ $s_{42} = (s_2 >> 119) \wedge \neg 0x20$
$s_{43} = (s_2 >> 150) \wedge \neg 0x2$ $s_{44} = (s_2 >> 149) \wedge \neg 0x3$ $s_{45} = (s_2 >> 145) \wedge \neg 0x7$
$s_{46} = (s_2 >> 144) \wedge \neg 0x8$ $s_{47} = (s_2 >> 143) \wedge \neg 0x9$ $s_{48} = (s_2 >> 140) \wedge \neg 0xC$
$s_{49} = (s_2 >> 138) \wedge \neg 0xE$ $s_{50} = (s_2 >> 120) \wedge \neg 0x20$
$s_{51} = (s_2 >> 151) \wedge \neg 0x2$ $s_{52} = (s_2 >> 150) \wedge \neg 0x3$ $s_{53} = (s_2 >> 146) \wedge \neg 0x7$
$s_{54} = (s_2 >> 145) \wedge \neg 0x8$ $s_{55} = (s_2 >> 144) \wedge \neg 0x9$ $s_{56} = (s_2 >> 141) \wedge \neg 0xC$
$s_{57} = (s_2 >> 139) \wedge \neg 0xE$ $s_{58} = (s_2 >> 121) \wedge \neg 0x20$
$s_{59} = (s_2 >> 155) \wedge \neg 0x2$ $s_{60} = (s_2 >> 154) \wedge \neg 0x3$ $s_{61} = (s_2 >> 150) \wedge \neg 0x7$
$s_{62} = (s_2 >> 149) \wedge \neg 0x8$ $s_{63} = (s_2 >> 148) \wedge \neg 0x9$ $s_{64} = (s_2 >> 145) \wedge \neg 0xC$
$s_{65} = (s_2 >> 143) \wedge \neg 0xE$ $s_{66} = (s_2 >> 125) \wedge \neg 0x20$
$s_{67} = (s_2 >> 156) \wedge \neg 0x2$ $s_{68} = (s_2 >> 155) \wedge \neg 0x3$ $s_{69} = (s_2 >> 151) \wedge \neg 0x7$
$s_{70} = (s_2 >> 150) \wedge \neg 0x8$ $s_{71} = (s_2 >> 149) \wedge \neg 0x9$ $s_{72} = (s_2 >> 146) \wedge \neg 0xC$
$s_{73} = (s_2 >> 144) \wedge \neg 0xE$ $s_{74} = (s_2 >> 126) \wedge \neg 0x20$
return $\left(\sum_{i=1}^{74} s_i \ mod \ p_{160k1}\right)$

Euclidean algorithm. If b divided a, we get two new values q and r. This four values had to fulfill the equation $b = q \cdot a + r$, where q is called the quotient and $0 \leq r < a$ is called the

Algorithm 4.11 Fast reduction modulo $p_{160r1} = 2^{160} - 2^{31} - 1$.

Require: An integer $c = (c_{319}, c_{318}, \ldots, c_1, c_0)$ in base 2 with $0 \leq c < p_{160r1}^2$
Ensure: $a \ mod \ p_{160r1}$
Define s_i as 160 bit integers
$s_1 = (c_{159}, c_{158}, \ldots, c_1, c_0)$ $s_2 = (c_{319}, c_{318}, \ldots, c_{161}, c_{160})$
$s_3 = ROTL_{31}(s_2)$
$s_4 = (s_2 >> 98) \wedge \neg 0x7FFFFFFF$
return $(s_1 + s_2 + s_3 + s_4 \ mod \ p_{160r1})$

Algorithm 4.12 Extended Euclidean algorithm for integers.

Require: positive integers a and b with $a \leq b$
Ensure: $d = gcd(a, b)$ and integers x, y satisfying $a \cdot x + b \cdot y = d$
  $u \leftarrow a$, $v \leftarrow b$
  $x_1 \leftarrow 1$, $x_2 \leftarrow 0$, $y_2 \leftarrow 1$
  while $u \neq 0$ **do**
    $q \leftarrow \lfloor v/u \rfloor$, $r \leftarrow v - q \cdot u$, $x \leftarrow x_2 - q \cdot x_1$, $y = y_2 - q \cdot y_1$
    $v \leftarrow u$, $u \leftarrow r$, $x_2 \leftarrow x_1$, $x_1 \leftarrow x$, $y_2 \leftarrow y_1$, $y_1 \leftarrow y$
  end while
  $d \leftarrow v$, $x \leftarrow x_2$, $y \leftarrow y_2$
  return (d, x, y)

Algorithm 4.13 Inversion in $\mathbb{F}_p$ using the extended Euclidean algorithm.

Require: prime p and $a \in [1, p-1]$
Ensure: a^{-1} *mod* p
  $u \leftarrow a$, $v \leftarrow p$
  $x_1 \leftarrow 1$, $x_2 \leftarrow 0$
  while $u \neq 1$ **do**
    $q \leftarrow \lfloor v/u \rfloor$, $r \leftarrow v - q \cdot u$, $x \leftarrow x_2 - q \cdot x_1$
    $v \leftarrow u$, $u \leftarrow r$, $x_2 \leftarrow x_1$, $x_1 \leftarrow x$
  end while
  return $(x_1 \ mod \ p)$

remainder. The extended Euclidean algorithm is an extension of the classical Euclidean algorithm. This algorithm can find two integers x and y which fulfill the Equation 4.18.

$$a \cdot x + b \cdot y = d = gcd(a, b) \tag{4.18}$$

Algorithm 4.12 is from [24] and illustrates the extended Euclidean algorithm.

We can use Algorithm 4.12 with inputs (a, p) to calculate $a \cdot x \equiv 1 \ mod \ p$, where p is prime, $a \in [1, p-1]$. If the algorithm finished, it solves the Equation 4.18 where $d = gcd(a, p) = 1$ and thus $a \cdot x_1 + p \cdot y_1 = 1 \ mod \ p$. Subsequent to this occurrence $a \cdot x_1 \equiv 1 \ mod \ p$ and so $a^{-1} = x_1 \ mod \ p$. Algorithm 4.13 is from [24] and uses the extended Euclidean algorithm to calculate the inverse element in $\mathbb{F}_p$

One disadvantage of this algorithm is the expensive division operation. There exist some other versions of Algorithm 4.13, like the binary algorithm for inversion in $\mathbb{F}_p$, in with the expensive division operation is replaced by a cheaper shift and subtraction operation. Those algorithms have a runtime of $2 \cdot k$, where k is the maximum of the bit length of a and b. One possible implementation of a binary algorithm for inversion is presented in Algorithm 4.14 and it is from [24]. It is also possible to create a division algorithm directly from the binary algorithm, by changing the initialization conditions, cf. [24]. One disadvantage for implementation in hardware is that the integers x_1 and x_2 may have a negative value. In that case some additional storage for the algebraic sign is required.

Montgomery Inversion Algorithm

The basic strategy in Montgomery's method is to replace modular reduction $z \ mod \ p$ by a less expensive operation $z \cdot R^{-1} \ mod \ p$ for a suitably chosen R. In 1995, Burton S. Kaliski,

Algorithm 4.14 Binary algorithm for inversion in $\mathbb{F}_p$.

Require: prime p and $a \in [1, p-1]$
Ensure: a^{-1} *mod* p
  $u \leftarrow a$, $v \leftarrow p$
  $x_1 \leftarrow 1$, $x_2 \leftarrow 0$
  while $u \neq 1$ and $v \neq 1$ **do**
    while u is even **do**
      $u \leftarrow u/2$
      if x_1 is even then $x_1 \leftarrow x_1/2$; else $x_1 \leftarrow (x_1 + p)/2$
    end while
    while v is even **do**
      $v \leftarrow v/2$
      if x_2 is even then $x_2 \leftarrow x_2/2$; else $x_2 \leftarrow (x_2 + p)/2$
    end while
    If $u \geq v$ then: $u \leftarrow u - v$, $x_1 \leftarrow x_1 - x_2$
    else $v \leftarrow v - u$, $x_2 \leftarrow x_2 - x_1$
  end while
  if $u = 1$ **then**
    return x_1 *mod* p
  else
    return x_2 *mod* p
  end if

Jr. [32] proposed an algorithm which is derived from the extended Euclidean algorithm and can be divided in two phases. Phase one, shown in Algorithm 4.15, is also called almost Montgomery inverse, takes the input values a and p. The algorithm produces two outputs r and k, where $r = a^{-1} \cdot 2^k \ mod \ p$ and $n < k < 2n$. Phase two, shown in Algorithm 4.16, uses the output of phase one and produce the final result $x = a^{-1} \cdot R \ mod \ p$, where $R = 2^n$ and $2^{n-1} \leq p < 2^n$. The Montgomery multiplication algorithm can be used to transform the inverse element x back into integer domain from Montgomery domain. Figure 4.2 is from [23] and shows the input/output behavior from the Kaliski algorithm.

There exist some optimizations of the Kaliski algorithm in the literature. Erkay Savas [51] described a correction algorithm which allows fast calculation of an inverse element when input and output values are in Montgomery domain. Figure 4.3 is also from [23] and shows different ways to compute the Montgomery inversion, cf. [23]. This work uses the Kaliski algorithms to compute the inverse element in $\mathbb{F}_p$. The input value, which has to be inverted, is always in integer domain and the following operation is always a multiplication, which is provided with Montgomery's method. After the multiplication,

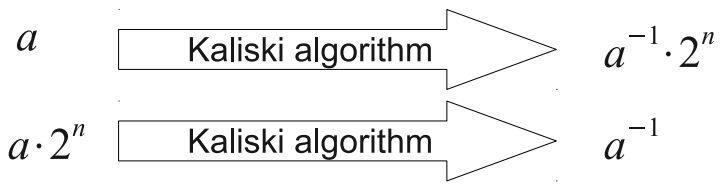

Figure 4.2: Input/output behavior for Kaliski algorithm [23].

Algorithm 4.15 Almost-Montgomery inverse (Kaliski phase one).

Require: prime p and $a \in [1, p-1]$
Ensure: $r\ in\ [1, p-1]$ and k, where $r = a^{-1} \cdot 2^k\ mod\ p$ and $n \leq k < 2n$
  $u \leftarrow p$, $v \leftarrow a$, $r \leftarrow 0$, $s \leftarrow 1$, $k \leftarrow 0$
  while $u > 0$ **do**
    if u is even then $u \leftarrow u/2$, $s \leftarrow 2s$
    else if v is even then $v \leftarrow v/2$, $r \leftarrow 2r$
    else if $u > v$ then $u \leftarrow (u-v)/2$, $r \leftarrow r+s$, $s \leftarrow 2s$
    else then $v \leftarrow (v-u)/2$, $s \leftarrow s+r$, $r \leftarrow 2r$
    $k \leftarrow k+1$
  end while
  $s \leftarrow r-p$
  if $r \geq p$ then **return** s and k
  else then **return** r and k

Algorithm 4.16 Kaliski phase two.

Require: p, $r \in [1, p-1]$ and k from AlmMonInv 4.15
Ensure: $a^{-1} \cdot R\ mod\ p$, where $a^{-1} \in [1, p-1]$ and $R = 2^n$
  for $i = 1$ to $k-n$ **do**
    if r is even then $r = r/2$
    else then $r = (r+p)/2$
  end for
  return r

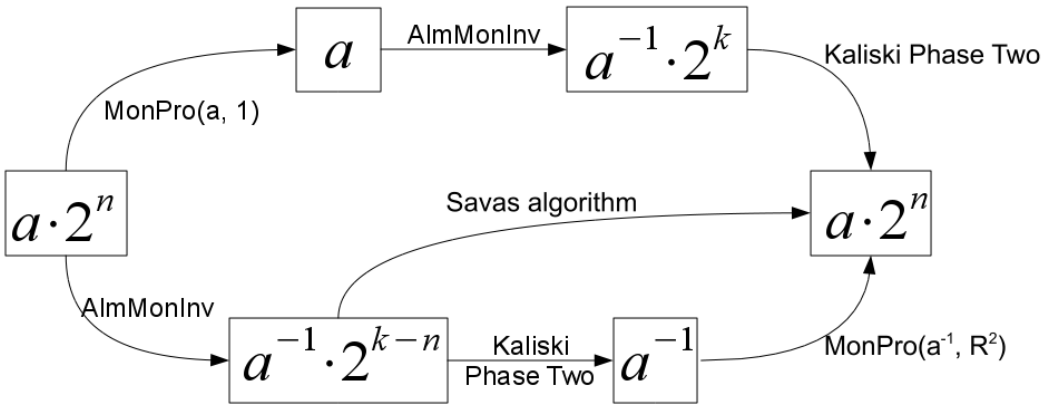

Figure 4.3: Different ways to compute the Montgomery inversion [23].

the result is available in integer domain, see Equation 4.8. Algorithm 4.15 and Algorithm 4.16 are modified to prevent SPA attacks. Dummy operations are insert to make in every loop the same sequence of operations. To prevent DPA attacks randomization is used. The integer a is multiplied with a random value r and then the inverse element of $(a \cdot r)^{-1} = a^{-1} \cdot r^{-1}$ is calculated. Finally the the random value r is removed by a second multiplication with $a^{-1} = a^{-1} \cdot r^{-1} \cdot r$.

4.2 Elliptic-Curve Arithmetic

The solutions of an equation, defined over $\mathbb{F}_p$, are the points at an elliptic curve E over the finite field $\mathbb{F}_p$. Depending on the finite field, there exist different forms of elliptic curves. The two big groups are curves defined over prime fields $E(\mathbb{F}_p)$ and curves defined over binary fields $E(\mathbb{F}_{2^m})$.

In a general form, an elliptic curve over a finite field $\mathbb{F}_q$ is defined by Equation 4.19, which is called *Weierstrass* equation. In this equation the coefficients $a_1, a_2, a_3, a_4, a_5, a_6$ are elements of $\mathbb{F}_q$ and $\Delta \neq 0$. Δ is called the discriminate of the elliptic curve, and [24] defined the discriminate of an elliptic curve E in *Weierstrass* form as following Equation 4.20.

$$E: \; y^2 + a_1 \cdot x \cdot y + a_3 \cdot y = x^3 + a_2 \cdot x^2 + a_4 \cdot x + a_6 \tag{4.19}$$

$$\begin{aligned} \Delta &= -d_2^2 \cdot d_8 - 8 \cdot d_4^3 - 27 \cdot d_6^2 + 9 \cdot d_2 \cdot d_4 \cdot d_6 \\ d_2 &= a_1^2 + 4 \cdot a_2 \\ d_4 &= 2 \cdot a_4 + a_1 \cdot a_3 \\ d_6 &= a_3^2 + 4 \cdot a_6 \\ d_8 &= a_1^2 \cdot a_6 + 4 \cdot a_2 \cdot a_6 - a_1 \cdot a_3 \cdot a_4 + a_2 \cdot a_3^2 - a_4^2 \end{aligned} \tag{4.20}$$

The *Weierstrass* equation, Equation 4.19 can be simplified considerably by applying an admissible change of variables. Depending on which field is used, a prime field $\mathbb{F}_p$ or a binary field $\mathbb{F}_{2^m}$, exist a simple *Weierstrass* equation, cf. [24].

- If the underlying field is a prime field $\mathbb{F}_p$, we can transform the elliptic curve E in *Weierstrass* form to the simple form shown in Equation 4.21. The coefficients a, b are elements of $\mathbb{F}_p$ and the discriminate change in this case to $\Delta = -16 \cdot \left(4 \cdot a^3 + 27 \cdot b^2\right)$.

$$y^2 = x^3 + a \cdot x + b \tag{4.21}$$

- If the underlying field is a binary field $\mathbb{F}_{2^m}$, there exits two possible forms of simple elliptic-curve equations, addicted from coefficient a_1. If $a_1 \neq 0$ we can transform the curve E to Equation 4.22, which is called *not supersingular*. The coefficients a, b are elements of $\mathbb{F}_{2^m}$ and the discriminate change to $\Delta = b$.

$$y^2 + x \cdot y = x^3 + a \cdot x^2 + b \tag{4.22}$$

  If $a_1 = 0$ we can transform the curve E to Equation 4.23, which is called *supersingular*. The coefficients a, b are elements of $\mathbb{F}_{2^m}$ and the discriminate change to $\Delta = c^4$.

$$y^2 + c \cdot y = x^3 + a \cdot x + b \tag{4.23}$$

All solutions from Equation 4.19 are points $P = (x, y)$ on the elliptic curve E for $x, y \in \mathbb{F}_q$. On every elliptic curve exist an additional point O, which is called the point of infinity. The number of points on the curve E, denoted $\#E(\mathbb{F}_q)$ is called the order of E. The *Hasse Theorem* 4.24 appraise the number of points on the curve and the interval $\left[q + 1 - 2 \cdot \sqrt{q}, q + 1 + 2 \cdot \sqrt{q}\right]$ is called the *Hasse interval*, cf. [8, 24].

$$q + 1 - 2 \cdot \sqrt{q} \leq \#E(\mathbb{F}_q) \leq q + 1 + 2 \cdot \sqrt{q} \tag{4.24}$$

The following Sections 4.2.1 and 4.2.2 only describe operations on elliptic curves defined over prime fields $\mathbb{F}_p$.

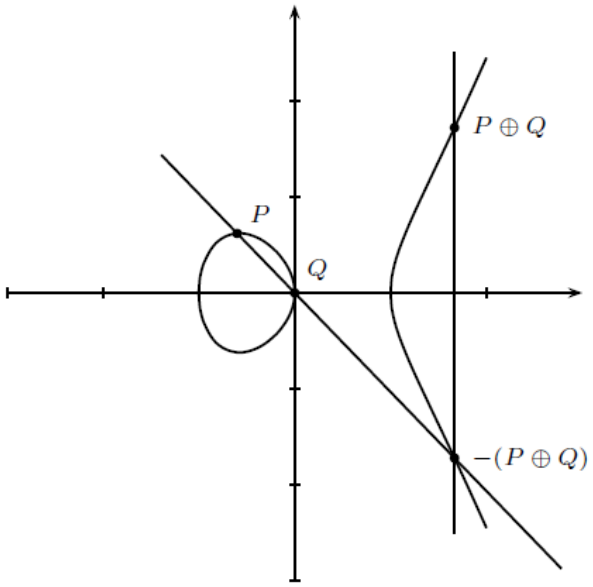

Figure 4.4: Geometric addition of EC points [12].

4.2.1 Curve Arithmetic

If E is an elliptic curve defined over the field $\mathbb{F}_p$, then the set of points $E(\mathbb{F}_p)$ together with the point of infinity O forms an abelian group. This group is used to construct the elliptic-curve cryptographic system. A simple way to explain the rules to add two points or double one point is a geometric approach. The following geometric description are from [24] and some other definitions are in [12].

> Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two distinct points on an elliptic curve. Then the sum R, of P and Q is defined as follows. First draw a line through P and Q; this line intersects the elliptic curve at a third point. Then R is the reflection of this point about the x-axis. The double R, of P, is defined as follow. First draw the tangent line to the elliptic curve at P. This line intersects the elliptic curve at a second point. Then R is the reflection of this point about the x-axes [1].

The addition operation is depicted in Figure 4.4, the double operation is depicted in Figure 4.5 and both pictures are from [12].

These formulas to add and double are presented next for elliptic curves over a prime field, e.g. [12, 24].

1. **Add the zero element to it self**: $O + O = O$
2. **Add the zero element to a point**: for all $P \in E(\mathbb{F}_p)$ $0 + P = P + 0 = P$
3. **Add a negative point**: a negative point is denoted by $-Q$. If $P(x, y)$ is a point in $E(\mathbb{F}_p)$ and $Q(x, -y)$ is the negative point to P then $P+Q = 0$ and $Q(x, -y) \in E(\mathbb{F}_p)$

[1] [24], site 79

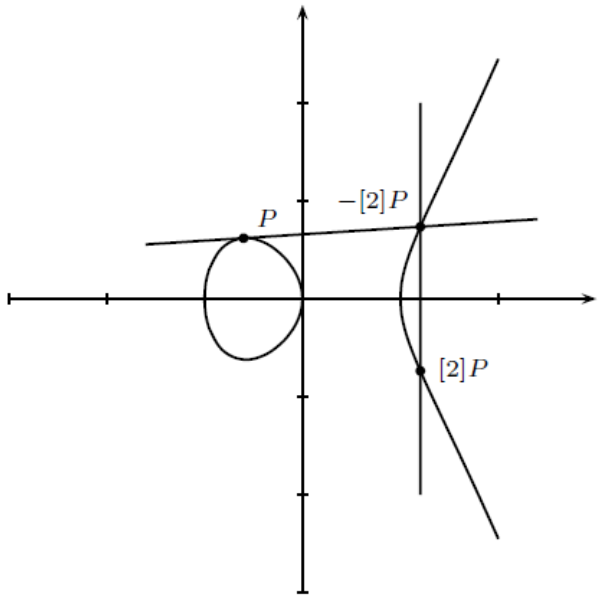

Figure 4.5: Geometric doubling of EC points [12].

4. **Add two points**: if P, Q are two points in $E(\mathbb{F}_p)$, denoted by $P(x_1, y_1)$ and $Q(x_2, y_2)$, and $P \neq Q$. The sum of $P+Q$ is $R(x_3, y_3$, where $x_3 = \left(\frac{y_2-y_1}{x_2-x1}\right)^2 - x_1 - x_2$ and $y_3 = \left(\frac{y_2-y_1}{x_2-x1}\right) \cdot (x_1 - x_3) - y_1$

5. **Double one point**: if P is a point in $E(\mathbb{F}_p)$, denoted by $P(x_1, y_1)$, and $P \neq -P$. The sum of $P + P$ is $S(x_4, y_4$, where $x_4 = \left(\frac{3 \cdot x_1^2 + a}{2 \cdot y_1}\right)^2 - 2 \cdot x_1$ and $y_4 = \left(\frac{3 \cdot x_1^2 + a}{2 \cdot y_1}\right)^2 \cdot (x_1 - x_4) - y_1$

These formulas for point addition and doubling are defined in affine coordinates. Both operations require a field inversion and several field multiplications. Since inversion in $\mathbb{F}_p$ is significantly more expensive than a multiplication it may be advantageous to represent points in a graticule where no inversion for add and double operations is required.

4.2.2 Point Representation

The formulas in Section 4.2.1 are defined in affine coordinates. These add and double operations can be calculated faster if some convenient coordinates are used. A point $P(x, y)$ in affine coordinates has a dimension of two and this can be denoted by $\mathbb{F}_p^2$. The same point can be represented by a different coordinate system which has for example a dimension of three $\mathbb{F}_p^3$. Now we can define an equivalence relation f, which transforms a point in $\mathbb{F}_p^2$ to a point in $\mathbb{F}_p^3 \setminus \{(0, 0, 0)\}$, where $\lambda \in \mathbb{F}_p$ and c and d are positive integers.

$$(X : Y : Z) = f(x, y) = \left\{\lambda^c \cdot x, \lambda^d \cdot y, \lambda \cdot 1\right\} \tag{4.25}$$

The point $(X : Y : Z)$ is called a projective point and the point (x, y) is called a representative point. The inverse transformation from $\mathbb{F}_p^3$ to $\mathbb{F}_p^2$ has two possible cases. All projective points with $Z = 0$ do not correspond to any affine point and so the set of this

points are called the line of infinity. For all points with $Z \neq 0$ exist an unique point in $\mathbb{F}_p^2$ and the inverse function f^{-1} is

$$(x, y) = f^{-1}(X, Y, Z) = \left\{ X/Z^c, Y/Z^d \right\} \tag{4.26}$$

To transform the *Weierstrass* equation, Equation 4.19 of an elliptic curve from affine form to projective form it is necessary to change x by X/Z^c and y by Y/Z^d and clearing denominators, cf [12, 24]. The following subsections present the two most commonly used coordinate transformations.

Standard-Projective Coordinates

This coordinate transformation uses $c = 1$ and $d = 1$. The projective point $(X : Y : Z)$ with $Z \neq 0$ corresponds to the affine point $(X/Z, Y/Z)$ and the point of infinity O corresponds to the projective point $(0, 1, 0)$. The negative point of $(X : Y : Z)$ is $(X : -Y : Z)$. The equation of an elliptic curve with standard-projective coordinates is described as follows.

$$Y^2 \cdot Z = X^3 + a \cdot X \cdot Z^2 + b \cdot Z^3 \tag{4.27}$$

Based on Equation 4.27 the formulas for add and double can be calculated as follow.

- **Point addition**: Let P and Q are two points in $\mathbb{F}_p^3$, denoted by $P\,(X_1 : Y_1 : Z_1)$ and $Q\,(X_2 : Y_2 : Z_2)$, and define that $P \neq \pm Q$. The sum of two points $P + Q = R\,(X_3 : Y_3 : Z_3)$ in standard-projective coordinates is defined in Equation 4.28. The mathematical description for point addition is from [12].

$$\begin{aligned} A &= Y_2 \cdot Z_1 - Y_1 \cdot Z_2 \\ B &= X_2 \cdot Z_1 - X_1 \cdot Z_2 \\ C &= A^2 \cdot Z_1 \cdot Z_2 - B^3 - 2 \cdot B^2 \cdot X_1 \cdot Z_2 \\ X_3 &= B \cdot C \\ Y_3 &= A \cdot \left(B^2 \cdot X_1 \cdot Z_2 - C\right) - B^3 \cdot Y_1 \cdot Z_2 \\ Z_3 &= B^3 \cdot Z_1 \cdot Z_2 \end{aligned} \tag{4.28}$$

- **Point doubling**: Let P a point in $\mathbb{F}_p^3$, denoted by $P\,(X_1 : Y_1 : Z_1)$, and define that $P \neq -P$. The sum of $P + P = S\,(X_4 : Y_4 : Z_4)$ in standard-projective coordinates is defined in Equation 4.29. The mathematical description for point doubling is from [12].

$$\begin{aligned} A &= a \cdot Z_1^2 + 3 \cdot X_1^2 \\ B &= Y_1 \cdot Z_1 \\ C &= X_1 \cdot Y_1 \cdot B \\ D &= A^2 - 8 \cdot C \\ X_4 &= 2 \cdot B \cdot C \\ Y_4 &= A \cdot (4 \cdot C - D) - 8 \cdot Y_1^2 \cdot B^2 \\ Z_4 &= 8 \cdot B^3 \end{aligned} \tag{4.29}$$

In comparison to add and double in affine coordinates no inversion is required. If one of the input points P or Q is given in affine coordinates, then the number of multiplications, needed for a point addition, decreases, cf. [12, 24].

Jacobian Projective Coordinates

This coordinate transformation uses $c = 2$ and $d = 3$. The projective point $(X : Y : Z)$ with $Z \neq 0$ corresponds to the affine point $\left(X/Z^2, Y/Z^3\right)$ and the point at infinity O corresponds to the projective point $(1 : 1 : 0)$. The negative point of $(X : Y : Z)$ is $(X : -Y : Z)$. The equation of an elliptic curve with Jacobian projective coordinates is described as follows.

$$Y^2 = X^3 + a \cdot X \cdot Z^4 + b \cdot Z^6 \tag{4.30}$$

Based on Equation 4.30 the formulas for addition and doubling can be calculated as follow.

- **Point addition**: This point addition formulas using mixed Jacobian - Affine coordinates. Let P and Q are two points in $\mathbb{F}_p^3$, denoted by $P\,(X_1 : Y_1 : Z_1)$ and $Q\,(X_2 : Y_2 : 1)$, and define that $P \neq \pm Q$. The sum of two points $P + Q = R\,(X_3 : Y_3 : Z_3)$ in Jacobian coordinates is defined in Equation 4.31. The mathematical description for point addition is from [24].

$$\begin{aligned} A &= \left(Y_2 \cdot Z_1^3 - Y_1\right) \\ X_3 &= A^2 - \left(X_2 \cdot Z_1^2 - X_1\right)^2 \cdot \left(X_1 + X_2 \cdot Z_1^2\right) \\ Y_3 &= A \cdot \left(X_1 \cdot \left(X_2 \cdot Z_1^2 - X_1\right)^2 - X_3\right) - Y_1 \cdot \left(X_2 \cdot Z_1^2 - X_1\right)^3 \\ Z_3 &= \left(X_2 \cdot Z_1^2 - X_1\right) \cdot Z_1 \end{aligned} \tag{4.31}$$

- **Point doubling**: This point doubling formulas using only Jacobian coordinates. Let P a point in $\mathbb{F}_p^3$, denoted by $P\,(X_1 : Y_1 : Z_1)$, and define that $P \neq -P$. The sum of $P + P = S\,(X_4 : Y_4 : Z_4)$ in Jacobian coordinates is defined in Equation 4.32. The mathematical description for point doubling is from [24].

$$\begin{aligned} X_4 &= \left(3 \cdot X_1^2 + a \cdot Z_1^4\right)^2 - 8 \cdot X_1 \cdot Y_1^2 \\ Y_4 &= \left(3 \cdot X_1^2 + a \cdot Z_1^4\right) \cdot \left(4 \cdot X_1 \cdot Y_1^2 - X_4\right) - 8 \cdot Y_1^4 \\ Z_4 &= 2 \cdot Y_1 \cdot Z_1 \end{aligned} \tag{4.32}$$

In this case no inversion is needed to compute addition of two points or the double of one point. For some special elliptic curves, the equation for point doubling 4.32 can be optimized. If $a = -3$, then the expression $\left(3 \cdot X_1^2 + a \cdot Z_1^4\right)$ can be calculated with only one field multiplication and one field squaring $\left(3 \cdot X_1^2 - 3 \cdot Z_1^4\right) = 3 \cdot \left(X_1 - Z_1^2\right) \cdot \left(X_1 + Z_1^2\right)$. Point doubling can be further more accelerated by using the fact that multiplications with 2, 4 or 8 can be done by several field additions, see [12, 24]. The elliptic curve defined in SECP160r1 has this special form. This improvement is used for point doubling in this thesis.

There are different field operation counts for point addition and doubling in various coordinate systems. Table 4.2 shows an overview about several add and double equations and uses the following notation. $C_1 + C_2 \leftarrow C_3$ means that the points which have to be added are in C_1 and C_2 coordinates, while their sum is is expressed in C_3 coordinates. The variables in Table 4.2 are defined as follow: A = Affine, P = standard projective, J = Jacobian, I = inversion, M = multiplication and S = squaring. If standard-projective coordinates or Jacobian coordinates are used, a transformation back to affine coordinates is required. This transformation requires one inverse and two multiplications for booth graticule and accessorily one squaring if Jacobian coordinates are used, e.g. [12, 24].

Table 4.2: Operation counts for point addition and doubling on $E(\mathbb{F}_p)$.

Doubling		Addition		Doubling with $a = -3$	
$2A \to A$	1I, 2M, 2S	$A + A \to A$	1I, 2M, 1S	$2A \to A$	1I, 2M, 2S
$2P \to P$	7M, 5S	$P + P \to P$	12M, 2S	$2P \to P$	7M, 3S
$2P \to P$	7M, 5S	$P + A \to P$	9M, 2S	$2P \to P$	7M, 3S
$2J \to J$	4M, 6S	$J + J \to J$	12M, 4S	$2J \to J$	4M, 4S
$2J \to J$	4M, 6S	$J + A \to J$	8M, 3S	$2J \to J$	4M, 4S

4.2.3 Scalar Multiplication

With elliptic-curve operations from Section 4.2.1 and Section 4.2.2 are only point addition and point doubling defined. Cryptographic operations are also need methods to compute $k \cdot P$, where k is an integer and P is a point on the elliptic curve $E(\mathbb{F}_p)$. This operation is called scalar multiplication, and the execute time depends strongly with this. The integer k is randomly selected from the interval $[1, n-1]$ and one possible representation is the binary representation of k. In this case $k = (k_{t-1}, k_{t-2}, \cdots, k_2, k_1, k_0)$, where $t \approx m = \lceil log_2 p \rceil$ cf. [12, 24]. There are many algorithms described in literature to perform the scalar multiplication. In this thesis only three algorithms are described, because those are the most interesting algorithms for this work.

Double-and-Add Algorithm

The left-to-right binary method, see Algorithm 4.17 is a standard method to perform scalar multiplication. This version is from [24] and is also called Double-and-Add algorithm. To calculate the scalar-point product an additive version of the basic repeated square and multiply method for exponentiation is used. The number of ones in the binary representation of k is $m/2$. Hence the expected running time of Algorithm 4.17 is approximately $m/2$ for point addition an m for point doubling, denoted $0.5mA + mD$. But the algorithm has some serious difficulties. The power consumption traces of a point addition operation and a point doubling operation are not the same, so an attacker can easily distinguish between these operations and derive the value k. This technique was described by Jean-Sebastien Coron [13] in 1999.

Algorithm 4.17 Double-and-Add algorithm for scalar multiplication.

Require: $k, P, t = \lceil log_2 k \rceil$
Ensure: $Q = k \cdot P$
  $Q \leftarrow P$
  for $i = t - 2$ to 0 **do**
    $Q \leftarrow ECDLB(Q)$
    if $k[i] = 1$ **then**
      $Q \leftarrow ECADD(Q, P)$
    end if
  end for
  return Q

Algorithm 4.18 Always Double-and-Add algorithm for scalar multiplication.

Require: $k, P, t = \lceil log_2 k \rceil$
Ensure: $Q = k \cdot P$
  $Q[0] \leftarrow P$
  for $i = t - 2$ to 0 **do**
    $Q[0] \leftarrow ECDLB(Q[0])$
    $Q[1] \leftarrow ECADD(Q[0], P)$
    $Q[0] \leftarrow Q\,[k[i]]$
  end for
  return $Q[0]$

Always Double-and-Add Algorithm

Coron described also an improved version of Algorithm 4.17 in his paper, see [13]. He used a technique which is called *Coron's dummy addition method*, which is one of the standard countermeasures against SPA. This algorithm is shown in Algorithm 4.18 which from [28] and which performs an *ECADD* and an *ECDBL* operation in every round. The runtime can appraise with $mA+mD$. So the always Double-and-Add algorithm, described by Coron is slower than the standard binary method described in Algorithm 4.17. The performance of this algorithm can be improved by using a combined addition and double operation (ECADDBL), which reuses some internal state variables.

Tetsuya Izu, Bodo M"oller and Tsuyoshi Takagi [28, 42] showed, that there exist a potential security problem in this method if we using projective coordinates. In the case that $k[i] = 0$, a point that is used in the current iteration is also an input point in the next iteration. *ECADD* and *ECDBL* involve squaring the Z coordinate so the same Z value will be squared again. Maybe a side channel can provide hints that the same squaring is performed again and consequently information in $k[i]$ leak out.

Montgomery Ladder Algorithm

Montgomery proposed in [44] some mathematical theories on elliptic curves which only work with the x-coordinate. These formulas were original described for the elliptic curve in Montgomery form and use the fact, that the sum of two points can be computed without the y-coordinate, if there differences is a known point. There exist some improvements for this technique to general elliptic-curves in Weierstrass form, e.g. [7, 20, 29]. The Montgomery ladder algorithm, see Algorithm 4.19 is from [7] and employ the Montgomery technique to calculate the scalar multiplication. Where P is a known point and $P = R[1] - R[0]$ throughout the algorithm.

In [7], Eric Brier and Marc Joye described the formulas for point doubling and addition with Montgomery technique and curves in Weierstrass form. If $\mathbb{F}_p$ is a prime field and the elliptic curve defined over the finite field is given by Equation 4.21, the formulas for double and add are from [7] and shown in the following itemization.

- **Point addition:** If P, Q are two points in $E(\mathbb{F}_p)$, denoted by $P(x_1, y_1)$ and $Q(x_2, y_2)$, and $P \neq Q$. The x-coordinate of the sum from $P + Q$ is (x_3 and the difference from $P - Q$ is (x, y) and is always known. Equation 4.33 is from [7] and shows the point

Algorithm 4.19 Montgomery ladder algorithm for scalar multiplication.

Require: $k, P, t = \lceil log_2 k \rceil$
Ensure: $Q_x = k \cdot P$
  $Q[0] \leftarrow P$, $Q[1] \leftarrow 2 \cdot P$
  for $i = t - 2$ to 0 **do**
    if $k[i] = 0$ **then**
      $Q[1]_x \leftarrow (R[0] + R[1])_x$, $Q[0]_x \leftarrow (2 \cdot R[0])_x$
    else
      $Q[0]_x \leftarrow (R[0] + R[1])_x$, $Q[1]_x \leftarrow (2 \cdot R[1])_x$
    end if
  end for
  return $Q[0]_x$

addition in affine coordinates.

$$(P + Q)_x = x_3 = \frac{-4 \cdot b \cdot (x_1 + x_2) + (x_1 \cdot x_2 - a)^2}{x \cdot (x_1 - x_2)^2} \tag{4.33}$$

There exists an additional equation to Equation 4.33 for point addition which is described in [28]

$$(P + Q)_x = x_3 = \frac{2 \cdot (x_1 + x_2) \cdot (x_1 \cdot x_2 + a) + 4 \cdot b}{(x_1 - x_2)^2} - x \tag{4.34}$$

- **Point doubling:** If P is a point in $E(\mathbb{F}_p)$, denoted by $P(x_1, y_1)$, $P \neq -P$ and $y_1 \neq 0$. The x-coordinate of the sum from $P + P$ is $S(x_4)$. Equation 4.35 is from [7] and shows the point doubling in affine coordinates.

$$(2 \cdot P)_x = x_4 = \frac{(x_1 2 - a)^2 - 8 \cdot b \cdot x_1}{4 \cdot \left(x_1^3 + a \cdot x_1 + b\right)} \tag{4.35}$$

If the y-coordinate of a point P is required, there exist an useful feature, described by Montgomery, to calculate it from the x-coordinate of P. This technique only required the x-coordinate of P, the x-coordinate of another point Q and the coordinates of the point $P - Q$. Equation 4.36 is from [7] and can be used to restore the y-coordinate. There are $P = (x_1, y_1)$ and $Q = (x_2, y_2) \in E(\mathbb{F}_p) \setminus \{0\}$ with $P \neq \pm Q$ and $P - Q = (x, y)$ with $y \neq 0$, cf. [7, 12, 24]

$$P_y = y_1 = \frac{2 \cdot b + (a + x \cdot x_1) \cdot (x + x_1) - x_2 \cdot (x - x_1)^2}{2 \cdot y} \tag{4.36}$$

This feature is not needed in this thesis because to generate an *ECDSA* digital signature only the x-coordinate of a point is required. The advantage of Algorithm 4.19 is that is insusceptible to SPA and so it is a good choice for this work.

The Montgomery ladder algorithm calculates in every round $P + Q$ and $2 \cdot P$. So some improvements of Equation 4.33, Equation 4.34 and Equation 4.35 are required. One possibility is to transform the equations from affine coordinates to projective coordinates.

In this case, no field inverse operation is required. Equation 4.37, Equation 4.38 and Equation 4.39 are from [7, 29] and illustrate that in standard-projective coordinates.

$$\frac{X_3}{Z_3} = \frac{Z}{X} \cdot \frac{(X_1 \cdot X_2 - a \cdot Z_1 \cdot Z_2)^2 - 4 \cdot b \cdot Z_1 \cdot Z_2 \cdot (X_1 \cdot Z_2 - X_2 \cdot Z_1)}{(X_1 \cdot Z_2 - X_2 \cdot Z_1)^2} \tag{4.37}$$

$$\frac{X_3}{Z_3} = \frac{2 \cdot (X_1 \cdot Z2 + X_2 \cdot Z_1) \cdot (X_1 \cdot X_2 + a \cdot Z_1 \cdot Z_2) + 4 \cdot b \cdot Z_1^2 \cdot Z_2^2}{(X_1 \cdot Z_2 - X_2 \cdot Z_1)^2} - \frac{X}{Z} \tag{4.38}$$

$$\frac{X_4}{Z_4} = \frac{\left(X_1^2 - a \cdot Z_1^2\right)^2 - 8 \cdot b \cdot X_1 \cdot Z_1^3}{4 \cdot \left(X_1 \cdot X_2 \cdot \left(X_1^2 + a \cdot Z_1^2\right) + b \cdot Z_1^4\right)} \tag{4.39}$$

$$\frac{X_4}{Z_4} = \frac{\left(X_1^2 \cdot Z_1^2 - a \cdot Z_1^2 \cdot Z_2^2\right)^2 - 8 \cdot b \cdot X_1 \cdot Z_1^3 \cdot Z_2^4}{4 \cdot Z_1 \cdot Z_2 \cdot \left(X_1 \cdot X_2 \cdot \left(X_1^2 \cdot Z_2^2 + a \cdot Z_1^2 \cdot Z_2^2\right) + b \cdot Z_1^3 \cdot Z_2^3\right)} \tag{4.40}$$

Tetsuya Izu and Tsuyoshi Takagi [29] presented in 2001 an improved point-doubling equation, see Equation 4.40, which share some intermediate variables of point addition. Therewith, they created a combined Double-and-Add operation (ECADDBL) which required $15M + 4S$ to compute $P + Q$ and $2 \cdot P$ and if $a = -3$ it reduces to $13M + 4S$. In 2002, see [28] they published an improved version of these ECADDBL operation, which is a little bit faster than the older version of [29]. That improved version only needs $13M+4S$ if $a \neq -3$ and $11M + 4S$ if $a = -3$. Both versions need only 7 auxiliary variables for one point Double-and-Add operation. But there are some hassles with that presented algorithm if we use it directly in hardware implementation where small area is required. A multiplication like $a = a \cdot b$ is no problem in software implementations but in hardware implementations an additional variable to save the result like $c = a \cdot b$ is needed. So the ECADDBL algorithm presented by Izu and Takagi needs 8 auxiliary variables under these requirement. Therefore, we used Equation, 4.38 and Equation 4.40 to create a new combined ECADDBL algorithm, which is smaller to implement in hardware and needs only 7 auxiliary variables. This algorithm is presented in Algorithm 4.20 and needs $11M + 6S$, if $a = -3$. A version for $a \neq -3$ is not created, because the elliptic curve, used in this work, is from *SECP160r1* and has $a = -3$.

Countermeasures against DPA

The always Double-and-Add algorithm, see Algorithm 4.18 and the Montgomery ladder algorithm, see Algorithm 4.19 are secure against SPA. But it is maybe possible to break it by using DPA. There exist some countermeasures against DPA, such as those by Jean-Sebastien Coron [13] or Marc Joye and Christophe Tymen [31]. For this ECDSA module the countermeasure described by Coron is used. The basic idea is to randomize the projective coordinates of the base point P. Let $P = (X : Y : Z)$ the base point described in *SECP160* given in standard-projective coordinates, then all $r \in \mathbb{F}_p \setminus \{0\}$,$(r \cdot X : r \cdot Y : r \cdot Z)$ represent the same point. Corons idea is to transform the base point from $(X : Y : Z)$ to $(r \cdot X : r \cdot Y : r \cdot Z)$ with a randomly selected r before starting the scalar multiplication. With this technique, the side-channel information available to the statistic analysis will be randomized. This countermeasure needs only two multiplications at the beginning of the scalar point multiplication. The countermeasure described by Joye and Tymen bases on the randomly selected isomorphisms between elliptic curves. For more information about that look at [31].

Algorithm 4.20 Own improved combined Double-and-Add algorithm.

Require: $P = (X_1, Z_1)\ Q = (X_2, Z_2)\ P = (x, y)\ b$
Ensure: $X_3, Z_3 = P + Q \quad X_4, Z_4 = 2 \cdot P$

$REG_1 = X_1;\ REG_2 = Z_1;\ REG_3 = X_2;\ REG_4 = Z_2$
$REG_5 \leftarrow REG_1 \cdot REG_4$
$REG_6 \leftarrow REG_3 \cdot REG_2$
$REG_7 \leftarrow REG_1 \cdot REG_3$
$REG_1 \leftarrow REG_2 \cdot REG_4$
$REG_2 \leftarrow REG_5 - REG_6$
$Z_3 \leftarrow REG_2^2$
$REG_2 \leftarrow REG_7 - REG_1$
$REG_2 \leftarrow REG_2 - REG_1$
$REG_2 \leftarrow REG_2 - REG_1$
$REG_4 \leftarrow REG_5 + REG_6$
$REG_7 \leftarrow REG_2 \cdot REG_4$
$REG_6 \leftarrow REG_1^2$
$REG_4 \leftarrow REG_6 \cdot b$
$REG_7 \leftarrow REG_7 + REG_4$
$REG_7 \leftarrow REG_7 + REG_7$
$REG_7 \leftarrow REG_7 + REG_4$
$REG_7 \leftarrow REG_7 + REG_4$
$REG_2 \leftarrow Z_3 \cdot x$
$X_3 \leftarrow REG_7 - REG_2$
$REG_2 \leftarrow REG_5^2$
$REG_2 \leftarrow REG_2 - REG_6$
$REG_2 \leftarrow REG_2 - REG_6$
$REG_2 \leftarrow REG_2 - REG_6$
$REG_6 \leftarrow REG_2 \cdot REG_5$
$REG_2 \leftarrow REG_4 \cdot REG_1$
$REG_6 \leftarrow REG_6 + REG_2$
$REG_4 \leftarrow REG_1 \cdot REG_6$
$REG_4 \leftarrow REG_4 + REG_4$
$Z_4 \leftarrow REG_4 + REG_4$
$REG_6 \leftarrow REG_2 \cdot REG_5$
$REG_6 \leftarrow REG_6 + REG_6$
$REG_6 \leftarrow REG_6 + REG_6$
$REG_6 \leftarrow REG_6 + REG_6$
$REG_2 \leftarrow REG_5^2$
$REG_5 \leftarrow REG_1^2$
$REG_2 \leftarrow REG_2 + REG_5$
$REG_2 \leftarrow REG_2 + REG_5$
$REG_2 \leftarrow REG_2 + REG_5$
$REG_1 \leftarrow REG_2^2$
$X_4 \leftarrow REG_1 - REG_6$
return $X_3, Z_3\ X_4, Z_4$

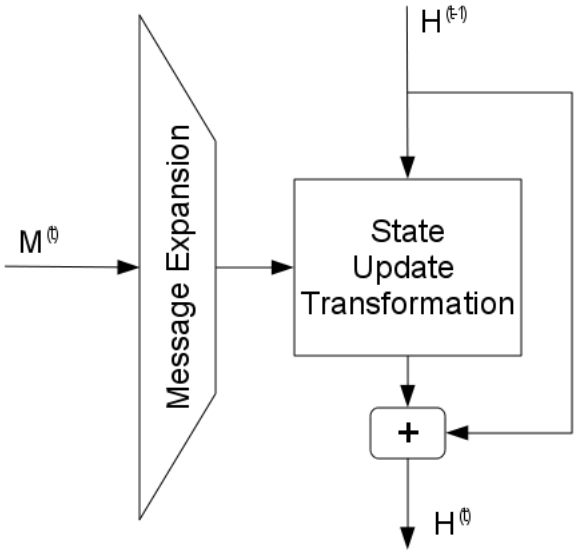

Figure 4.6: Overview of the SHA-1 compression function.

4.3 SHA-1

The ECDSA digital-signature algorithm needs also a hash function to compute the fingerprint of a message. There exist a lot of different hash functions, but the most current used hash functions are from the SHA family. Table 4.3 shows some primary information about SHA family and the informations are from [19].

Since a *SECP160* elliptic curve is used, it makes sense to use SHA-1 with a 160-bit hash value. The following section gives a short description of SHA-1.

At the first, the input message M has to be padded and split into message blocks M^t with a size of 512 bits. Suppose that the length of the input message M is l bits, a padded message block can be create as follow. Append the bit 1 followed by k zero bits to the end of the message M. Where k is the smallest non negative solution to the equation $l + 1 + k = 448 \; mod \; 512$. At last, append a 64-bit block to the end, which includes the number l by using a binary representation. All message blocks M^t have now a length of 512 bits. The actual hash function can be divided into two parts, the message expansion and the state update transformation. Figure 4.6 gives an overview of these two parts.

The message expansion can be defined as follow. A message block M^t can be split into 16 32-bit words, denoted by M_i, with $0 \leq i \leq 15$. The message expansion function expanded the message block M^t linearly into 80×32-bit words W_i and this function is

Table 4.3: Secure hash algorithm properties [19].

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)	Security (bits)
SHA-1	$< 2^{64}$	512	32	160	80
SHA-256	$< 2^{64}$	512	32	256	128
SHA-384	$< 2^{128}$	1 024	64	384	192
SHA-512	$< 2^{128}$	1 024	64	512	256

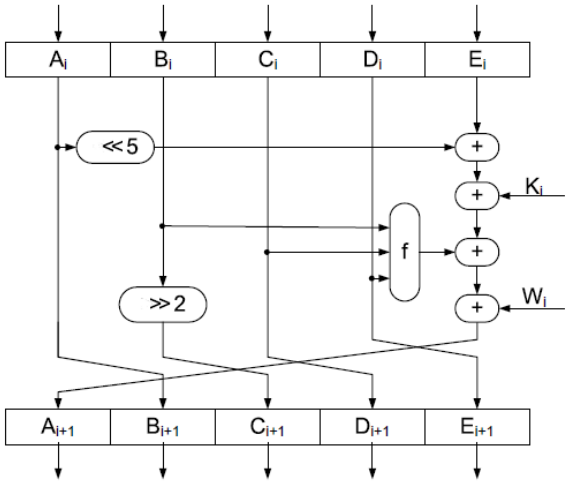

Figure 4.7: One step of the state update transformation [15].

described in Equation 4.41.

$$W_i = \begin{cases} M_i & \text{for } 0 \leq i \leq 15 \\ (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) << 1 & \text{for } 16 \leq i \leq 79 \end{cases} \tag{4.41}$$

The state update transformation starts by copying the last hash values H^{t-1} into the five 32-bit state variables $A, \cdots, E$. If the first message block is processed, initial hash values are used. For SHA-1, the initial hash-values $H^{(0)}$ are constants of the following five 32-bit words, shown in Equation 4.43.

$$\begin{aligned} H_0^{(0)} &= \text{0x67452301} \\ H_1^{(0)} &= \text{0x}efcdab89 \\ H_2^{(0)} &= \text{0x}98badcfe \\ H_3^{(0)} &= \text{0x10325476} \\ H_4^{(0)} &= \text{0x}c3d2e1f0 \end{aligned} \tag{4.42}$$

These state variables are updated in 80 ($0 \leq i \leq 79$) steps afterwards, by using the expanded message W_i and a round constant K_i in step i. A single step of the state update transformation is shown in Figure 4.7 and described in Algorithm 4.21. The function f in Figure 4.7 depends on the step number and is defined in Equation 4.44. The round constants K_i are defined in Equation 4.45, which uses a hexadecimal number system. After 80 state update transformations the state variables are added with the last hash values H^{t-1} and then the processing of one message block is complete.

$$H^t_{(0,\cdots,4)} = \left(A^t, \cdots, E^t\right) + H^{t-1}_{(0,\cdots,4)} \tag{4.43}$$

Algorithm 4.21 SHA-1 state update transformation.

Require: $A^{t-1}, \cdots, E^{t-1}$
Ensure: $A^t, \cdots, E^t$
  for $i = 0$ to 79 **do**
    $T \leftarrow ROTL_5(A) + f_i(B, C, D) + E + K_i + W_i$
    $E \leftarrow D$; $D \leftarrow C$
    $C \leftarrow ROTR_2(B)$
    $B \leftarrow A$; $A \leftarrow T$
  end for
  return $A^t, \cdots, E^t$

$$f_i = \begin{cases} (B \wedge C) \oplus (\neg B \wedge D) & 0 \leq i \leq 19 \\ (B \oplus C \oplus D) & 20 \leq i \leq 39 \\ (B \wedge C) \oplus (B \wedge D) \oplus (C \wedge D) & 40 \leq i \leq 59 \\ (B \oplus C \oplus D) & 60 \leq i \leq 79 \end{cases} \tag{4.44}$$

$$K_i = \begin{cases} 0\text{x}5a827999 & 0 \leq i \leq 19 \\ 0\text{x}6ed9eba1 & 20 \leq i \leq 39 \\ 0\text{x}8f1bbcdc & 40 \leq i \leq 59 \\ 0\text{x}ca62c1d6 & 60 \leq i \leq 79 \end{cases} \tag{4.45}$$

After processing all padded message blocks M^t, the resulting 160-bit message digest, of an input message M is the value

$$H_0^t | H_1^t | H_2^t | H_3^t | H_4^t \tag{4.46}$$

For more information about SHA-1 see [19]. The SHA-1 module which is used in this work, only handles with one padded message block. The padding must be done externally. A message with 447 bits without padding is long enough for an RFID tag with ECDSA to ensure a secure identification. An extension to process more than one message block can be done but is not provided yet.

Chapter 5

Implementation of the ECDSA Hardware Module

The aim of this chapter is to design an elliptic-curve processor with full ECDSA functionality and optimization for low-area and low-power consumption. Such a module can be used to implement secure RFID tags, smartcards, or other devices. This chapter gives an overview of a hardware ECDSA design in the first section. The following sections give details about all components which we need in our module. We also describe some measures for low-power optimization and measures to prevent Simple Power Analysis (SPA) and Differential Power Analysis (DPA) attacks. The results of our implementation are given in the last section.

5.1 Overview

The whole design process can be split into two parts. In the first part, which we describe only briefly, is built a high-level model of the elliptic-curve processor in a language like *JAVA* or *C++*. This high-level model describes the elliptic-curve processor at the system or architecture level. An object-oriented program language is used to describe all parts

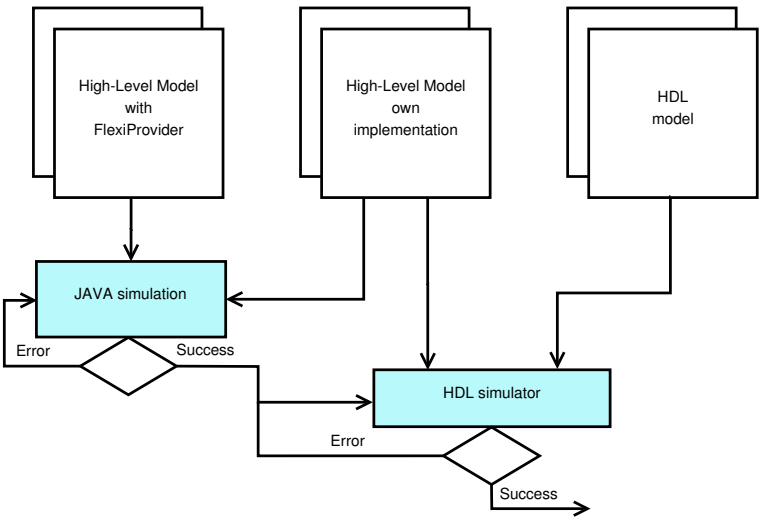

Figure 5.1: Verification process between different models.

of the ECDSA module, like memory, arithmetic-logic unit (ALU) or the controller. The segmenting into smaller parts makes the comparing of different implementations easy. We implemented different algorithms which perform the same operation to analyze them on reference to memory and run-time requirements. The *FlexiProvider* Java Cryptography Architecture (JCA/JCE), see [9], is used as origin to generate test vectors. The testing of our implementation is a two step approach. In the first step, we use the test vectors, generated with *FlexiProvider*, to verify our own high-level software model. In a second step, we generate new test vectors with our own software implementation to verify the hardware-description language (HDL) model. This approach has the advantage, that it is possible to produce also test vectors for sub-modules, like random-access memory (RAM) or the ALU. Figure 5.1 shows the relations, between the two software models and the hardware model.

In the second part, we describe the hardware architecture of our elliptic-curve processor. The complete hardware architecture is an application-specific integrated circuit (ASIC) design with digital standard-cells from *c35b4* CMOS libraries published by Austriamicrosystem AG [2]. We can divide the architecture of our ECDSA module into four parts. These sub-modules and the connections between them are shown in Figure 5.2.

- **AMBA**: The ECDSA module provides an AMBA APB interface [39] as interconnection to other modules. The interface offers all ports from the AMBA specification. Section 5.2 gives detailed information about the AMBA interface.
- **Memory**: The memory sub-module includes a RAM module, a Read-only memory (ROM) module and an Electrically Erasable Programmable Read-Only Memory

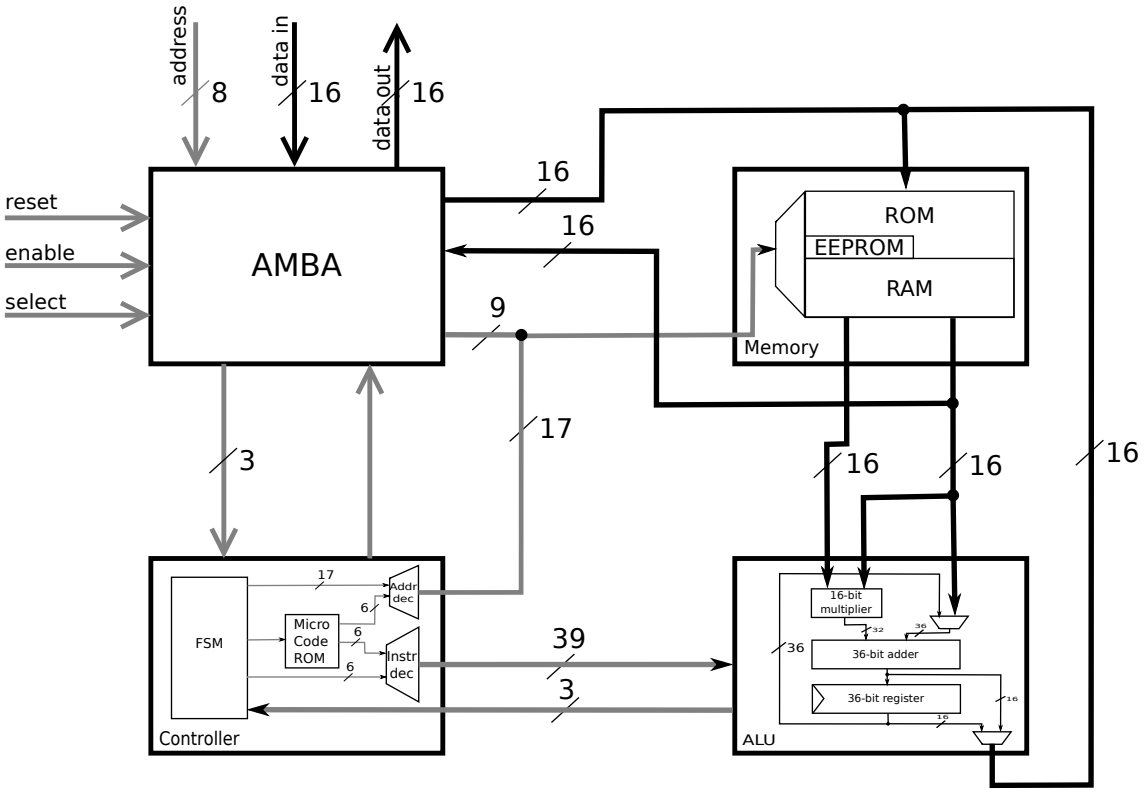

Figure 5.2: Elliptic-curve processor architecture.

(EEPROM) module. It offers a 16-bit dual-port interface to read data operations, a 16-bit single-port interface to write data operations, and it has an address space of 8 bits. Section 5.3 shows detailed information about the memory sub-module and compares different implementations.

- **ALU**: The ALU sub-module includes a register, a multiplication and addition unit and some logical elements. Section 5.4 provides detailed information about the multiply-accumulate unit because it is the most important part in the ALU sub-module.
- **Controller**: The control unit includes all algorithms which we need to generate digital signatures using ECDSA. It has a so called *Hybrid* design, because some parts of the control unit are implemented in the finite-state machine (FSM) approach and some other parts in a micro-code control architecture. Section 5.5 gives detail information about the FSM part and the micro-coded parts.

5.2 APB Interface

The interconnection to other modules or components is done by an AMBA APB interface. The Advanced Microcontroller Bus Architecture (AMBA) is published by ARM Ltd. and specifies different on-chip communication standards, like Advanced High-performance Bus (ABH), Advanced System Bus (ASB) and Advanced Peripheral Bus (APB). For this work the APB is used, because it is optimized for low-power consumption and has a simple interface. Figure 5.3 is from [39] and shows the APB slave interface, which has been implemented in our ECDSA module. Figure 5.4 shows three states, which we need for communication over the AMBA APB bus. In fact, the communication process comprises three states:

- **IDLE**: The system is in the IDLE state, if no communication over the bus is required.
- **SETUP**: The bus goes into the SETUP state if a communication is required. In this state, the module is selected *(PSELx)* and the address is set using the address line *(PADDR)*. In order to select read or write access the *(PWRITE)* line is used. After one clock cycle, the state of the bus is ENABLE.

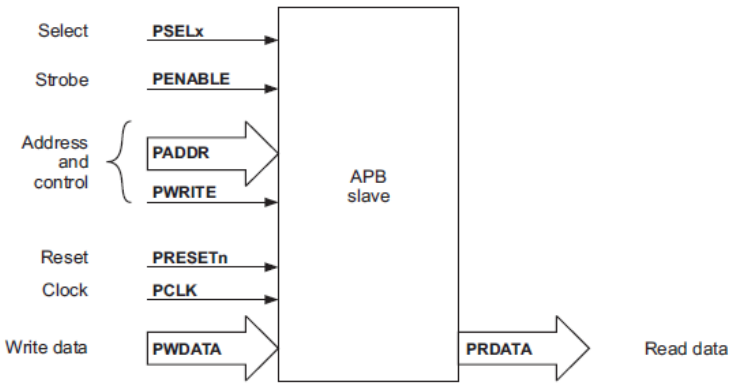

Figure 5.3: APB slave interface description [39].

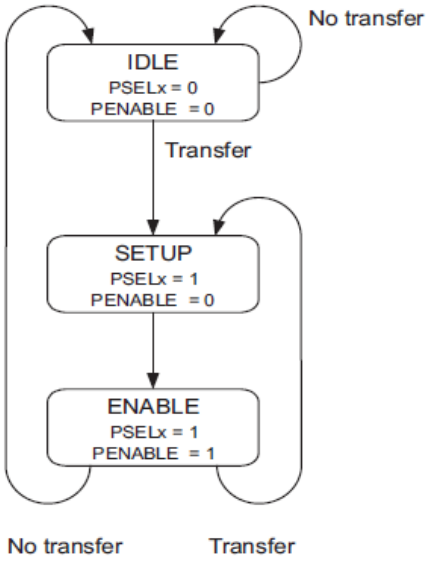

Figure 5.4: APB state diagram [39].

- **ENABLE**: This is the data transfer state. All signals from SETUP state have to be stable and the *(PENABLE)* signal goes to high. The read or write operation is done after the next clock cycle and the bus goes back to the SETUP state if an other transfer follows or it goes to the IDLE state otherwise.

Each port to read and write data (*(PRDATA)* or *(PWDATA)*) has a size of 16 bits and the address port has 8 bits in our APB implementation. Table 5.1 shows the specification for memory-mapped I\O. The padded message has to be load in blocks of 128 bits into the memory. The ephemeral key and the random value for randomized projective coordinates have to be loaded to the correct addresses and afterwards the signature generation can be started. Before the signature generation is finished, a second randomization value is required. This second random value is used to randomize the ephemeral key and assure the Montgomery inversion algorithm against SPA and DPA attacks. This randomization requires less clock cycles then a time-invariant implementation of Montgomery inversion algorithm. If the *FINISH* signal goes to high, the values r and s can be read from memory. The *PRESETn* line is specified by AMBA APB specification as active low reset signal and in our design a signal change is only allowed at the positive edge of the clock signal *PCLK*. Figure 5.5 and Figure 5.6 show the timing diagrams of a read operation and a write operation.

5.3 Memory

The memory sub-module contains three different storage elements. A RAM is used to store intermediate values, a ROM table which contains constant values such as elliptic-curve parameters or initial hash-values to calculate the SHA-1 fingerprint, and an EEPROM module that stores the private static key. The interface is equal to a dual-port RAM but with some simplifications. Each port in a generic dual-port RAM can be used to write or read data from any memory location. In our algorithmic design for elliptic-curve operations, a second write port brings no speed up. Hence, our memory implementation

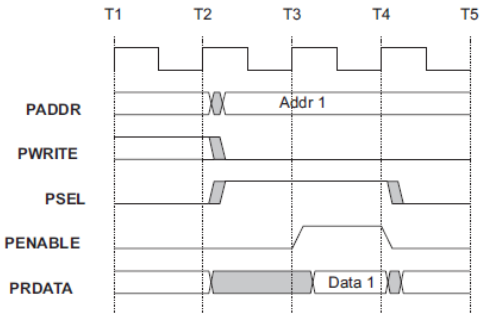

Figure 5.5: APB read operation [39].

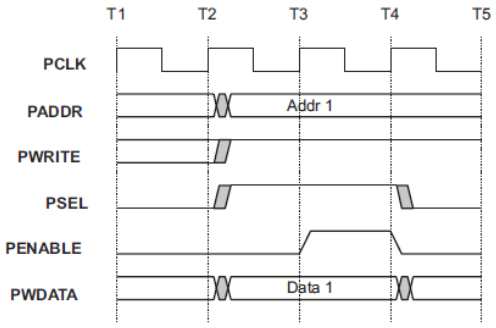

Figure 5.6: APB write operation [39].

includes only one write port to reduce the required area and power consumption. The 'Port A' of our memory module has only read access and the write operations can only be done on the address of 'Port B'. The RAM also has a dual-port design with only one write port and has the same port specification as the memory sub-module. The EEPROM module is connected to memory 'Port A' and the ROM table is split into two parts and each part is connected to 'Port A' or to 'Port B', but not to both. The memory module includes 1 440 bits of RAM, 1 328 bits of ROM, and a 160-bit EEPROM module. Figure 5.7 shows the architecture of the memory sub-module. Figure 5.8 shows the address space of the memory sub-module. The memory address is split into two parts. The higher 4 bits of the address are used to select a 160-bit or a 176-bit word and the lower 4-bit of the address are used to select the 16-bit element from the word. Each port has a

Table 5.1: APB memory mapped I\O specification.

Address	**Read**	**Write**
$0x00 \cdots 0x0A$	ECDSA signature value r	first 128-bit block of the padded message
$0x10 \cdots 0x1A$	ECDSA signature value s	second 128-bit block of the padded message
$0x20 \cdots 0x29$		third 128-bit block of the padded message
$0x30 \cdots 0x39$		fourth 128-bit block of the padded message
$0x50 \cdots 0x59$		value for randomized projective coordinates (160 bits)
$0x60 \cdots 0x69$		random value for Montgomery inversion (160 bits)
$0x70 \cdots 0x79$		ephemeral ECDSA key
$0x90 \cdots 0x99$		EEPROM static private ECDSA key
$0xA0$	$0x00$ - ECDSA signature creation run $0x01$ - finish $0x02$ - get new random at address $0x60 \cdots 0x69$	$0x01$ - start ECDSA signature generation $0x02$ - restart after new random insert $0x03$ - make soft reset

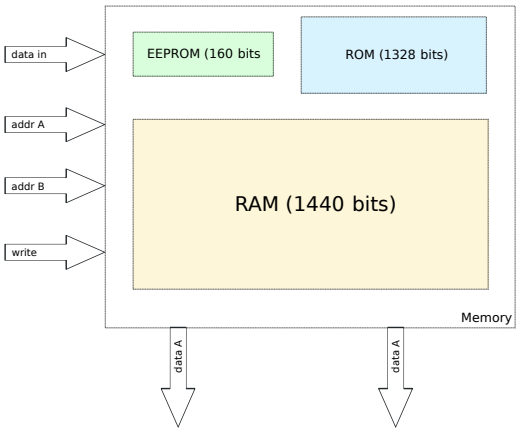

Figure 5.7: Memory overview.

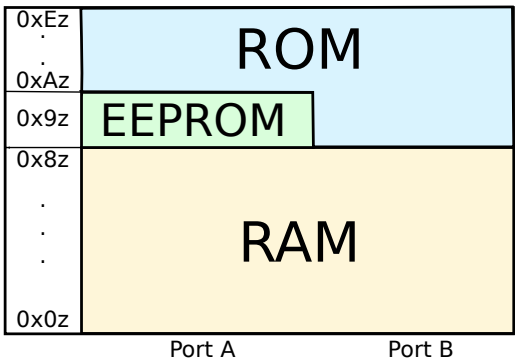

Figure 5.8: Memory address space.

Table 5.2: ROM address space.

Address	**'Port A'**	**'Port B'**
$0x90 \cdots 0x99$		EC parameter b
$0xA0 \cdots 0xAA$	EC parameter p	Montgomery multiplication factor R
$0xB0 \cdots 0xB9$	EC basepoint	DPL(EC basepoint)
$0xC0 \cdots 0xCA$	EC parameter n	
$0xD0 \cdots 0xD9$	SHA-1 initial hash-value $H^{(0)}$	
$0xE0 \cdots 0xE9$	SHA-1 round constant K_i	

size of 16 bits and consequently the smallest memory element has also a size of 16 bits. The elliptic-curve operations require seven RAM entries with 10×16-bit elements and additional two 10×16-bit elements are required to store the ephemeral key and the hash fingerprint. Therefore, we need nine entries with 160 bits (1 440 bits) to calculate the scalar multiplication. The signature values r and s are calculated in a prime field with the size of 161 bits. Therefore, we need entries with 11×16-bit elements, but the signature-generation step requires only eight entries with 176 bits. In order to make the memory as small as possible, we implement our RAM with two different address spaces. One 160-bit word is shared to implement either nine 160-bit entries or eight 176-bit entries. The description above shows only the addressing of the RAM module. In Subsection 5.3.1, we give some information about different RAM design, like an ASIC design in comparison to a RAM macro-module, or synchronous read operations in comparison to asynchronous read operations.

The ROM table includes five entries with 160 bits and three entries with 176 bits. Table 5.2 shows the ROM-table entries sorted by port and address.

5.3.1 RAM

There exist different types of dual-port RAM architectures in the literature. From the algorithmic aspect, we need at least 1 440 bits of RAM to generate an ECDSA digital signature. One possibility is to use a synchronous dual-port RAM macro, such as published by Austriamicrosystem AG [2]. However, there exist no RAM macro from Austriami-

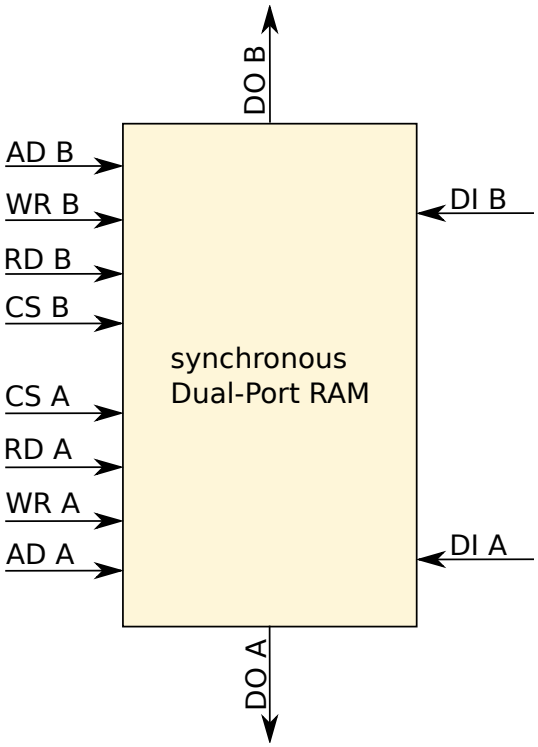

Figure 5.9: Synchronous dual-port RAM schematic (taken from Austriamicrosystem AG [2]).

crosystem AG which has exact 1 440 bits, consequently we had to use the next bigger one, which has a size of 2 048 bits. Figure 5.9 shows the schematic of an Austriamicrosystem AG dual-port RAM. This synchronous RAM needs two read lines (RDx) to enable a read operation and two write lines (WRx) to enable a write operation. The lines for addressing and data have the same sizes as we described above. Therefore, we need a 19-bit control word for this dual-port RAM, under the acceptance that we only use one port for write operations. This RAM architecture needs 8 730 GEs of area [2] and one ECDSA signature-generation process needs approximately 707 000 clock cycles. The number of clock cycles is a result from our JAVA high-level model.

Another way is to design an ASIC RAM with digital standard-cells. For that, we use cells from *c35b4* CMOS library [2]. Figure 5.10 shows an ordinary memory element with a size of 16 bits. The D-Type Flip Flop has the value from input 'd' and holds it at output 'q', during every positive clock edge. A Multiplexer is used to choose, whether a new value should be stored or the last value should be back coupling. This basic memory design needs 16 D-Type Flip Flops and 16 Multiplexers to build a 16-bit memory element. However, this is consequently no good solution for a low-area design. One possible optimization for this basic memory element is shown in Figure 5.11. The 16 Multiplexers are removed and instead a clock-gating cell is insert. This memory architecture has several advantages. First, the area which is required decreases because one clock-gating cell requires less area then 16 Multiplexers. Second, this design needs less power than the ordinary memory element, because the D-Type Flip Flops only change its state if a new information has to be stored and not in every clock cycle. More information about glock gating is given in Section 5.7. If we have a single 16-bit memory element, the design shown in Figure

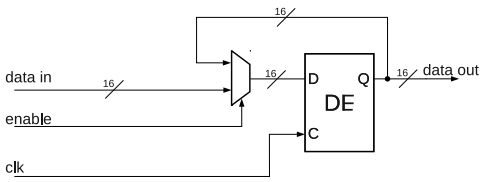

Figure 5.10: Ordinary 16-bit RAM element with D-Type Flip Flop.

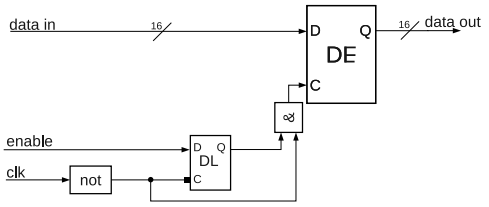

Figure 5.11: Clock gated 16-bit RAM element with D-Type Flip Flop.

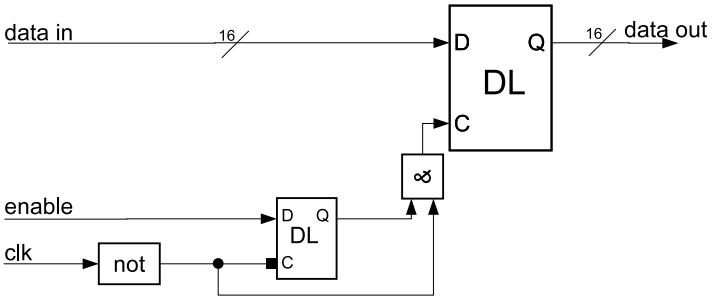

Figure 5.12: Latch-based 16-bit RAM element.

5.11 is a good choice. But for our 1 440-bit RAM we need 90×16-bit memory elements and therefore some more optimizations are possible. In our work, we used a latch-based RAM to further optimize the area of our ECDSA module. We adopted the design from Figure 5.11 and replaced the D-Type flip flop by a latch. But to implement a RAM with synchronous write capability, some more changes were required. Figure 5.12 shows one 16-bit memory element with a latch as storage element. The 1 440-bit RAM consists of 90×16-bit latch-based storage elements. We had to insert two additional 16-bit latch-based storage elements at the bottom of the two read ports, to get an asynchronous read and a synchronous write capability. The full latch-based RAM architecture is shown in Figure 5.13.

Table 5.3, Figure 5.14 and Figure 5.15 give an outline about this different RAM designs. The synchronous dual-port RAM from Austriamicrosystem AG requires the least area, but if we compare the product of area and clock cycles, the latch-based RAM is the best solution. Therefore, we used this RAM design for our ECDSA module.

Table 5.3: Comparison of different RAM architectures

RAM type	**Area [#GE]**	**Time [kCycles]**	**Area · Time**
Synchronous Dual-Port RAM macro	8 730	707	6.17
Ordinary D-Type Flip Flop RAM	13 100	512	6.70
Clock gated D-Type Flip Flop RAM	12 050	512	6.16
Latch-based RAM	10 300	512	5.27

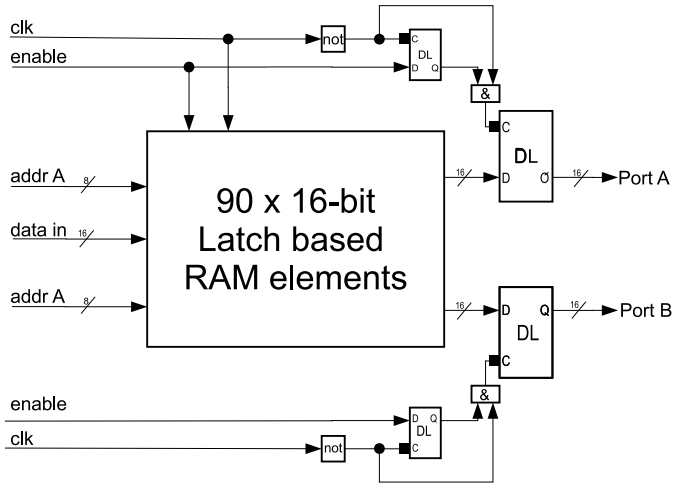

Figure 5.13: Asynchronous read / synchronous write latch-based RAM.

5.4 Arithmetic Logic Unit (ALU)

The most important component of the ALU is the 16 × 16-bit multiply-accumulate unit. This multiply-accumulate unit is used for Montgomery multiplication and multiplication with implicit fast reduction. Also a 16-bit adder for addition and subtraction operations are needed. Some logical elements are needed for SHA-1 computation. The 16-bit adder is shared with the multiply-accumulate unit to make a design which is optimized for area. In order to store some intermediate values, we used a 36-bit accumulator.

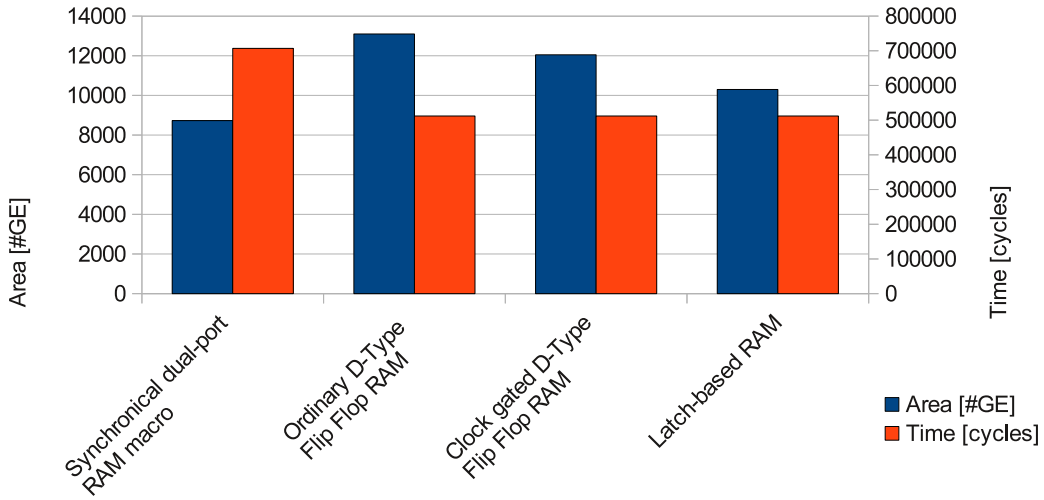

Figure 5.14: Comparison of different RAM architectures - area and time requirements.

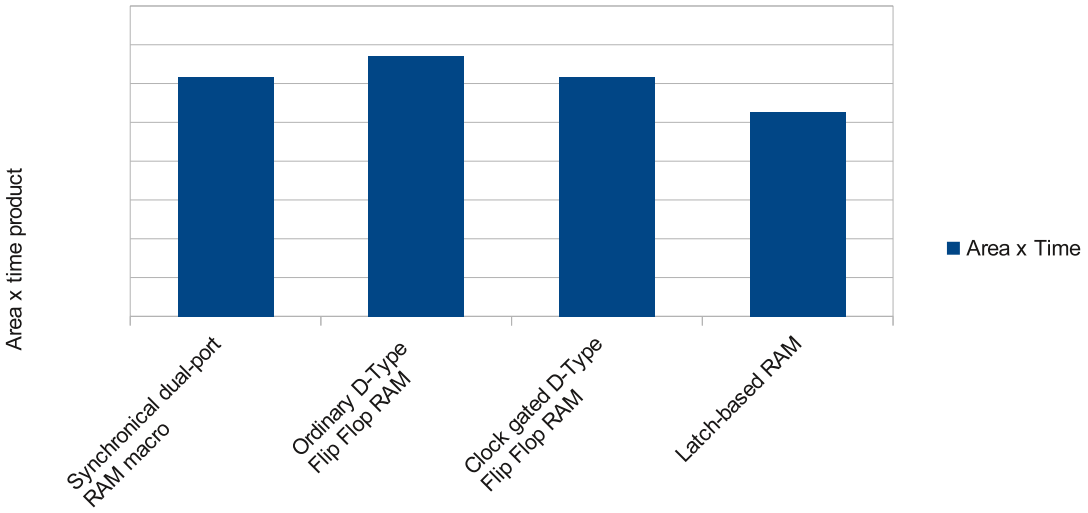

Figure 5.15: Comparison of different RAM architectures - area/time product.

5.4.1 Multiplication

The multiply-accumulate unit is the core component of the ALU. This component consists of a 16 × 16-bit multiplier, a 36-bit adder, and a 36-bit register. Figure 5.16 shows the block diagram of the multiply-accumulate unit. The multiply-accumulate unit multiplies two 16-bit values and adds the multiplication result to the values actually stored in the registers. The 36-bit adder consists of two 16-bit adder and one 4-bit adder, because now we can use sleep logic if only one 16-bit adder is required. This multiplication and addition operation is done within one clock cycle. To get a multiply-accumulate unit with

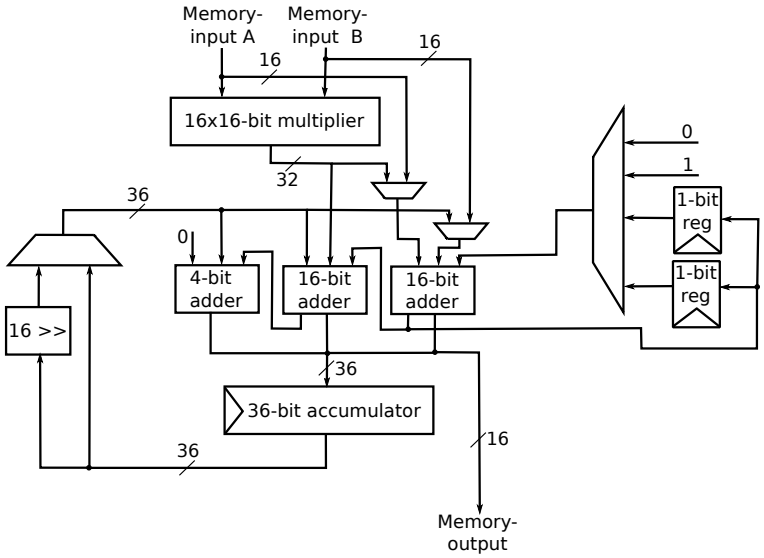

Figure 5.16: Multiply-accumulate unit.

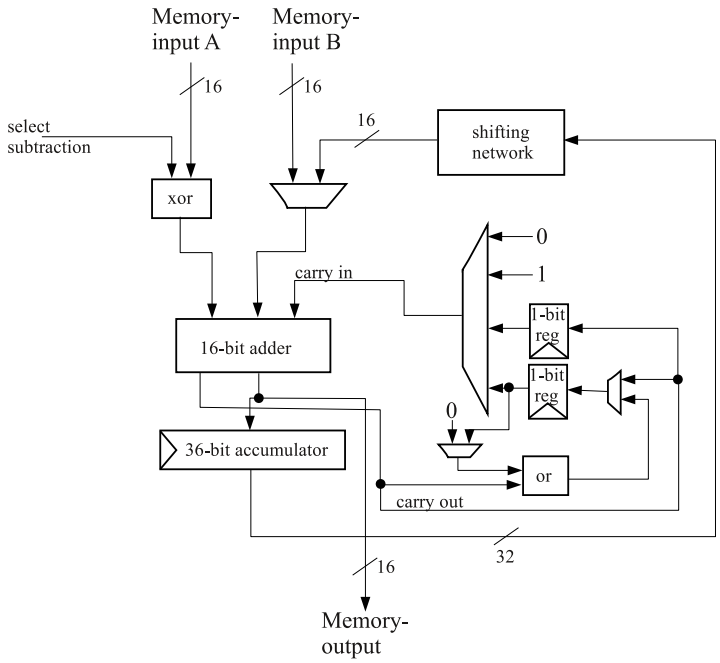

Figure 5.17: 16-bit adder unit.

low-power consumption sleep logic is used to switch off the multiplier and the adder to reduce switching activity. We implemented the multiplier with the *Verilog* * operation and consequently the synthesis is done by the Cadence-design tools.

5.4.2 Addition

The 36-bit adder from the multiply-accumulate unit is also used for simple addition and subtraction operations in two different finite fields. To reduce chip area and to fulfill the low-power requirements, we used only the first 16-bit adder unit for simple addition and subtraction operations. The other two adder units, see Figure 5.16, can be switched off by a sleep logic, which reduces the switching activity. A subtraction operation is required to calculate the negative value of b, cf. Algorithm 4.3. This negative value is called *two's complement* and it is calculated by a XOR operation of b and $0xFFFF$ and an addition of 1 afterwards. In order to implement the multiplication with implicit fast reduction, see Algorithm 4.8, additionally an interleaved addition operation and some shift operations are needed. Figure 5.17 shows the 16-bit adder unit with carry save registers in a block diagram. The shift operations are done by a fixed wired network using multiplexers.

5.4.3 Other Elements

A few additional components are needed to calculate the ECDSA digital signature. The SHA-1 calculation requires two logical binary operation, a 16-bit XOR operation and a 16-bit AND operation and three bit-wise rotation operations. This rotation operations are also done by a fix wired network. We also implemented a 8-bit shift register, which stores the actual needed 8-bit part of the ephemeral key during the scalar-multiplication process.

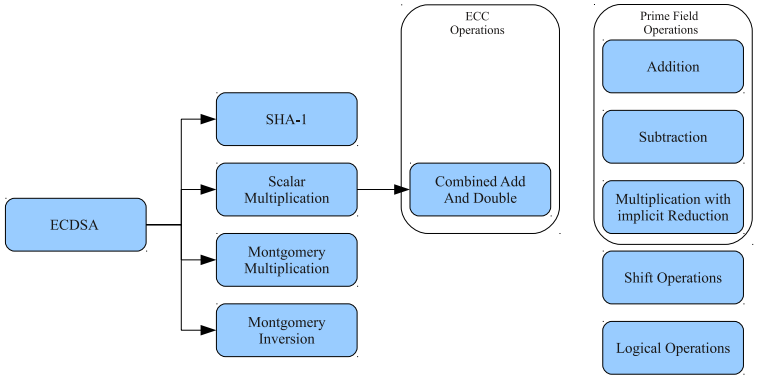

Figure 5.18: Hierarchical structure of the control unit.

5.5 Control Machine

The control unit generates all signals to control the datapath. We implemented all algorithms, which are needed to generate a valid digital signature. The ECDSA digital-signature algorithm has a very complex design. Therefore, we used a control machine with a hierarchical architecture. Figure 5.18 gives an overview on the hierarchical structure of the control unit. The highest level is ECDSA, which we described before in Algorithm 3.1. This procedure uses the functionality of the next lower level to fulfill all needed operations. The operations at this second stage are the SHA-1 hash function from Section 4.3, the ECC scalar multiplication from Section 4.2.3, the Montgomery multiplication algorithm from Section 4.1.2 and the Montgomery inversion algorithm from Section 4.1.4. The third stage includes all elliptic-curve operations, which we required to calculate the scalar multiplication. In our case, this stage contains only one element, the combined Double-and-Add algorithm, see Algorithm 4.20. The last stage in our control unit includes all prime-field operations for the two different prime fields and some shift and logical operations. This operations are used from the combined Double-and-Add algorithm and from the algorithms in stage two.

From an architectural aspect, we used a so called *hybrid* control-unit, cf. [4], which means that our control unit uses a FSM and also some micro-coded parts. We used this architectural approach, because some algorithms can be simply implement in a micro-coded based control unit but not in a FSM and vice versa. The micro-coded approach is used to implement one round into the SHA-1 algorithm, the multiplication with implicit fast reduction and for the Montgomery multiplication algorithm. All other parts of ECDSA has been implemented as a finite-state machine. The following two subsections deliver insights of the FSM and micro-coded control.

5.5.1 Finite-State Machine (FSM)

A finite-state machine describes the algorithm in a set of states. The actual state is stored in a so called *state register*. Depending on the current state and the current input signals, the FSM could be persisted in the actual state or if necessary, change to another state in every clock cycle. It is important to define a starting point, which is the first state after power up in order to assure a correct data flow. A FSM is a good choice for algorithms with a simple and linear data flow, because these algorithms can be described in a few states with less feedback signals from the datapath. In case of complex algorithms, a

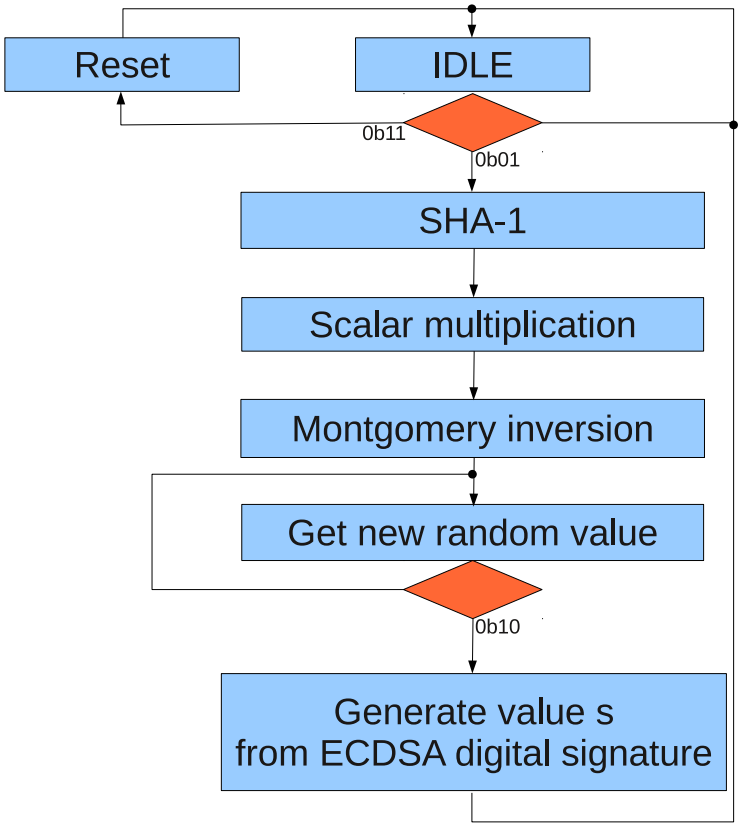

Figure 5.19: Overview of the finite-state machine.

FSM is not a good choice in most cases. Because a lot of states are required and the FSM becomes a very confuse layout. In our work, we used a FSM to describe almost all algorithms. Figure 5.19 gives an overview of the FSM, which we used as main part of the controller. This overview includes only some state blocks, which are presentable for the application flow. The complete FSM which we used to describe the ECDSA algorithm has 103 states. A soft reset with the control word $0x3$ is possible at every time in the ECDSA digital signature generation process and not only from the *IDLE* state.

5.5.2 Microprogramming

A micro-coded control unit uses a ROM to store the control signals and a counter to generate the addresses for the micro-code ROM. Figure 5.20 shows a basic example of a micro-coded control unit. The main parts are the ROM table, a register which holds the current address, an incrementation unit, and an operation decode logic. Small ROM tables can be synthesized from a set of combinatorial logical gates. If a big ROM table is required, a hard-macro ROM table has advantages, because this needs less area than a synthesized ROM table with combinatorial logical gates. The operation-decode logic uses a ROM table entry and decodes it to get the control signals for the ALU and the memory. This concept requires less area because the ROM tables store the information in a compressed form and only one decode logic is needed. Figure 5.20 shows a micro-coded control unit which requires no feedback from the ALU. This is the simple case, in which only a register and an incrementation unit is required to generate the ROM

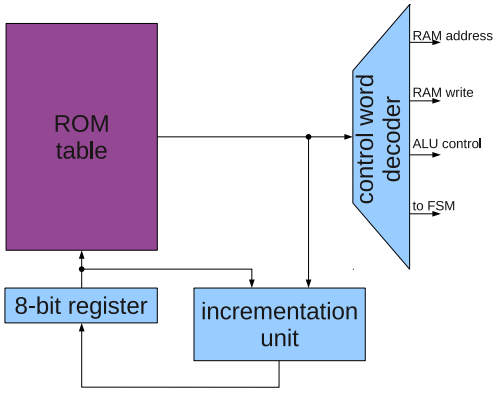

Figure 5.20: Overview of micro-coded control unit without feedback signals.

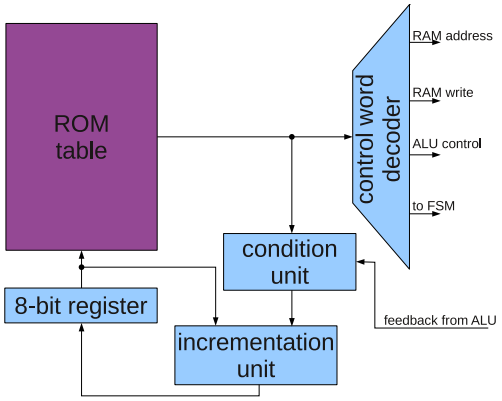

Figure 5.21: Overview of micro-coded control unit with feedback signals.

address for the next clock cycle. But most complex algorithms need some feedback signals from the datapath and in this case an additional condition unit is needed. Figure 5.21 shows a micro-coded control unit with a condition unit to handle feedback signals. The condition unit can make some changes on the ROM address, according to the level of the feedback signals. But this feedback from the datapath could make some problems if a pipelined architecture is used. In some cases, we must break the pipeline to retrieve the correct feedback signals from the ALU. The algorithms, which we used in this work need some feedback signals from the ALU, like the carry bit, the is-zero bit, and a bit of the ephemeral key, but we use no pipeline architecture in our datapath. Figure 5.22 shows the micro-coded part of the control unit, which is used in this work. It includes three ROM tables which hold the control signals for SHA-1, the Multiplication with implicit fast reduction and for the Montgomery multiplication algorithm. We use only one address register, one incrementation unit, and one operation-decode logic for all three ROM tables to get a design which requires less area. The 8-bit address register is also shared with the finite state machine, in which we used it as simple counter. Therefore, we insert an *AND* to turn of the addressing of the ROM tables. The controller of this ECDSA module uses both control-unit architectures as so called *hybrid* control-unit, cf. [4]. This design requires some additional logic to change between the FSM and the micro-coded part, but by using of the optimal control architecture for each algorithm, we get a control unit with the required minimal area.

5.6 Countermeasure against SPA, DPA, and Timing Analysis

Simple Power Analysis (SPA) and Differential Power Analysis (DPA) are two types of side-channel attacks. Side-channel attacks use side-channel information to attack a cryptographic device. Side-channel information are all pieces of information which you get physically from the device, e.g. the power consumption. This type of attacks try to break the implementation of an cryptographic algorithm and not the cryptographic algorithm itself. There exist several side channels like timing, power consumption, or electromagnetic emissions. These side-channel information can maybe used to extract information which

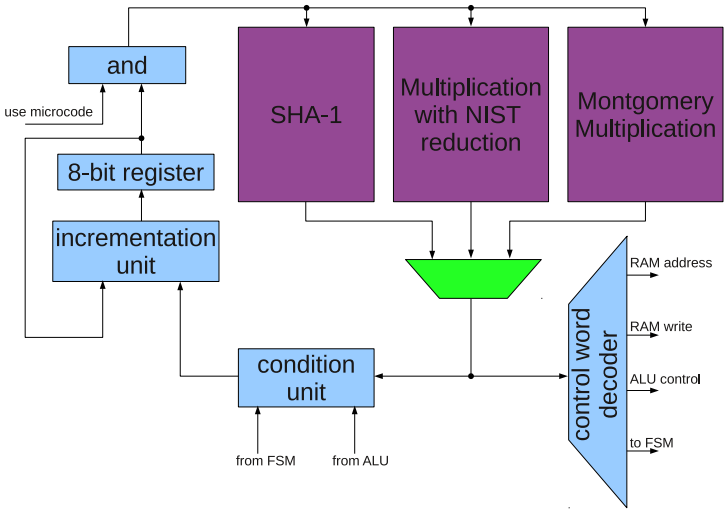

Figure 5.22: Micro-coded part of the control unit.

should have been protected, like a private key for instance. Therefore, it is important to protect a device against side-channel attacks. Timing analyses use the execution time to extract information. Simple power analysis and differential power analysis are two techniques, which use the power consumption or the electro-magnetic radiation of a device as information channel. The power consumption of a device based on semiconductor logic-gates depends on the number of transistors which change their charge in one clock cycle. If the algorithm, processed on the device, is well-known, the power consumption can be used to draw conclusions from the actually processed data, like a private key, cf. [4]. In the following, we will describe some side-channel attacks and the countermeasures which we have implemented in our module, to prohibit them.

Timing Analysis

If there exist a correlation between the execution time of an algorithm and a secret which is used by this algorithm, the secret can maybe extract by using timing analysis. Therefore, it is important to design an algorithm time-invariant, which means there exist no correlation between the execution time and a secret. One example of an algorithm which is not time-invariant in relation to the ephemeral key is the simple Double-and-Add algorithm, see Algorithm 4.17. Because a point-addition operation requires less time then a point-doubling operation. Therefore, we use in our work the Montgomery ladder algorithm, see Algorithm 4.19.

Hisayoshi Sato, Daniel Schepers and Tsuyoshi Takagi [50] show that also the Montgomery multiplication algorithm can be attacked by timing analysis. The last subtraction in the Montgomery multiplication algorithm, see Algorithm 4.6 is not required in every multiplication operation, consequently it could be a target for a timing attack. We use an own improved Montgomery multiplication algorithm in our work, see Algorithm 4.7, which makes this subtraction at all times and uses the last carry bit to decide which RAM address should be used in the next operation. Therefore, a timing attack such as described by Sato, Schepers and Takagi should not by possible anymore.

Simple Power Analysis (SPA)

Simple power analysis uses only a few or even one trace of an encryption operation or signature-generation operation to extract information. That means, that the power trace is directly interpretable. One possibility to attack the ECDSA algorithm with simple power analysis is the scalar multiplication. If we use the simple Double-and-Add algorithm, see Algorithm 4.17, it is easily possible to extract the ephemeral key from the power consumption trace, because a double operation has a completely different power consumption as an add operation. In our work, we use the Montgomery ladder algorithm, see Algorithm 4.19, to prevent the scalar multiplication against this type of attack.

Differential Power Analysis (DPA)

The DPA uses statistical methods to detect extremely small differences in power consumption traces or electromagnetic-radiation traces. Therefore, it is a very powerful attack to extract some secrets. But this type of attack requires a lot of traces to make a good statistical analysis possible. For real attacks, several thousand power-consumption traces are required and they must all use the same secret information. We use Coron's randomized projective coordinates [13] to protect the scalar multiplication against differential power analysis, cf. [41]. An attack with differential power analysis of the multiplication of the private key k_{prv} with the signature value r is also possible, cf. [27]. This multiplication is done by a Montgomery multiplication algorithm and this one is also vulnerable to DPA attacks. Hutter et a. give a solution to prevent this type of attack in [27]. An other possible solution to prevent this attack is a randomization of the Montgomery multiplication algorithm, but this randomization is not covered within this work and marked as future work.

5.7 Low-Power Optimization

Mobile devices have low-power requirements. Therefore, low-power optimization is also important for our ECDSA module. The power consumption of a *Complementary Metal Oxide Semiconductor (CMOS)* integrated circuit can be split into two parts. A static power consumption, which is caused by *sub-threshold currents* or *drain leakage currents* and a dynamic power consumption, which depends on a so called *short circuit current* and the charging and discharging of capacitors. The dynamic power consumption has the larger proportion on the full power consumption. This depends on the switching activity α, the parasitic capacity C_L, the supply voltage V_{DD} and the clock frequency f. Equation 5.1 show the relation between this four parameters.

$$P = \alpha \cdot C_L \cdot V_{DD}^2 \cdot f \tag{5.1}$$

The parasitic capacity is a technology depending parameter and can not be changed. Also the supply voltage depends on the technology and can not made arbitrarily small. Consequently, the switching activity and the frequency are the important points for low-power optimization. There exist two major concepts to reduce the switching activity and we use both in our work.

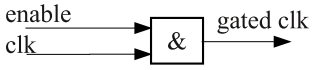

Figure 5.23: Clock gating with an AND gate.

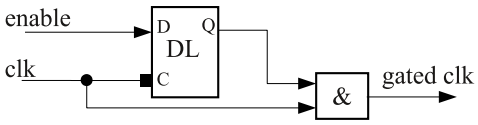

Figure 5.24: Clock gating with a clock-gating cell.

Clock Gating

Clock gating is a technique to reduce the switching activity of sequential CMOS logic. An ordinary 16-bit register element, shown in Figure 5.10, changes its internal state in every clock cycle, even if there is no new data at the $data_{in}$ port. Consequently, clock gating uses the approach, that a switching activity in the Flip Flop is only necessary if the $data_{in}$ port also changes. The *enable* signal, which can be seen in Figure 5.10, can also be used as a clock gating signal. A simple type of clock gating is shown in Figure 5.23, but this type should not be used. Because, the *enable* signal can produce glitches on the clock tree, which can lead to errors in the memory behavior. In order to prevent glitches on the clock tree, a clock gating cell has to be used. Figure 5.24 shows the block diagram of a clock-gating cell, consisting of a latch and an *AND* gate. Clock gating on a single Flip Flop is not useful, but it make senses for register which consists of several Flip Flips. Because, in this case we can use one clock-gating cell for all Flip Flops in the register, instead of an own multiplexer for every Flip Flop, compare Figure 5.10 and Figure 5.11. Therefore, clock gating can be used to reduce dynamic power consumption and a clock gated storage element requires furthermore less area, see Section 5.3. We use clock gating in all parts of our ECDSA module, but most beneficially in the memory. The clock-gating cells are built in automatically by the Cadence-design tools.

Operand Isolation

A second approach to reduce the dynamic power consumption is operand isolation, which is sometimes also called *sleep logic*. In a combinatorial circuit without operation isolation, an operation running through all combinatorial gates in each clock cycle. But in most cases, only a small combinatorial sub-circuit is actually used for an operation, like a multiplication or an addition. All other sub-circuits have also a switching activity and require electrical power, but these results are discarded. For this reason, it makes sense that only the actually used sub-circuit has switching activity and all other parts are on a constant value. This task is easy to implement, because we have only to insert one *AND* gate in the datapath. The *AND* gate can be controlled by a select signal, which we generate from the control word. If the select signal is 0, the sub-circuit is disconnect from the datapath. In our work, we use this technique manually for the multiplication sub-circuit and the addition sub-circuit in the ALU, because these are the parts with the highest power consumption. Additionally, we use operand isolation for the data input port of our memory, if no write operation is required and to switch off the ROM tables which actually are not in use. The Cadence-design tools supporting also operand isolation in an automatic way, but these improvements have only a small impact on the power consumption in our work. Figure 5.25 shows the power consumption of our ECDSA module. The blue trace (upper line) shows it without operand isolation and the red trace

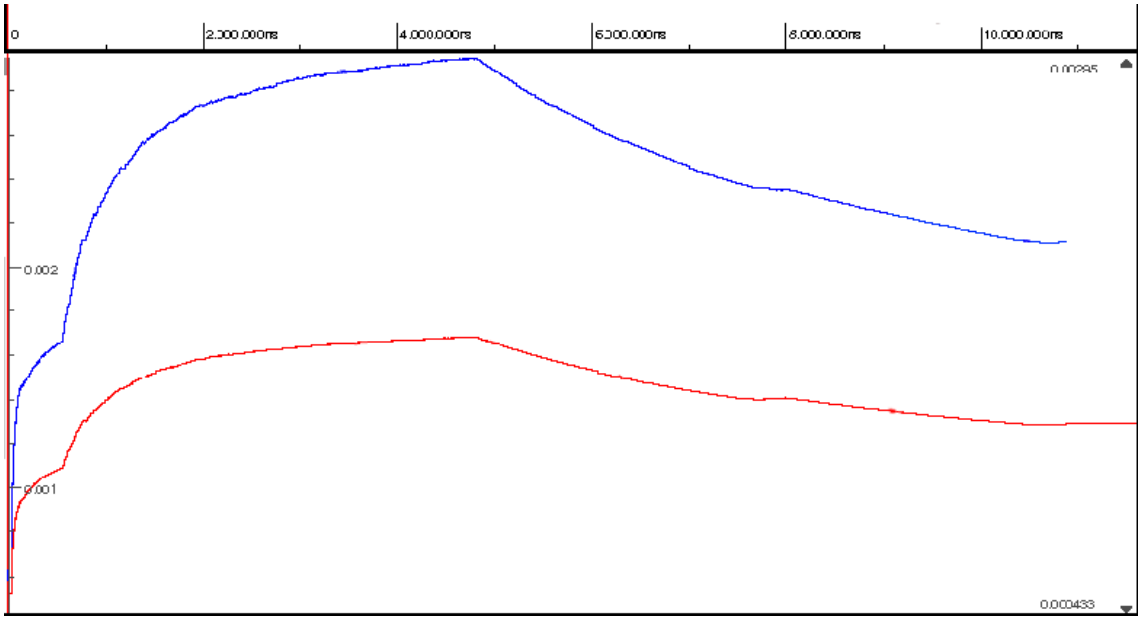

Figure 5.25: Operand isolation benefit (simulation with $V_{DD} = 3.3\,\mathrm{V}$ and $f = 5\,\mathrm{MHz}$).

(lower line) shows it with operand isolation. With operand isolation, we get a power-reduction of approximately 55%.

5.8 Results

We modeled the ECDSA module in several steps from a high-level model in Java to the post-layout simulation with the power simulation-tool NanoSIM from Synopsys. The results can be divided into two subsections, a small functional subsection and a bigger subsection with synthesis results. We discuss the area and power consumption of our ECDSA module and some timing constraints.

5.8.1 Functional

The aim of this work was to create an elliptic-curve based IP module with low-power and low-area requirements. This IP module generates an ECDSA digital signature, which use the elliptic-curve parameters from $SECP160_{r1}$ [48]. We used the Montgomery ladder algorithm to perform the scalar multiplication. Standard-projective coordinates are used to implement a combined Double-and-Add algorithm, which requires only 11 multiplications, 6 squaring, and 24 addition operations, see Algorithm 4.20. This algorithm used only the x-coordinate of an elliptic-curve point. In order to generate the full ECDSA digital signature, some additional functionality is required. Therefore, we implement the SHA-1 hash algorithm to calculate a SHA-1 fingerprint from the message. Furthermore, some modular multiplication and inversion algorithms for general moduli are required. We used a modified version of the Montgomery multiplication algorithm, see Algorithm 4.7 for multiplication and the Montgomery inversion algorithm to calculate the inverse element. Thus, this module contains all functionality, which is required for a digital signature generation, without a cryptographic secure random number generator (CSRNG). A

Table 5.4: Chip area of the ECDSA module for a clock frequency of 5 MHz.

Module	**Area [μm^2]**	**Area [#GE]**	**Area [%]**
Memory	602 438	10 953	59.81
ALU	149 313	2 715	14.82
Controller	249 795	4 542	24.80
Others	5 788	105	0.57
ECDSA module	**1 007 334**	**18 315**	**100.00**

Java high-level model is used to generate test vectors and to verify the functionality of our ECDSA module. We have tested the module after several designs steps, like synthesizes step or place-and-route step. All simulations have been done successfully and we could show a correct implementation of an ECDSA algorithm on circuit level.

5.8.2 Synthesis

We used the Cadence-design tools, with the IAIK design flow to synthesize our HDL model. The HDL model is described in *Verilog* and we used digital standard-cells from *c35b4* CMOS libraries published by Austriamicrosystem AG (AMS) for the synthesis. Therefore, all following process-dependent results refer on these 0.35μm process, unless otherwise is specified. Figure 5.26 shows the layout of our ECDSA module after the place-and-route design process. The three big sub-modules are marked on this figure to get an impression of their proportions. We do not show the AMBA interface as fourth sub-module because it requires only a very small area and can not be assigned to a special place. The memory consist of a RAM and a ROM table, at which the ROM table is imbedded in the middle of the memory block. We subdivide the controller in Figure 5.26 in the micro-coded part on the bottom site and the FSM on the upper site. The upper and right marked area in the ALU is the 16×16-bit multiplication unit, which requires almost the half area of the ALU. Table 5.8.2 shows the area requirements of the ECDSA module in terms of μm^2, Gate Equivalents (GE) and as a percentage of the total area. The area of digital circuits is usually specified in GEs, because these values are independent from the technology. One *GE* is equal to the size of a two input NAND gate, which requires an area of $55\mu m^2$ at the *c35b4* CMOS libraries. Figure 5.27 shows the area consumption also in a graphical way. The memory requires the largest part of the area. In Table 5.8.2, the area requirements for the memory include the RAM and the ROM table are shown, but not the area for the EEPROM, because we have only a timing model of the EEPROM. It is interesting that the ALU requires less area then the control machine. The reason is that we design an ALU which contains only basic operations which we can use to perform operations in both finite fields. This relatively simple ALU leads to a more complex control unit. This design has also a positive influence to the power consumption.
Table 5.5 illustrates the detail area consumption of the ALU. The multiplier is the largest part of the ALU and the whole multiply-accumulate unit requires more than 60 % of the area. Figure 5.28 shows the area distribution of the control unit. In our work, we use the micro-coded approach only for three algorithms, but this sub-system requires only slightly less area then the finite-state machine. This fact is given by the complexity of these three algorithms. The AMBA interface and some additional logic is summarized at the *Others* item in Table 5.8.2. Our work is a full ASIC designed ECDSA processor. Some other works like [54, 26] require less area, but they use RAM macros instead of a latch based

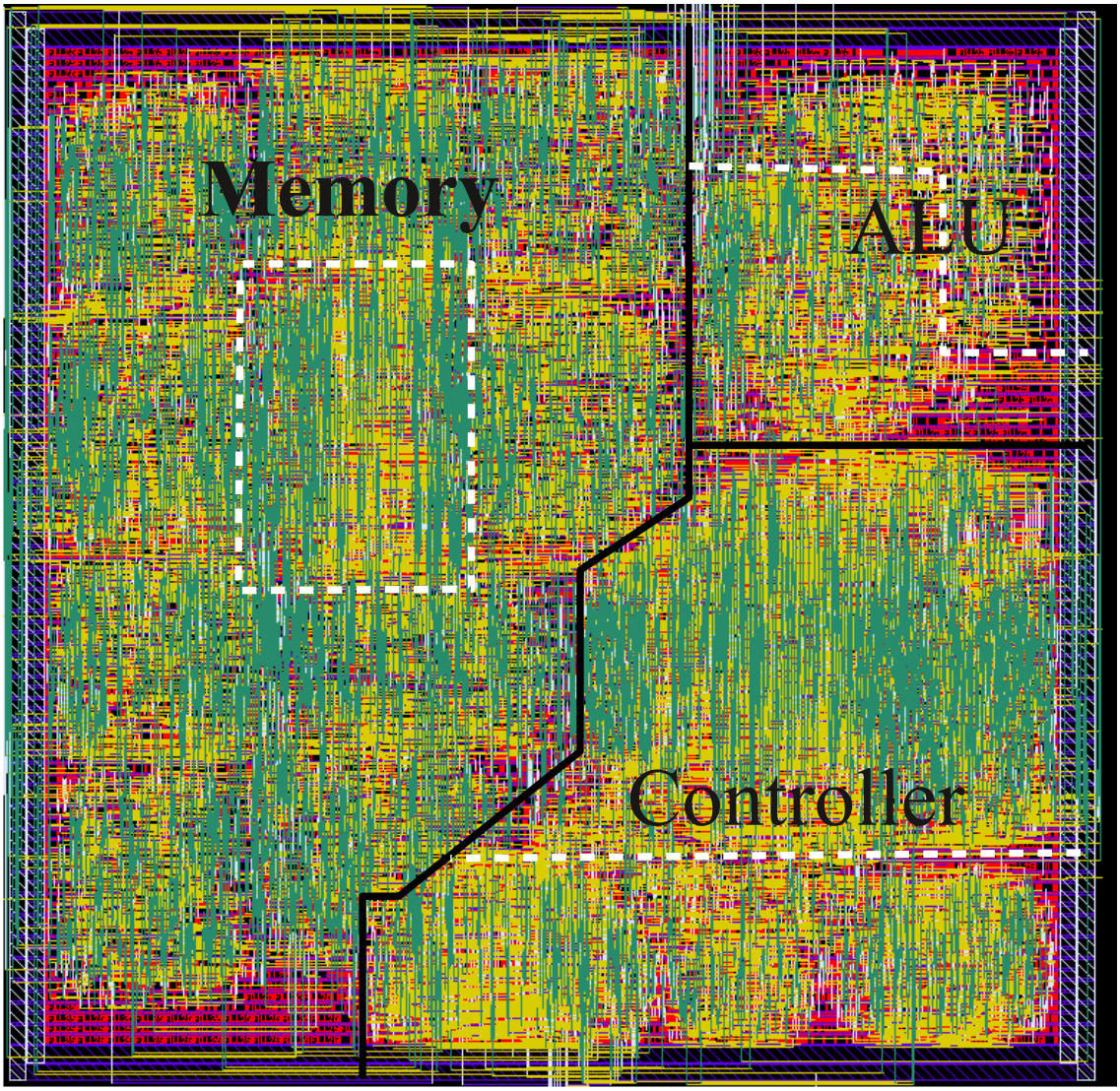

Figure 5.26: Layout after place-and-route.

custom design. Implementations, which use RAM macros can be difficult for passive tags in high volume, but it could half the chip area for the memory module.

Next, we will discuss the cycle count and the maximum clock frequency of our ECDSA implementation. The most time intensive part of the signature-generation process is the

Table 5.5: Area requirements of the ALU sub-modules.

Sub-module	**Area [#GE]**	**Area [%]**
Multiplier	1 307	48.16
Adder	159	5.85
Register	191	7.02
Logical units	1 058	38.97
ALU	**2 715**	**100.00**

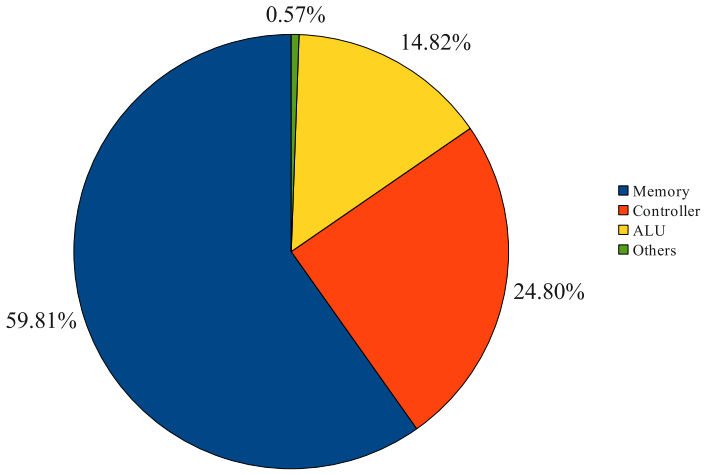

Figure 5.27: Chip area of the ECDSA module.

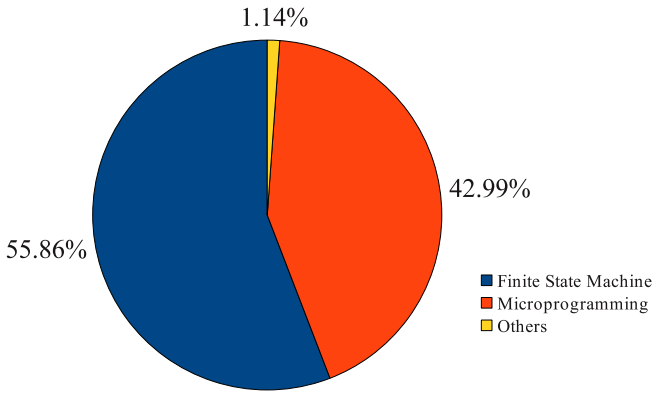

Figure 5.28: Area requirements of the control unit sub-modules.

scalar multiplication. We use a combined Double-and-Add algorithm which requires 11 multiplications, 6 squaring and 24 addition operations in the finite field $\mathbb{F}_{P160}$. One Double-and-Add operation requires 2 959 clock cycles. Figure 5.29 shows the distribution of the individual finite-field operations. There, we have summarized the multiplication and the square operation because we use no special squaring algorithm in order to reduce the area requirement. With an optimized squaring algorithm, the number of required clock cycles can be reduced. The whole Montgomery ladder algorithm requires about $2\,959 \cdot 159 = 470\,481$ clock cycles only for the point Double-and-Add operation. Additional clock cycles are needed for Coron's randomized projective coordinates countermeasure and the key management. Table 5.6 shows the clock cycle-count distribution for the full ECDSA signature algorithm. The scalar multiplication includes all needed operations without the Montgomery inversion. Table 5.6 also illustrates that the scalar multiplication is the most time intensive part of the ECDSA signature-generation pro-

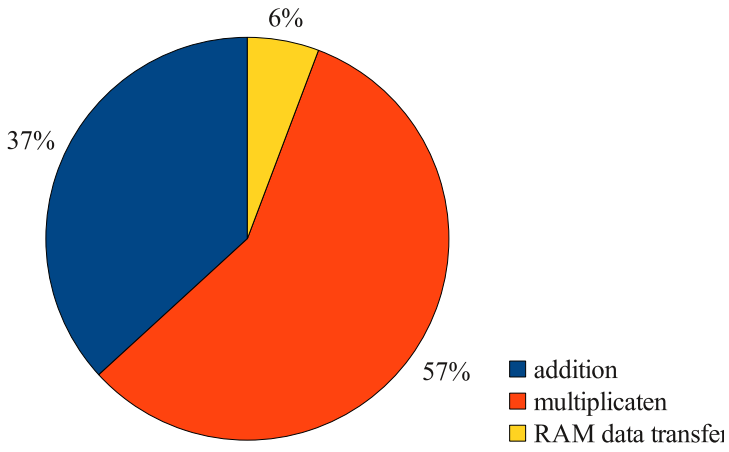

Figure 5.29: Cycle count of the combined Double-and-Add algorithm

cess. Therefore, it is important to optimize this algorithm to reduce the clock-cycle count and the power consumption. The full ECDSA digital signature algorithm requires also 7 Montgomery multiplications, which requires $7 \cdot 276 = 1\,932$ clock cycles and 2 Montgomery inversion, which requires $2 \cdot 14\,801 = 29\,614$ clock cycles. Figure 5.30 shows these results also in a graphical way. The second part to characterize the performance of the ECDSA

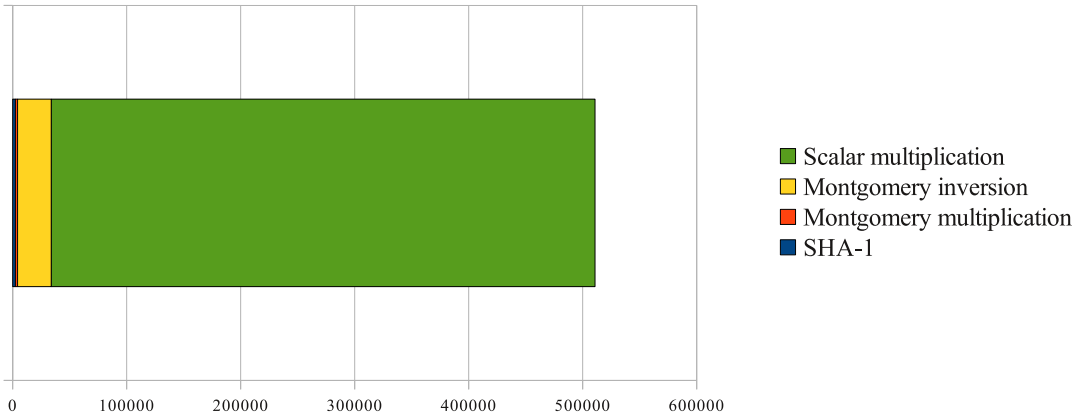

Figure 5.30: Cycle-count distribution for the full ECDSA signature generation.

Table 5.6: Cycle-count distribution for the full ECDSA signature algorithm.

Scalar multiplication	**Montgomery multiplication**	**Montgomery inversion**	**SHA-1**	**Total**
476 873	1 932	29 614	2 412	**510 831**
93.35 %	0.38 %	5.80 %	0.47 %	**100** %

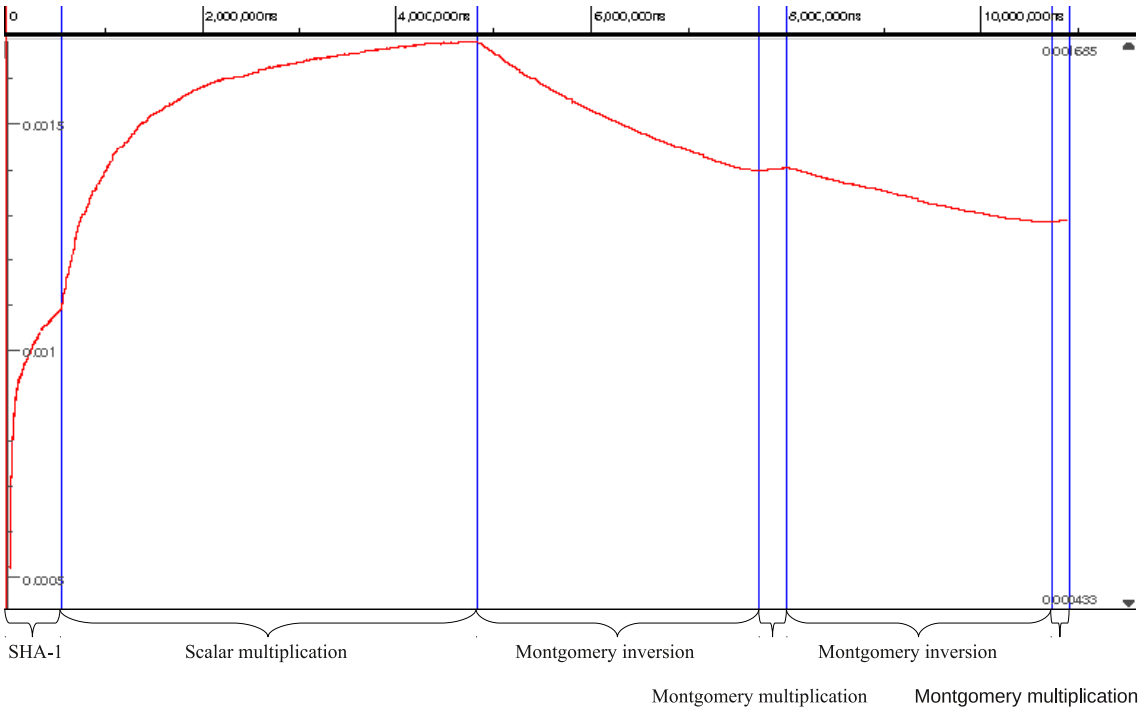

Figure 5.31: Current of a signature-generation process with $V_{DD} = 3.3\,\mathrm{V}$ and $f = 5\,\mathrm{MHz}$.

module is the maximum clock frequency. This value is generally limited by the longest path in our digital circuit. We evaluate our module with a clock frequency of 5 MHz and a core voltage of 3.5 V. The results after the place-and-route design step indicate a maximum clock frequency of 70 MHz. Our ECDSA module requires approximately 103 ms at a clock frequency of 5 MHz to generate the full digital signature. This result gives an operation throughput of 9.7 Ops/s (ECDSA signature-generations operations per second). If we reduce the core voltage to 2.5 V the maximum clock frequency decrease to 50 MHz.

At last, we will discuss the power consumption of our ECDSA module. The power consumption results are generated with *Synopsys NanoSIM*, which is a near-spice simulator. Thus, these results are more accurate than the power consumption results from *Cadence-Synthesis Tools*. The *NanoSIM* simulation uses a *Value-Change-Dump (VCD)* file, which contains the input values for the power simulation. This *VCD* file can be generated during the HDL simulation and thus contains data from a real signature-generation operation. Figure 5.31 illustrates the power simulation result for one signature-generation process. This simulation uses only an 8-bit ephemeral key, because the *NanoSIM* power simulation is very time intensive. A simulation of the full ECDSA operation with a 160-bit ephemeral key takes approximately more than a month. In Figure 5.31, we marked severally steps of the ECDSA signature-generation process. After start, the SHA-1 hash value is calculated, which is followed by the scalar multiplication. The scalar multiplication is the part which requires the most power. The *Montgomery* inversion looks very time intensive in Figure 5.31, but this is an illusion because we use only an 8-bit ephemeral key. The power consumption can be calculated from the supply current and the core voltage with Equation 5.2.

$$P = U \cdot I \tag{5.2}$$

Table 5.7: Power consumption for different operation conditions and a clock frequency of 5 MHz.

Operation	**Core voltage [V]**	**Average current [mA]**	**Power [mW]**
ECDSA 8-bit	3.3	1.21	3.99
ECDSA 8-bit	2.5	0.84	2.10
ECDSA 160-bit	3.3	1.68	5.54
ECDSA 160-bit	2.5	1.17	2.92

Therefore, the power-consumption trace is very similar to the trace of the current in Figure 5.31. Hence, we can also see the advantage of the Montgomery ladder algorithm. The power trace during the scalar multiplication has a smooth power consumption. A reduction of the core voltage decreases quadratically the power consumption. Therefore, it makes sense to reduce the core voltage from 3.3 V to 2.5 V to minimize power consumption. Table 5.7 shows the results of the *NanoSIM* power simulation. ECDSA 8-bit operation in Table 5.7 means the ECDSA signature-generation process with a 8-bit ephemeral key, which we use for the power simulation with *NanoSIM*. The average current and the power for the full ECDSA operation with a 160-bit ephemeral key are only estimated. We use the current during the scalar multiplication to estimate the average current over the full ECDSA operation, because the scalar multiplication requires the most time, see Table 5.6 and the highest current, see Figure 5.31.

5.8.3 Summary and Comparison

In this subsection we will compare our ECDSA module with related work. Table 5.8 shows the comparison to other modules, but not all of them implement the full ECDSA algorithm. In some works only the scalar multiplication was implemented. All this implementations have a different functional scope and also different technologies and clock frequencies. Therefore, it is difficult to find a fair comparison between all these works. The results from Lee et al. [38] and Hein et al. [25] are given as a comparison, because they used arithmetic over a binary field which is quite simpler than the arithmetic over a prime field. But if we compare our result with the result from Hein we have almost the same

Table 5.8: Comparison of our ECDSA module to related work.

Design	**Field**	**Core voltage [V]**	**Techn. [μm]**	**Area [#GE]**	**Clock Cycles**	**Power [μW] @1 MHz**
Yong Ki Lee [38]	$F_{2^{163}}$	1.5	0.13	12 506	275 816	32.4
Hein [25]	$F_{2^{163}}$	2.5	0.35	11 904	296 000	516.0
Auer [4]	F_{p192}	2.5	0.35	24 745	1 031 000	613.0
Fuerbass [22]	F_{p192}	3.3	0.35	23 656	502 000	1 692.0
Hutter [26]	F_{p192}	3.3	0.35	19 115	859 188	1 508.0
Wenger [54]	F_{p192}	1.8	0.18	11 686	1 377 000	114.0
This work	F_{p160}	3.3	0.35	18 315	510 831	1 108.0
This work	F_{p160}	2.5	0.35	18 315	510 831	585.0

Table 5.9: Synthesis results with a 180nm technology from *UMC*.

Operation	**Techn.** [μ**m**]	**Core voltage** [**V**]	**Area** [μm^2]	**Power** [μ**W**]@**1 MHz**
ECDSA 160-bit	0.18	1.6	212 143	192

power consumption. Auer [4], Hutter et al. [26] and Wenger et al. [54] also implement the full ECDSA algorithm with SHA-1 as hash function. Therefore, these three works are easy to compare with our implementation. The results from Hutter are very similar to our work, if we take into consideration the other prime field. According to the result from Auer, our work requires less area, less clock cycles and also the required power is slightly smaller. Wenger uses in his work a single-port RAM macro, which requires much less area to our dual-port ASIC RAM. Consequently, his implementation requires less area than our module, but the run time of his ECDSA implementation is about 2.5 times longer. As a conclusion we show the synthesis result, if we use a 180 nm technology from *UMC* [1]. Table 5.9 shows the main module information after the place-and-route step. The power result is generated by using the *Cadence-First-Encounter* tool, but these are only a rough estimation. If we use this 180 nm technology, we could reduce the power consumption about 50 % but the ECDSA implementation by Wenger requires also less power. A possible reason is the usage of the single-port RAM macro, which requires also less power than an ASIC dual-port RAM.

Chapter 6

Conclusions

The aim of this work was to design an elliptic-curve processor with full ECDSA functionality and optimization for low-area and low-power consumption. Such a processor can be used to perform a cryptographic-secure authentication in combination with an RFID system. This secure RFID tags can be used for door-locking applications, micro-payment applications, or for anti-counterfeiting purposes.

In order to fulfill this challenge to implement a secure RFID tag, which requires low area and low power, we modeled the elliptic-curve processor in several steps: from a high-level model in JAVA to a post-layout power simulation with the tool NanoSIM from Synopsys. The high-level model is used for test-vector generation and it is used to verify all design steps down to post-layout power simulation. All simulations have been done successfully and we could show a correct implementation of our ECDSA module. Our elliptic-curve processor includes all functionalities which are required to generate an ECDSA digital signature, except of a cryptographic secure random number generator. The calculation of the SHA-1 hash value can also be done by our ECDSA module.

We use the elliptic-curve parameters from $SECP160r1$, which is a standardized elliptic curve from SECG. All algorithms, which are required to generate an ECDSA digital signature, are optimized for this elliptic curve. In order to fulfill a fast multiplication in the prime field $\mathbb{F}_{P160r1}$, we designed a new multiplication algorithm with implicit fast reduction. This multiplication algorithm requires only $t^2 + 5 \cdot t$ clock cycles, where t represents the number of words. The scalar multiplication is done by using a *Montgomery* ladder algorithm in combination with a combined Double-and-Add algorithm, to prevent side-channel attacks. Our combined Double-and-Add algorithm is a modified version of the algorithm proposed by Tetsuya Izu and Tsuyoshi Takagi [29], which is optimized to our memory architecture and datapath layout. This own version of the combined Double-and-Add algorithm requires 17 multiplications and 24 addition operations to calculate one point addition and doubling operation. From a hardware architecture point of view, we use an ASIC design for all sub-modules with a datapath of 16 bits. The memory consists of a RAM with 1 440 bits, ROM table with 1 328 bits and an EEPROM with 160 bits. A special feature of our memory is that we can select between two different address spaces. One address space for operations modulo $\mathbb{F}_{P160}$ with an address space of 9×160 bits and a second address space for operations modulo $\mathbb{F}_{n161}$ with an address space of 8×176 bits. The core component in the ALU is the multiply-accumulate unit, which can calculate a 16×16-bit multiplication and a 36-bit addition within one clock cycle. The 36-bit adder is also used for simple addition and subtraction operations and includes all functionalities to execute two interleaved addition operations. Our ECDSA module use a *hybrid* control-

Table 6.1: Summarization of our results $V_{DD} = 3.3$V.

Technology	**Area**		**Runtime**	**Power**	$\mathbf{f_{max}}$
	[μm^2]	[#GE]	[cycles]	[μW]	[MHz]
ECDSA 0.35 μm AMS	1 007 334	18 315	510 831	1 108.0@1 MHz	70

unit, which means that one part is implemented as a FSM and some other parts in a micro-coded approach.

In relation to other works, our implementation requires less clock cycles and a slightly lower chip area. The required electrical power is also low in relation to other works. Only one other implementation requires less power, but they used a totally different memory architecture. Table 6.1 shows a summarization of our results.

6.1 Optimizations and Outlook

In face of the good results, further optimizations of our implementation are possible. The following enumeration gives you some possible areas for improvements.

- **Memory Architecture:** Actually, we use an ASIC dual-port memory architecture with asynchronous-read and synchronous-write capability. A change to an other memory architecture, like a synchronous dual-port RAM macro, or a synchronous single-port RAM macro, would reduce the size of area and the required electrical power. In contrast, the required clock cycles for one ECDSA digital signature generation becomes larger. In a second approach, we could use a more sophisticated sleep logic, which should reduce the required power.

- **RAM Size:** Our implementation requires 9×160 bits of RAM to generate an ECDSA digital signature. If we change the ECDSA authentication protocol, so that the scalar multiplication is calculated first and the message which should get signed is transmitted to the tag afterwards, we can reduce the required RAM elements to 8×160 bit. This change would reduce the RAM area to about 1 070[GEs].

- **Controller:** Actually, we use a *hybrid* control-unit in our ECDSA module. A change to a pure micro-coded control unit which uses a hard-macro ROM table would also reduce the size of area of our ECDSA module. But, there we have to solve some problems with condition jumps and pipeling.

- **Square Operation:** Actually, a square operation is done by the same algorithm as for a multiplication operation. If we use a special squaring unit, then we can reduce the required clock cycles for an ECDSA digital signature-generation process. But this change would lead to a bigger size of area, because an additional squaring algorithm is required.

- **DPA Resistant Multiplication:** An other possible solution to get a DPA resistant multiplication algorithm is to randomize the particular 16×16-bit multiplication operations and hide them time domain. We did not implement such a countermeasure and therefore it could be a topic of a future work.

Appendix A

Definitions

A.1 Abbreviations

ALU	Arithmetic Logic Unit
AMBA	Advanced Microcontroller Bus Architecture
AMS	Austriamicrosystems AG
ANSI	American National Standards Institute
ASIC	Application-Specific Integrated Circuit
CIHS	Coarsely Integrated Hybrid Scanning
CIOS	Coarsely Integrated Operand Scanning
CSRNG	Cryptographic Secure Random Number Generator
DPA	Differential Power Analysis
EC	Elliptic Curve
ECC	Elliptic-Curve Cryptography
ECADD	Elliptic-Curve Point Addition Operation
ECDBL	Elliptic-Curve Point Doubling Operation
ECDLP	Elliptic-Curve Discrete Logarithm Problem
ECDSA	Elliptic-Curve Digital Signature Algorithm
FIOS	Finely Integrated Operand Scanning
FIPS	Finely Ingegrated Product Scanning
FIPS	Federal Information Processing Standards
gcd	Greatest Common Divisor
HF	High Frequency
LF	Low Frequency
LFSR	Linear Feedback Shift Register
MW	Microwave
NIST	National Institute of Standards and Technology
RFID	Radio-Frequency Identification
SECG	Standards for Efficient Cryptography Group
SECP160	Elliptic Curves over Prime Fields with 160 bits standardized from SECG
SOS	Seperated Operand Scanning
SPA	Simple Power Analysis
UHF	Ultra-High Frequency
UMC	United Microelectronics Corporation
VCD	Value Change Dump

A.2 Used Symbols and Operations

$\wedge$	Bitwise AND operation
$\vee$	Bitwise OR operation
$\neg$	Bitwise NOT operation
$\oplus$	Bitwise XOR operation
$<<$	Left shift operation
$>>$	Right shift operation
$ROTL_n(a)$	Rotate left operation, $ROTL_n(a) = (a << n) \vee (a >> W - n)$
$ROTR_n(a)$	Rotate right operation, $ROTR_n(a) = (a >> n) \vee (a << W - n)$
$\mathbf{MonPro}(a, b)$	Montgomery product from a and b, $c_R = a * b$
$\mathbf{AlmMonInv}(a)$	Almost Montgomery inverse from a, $a^{-1} \cdot 2^k$

Bibliography

[1] United Microelectronics Corporation (UMC) 0.18 μm CMOS Libraries.

[2] A. AG. Memory Compiler for Dual-Port RAM in 0.35m CMOS (C35), 2010.

[3] ANSI. *ANSI X9.62:2005: Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)*. American National Standards Institute, pub-ANSI:adr, 2005.

[4] A. Auer. Scaling Hardware for Electronic Signatures to a Minimum. *Master's thesis, TU Graz (October 2008)*, 2009.

[5] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls, and I. Verbauwhede. Public-key cryptography for RFID-tags. In *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops' 07. Fifth Annual IEEE International Conference on*, pages 217–222. IEEE, 2007.

[6] A. Bosselaers, R. Govaerts, and J. Vandewalle. Comparison of three modular reduction functions. In *Advances in CryptologyCRYPTO93*, pages 175–186. Springer, 1993.

[7] E. Brier and M. Joye. Weierstrass Elliptic Curves and Side-Channel Attacks. In *PKC: International Workshop on Practice and Theory in Public Key Cryptography*. LNCS, 2002.

[8] D. R. L. Brown. *SEC1: Elliptic Curve Cryptography*, May 2009.

[9] J. Buchmann. FlexiProvider - Harnessing the power of the Java Cryptography Architecture.

[10] N. S. Chang, C. H. Kim, Y. Park, and J. Lim. A Non-redundant and Efficient Architecture for Karatsuba-Ofman Algorithm. In *ICISC: International Conference on Information Security and Cryptology*. LNCS, 2005.

[11] J. Chung and A. Hasan. More Generalized Mersenne Numbers (Extended Abstract). In *SAC: Annual International Workshop on Selected Areas in Cryptography*. LNCS, 2003.

[12] H. Cohen, G. Frey, and R. Avanzi. *Handbook of elliptic and hyperelliptic curve cryptography*. CRC Press, 2006.

[13] J. S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In *CHES: International Workshop on Cryptographic Hardware and Embedded Systems, CHES, LNCS*, 1999.

[14] R. E. Crandall. Method and Apparatus for Public Key Exchange in a Cryptographic System. Technical report, US Patent 5463690, Oct. 1995.

[15] C. De Canniere and C. Rechberger. Finding SHA-1 characteristics: General results and applications. *Lecture Notes in Computer Science*, 4284:1, 2006.

[16] C. Elwart. RFID in Wirtschaft und Gesellschaft – Analyse aktueller Anwendungsfälle zur Ausdifferenzierung des Mikropolis-Modells. *Universität Hamburg*, 2010.

[17] H. Fan, J. Sun, M. Gu, and K. Lam. Overlap-free Karatsuba-Ofman Polynomial Multiplication Algorithms for Hardware Implementations. This is a major version. I added 1 example and 1 figure, and corrected some typoes, 2007.

[18] K. Finkenzeller and D. Müller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication*. Wiley, 2010.

[19] N. FIPS. 180-2: Secure Hash Standard (SHS). Technical report, Technical report, National Institute of Standards and Technology (NIST), 2 001. http://csrc. nist. gov/publications/fips/fips180-2/fips180-2withchangenotice. pdf, Aug. 2002.

[20] W. Fischer, C. Giraud, and E. W. Knudsen. Parallel scalar multiplication on general elliptic curves over $\mathbb{F}_p$ hedged against Non-Differential Side-Channel Attacks, Jan. 09 2002.

[21] R. Fischlin. Effiziente Arithmetik in $\mathbb{Z}_m$ und $\mathbb{F}_{p^n}$, Februar 2001.

[22] F. Fürbass and J. Wolkerstorfer. ECC processor with low die size for RFID applications. In *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pages 1835–1838. IEEE, 2007.

[23] A. A. Gutub and A. F. Tenca. Efficient scalable VLSI architecture for Montgomery inversion in GF(p). *Integration*, 37(2):103–120, 2004.

[24] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Science + Business Media LLC, 2004.

[25] D. Hein, J. Wolkerstorfer, and N. Felber. ECC is Ready for RFID–A Proof in Silicon. In *Selected Areas in Cryptography*, pages 401–413. Springer, 2009.

[26] M. Hutter, M. Feldhofer, and T. Plos. An ECDSA processor for RFID authentication. *Radio Frequency Identification: Security and Privacy Issues*, pages 189–202, 2010.

[27] M. Hutter, M. Medwed, D. Hein, and J. Wolkerstorfer. Attacking ECDSA-enabled RFID devices. In *Applied Cryptography and Network Security*, pages 519–534. Springer, 2009.

[28] T. Izu, B. Möller, and T. Takagi. Improved Elliptic Curve Multiplication Methods Resistant against Side Channel Attacks. In *INDOCRYPT: International Conference in Cryptology in India*. LNCS, Springer-Verlag, 2002.

[29] T. Izu and T. Takagi. A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks. In *PKC: International Workshop on Practice and Theory in Public Key Cryptography*. LNCS, 2002.

[30] M. Johnson, B. Phung, T. Shackelford, and S. Rueangvivatanakij. Modular Reduction of Large Integers Using Classical, Barrett, Montgomery Algorithms, 2002.

[31] M. Joye and C. Tymen. Protections against Differential Analysis for Elliptic Curve Cryptography. In *CHES: International Workshop on Cryptographic Hardware and Embedded Systems, CHES, LNCS*, 2001.

[32] B. S. J. Kaliski. The Montgomery inverse and its applications. *Computers, IEEE Transactions on*, 44(8):1064–1065, Aug 1995.

[33] C. Kaya Koc, T. Acar, and B. J. Kaliski. Analyzing and comparing Montgomery multiplication algorithms. *Micro, IEEE*, 16(3):26–33, Jun 1996.

[34] P. Kitsos and Y. Zhang. *RFID Security - Techniques, Protocols and System-on-Chip-Design.* Springer Science + Business Media LLC, 2008.

[35] M. Lampe, C. Flörkemeier, and S. Haller. Einführung in die RFID-Technologie. *Das Internet der Dinge*, pages 69–86, 2005.

[36] K. Latif, A. Mahboob, and N. Ikram. A parameterized design of Modular Exponentiation on reconfigurable platforms for RSA cryptographic processor. In *Computer, Control and Communication, 2009. IC4 2009. 2nd International Conference on*, pages 1–8, Feb. 2009.

[37] Y. Lee, L. Batina, and I. Verbauwhede. Untraceable RFID authentication protocols: Revision of EC-RAC. In *RFID, 2009 IEEE International Conference on*, pages 178–185. IEEE, 2009.

[38] Y. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede. Elliptic-curve-based security processor for RFID. *IEEE Transactions on Computers*, pages 1514–1527, 2008.

[39] A. Ltd. AMBA Specification Rev. 2.0. *ARM IHI 0011A*, 1999.

[40] M. Machhout, M. Zeghid, W. E. hadj yoursef hadj yoursef, B. Bouallegue, A. Baganne, and R. Tourki. Efficient Large Numbers Karatsuba-Ofman Multiplier Design for Embedded Systems. *International Journal of Electronics, Circuits and Systems*, 3(1), 2009.

[41] M. Medwed and E. Oswald. Template attacks on ECDSA. *Information Security Applications*, pages 14–27, 2009.

[42] B. Möller. Securing Elliptic Curve Point Multiplication against Side-Channel Attacks. In *ISW: International Workshop on Information Security, LNCS*, 2001.

[43] P. L. Montgomery. Modular Multiplication Without Trial Division. *Mathematics of Computation*, 44(170):519–521, 1985.

[44] P. L. Montgomery. Speeding the Pollard and elliptic Curve Methods of factorization. *Mathematics of Computation*, 48:243–264, 1987.

[45] National Institute of Standards and Technology. Digital Signature Standard (DSS). FIPS Publication 186-3, June 2009.

[46] NSA. The Case for Elliptic Curve Cryptography. Only available one a web page., Jan. 2009.

[47] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *Advances in Cryptology - CRYPTO' 92*, pages 31–53. Springer, 1993.

[48] C. Research. *SEC2: Recommended Elliptic Curve Domain Parameters*. Certicom Corp., September 2000.

[49] E. Rolf, V. Nilsson, and A. Perlos. Near Field Communication (NFC) for Mobile Phones. 2006.

[50] H. Sato, D. Schepers, and T. Takagi. Exact analysis of Montgomery multiplication. *Progress in Cryptology-INDOCRYPT 2004*, pages 1387–1394, 2005.

[51] E. Savas and C. K. Koç. The Montgomery Modular Inverse – Revisited. *IEEE Transactions on Computers*, 49(7):763–766, July 2000.

[52] C. Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[53] J. A. Solinas. Generalized Mersenne Numbers. Technical report, National Securtiy Agency, Sept. 09 1999.

[54] E. Wenger, M. Feldhofer, and N. Felber. Low-resource hardware design of an elliptic-curve processor for contactless devices. *Information Security Applications*, pages 92–106, 2011.

[55] A. Werner. *Elliptische Kurven in der Kryptographie*. Springer, 2002.

[56] J. Wolkerstorfer. Is Elliptic-Curve Cryptography Suitable to Secure RFID Tags?, 2005. Presentation.

[57] A. D. Woodbury. Efficient algorithms for elliptic curve cryptosystems on embedded systems. Master's thesis, Worcester Polytechnic Institute, 2001.

[58] G. Zaverucha. Generalized Mersenne Numbers in Pairing-Based Cryptography. Master's thesis, Dalhousie University, July 2006.

[59] J. Zhang, T. Xiong, and X. Fang. Hardware Implementation of Improved Montgomery Modular Multiplication Algorithm. In *Communications and Mobile Computing, 2009. CMC '09. WRI International Conference on*, volume 3, pages 370–374, Jan. 2009.