Direct Anonymous Attestation based on Elliptic Curve Cryptography A feasibility Study for RFID

Master Thesis

Nerma Šarić

Institute for Electronics (IFE)
Graz University of Technology
http://www.ife.tugraz.at/

in cooperation with Infineon Technologies Austria AG

Advisor: Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Pribyl
Assessor: Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Karl-Christian Posch
Supervisor at Infineon: Dipl.-Ing. Holger Bock

Graz, March 2013.

in cooperation with
Infineon Technologies Austria AG
Development Center Graz

EIDESSTATTLICHE ERKLÄRUNG

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 entnommenen Stellen als solche kenntlich gemacht habe.

Graz,am ....

(Unterschrift)

STATUTORY DECLARATION

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.

....

Date (signature)

Acknowledgments

I would like to thank Infineon Technologies to make all the equipment available and therefore enable me to create my diploma thesis, especially to my advisor Dipl.-Ing. Holger Bock for his support. I would also like to take the opportunity to thank Dipl.-Ing. Dr.techn. Andreas Genser.

I also want to thank Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Pribyl and Ao.Univ-Prof. Dipl.-Ing Dr.techn. Karl-Cristian Posch of the Technical University Graz for supporting me and making this thesis work possible. My deepest gratitude goes to Dipl.-Ing. Dr.techn. Marc Jörn Schmidt who was a wonderful and hard-working mentor from whom I have learned a great deal.

My special thanks go to thank Univ.-Prof. Dipl.-Ing. Dr.techn. Lothar Fickert, who helped me during my studies and with this thesis.

Last but not least I want to thank my family and Nedžad Hrustić for their love, support encouragement during my study. As ever, my family have been there for me in different ways.

Thank you all for being patient with me!

Nerma Šarić

Kurzfassung

Radio Frequency Identification (RFID)-Tags existieren bereits seit Jahren. Dies sind meistens passive Transponder, die für die kontaktlose Kommunikation verwendet werden. RFID-Tags werden versorgt mit einem elektromagnetischen Feld des RFID-Lesers. Der Ausgangspunkt dieser Arbeit war SLE78CLxxxP Dual 16-Bit-Sicherheits-Controller von der Infineon, der $CCEAL6 + high/EMVCo$ zertifiziert ist. Daher ist es ideal für höchste Sicherheitsanwendungen wie digitale Signaturen (z.B. ePassport, eHealth-Karte, eDriver License, eVisa, eSignature).

Diese Arbeit beschreibt die Implementierung eines angepassten Direct Anonymous Attestation (DAA) Algorithmus, sodass das Authentifizierung, Anonymität und Unverfolgbarkeit des RFID-Tags zur Verfügung gestellt ist. Dies ist ein optimiertes anonymes Authentifizierungsschema, die Limitierungen eines RFID-Tags in Speicherbedarf und Berechnungs-Leistung angepasst ist. Das System wird auf Basis elliptischer Kurven-Kryptographie (ECC), die Public-Key-Kryptographie mit kleinen Schlüssel-Grössen beruht, entwickelt. Das ermöglicht, dass genug Speicher für Algorithmen-Berechnung vorhanden ist und damit wird die Ausführungszeit verbessert.

Diese Arbeit stellt die Umsetzung des DAA Unterschrifts Protokol für einen RFID-Tag dar. Der Signatur-Generation-Algorithmus wurde mit µKeil, einer Modellierungssprache für effiziente Low-Level-Programmierung in C, implementiert.

Dieser implementierte Algorithmus hat keine eingebaute Unterstützung für den Widerruf der Sicherheits-Parameter.

Prüfung und Evaluierung hat ergeben, dass dieser DAA-Schema erfolgreich ausgefürt wird und damit auch, dass es auf einem RFID-Tag implementiert werden kann. Mit diesem Ergebnis wurde ein grosser Schritt, um anonyme RFID-Kommunikation zu sichern, getan.

Stichwörter: RFID, Elliptic Curve Cryptography, Direct Anonymous Attestation, Digitale Signatur, Authentifizierung, Anonymität, Datenschutz

Abstract

Radio frequency identification (RFID) tags already exist for years. They are mostly passive devices which are used for contactless communication. RFID tags derive its power from the electromagnetic field of the RFID reader. The starting point of this thesis was Infineon's SLE78CLxxxP dual 16-bit security controller, which is certified $CCEAL6 + high/EMVCo$. That makes it ideal for highest security applications like digital signatures (e.g. ePassport, eHealth card, eDriver's License, eVisa, eSignature).

This thesis describes an implementation of a lightweight direct anonymous attestation (DAA) scheme, which provides authentication, anonymity and untraceability of RFID tags. This version is an optimized anonymous authentication scheme adapted for memory and computing constraints of the RFID tag. The scheme is based on elliptic curve cryptography (ECC) (which is public-key cryptography with small key-sizes). That results in a small memory requirements, which is important for tag's computation performance.

This work presents an implementation of the DAA signature generation protocol for RFID tags. The signature generation algorithm has been implemented using µKeil, a modeling language for efficient low-level programming in C. This scheme has no built-in support for revocation of security parameters.

Verification and evaluation has shown that the DAA algorithm is feasible for RFID tag.
With this result a big step to secure anonymous RFID communication has been done.

Keywords: RFID, Elliptic Curve Cryptography, Direct Anonymous Attestation, Digital Signature, Authentication, Anonymity, Privacy

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The importance of devices that use RFID and security technology has grown over the last few years. The reader communicates with the tag through electromagnetic field of radio waves, which is powering the tag. The communication is contactless, and the reader with the help of a software, which runs on a PC or a server connected to the reader, identifies the object. RFID tags have been embedded into different products like student cards, library books, health cards, citizen cards and even passports. Furthermore, the increased involvement of the tags in government and billing procedures increases need for secure communication.

A person and/or object who carries a device equipped with RFID tags is potentially traceable by an adversary. This would violate the person's privacy. Therefore, designing RFID protocol, which is privacy friendly, is a major concern before this technology becomes even more widespread. Privacy-friendly in this context means that the identity of the tags cannot be interferred from the protocol messages (i.e. anonymity), and that two tags cannot be linked (i.e. untraceability).

A possible privacy-friendly RFID protocol is Direct Anonymous Attestation (DAA). The DAA [2] algorithm plays an important role in privacy enhanced technologies. It allows an entity (e.g., a user, or a hardware device) to create a signature without revealing its identity. Anonymous signatures also enable anonymous entity authentication. There are various DAA algorithm implementations on Trusted Computing Platforms [13] or Java enabled platforms [8]. These implementations are made on standardized hardware, which is providing remarkable preformance in processing power and storage.

To run a complex DAA algorithm on a RFID tag, with passive power, several implementation optimizations need to be made. Reasearch showed that Elliptic Curve Cryptography (ECC) is well suited for such an optimization. It is based on difficult mathematical problems, so that ECC sytems with much smaller key size, provide equivalent security as the other existing public-key schemes. Due to short key length processing and memory requirements are much smaller. DAA based on ECC is very attractive for RFID tag implementation, where chip in processing power and storage is strongly limited.

The core component of Infineon's SLE78CLxxxP family RFID tag is a dual 16-bit processor. It has a Contactless Security Controller called Integrity Guard. This tag has purpose built co-processors for RSA/ECC and DES/AES operations, which are used to accelerate time-consuming cryptographic computations. For the implementation Infineon CryptoLibrary (CryptoLibSLE70) designed for this co-processors has been used.

This thesis presents an anonymous authentication scheme tailored to the resource constraints of RFID tags. Anonymous authentication means that RFID tag authenticates to Verifier V as a member of a DAA group created by an issuer I, but V does not learn the actual identity of the tag.

The DAA implementation was made on Infineon SLE78 RFID tag with CryptoLibSLE70 modeled in µVision (Keil), a modeling language based on C that combines source code editing, program debugging and complete simulation.

1.1 Thesis Outline

The thesis starts with a short introduction to number theory in Chapter 2. This chapter deals with symmetric and asymetric cryptography and their comparison, digital signatures and hash functions.

Chapter 3 builds up on the basics of public-key cryptography introduced in Chapter 2 and discusses the area of elliptic curves and ECC. Operations over elliptic curves and a comparison of different implementation variations are described here. Special attention is paid to finte fields arithmetic, which is necessary to understand ECC.

Chapter 4 deals with the theory of anonymous authentication. The concept of DAA and concrete DAA scheme based on RSA and Lightweight Anonymous Authentication Scheme for Embedded Mobile Devices are presented.

Chapter 5 gives an overview of the design flow for developing a software implementation. It starts with description of the actual state in hardware and software. The second part is about the parameters which were included in software implementation. Finally, the integration of the parameters and the software implementation are presented.

Chapter 6 presents the implementation results of the propsed protocol measured with the tag in the strong electromagnetic field. The Results are discussed and compared to preformance evaluation of the related work. Their possible solutions are then presented.

Conclusions and achievements of this work are drawn in Chapter 7 with suggestions for future work to improve the protocol implementation.

An appendix completes the work with a glossary and a bibliography.

Chapter 2

Theoretical Background

2.1 Cryptography

Cryptography is defined [7],[23] as the study of mathematical methodes to allow secure communication over a non-secure channel. Main goals related to aspects of information security are:

Confidentiality stands for secrecy and privacy. This means that no third party should be able to access the unencrypted information.

Data integrity is about the unauthorized change or alteration of data. It must be possible to detect manipulation of data by unauthorized parties. Manipulation can be the insertion, deletion or substitution of data.

Authentication and identification go hand in hand. Two different forms of authentication can be distinguished: Entity authentication is about proving ones authenticity using certain information about oneself to a second party. Origin authentication proves that the provided data/message/Information is from a certain sender.

Nonrepudiation prevents an entity from denying previous commitments or actions, In the case of a signed contract, this attribute makes sure that a party cannot back out, by denying that the contract was signed.

2.2 Symmetric cryptography

In this cryptographic scheme a secret is shared between communications partners. Its history goes back to ancient Egypt (1900 b.c.), over *Caesar cipher* and the *Vignere cipher*, which was the first to use passphrase for encryption. A special case of symmetric cryptography is a *one-time-pad*, which allows perfect security but the usability of this scheme is restricted. However, it is used in quantum cryptography where the key is exchanged over a photon beam and it is provably secure.

The *Kerckhoff principle* from 19^{th} century is still of great importance and it says that a security of cryptosystem must only rely on the secrecy of the keys (the shared secret) and never on the secrecy of used algorithm.

Nowadays the most used algorithm is probably the Advanced Encryption Standard (AES) wich was chosen by US National Institute of Standard and Technology (NIST) to replace the older Data Encryption Standard (DES).

Symmetric encryption still plays an important role in secure communication, because of the much higher performance as asymmetric cryptography.

2.3 Asymmetric Cryptography

Asymmetric or Public - key cryptography is called asymmetric because instead of a one shared key for all participants, each of them has a *key pair*. There is a public key, which is available to the public and the private key which is kept secret and altough different, the two parts of the key pair are mathematically linked. Public key is used to encrypt the plaintext for its owner and only owner of the corresponding private key can decrypt ciphertext and read the encrypted message.

Public - key cryptography uses mathematical functions also called trapdoor one-way functions. These functions presumably have no efficient solution. It is computationally easy to generate public and private key, encrypt plaintext and decrypt ciphertext, but it is infeasible for anyone to derive the private key.

Currently three classes of mathematical problems are used for asymmetric cryptosystems:

1. The integer factorization problem in RSA [25]
2. The dicrete logarithm problem in ElGamal [11]
3. The elliptic curve discrete logarithm problem (ECDLP) in ECC [7]

This thesis is based on ECC system. Only this system is described in detail in Chapter 3.

2.4 Comparison of Symmetric and Asymmetric Cryptography

At this point it is interesting to compare the two previously introduced cryptographic systems.

Advantages of symmetric - key cryptography:

- High performance implementations are possible
- Short key lengths
- Can be used as pseudo - random number generators
- Symmetric - key ciphers can be combined. This results in strong product ciphers.

Disadvantage of symmetric - key cryptography:

- The key must remain secret within all participating parties
- Too many keys: a lot of key pairs need to be managed in large network
- Need for secure channel for secret key exchange
- Cryptographic practice dictates that the key is changed frequently. If possible it should be even within communication sessions

Advantages of public - key cryptography:

- The private key must be kept secret by one entity only (authenticity)

- There is no problem of key distribution. Everyone publishes thier public keys
- The key pair can remain unchanged for long periods of time. This depends on the mode of usage.
- There are many efficient digital signature mechanisms. They mostly only require small public keys.
- The total number of required key pairs in a large network is much smaller then in a symmetric - key scenario.

Disadvantages of public - key encryption:

- In comparison to symmetric - key schemes, the pubic - key schemes are computationally much more intensive.
- The required key size is larger compared to symmetric - key encryption methods.

Optimally one should use the advantages of both worlds. Use the asymmetric - key cryptography for establishing a secure communication channel by sharing a secret key. Then use symmetric - key cryptography for the real communication and transfer of the data, bacause of the much higer performance.

2.5 Digital Signatures

Fundamental components on cryptography are digital signatures. They are used for signing documents and they are a method for providing authenticity, authorization and nonrepudiation. Therefore it can be determined whether a document was signed with a valid signature and if it was or was not changed after signing. A wide spread algorithm for digital signatures which is standardized by the National Institute of Standards and Technology (NIST) in 1991 is specified in the Digital Signature Standard (DSS) [7], [24] and called Digital Signature Algorithm (DSA). The purpose is to bind an entity to a piece of information. During the process of signing, some secret information held by the signing entity is used to generate a signature.

Digital signatures are perfect applications for public - key cryptography and the signing and verification procedure can be distinguished. The private key, one which is only known by the owner, is used by a digital signature algorithm to calculate the signature. The signature can now be sent with the signed document. The receiver uses the public key of the signer to check the validity and authenticity of this signed document.

One standardized algorithm based on ECC is Elliptic Curve Digital Signature Algorithm (ECDSA), which is the Digital Signature Algorithm (DSA)-analogue for elliptic curves.

2.6 Hash Functions

The book [1], [27] states that a hash function is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called hash - values. A hash value can be seen as a compact representation of an input string. During this compression, the number of bits (from input to hash) gets reduced. This means that it is theoretically possible to find two input strings that generate the same hash value. This is called a collision. Hash algorithms are designed in a way that it is hardly possible to find a collision. It should be also computationally infeasible to find an input x for predefined hash value y so that $h(x) = y$.

In cryptography, hash algorithms play a very important role for digital signatures and data - integrity checks. During a signature generation, not the whole message is signed, but the hash value of the message. So finding a message with the same hash value as the originally signed message should be computationally infeasible. To check the validity of a signature, the hash function must be publicly known.

According to [7], a hash function is an one-way function h which has at least two of these following properties:

- An input m of arbitrary finite bit-length is mapped by h to an output $h(m)$ with finite bit-length n (compression).
- Given the input m and h it should be easy to compute $h(m)$ (ease of computation). The following three conditions must hold if the hash function is to

be used for cryptographic applications.

- Given the output $h(m)$ of a hash function it should be computationally infeasible to find any input m' which hashes to the same output $h(m') = h(m)$ (preimage resistance).
- Given an input m and the output $h(m)$ of a hash function it should be computationally infeasible to find a second input m' which has the same output $h(m) = h(m')$ (2^{nd} preimage resistance).
- It should be computationally infeasible to find any two distinct inputs m and m' which hash to the same output $h(m) = h(m')$ (collision resistance).

Common representative of hash function is the SHA-1 algorithm (SHS) with an 160-bit output which is standardized by the NIST. There are also hash functions with an output of 256 bits (SHA-256) and 512 bits (SHA-512). For an in-depth discussion of hash functions see [27].

Chapter 3

Elliptic Curve Cryptography

The use of elliptic curves in cryptography was proposed first by Victor Miller [20] and Neal Koblitz [16] in 1985. Since back then the popularity for elliptic curves grew more and more. In the work [21] by Nils Gura et al. different RSA and ECC algorithms on embeded processors are compared. [7] states that a 224-bit elliptic curve and 2048-bit RSA algorithm have the same level of security. NIST P-224 curve arithmetic and RSA-2048 algorithm have been implemented on ATmega128 (8-bit processor). The code size of both algorithms is very similar, but the big difference is in the required size of data memory: RSA-2048 data memory has to be more than three times larger then the 224-bit elliptic curve memory. This clearly shows that the elliptic curve algorithms have advantage over other asymmetric algorithms, altough cryptographic operations relying on elliptic curves are more complex to calculate.

3.1 The Elliptic Curve Discrete Logarithm Problem (ECDLP)

The mathematical basis for the security of elliptic curve cryptosystems is the computational intractability of the Elliptic Curve Discrete Logarithm Problem (ECDLP) leading to smaller key-sizes which make elliptic curves attractive. The difficulty to solve ECDLP on E is the problem of finding an integer k (if such an integer k exists) such that $kP = Q$ for a given point $Q \in E$.

Q and P are points on the elliptic curve E wich is a group defined over a finite field $GF(p)$ and k is an integer. The operation kP is called point multiplication.

The calculation of the point multiplication over the curve $E(Fp)$ involves several mathematical operations. For an easier understanding of these calculations it is important to keep in mind that there are two levels of abstraction which must be strictly kept apart:

1. The higher level of abstraction with regard to the elliptic curve itself, the points on the curve and the operations which can be performed with these points.
2. The lower level of abstraction (the ”basic”) concerns the underlying algebraic structure (for cryptographic applications: a finite field) of the elliptic curve. Performing a point operation on the elliptic curve involves multiple operations in the underlying finite field which often is $GF(p)$ or $GF(2^m)$. Therefore, basics in finite field arithmetic are crucial for understanding elliptic curves and elliptic curve cryptography.

The remainder of this chapter is organized as follows. After a mathematical definition of elliptic curves, two different point representations are discussed along with elliptic curves over different (finite) fields and the operations defined on them.

Good introductions and references of the comprehensive area of application of finite fields, elliptic curves and ECC can be found in [19], [18], [17], [7].

3.2 Elliptic Curve Arithmetic

3.2.1 Weierstrass Equation

A Weierstrass equation which defines an elliptic curve over a field is a homogeneous equation of degree 3 of the form:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad a_1, a_2, a_3, a_4, a_6 \in K \tag{3.1}$$

This equation is behind every elliptic curve. For convenience, this equation is written using non-homogeneous (affine) coordinates ($x = X/Z$, $y = Y/Z$). There is exactly one point in E with Z -coordinate equal to 0, namely $(0 : 1 : 0)$. This point is called *point at infinity* and is denoted by O.

In order to ensure that the curve is *"smooth"*, there is also the discriminant $\Delta \neq 0$ to consider. Δ is the discriminant of E. It is defined as follows:

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

The smoothness of the elliptic curve E is necessary, so that there are no points on the curve with two or more distinct tangent lines.

As allready stated E is defined over K. K is called the underlying field and it is written $E(K)$. That is because the coefficients in Equation 3.1 are elements of K. Choices for underlying fields are prime fields $GF(p)$ and binary extension fields $GF(2^m)$. For both of the fields there are recommended parameters in the FIPS 186-3 standard [24].

An elliptic curve point is a tuple of integers (x, y), which fullfills the 3.1 and so lies on the curve. The Weierstrass equation can be simplified. For prime fields $GF(p)$ or for all fields with char $char(K) \neq 2, 3$ special form is obtained:

$$E : y^2 = x^3 + ax + b \quad a, b \in K \tag{3.2}$$

The constraint $char(K) \neq 2, 3$ must hold, otherwise there would be a division by zero in the equations used for performing the transformations. 3.1 represents an elliptic curve defined over $\mathbb{R}$ or $E(\mathbb{R})$.

The equation which defines a non-supersingular elliptic curve over a binary extension field $GF(2^m)$ is

$$E : y^2 + xy = x^3 + ax^2 + b \quad a, b \in K \tag{3.3}$$

In 3.2 elliptic curve over $GF(23)$ is presented

The points on an elliptic curve form a group. The group operations as point addition, point doubling are described in the following sections. The neutral element of the group is the point at infinity O.

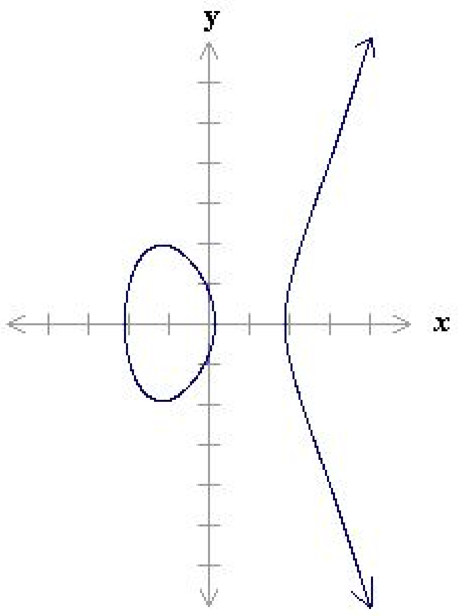

Figure 3.1: Elliptic curve over $\mathbb{R}$, $y^2 = x^3 - 4x + 0.67$

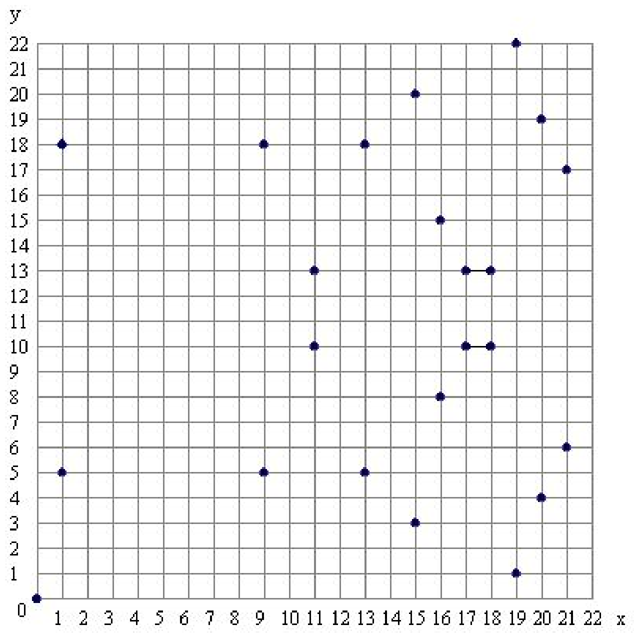

Figure 3.2: Elliptic curve over $GF(23)$: $y^2 = x^3 + x$

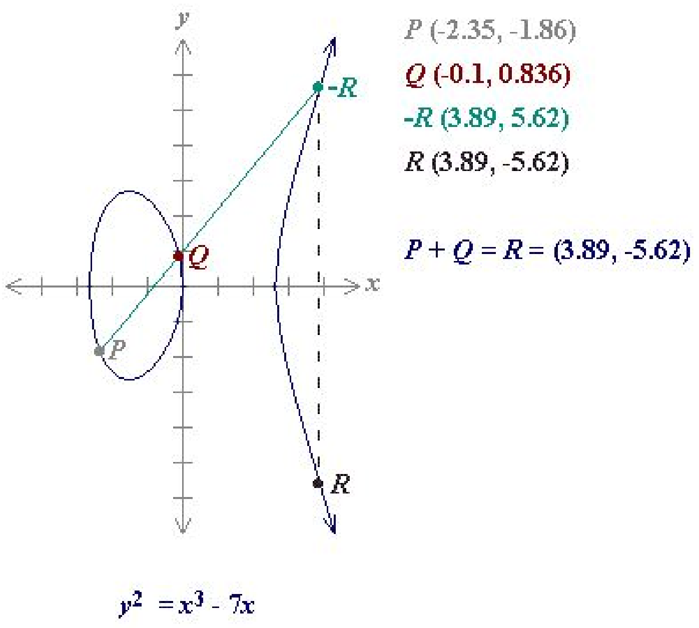

Figure 3.3: Graphical point addition $R = P + Q$

3.2.2 Point Addition

When $E(\mathbb{R})$ is an elliptic curve over $\mathbb{R}$ the addition of two distinct points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ is defined graphically as shown in 3.3. Addition is done by placing a straight line through the points P and Q which will be extended until it intersects the elliptic curve in exactly one point $-R$. The point of intersection is mirrored along the x-axis, which leads to the resulting point $R = (x_R, y_R)$. Performing the addition geometrically may be easier to remember. Unfortunately, no geometric approach exists for elliptic curves over $GF(p)$ and $GF(2^m)$.

Analytically, addition of P and Q is accomplished by

$$R = (x_R, y_R) = P + Q$$

which is calculated in the following way:

$$x_R = \lambda^2 - x_P - x_Q$$
$$y_R = \lambda(x_P - x_R) - y_P$$

With λ denoting the slope of the straight line:

$\lambda = \frac{y_Q - y_P}{x_Q - x_P}$

The equation for $GF(2^m)$ is:

$x_R = \lambda^2 + \lambda + x_P + x_Q + a$
$y_R = \lambda(x_P + x_R) + y_P$

with λ:

$\lambda = \frac{y_Q + y_P}{x_Q + x_P}$

This formulas cannot be used if x_P equals x_Q. Calculation of the addition always takes place in the underlying field. For $E(\mathbb{R})$ these are the real numbers leading to floating-point calculations.

3.2.3 Point Doubling

If the two points, P and Q, are equal ($P = Q$), no conventional point addition as described in the previous section can be performed because a straight line through P and Q is not defined (λ undefined). A doubling algorithm has to be applied instead. In this case the tangent to the curve at the point P is used instead and everything else is similar to the addition, as shown in 3.4.

Analytically, doubling of P is accomplished by

$R = (x_R, y_R) = P + P = 2P$

For prime fields $GF(p)$ is calculated in the following way:

$x_R = \lambda^2 - 2x_P$
$y_R = \lambda(x_P - x_R) - y_P$

with λ denoting the slope of the tangent to the curve at point P:

$\lambda = \frac{3x_P^2 + a}{2y_P}$

The equation for $GF(2^m)$ fields is:

$x_R = \lambda^2 + \lambda + a$
$y_R = x_P^2 + (\lambda + 1) + x_R$

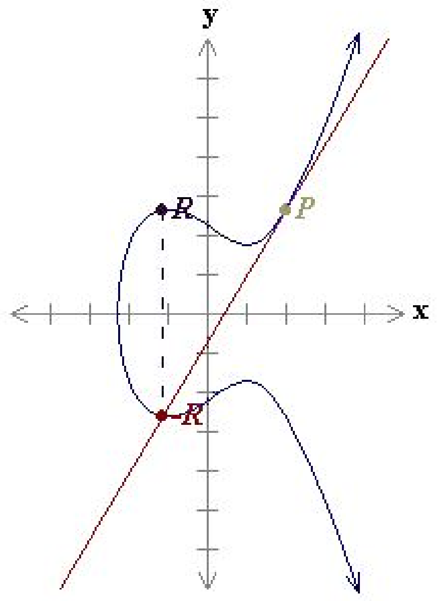

Figure 3.4: Graphical point doubling $R = 2P$

with λ:

$\lambda = x_p + \frac{y_P}{x_P}$

If $P = -P$ then the result is the point at infinity O for both prime $GF(p)$ and binary $GF(2^m)$ fields, what is represented in 3.5.

3.2.4 Point Multiplication

The point addition and doubling formula can now be used to derive a multiplication method. A multiplication method is in Algorithm 1.

The scalar k is scanned bitwise, and depending if the current bit is set, the intermediate result Q is only doubled or doubled and added to the base point P. If k is chosen randomly, on average t point doubling and $t/2$ point additions are required to calculate the point multiplication. Here, t is the number of bits of k.

In order to optimize Algorithm 1, the succeeding point addition and doubling can be merged into a single function that calculates both results at the same time.

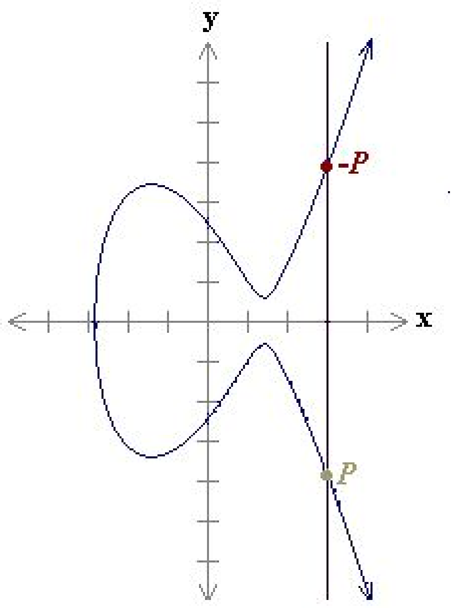

Figure 3.5: Graphical presentation of $P + (-P) = O$

Algorithm 1 Point Multiplication

```
Require: $k = (k_{t-1}, ..., k_1, k_0)_2, k_{t-1} = 1\ P \in E(F_p)$.
Ensure: $k \cdot P$.
1: $Q[0] \leftarrow P$
2: $Q[1] \leftarrow 2P$
3: for $i \leftarrow t - 2$ to 0 do
4:     $Q[1 \otimes k_i] \leftarrow Q[k_i] + Q[1 \otimes k_i]$
5:     $Q[k_i] \leftarrow 2Q[k_i]$
6: end for
7: return $(Q)$
```

Various optimizations can be done to reduce the number of elliptic curve operations preformed in the point multiplication. In [7] mentioned are window methods, non-adjacent form (NAF), simultaneous point multiplication, joint sparse form (JSF) and others.

For elliptic curves over binary fields $GF(2^m)$ the Montgomery Method is more efficient algorithm to calculate the point multiplication. For prime fields it is less efficient. Another advantage of the Montgomery method is that in every step both point addition and a point doubling are preformed. This gives a certain protection against side channel attacks, which use timing or power analysis to recover all bits of private integer k.

3.2.5 Finite Fields Arithmetic

A finite field or Galois field (GF) is a field $\mathbb{F}$, which contains a finite number of elements. The order ł$\mathbb{F}$l is the number of elements in $\mathbb{F}$ and it contains two operations $(+, \cdot)$. The finite field over which elliptic curve E is defined is a prime field $GF(p)$ or binary extension field $GF(2^m)$. Finite field compiles the following arithemtic operations:

1. $(\mathbb{F}, +)$, is an anbelian group with the neutral element 0
2. $(\mathbb{F}/0, \cdot)$ is an anbelian group with the neutral element 1
3. The distributive law $(a + b) \cdot c = a \cdot c + b \cdot c$ for $a, b, c \in \mathbb{F}$ holds true

The finite field is defined as the "normal" fields like $\mathbb{N}$, $\mathbb{Z}$ or $\mathbb{R}$, with the difference that its set is finite. The number of elements in field is called order of the group. Finite fields only exists if the order is a pk, with p as a prime number and $k \in \mathbb{N}$. If $k = 1$, the field is a prime field and if $k > 2$, the field is then called an extension field. Extension fields with an order of 2^m are called binary extension fields or characteristic - two finite fields. This two type of fields are most used in ECC applications.

Prime field elements can be represented by integers and the arithmetic is performed modulo the prime modulus p. With extension fields various representations exist.

3.2.6 Mathematical Operations

A prime field $GF(p)$ is a finite field with p elements, where p has to be a prime number, otherwise the elements only define a ring instead of a field. The basic mathematical operations applicable in the prime field $GF(p)$ are addition, subtraction, multiplication and inversion, which are performed modulo this prime p. Simple addition and multiplication algorithms result into majority of the runtime of the DAA Algorithm. For inversion algorithms, the reader is encouraged to read the Chapter 2.2.5 in the Guide to Elliptic Curve Cryptography [7].

3.2.6.1 Reduction

Given any integer z, then $z\ mod\ p$ (the integer remainder in the range [0, p - 1] after z is divided by p) is called the modular reduction of z with respect to modulo p. This can also be achieved by multiple subtractions.

Modular reduction is needed to ensure that results of prime field operations will stay in [0, p - 1] range. In general, modular reduction is an expensive operation, and should be avoided whenever possible. There are two prominent reduction algorithms available: Barrett and Montgomery reduction. Barrett reduction is a direct replacement for classical modular reduction, whereas Montgomery reduction is only useful when a sequence of multiple modular operations is calculated. Both are described in the book [7]. The Montgomery Multiplication is, as the name suggests, not a pure modular reduction, but a field multiplication wich includes efficient reduction. Details can be found in the following section.

The advanage of NIST primes, used as modulo are their special forms. They can be expressed as sum or difference of small numbers and powers of 2. The FIPS 186-3 standard defines several primes in the size of 192-bit to 521-bit [24]:

$$p^{192} = 2^{192} - 2^{64} - 1$$
$$p^{224} = 2^{224} - 2^{96} + 1$$
$$p^{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$$
$$p^{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$$

All the numbers are written this way for the efficient software implentations. The exponents are chosen to fit exactly multiples of 32-bit words. These primes, represented as exponents of 2, allow very efficient reduction: only a few subtractions and additions are needed.

This thesis uses the NIST - P256 parameters. Algorithm for the fast reduction, shows how the upper 256-bit of the multiplication must be added to the lower 256-bit in order to preform a reduction.

Algorithm 2 Fast reduction modulo $p^{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$

```
Require: An integer $c = (c_{15}, c_{14}, c_{13}, c_{12}, c_{11}, c_{10}, c_9, c_8, c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0)$ in
    base $2^{32}$, with $0 \leq c \leq p_{256}^2$.
Ensure: $c(mod\ p_{256})$.
 1: $s_1 \leftarrow c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0$
 2: $s_2 \leftarrow c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0$
 3: $s_3 \leftarrow c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0$
 4: $s_4 \leftarrow c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0$
 5: $s_5 \leftarrow c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0$
 6: $s_6 \leftarrow c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0$
 7: $s_7 \leftarrow c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0$
 8: $s_8 \leftarrow c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0$
 9: $s_9 \leftarrow c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0$
10: return $(s_1 + 2s_2 + 2s_3 + 2s_4 + s_5 - s_6 - s_7 - s_8 - s_9(mod\ p_{256})$
```

3.2.6.2 Addition and Subtraction

Operation called *addition modulo p* is $a + b = r$ for each $a, b \in GF(p)$ where r is the remainder of $(a + b)/p$. By replacing b with its inverse element $(-b)$ regarding addition leads to the definition of subtraction: $a + (-b) = a - b = r$ for each $a, b \in GF(p)$ where r is the remainder of $(a + (-b))/p = (a - b)/p$. This operation is also called *subtraction modulo p*.

The addition and subtraction algorithms are fast and efficient and a lot easier to implement then the multiplication algorithms, which are handled in the next section.

Algorithm 3 shows the addition procedure. To understand this Algorithm, very important is the carry propagation. When two W-wide words are added, the result has $W + 1$ bits. The extra, most significant bit is stored in the carry bit ϵ. The algorithm first adds a and b, and when the sum is larger than or equal to p, p is

Algorithm 3 Prime field addition in $GF(p)$

```
Require: Two integers $a, b \in [0, p-1]$ and a modulus $p$
Ensure: $c = (a+b)(mod\ c)$.
 1: $(\epsilon, C) \leftarrow A[0] + B[0]$
 2: for $i$ from 1 to $t-1$ do
 3:     $(\epsilon, C) \leftarrow A[i] + B[i] + \epsilon$
 4: end for
 5: if $\epsilon = 1$ or $c \geq p$ then
 6:     $(\epsilon, C[i]) \leftarrow C[0] - P[0]$
 7:     for $i$ from 1 to $t-1$ to 0 do
 8:         $(\epsilon, C[i]) \leftarrow C[i] - P[i] - \epsilon$
 9:     end for
10: end if
11: return $(C)$
```

subtracted from the intermediate result stored in c. In the case of the subtraction, preformed within lines 6 - 9, ϵ is used as a borrow bit.

3.2.6.3 Multiplication

Modular multiplication is the most important operation in an ECC, as it has the most significant impact on performance. It defines to a large extent the amount of resources needed for a hardware implementation. Multiplication is a heavy-weight operation that cannot be calculated in a single step, especially if a and b are of cryptographic size. It has to be split into several smaller operations. Modular multiplication $a \cdot b\ (mod\ p)$ can be calculated by conventional integer multiplication of a and b followed by a modular reduction. There is an algorithm for *MSB-to-LSB* or from *LSB-to-MSB* bit serial multiplication. The product $a \cdot b$ is calculated by multiplying the multiplicand a by the individual bits of multiplicand b.

3.2.6.4 Montgomery Multiplication

Peter Lawrence Montgomery presented a lot of different methods to improve ECC related operations. One of those improvements is a technique which allows implementing modular arithmetic s in prime field, so that all arithmetic operations can be executed in the Montgomery domain. We can multiply two numbers and reduce them at the same time, by introducing a third term (r):

$c = a \cdot b \cdot r^{-1}(mod\ p)$

There are two requirements for r:

- $r > p$
- $gcd(r, p) = 1$

Usually p is prime and r a multiple of 2, so that those requirements are fullfilled by default.

This method is even more effective when a lot of multiplications are needed. Only initial multiplication with r need to be done.

$\tilde{a} = a \cdot r\ mod\ p$ and $\tilde{b} = b \cdot r\ mod\ p$, so that montgomery multiplication results in:

$Mont(\tilde{a}, \tilde{b}) \equiv \tilde{a} \cdot \tilde{b} \cdot r^{-1}(mod\ p)$
$\equiv (a \cdot r) \cdot (b \cdot r) \cdot r^{-1}(mod\ p)$
$\equiv (a \cdot b) \cdot r\ (mod\ p)$

The result is also multiplied with r. To recover the result, a multiplication with it is needed:

$Mont(\tilde{a}, 1) \equiv \tilde{a} \cdot 1 \cdot r^{-1}(mod\ p)$
$\equiv (a \cdot r) \cdot 1 \cdot r^{-1}(mod\ p)$
$\equiv a\ (mod\ p)$

For the Motgomery multiplication implementation, an extra parameter (p') is needed, which can be calculated using the extended Euclidean algorithm . It is defined as $r \cdot r^{-1} - p \cdot p^{-1} = 1$. Algorithm 4 is known as Finley Integrated Product Scanning Form (FIPS) from the paper [12]. For the multiplication only the storage of the lower W bits of p' are needed ($p'_0 = p\ (mod\ 2W)$).

Algorithm 4 FIPS Montgomery Multiplication

Require: Two integers $a, b \in [0, p-1]$ and precomputed p'_0

Ensure: $c = a \cdot b \cdot r^{-1} (mod\ p)$.

```
1:  $ACC \leftarrow 0$
2:  for $i$ from 1 to $t-1$ to 0 do
3:      for $j$ from 0 to $i-1$ to 0 do
4:          $ACC \leftarrow ACC + A[j] \cdot B[i-j]$
5:          $ACC \leftarrow ACC + C[j] \cdot P[i-j]$
6:      end for
7:      $ACC \leftarrow ACC + A[i] \cdot B[0]$
8:      $C[i] \leftarrow ACC[0] \cdot p'_0\ (mod\ 2W)$
9:      $ACC \leftarrow ACC + C[i] \cdot P[0]$
10:     $ACC \leftarrow ACC >> W$
11: end for
12: for $i$ from $t$ to $2t-1$ do
13:     for $j$ from $i-t+1$ to $t-1$ to 0 do
14:         $ACC \leftarrow ACC + A[j] \cdot B[i-j]$
15:         $ACC \leftarrow ACC + C[j] \cdot P[i-j]$
16:     end for
17:     $C[i-s] \leftarrow ACC[0]$
18:     $ACC \leftarrow ACC >> W$
19: end for
20: if $\epsilon = 1$ or $c \geq p$ then
21:     $c \leftarrow c - p$
22: end if
23: return $(c)$
```

Chapter 4

Direct Anonymous Attestation (DAA)

Anonymous credential systems [2], [5], [3] enable the authentication without disclosing users identity. In an anonymous credential system, users authenticate themselves by proving the possession of credentials from a credential issuer. However, without additional countermeasures, users could copy and share their credentials. Then the unauthorized users could access services. A service provider cannot detect if a credential has been copied, and credentials cannot be revoked. Therefore, security measures are necessary to protect authentication secrets. Since anonymous authentication introduction by Chaum [5], various anonymous credential systems have been studies and proposed. For this thesis, the Camenisch- Lysyanskaya (CL) credential system [3] is of particular importance since it is the basis for most of DAA schemes. Several variants of CL credentials exist, which are based on the strong RSA assumption [5], [9] or on pairings over elliptic curves [4].

The concept of DAA, and a concrete scheme, were first introduced by Brickell, Camenisch, and Chen [2]. For a historical perspective, the reader is encouraged to read [9]. This RSA-based Direct anonymous attestation (RSA-DAA) [2] is an anonymous credential scheme, which was adopted by the Trusted Computing Group (TCG) and included in version 1.2 of the TPM specification [13]. DAA has been specified by the TCG [13]. DAA is a remote authentication method for a special module, called the Trusted Platform Module (TPM). A TPM can be authenticated without recovering the privacy of the user, who uses the harware with a TPM platform. TCG is an industry standardization body that aims to develop and promote an open industry standard for trusted computing hardware and software to enable more secure data storage, online business practices, and online transactions while protecting privacy and individual rights.

Anonymity of a DAA signature is not revocable, because it is a group signature, which cannot be opened. The user of a device in agreement with the recipient, can decide if the signatures will be linkable. To do so one has to use pseudonyms, which are available in DAA. Even more, DAA allows device revocation. If any device has been compromised, i.e. the key from TMP has been extracted and published, a verifier can detect a rogue signature, which was computed with a compromised key. Compromised keys are known and available in the revocation lists.

The computation of the DAA protocols is split between the resource-constrained TPM chip and the software running on the platform, e.g. PC. The TCG specified a Mobile Trusted Module (MTM) [18] for mobile devices with optional support for DAA. However, RSA-DAA is complex and computationally intensive, and thus not suitable for the hardware protection mechanisms of mobile embedded devices such as smartcards, or special-purpose processor extensions with very limited computational power and memory capacity.

The scheme that retain the same functionality as RSA-DAA but is more efficient is the DAA scheme based on elliptic curves and pairings. The advantage of using DAA based on ECC is obvious: both the key and signature length can be much shorter, and computational load placed on the TPM less severe. As a result, DAA based on ECC is typically more efficient in computation, storage and communication cost than RSA-DAA [28], [8], [21].

In this thesis, an optimized adaptation of an existing elliptic curve and pairings based lightweight anonymous authentication scheme for RFID tag and its implementation is presented. It is based on lightweight anonymous authentication scheme for mobile embedded devices on a common mobile hardware platform presented in [28]. This scheme is tailored to the hardware resource constraints and it provides anonymity, untraceability and secure device authentication. The TPM is a secure microcontroller or an additional security extension of a processor, which provides a concrete set of standardized functions. TPM on Infineon SLE78CLxxxP familiy RFID tag is not available, but RSA/ECC co-processor and Secure Memory (EEPROM) are available and those will be used for, in TCG standardized, TMP operations.

This chapter explains the anonymous attestation scheme, as the method for remote authentication of a device, while preserving the privacy od the user of the device. In the first part, definitions and formal specifications are given. In the second part of this chapter, lightweigt anonymous authentication scheme, with its advantages, is presented.

4.1 Introduction to DAA

Assume that the user of trusted hardware module (TPM) which is integrated into a platform such as a laptop or a mobile phone, communicates with a verifier who wants to be assured that the user indeed uses a platform that can be trusted, i.e., the verifier wants TPM to authenticate itself. This problem is called remote attestation.

On the other hand, privacy of the user has to be protected. The verifier should only know that the TPM and its key are not corrupted (valid), but it cannot know which TPM the user has. If that was the case, all signatures of the user, who used one TPM would be linkable to each other. This problem can be solved using any public-key signature (or an authentication) scheme. Every asymmetric cryptography scheme computation has a key pair as a result: a secret key, which is embedded into each TPM and a public key. Both keys are then used for authentication protocol, which is used by a verifier and a TPM platform. The problem is, that each TPM would use the same private key and in that case if one TPM with its key is compromised, every other TPM with the same key would be compromised too. That means, the TPM would be indistinguishable and the verifier using a revocation list could no longer distinguish between honest TMPs and the fake ones. That is why a detection of rogue TPMs needs to be a further requirement.

The solution first developed by TCG uses a trusted third party, privacy certification authority (Privacy CA). A key pair generated using a RSA for each TPM is called an Endorsment Key (EK) and all EKs of (valid) TPMs are known to the Privacy CA.

A TPM can generate a second key pair called an Attestation Identity Key (AIK), to identify itself to a verifier. By sending the AIK public key w.r.t. the EK to the Privacy CA, TMP authenticates itself, by authenticating the AIK. If the EK is valid, the Privacy CA issues a certificate to the TPMs AIK. Verifier can verify the signature w.r.t. AIK and the issued certificate. More detailed description is in [3].

This solutions has the obvious drawback that the Privacy CA needs to be involved in every transaction. If the Privacy CA and the verifier collude, or the Privacy CA's transaction records are revealed to the verifier in any other way, the verifier will still be able to uniquely identify a TPM.

4.2 Formal Specification of Direct Anonymous Attestation

This section provides the formal model of DAA using real-system/ideal-system as in [10]. In the real system cryptographic protocol run a number of players. They are an Adversary A, who controls some of the players, and an Environment E, which provides the input data to the legitimate players. The E receives an output of the legitimate players, too and interacts arbitrarily with all dishonest players, wich are assumed to be the A.

Same players are available in the ideal system. Only difference is that they do not run cryptographic protocols, but receive all they input and output data from an ideal all-trusted party T. This player uses the other players inputs and computes the output, realizing the functionality of a cryptographic protocols which are used in a real system.

A cryptographic protocol implements securely a functionality for every adversary A and every environment E, if there exists a simulator S, which is controlling the same parties in the ideal system as A does in the real system. By using S, it is possible that E cannot know whether it is run in the real or ideal system. That means, E does not know if it interacts with A (from real system) or if it interacts with the S

In the functionality of DAA the following kinds of players can be distinguished:

- issuer I,
- trusted platform module (TPM) M_i with identity ID_i,
- host H_i that has TPM M_i "built in"
- rogue detection oracle O notifying which TPMs should be revoked,
- verifiers V_j.

The issuer is a trusted third party, a Privacy Certification Authority (Privacy CA). It grants certificates to users, allowing them to authenticate themselves towards a verifier. A device consists of a TPM and a host, which are both needed for authentication. The host is not able to authenticate without being connected to a valid TPM. Just together, the host and TPM can authenticate by proving they have a certificate and know all the secret values the certificate was built on.

A verifier is the entity to whom the device wants to authenticate. Uptil now it is clear that, before the device can authenticate towards a verifier, it first has to obtain valid credentials from an issuer and to do so, it cannot be on a rogue detection list.

In Figure 4.1 this description of the DAA scheme, showing the different entities involved, as well as the protocols executed between them is visualised.

In the following specification ,a counter value *cnt* allows TPM to generate multiple DAA keys from a sigle secret. A basename *bsn* is used for the property of a possible link between multiple DAA signatures signed under the same DAA key. Basename *bsn* is controlled by a signer and a verifier. By *Join*, the procedure in which the TMP gets issued an anonymous certificate is denoted. Then it *"joins"* the group of certified or attestes TPMs. In *DAA-Sign/Verify* procedure, the TPM and its host H_i can convince a verifier that the TMP is certified. The word *"sign"* is used as

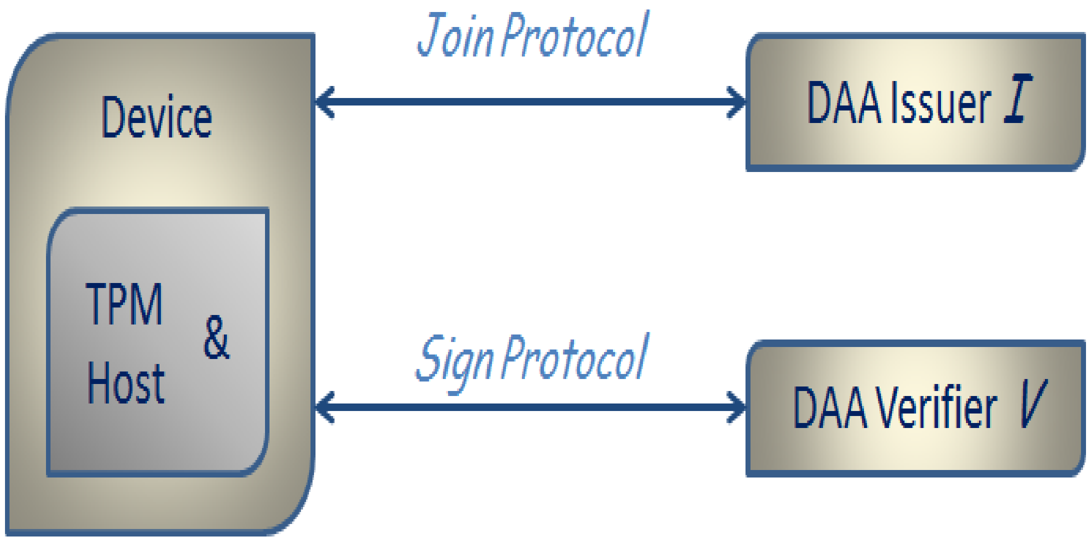

Figure 4.1: Overview of the DAA protocol with the different entities and protocols involved.

the verifier gets the result of the procedure, a piece of information that can be used as a proof that he or she was communicating with certified TPM. It can also be used to sign messages, in particular, AIK generated by the same TPM. The *rogue tagging* operation corresponds to the event when someone finds a TPM's DAA keys and want to publish those as invalid. The ideal system supports these operations:

Setup Each player indicates to issuer I whether or not it is corrupted. Each M_i sends its ID_i to T who forwards it to the respective host H_i.

Join Join protocol runs between a device and the issuer to build a certificate. The host H_i contacts I and requests to become a member with respect to a counter value *cnt*. I sends the corresponding M_i the counter value *cnt* and asks if it wants to become a member w.r.t. *cnt*. If I agrees, and if M_i was not tagged rogue w.r.t. some counter value, H_i can become a member.

DAA-Sign/Verify This protocol runs between a device and a verifier where the device performs a proof of knowledge. A host H_i wants to sign a message m for V_j with respect to some basename *bsn* and some counter value *cnt* for verifier V_j . H_i sends m, *bsn* and *cnt* to I. If H_i/M_i are not "joined" (a member), then I denies the request w.r.t. *cnt*. If it is not the case, I sends a message m and *cnt* to the device with M_i on it and asks whether it wants to sign. If yes, I asks it with respect to wich basename *bsn* is M_i signing, or if it wants to abort. If the platform with TPM decides to sign, I proceeds as follows:

- If M_i has been tagged rogue with respect to counter value *cnt*, I lets V_j know that a rogue TPM has signed m.
- If signature is done w.r.t. no basename then I informs V_j that m has been signed w.r.t. *bsn*.
- If signature is done w.r.t. *bsn* then I checks if it is signed from a H_i/M_i and if the message m is signed w.r.t. *bsn* and *cnt*. Then I chooses the corresponding pseudonym P from its database. If the m is not signed w.r.t. *bsn* and *cnt*, I looks up a new random pseudonym choosing corresponding security parameters. At the end, I informs V_j that a platform with a valid TMP signed m using a pseudonym P.

Rogue Tagging: A Oracle O informs I that he platform with identity ID_i with respect to *cnt* as a rogue. If the TPM with ID_i is not corrupted, I denies the request. Otherwise, I marks the M_i with ID_i as rogue w.r.t. counter value *cnt*.

The ideal system model captures unforgeability and anonymity/pseudonymity. A signature can only be produced with the platform which has a valid TPM built in. That means, that the TPM is not tagged roggue and it has to be a member ("join"). The signing w.r.t. the same basename, which was done by the same TPM can be and are linkable to each other through a pseudonym P. By signing a message with different basenames or, when TPM does not use any basename during signing, the signatures cannot be linked. These properties stand, even if the host H_i wich has TPM buit in is corrupted. But only if both, the host H_i and the platform M_i which is built in are honest, anonymity/pseudonymity can be guaranteed. Otherwise not, because a dishonest party can announce its identity.

4.3 Lightweight Anonymous Authentication Scheme for Embedded Mobile Devices

Due to resource constraints of mobile devices, a lightweight anonymous authentication scheme was introduced in [28]. This scheme optimizes and adapts DAA scheme from [6]. This protocol has features which are important for practical applications. Device can authenticate to a verifier without sharing any informations which could reveal its identity or to allow to a verifier the tracking of a device. It ensures that adversaries cannot impersonate legitimate devices to a honest verifier. And the protocol supports revocation list which makes roggue tagging anf it is there for revocation of authentication credentials.

A formal security model for anonymous authentication of mobile devices is a random oracle model. The scheme is secure under the decisional Diffie-Hellman (DDH), the discrete logarithm (DL) and the bilinear LRSW assumption, which is a dicrete-logarithm-based assumption proposed in [8]. For the detailed security aspects of this scheme and proof-of-concepts, the reader is encouraged to read [28] from page [4] to page [15] and [22].

How the lightweight anonymous authentication scheme works, is presented in Figure 4.3.

Verifier V_j outsources the accounting and billing, with subscription and other fees, to the issuer I. To access service, device R_i has to subscribe this service at I, which then issues an anonymous credential cred for R_i on behalf of V_j. R_i can now use cred to anonymously authenticate to V_j and get access to services provided by V_j. The requirements to an anonymous authentification are:

- *Correctness*: Users with valid credentials must be able to (anonymously) authenticate to the Verifier V_j.
- *Unforgeability*: Users must not be able to forge an authentication, i.e., they

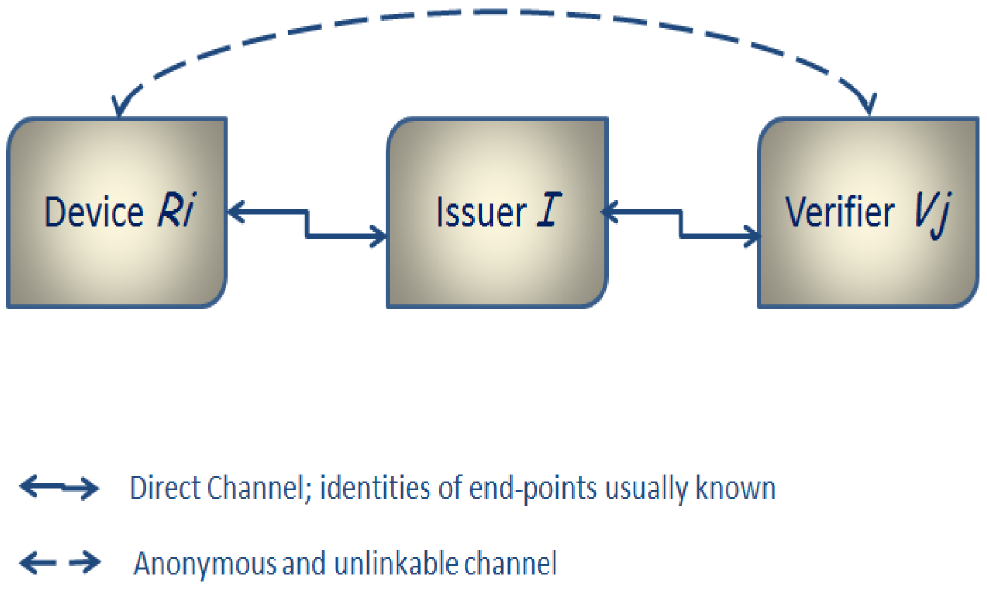
Device *Ri*
Issuer *I*
Verifier *Vj*
Direct Channel; identities of end-points usually known
Anonymous and unlinkable channel

Figure 4.2: Anonymous authentication scheme communication model.

must not be able to authenticate without having obtained a valid credential.

- *Unclonability*: Valid credentials cannot be copied (cloned).
- *Unlinkability*: Sessions must be unlinkable (full anonymity).
- *Revokability*: It must be possible to revoke users.
- *Practicability*: All protocols should be efficient and based on well-established standards. The implementation should be fast and based on widely used soft- and hardware.

4.3.1 Protocol Specification

The players in the scheme are a credential issuer I, a set of verifiers V_j, and a set of devices R_i. This anonymous authentication scheme is a three party protocol that is executed between a verifier and a device, which is composed of a semi-trusted Host H_i and a Secure Component S_i [14], [28].

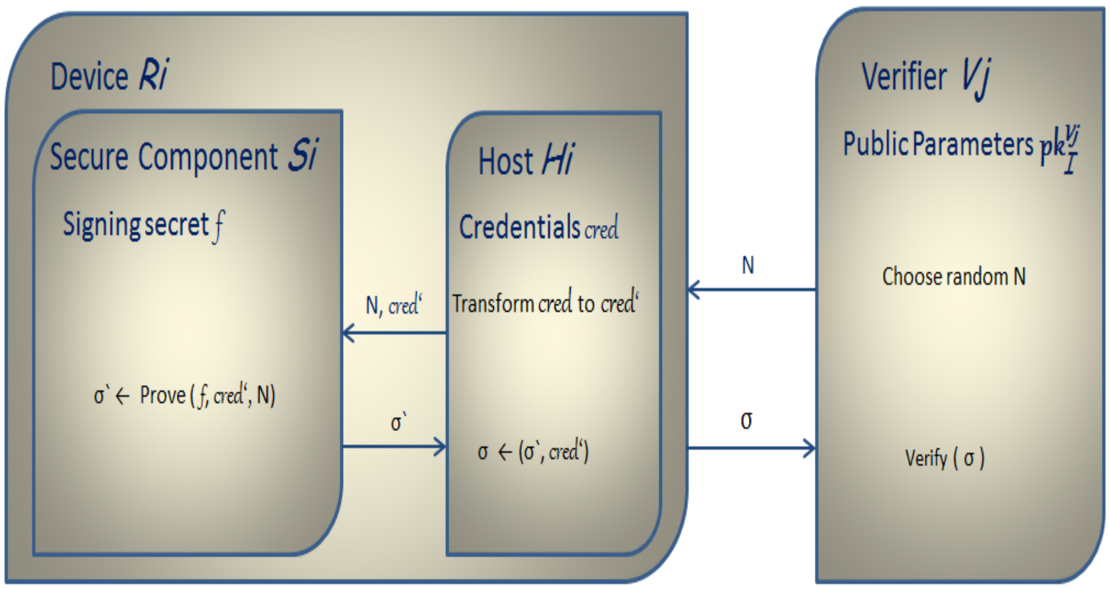

Figure 4.3: Protocol owerview.

The goal of the protocol is to authenticate R_i to V_j, so that V_j only learns that R_i is legitimate without allowing V_j to identify or trace R_i. R_i is called legitimate if it has been initialized by issuer I. The goal of this protocol is to split the computations which have to be preformed by R_i between S_i and H_i. In S_i, at least all security critical operations are preformed and if there is sufficient computing power, then all computations can be preformed by S_i. H_i preforms all privacy-related computations.

In Figure 4.3 is a protocol overview.

System initialization With a security parameter $l = (l_q, l_h, l_e, l_n) \in \mathbb{N}^4$ the issuer I generates the secret key sk_I and the corresponding public parameters pk_I^V, which is associated with verifier V_j and the revocation list RL. I uses *GenPair* [28] algorithm, which generates three Groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ of large prime exponent $q \approx 2^{lq}$, and their generators P_1 and P_2, and an admissible pairing $e : \mathbb{G}_1 x\ \mathbb{G}_2 \rightarrow \mathbb{G}_T$. The secret key of I, sk_I are two secrets $x, y \in \mathbb{Z}_q$. By multiplication of the secret with generator of a $\mathbb{G}$, I generates $X \leftarrow xP_2$ and $Y \leftarrow yP_1$, with P_1 as a generator of $\mathbb{G}_1$ and P_2 a generator of $\mathbb{G}_2$ and it chooses a collision resistant one-way hash function *Hash*. At the end, issuer I initializes the revocation list $RL \leftarrow 0$. The public parameters pk_I^V are: l, q, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T, P_1, P_2, e, X, Y$, $Hash$ and RL.

Device initialization The issuer I generates a secret signing key f and a corresponding anonymous credential $cred = (D, E, F, W)$ on behalf of V. That means, I chooses $f, r \in \mathbb{Z}_q$ and generates $D \leftarrow rP_1$, $E \leftarrow yD$, $F \leftarrow (x + xyf)D$ and $W \leftarrow fE$. Finally, the device R_i is initialized with secret signing key f and credentials *cred*. The signing key f has to be securely stored in Secure Component S_i. The credential *cred* can be stored in and used by the (semi-trusted) host H_i of R_i.

Anonymous authentication A mobile device R_i anonymously authenticates to a verifier V in this protocol, as shown in 4.3. The verifier V challenges R_i to sign a random message N. After the device receives N, H_i randomizes the credential *cred* to *cred*′ and computes the hash digest h of *cred*′. After computation, H_i passes to the secure component S_i the result of hash digest h, the value E' of *cred*′ and N. S_i then computes a signature of knowledge σ' in a similar way as in [6] and returns it to H_i. The host H_i composes the final anonymous signature σ $(\sigma', cred')$ and sends it to V.

Upon receipt of σ, V verifies if *cred*′ has not been revoked, if *cred*′ is a valid (randomized) credential w.r.t. pk_I^V , and if σ' is a valid signature of knowledge on N w.r.t. *cred*′ and pk_I^V . If the verification is successful, then V accepts R_i as a legitimate device and returns 1. Otherwise V rejects R_i and returns 0.

Device revocation To revoke a device R_i, I adds the signing secret f of R_i to the revocation list RL, and sends the updated revocation list RL to V using an authentic channel.

This lightweight DAA protocol for mobile devices [28] has been implemented in C on ARM TrustZone. For all cryptographic functions and operations MIRACLE (Multiprecisional Integer and Rational Arithmetic C/C++ library) [15] crypto library has been used. The preformance evaluation will be discussed in Chapter 6.

4.3.2 Pairings

Elliptic curves suitable for constructing pairings are called pairing-friendly elliptic curves. For practical purposes, curves with small embedding degrees are suitable, because they guarantee the trade-off between efficiency and security. This leads to the use of supersingular curves. Supersingular curves have been proven to be the most eficient curves for pairing [26]. Barreto and Naehrig [1] defined a new type of pairing-friendly curves which are known as BN-Curves. These are elliptic curves of the form:

$$E : y^2 = ax^3 + ax + b$$

for $a = 0$ and $b \neq 0$, where the curve order and the finite field are defined by the polynomials $q(s)$ and $p(s)$:

$$\begin{aligned} q(s) &= 36s^4 - 36s^3 + 18s^2 - 6s + 1, \\ p(s) &= 36s^4 - 36s^3 + 24s^2 - 6s + 1. \end{aligned}$$

To generate such curves, random values of s of the correct form have to be searched, until $q(s)$ and $p(s)$ are both prime. Based on this curve groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are selected. Using these groups, the pairings over elliptic curve is implemented. BN-curves are most suitable for 128-bit security, corresponding to 3.072-bit RSA and 256-bit ECC.

Chapter 5

Direct Anonymous Attestation Implementation

In this chapter, all the details of the first implementation of a (simplified) lightweight DAA protocol suitable for RFID tags is presented. It includes the complete descriptions of two new functions and their classes including their properties. These functions implement the computations on a RFID tag with pairing friendly types of curves.

Because a RFID tag contains a small chip with limited resources and it has no TPM, a requirement for my implementation is that the operations, which are computed on the tag, are minimal. The Infineon SLE78CLxxxP chip has a dual 16-bit processor, RSA/ECC co-processors and an EEPROM memory unit, which can be used for securely storing of the signing key. Based on protocol presented in [28], a DAA scheme suitable for a RFID tag is implemented. Of course, security must be maintained, but the security aspects of this scheme are out of scope of this thesis.

5.1 Analysis of Actual State

5.1.1 Hardware

The starting point of this thesis were Infineon's SLE78CLxxxP dual 16-bit security controllers, which are equipped with a security concept called Integrity Guard, which provides a security controller with comprehensive error-detection capabilities and full encryption along the entire the data path, including the two CPUs, memories (EEPROM, Flash, ROM and RAM), caches and buses. User RAM (000000-001FFF) has 8kb.

The core of these tags are two cetral processing units (CPU). They check one another continuosly. They cross-check all arithmetic operations and detect if they are executed correctly. If an attack or error is detected, the security controller triggers an alarm and it aborts all operations. Another advantage of the Integrity Guard is that it enables that all computations are made with encrypted data. Usually, processing of data on a conventional security controllers, can only be made on decrypted data. This fact is targeted in some attack scenarios. Due to Integrity Guard, this vulnerability is eliminated. All security sensitive operations are encrypted durinng processing and transmission along the whole data path.

The tag is equipped, among many other modules, with two co-processors. These hardware modules accelerate DES/AES operations for symmetric cryptography and support RSA up to 4096-bit and ECC up to 521-bit operations used in asymmetric cryptography. The Infineon SLE78CLxxxP is also certified $CCEAL6 + high/EMVCo$, which makes it ideal for highest security applications like banking and digital signature (e.g. ePassport, eHealth card/eSocial card, eDriver's License, eVisa, eResidence Permit, Vehicle Registration Card/eCar Registration, eSignature).

5.1.2 Software

The software used for implementation is Infineon's SLE70 CryptoLibrary, which can be differentiated into: the cryptographic libraries with implemented RSA, ECC and Hash functions and library's Toolbars. Also avilable for this thesis is a software for Issuer with its credentials. In ECC Library there are different parameters, classes and RSA and ECC functions. DAA scheme implementation uses some of them, which are described in detail in Section 5.6.

The CryptoLibrary is using a `CLONG` structure to represent very long unsigned integer numbers. This coding scheme is used internally as well as externally. A `CLONG` object always contains an unsigned integer, with maximum of `CLONG`.BitLength bits.

The significant bitlength corresponds to the `CLONG.BitLength` value minus number of leading zero bits. Therefore, the `TrueBitLength` value depends on the content of the currently represented integer number. All `CLONG` parameters of a High Level API are checked using `IsValidInputClong` or `IsValidOutputClong` for validity.

The library handles the local variables via the stack. The overlaying function of the Keil compiler for local parameters is not used. If a function requires a large working memory area (> 50 bytes) to store long parameters temporarily (e.g. randomized key parameters), the `CLONG` parameter has to be provided, which is giving a reference to the temporary working area. Before proceeding with the description of the implementation, a general issue that needs to be considered throughout has to be noted. Specifically, every group element received by any entity needs to be checked for validity, i.e., that it is within the correct group. In particular, it is important that the element does not lie in some larger group which contains the group in question. This strict stipulation avoids numerous attacks such as those related to small subgroups. In CryptoLibrary available operations `IsValidInput` and `IsValidOutput` check if the ECC parameter are valid given ECC equation, discussed in Section 5.6.

5.2 Goals to Achieve

The goal of this thesis is to create a fully working implementation of DAA on a RFID tag. The main requirement for a practical implementation of the (lightweight) DAA scheme is an acceptable execution time. This is opposed to a RFID tag's limited resources, in particular computational performance, so that the execution time can be achieved. The DAA protocol require complex computations on very large numbers (long integers). That is why efficient low-level programming in C was necessary.

For an implementation, several complex (cryptographic) operations are needed: secure random number generation and cryptographic hashing, which are directly available through the cryptolibrary. Large number multiplication and modular multiplication are available as well. For effective implementation ECC parameters together with best suitable functions have to be chosen.

The main challenge is to save public and secret parameters and to implement the scheme, so that the execution time is not longer than 450ms.

5.3 Elliptic Curve Cryptography

To implement the scheme, the curve parameters a and b, and domain parameters must be agreed on by both parties involved in secured and trusted communication using ECC. The domain parameters for prime fields are described below. The generation of domain parameters is out of scope of this thesis. Generally the protocols implementing the ECC specify the domain parameters to be used. There are several standard domain parameters defined by NIST [24].

Domain parameters for ECC over field $GF(p)$ The domain parameters defining an elliptic curve over $GF(p)$ are p, a, b, $\mathbb{G}$, n and h. p is a prime number of finite field $GF(p)$. a and b are the parameters defining the curve: $y^2 \ mod \ p = x^3 + ax + b$ $mod \ p$. $\mathbb{G}$ is the generator point, i.e. a point on the elliptic curve chosen for cryptographic operations. n is the order of the elliptic curve. The scalar, which is used in a point multiplication, have to be chosen as a number between 0 and $n - 1$. h is the cofactor of the elliptic curve.

For an efficient hardware implementation, the curves over $GF(2^m)$ are used more often. Two main advantages regarding the Binary Finite Field $GF(2^m)$ are that the bit additions are performed *mod* 2 and represented in hardware by simple XOR gates and the bit multiplications are represented in hardware by AND gates. The main drawback of $GF(p)$ in hardware implementation is the carry propagation for addition which results in a long critical path. For $GF(2^m)$ no carry chain is requred because addition is performed by a bit by bit XOR operation. For software implementations, which is the case in this thesis, $GF(p)$ fields are better suited, because most processors support integer multiplication efficiently.

In this thesis a pairing-friendly curve must be implemented, so that the computations with given issuer credentials could be fulfilled.

Affine coordinates Affine coordinates help to reduce the size, as they do not need to store a third coordinate during calculation, like projective coordinates. A point is directly represented by the X and Y coordinates as integer numbers. This method gives an initial advantage when thinking of the final computation and memory allocation on the tag. Projective coordinates need a different amount of field operations for a point addition and a point duplication. To be as fast as projective coordinates, the overall operation cost for a point operation should be the same in affine coordinates. The computation time for a field multiplication is the same for both coordinate systems. Using affine coordinates instead of projective coordinates results in a smaller code size because calculation with affine coordinates take less steps than calculation with projective coordinates.

Based on these facts, parameters for DAA scheme implementation on RFID tag are chosen. All mentioned points are interesting for a small and efficient firmware implementation of elliptic curve operations as they are desired in this thesis.

5.4 Free choice of domain parameters

Choosing appropriate parameters for pairing-based cryptography is still an active area of research. Both prime and binary extension fields are used in real world applications. The implementation of pairing based ECC solutions is highly dependent on the problem being solved, the implementation platform and the level of security intended to be achieved. For efficient implementation of ECC, it is important for the point multiplication algorithm and the underlying field arithmetic to be efficient.

Selection of a proper coordinate system also affects the performance of ECC system as computational cost of addition and multiplication operations depends on the coordinate system used. All feasible domain parameters for elliptic curves are supported by this device and CryptoLibrary. The criteria for choosing curve parameters is discussed above. The chosen pairing-friendly elliptic curve has the form $y^2 = x^3 + 3$ over a prime field.

5.4.1 Hash Function

In principle, any collision-resistant hash function is suitable for use in the implemented DAA protocol. To facilitate interoperability, the SHA-256 with 256-bit output has been recognized as best suitable. This hash function is defined in [24].

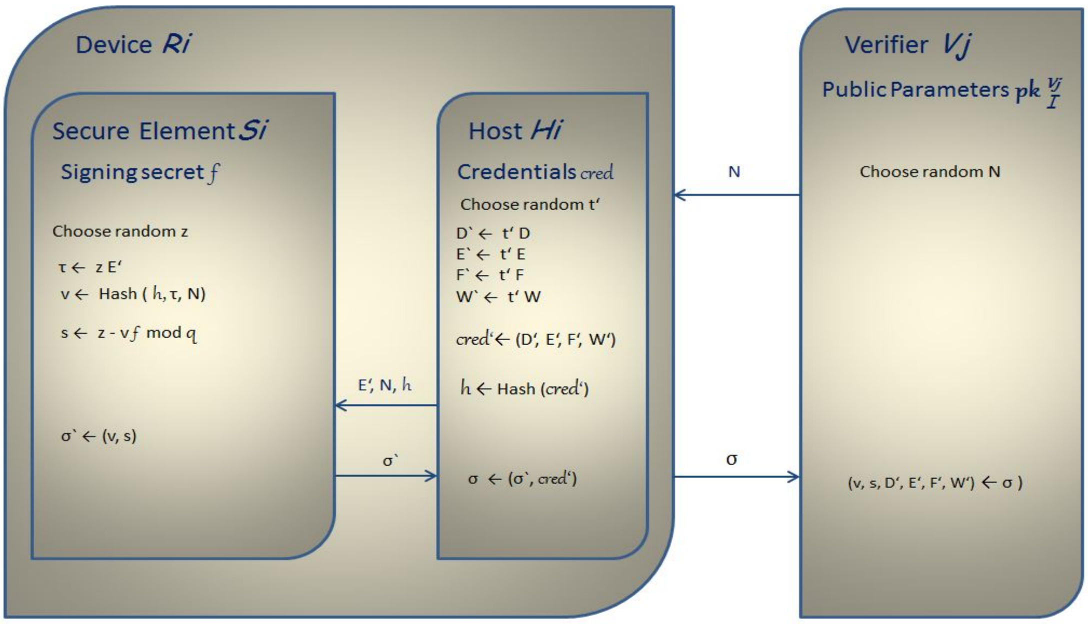

Figure 5.1: Implemented DAA protocol.

The number of bits in the output of the hash used, should be equal or close to the number of bits needed to represent the group order.

5.5 ECC and Pairing Based DAA Scheme

In Section 4.3 the implementation of a simplified version of the DAA scheme on mobile devices is described, together with an extensive description of the simplifications made to the protocols, including motivations for them. The protocol for DAA for RFID is based on lightweight anonymous authentication scheme presented in Figure 5.1.

A DAA scheme involves a set of issuers, devices, and verifiers. An Issuer is in charge of verifying the legitimacy of devices, and of issuing a DAA credentials to each device. A device can prove membership to a Verifier by providing a valid DAA signature. To do so, it requires that the device holds a valid DAA credentials. The Verifier can verify the membership credential from the signature, but it cannot learn the identity of the signer. The ECC-based commitments and DAA rely on the discrete logarithm assumption and Camenisch Lysyanskaya (CL) credential system, which are described in detail in [4]. Based on these definitions, the rest of this section describes implemented DAA scheme, which is based on the one in [28] and relies on its security proof.

5.5.1 The Setup Algorithm

To initialise the system, one needs to select parameters for each protocol, as well as the parameters for each Issuer. Prior to initialization each device has a private secret key embedded into it and each Issuer has access to the corresponding public parameters.

Each device has a single set of credentials, but it can create multiple signatures, even associated with a single issuer. This is allowed by randomization of orginal credentials, which is later on sent in addition to the computed signature.

5.5.2 The Join Protocol

This is a protocol between a given device R_i and an Issuer I. The protocol is virtually identical to that of [28]. The issuer implementation was not the assignment of this thesis. Issuer-algorithm with secret and credentials was diposed for this thesis and one part of it with its issued credentials is presented in Appendix A. This credential set has been used for this feasibility study.

5.5.3 The Sign/Verify Protocols

This is a protocol between a given device R_i and a Verifier V_j and it is identical to that shown in Figure 5.1. The main difference between implemented version and the protocol in [28] is the device revocation. The Verifier does not verify if $cred'$ has been revoked. An Implementation for a complete Verifier is out of scope of this thesis, and with it is the fact, that the revocation list was not a priority task. An implementation of verifier's pairing operations, which are time-consuming and have to be implemented for roggue tagging, is avoided.

With given signing key and credentials, from I, tag can anonymously sign a random message N, received from Verifier V_j.

Due to Integrity Guard and EEPROM where signing key is stored, some of the computations can be implemented on a "Secure Element". The Secure Element S_i contains of a secure EEPROM on a tag and encrypted communication and computation of values with private signing key. Implementation of this scheme will be divided between Secure Element S_i and a Host H_i. Communication and computation of host's calculations can be, but it does not have to be encrypted. For better preformance evaluation, the computation is not implemented with encrypted values.

The Host H_i computes all privacy-related operations. One set of credentials *cred*, H_i can randomize in *cred′* and use the same credentials for different messages from the same verifier V_j. To randomize credentials, four multiplications with random number have to be implemented. A random number generator (RNG) is available on SLE78CLxxxP and will be used for every random number generation.

The Secure Element S_i stores signing key and computes all security critical operations. For computing a signature on a random message N, one multiplication with random number, one hash function and one modular multiplication have to be implemented.

5.6 Implementation Details

To instantiate the DAA scheme, parameters have to be chosen and they are computed with pairing groups based on Barreto-Naehrig (BN) curves [1]. As already mentioned the issuer was implemented and credentials were prepared for this feasibility study. Based on these facts, a pairing-friendly elliptic curve was chosen: 256-bit elliptic curve with $a = 0$ and $b = 3$: $y^2 = x^3 + 3$. In Figure 5.2 the main hierarchy is represented.

For this thesis very large integer numbers are needed for scheme implementation. The CryptoLibrary is using the `CLONG` structure to represent very long unsigned integer numbers. A `CLONG` object always contains an unsigned integer, with maximum of `CLONG`.BitLength bits. Functions `ECC DH()` and `CryptoModAdd` were used

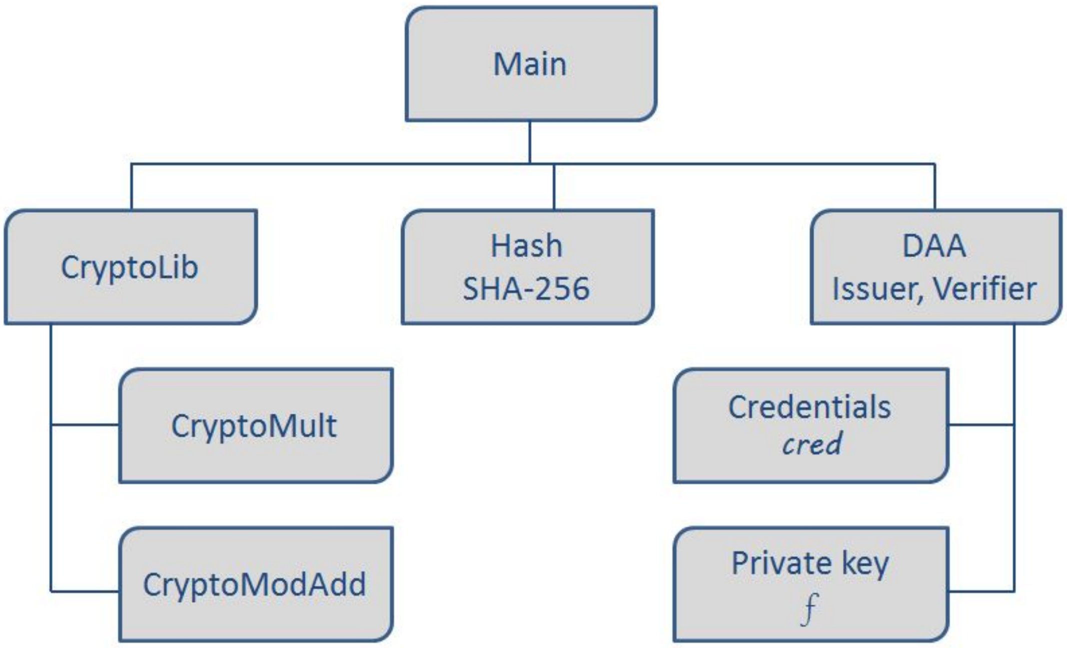

Figure 5.2: DAA scheme hierarchy

for point multiplication and modular multiplication in S_i and H_i. Hash function SHA-256 was available in CryptoLibrary, too. All Issuer parameters are presented in A.

Implemented scheme hierarchy is graphically presented in Figure 5.3.

5.6.1 Includes

clToolbox.h is a part of Infineon Library. Functions `ECC DH()` and `CryptoModMult` which were used for point multiplication and modular multiplication in S_i and H_i are defined in this include-file. The operation `ECC DH()` executes a skalar multiplication and the operation `CryptoModMult` executes $Result = (OperandA\ X\ OperandB)\ mod\ Modulus$. In both cases, operands must point to a `CLONG` that contains valid parameters, and a *Result* is the pointer where the result will be stored.

cl70Types.h is also a part of Infineon CryptoLibrary where types definitions for unsigned integers (e.g., `UINT8`), global variables and its pointers and function declarations are defined.

In LightweightMemoryManagment.h pointers for allocated working memory are defined. Especially Pointers for currently used `CLONG` structure and AffinePoint-tupel (X, Y).

RNG.h defines `getRandomBytes` command from integrated RNG register.

Sha256.h defines SHA-256 function with its parameters (pointer to buffer, to message and length of a buffer), description for calculations and buffer for final hash calculation.

ECCCurve.h defines ECC curve parameters. `ECCCPARM`, from CryptoLibrary contains all parameters relevant to an elliptic curve and supports ECC curves over prime fields $GF(p)$ as well as over $GF(2^m)$ fields. Objects within `ECCCPARM` are `AFFINEPoint Basepoint`, which is a point lying on the curve, `CLONG BasePointOrder`, `UINT8` Characteristic (0 for $GF(p)$ and 1 for $GF(2^m)$), `CLONG CoefA`, `CLONG CoefB`,

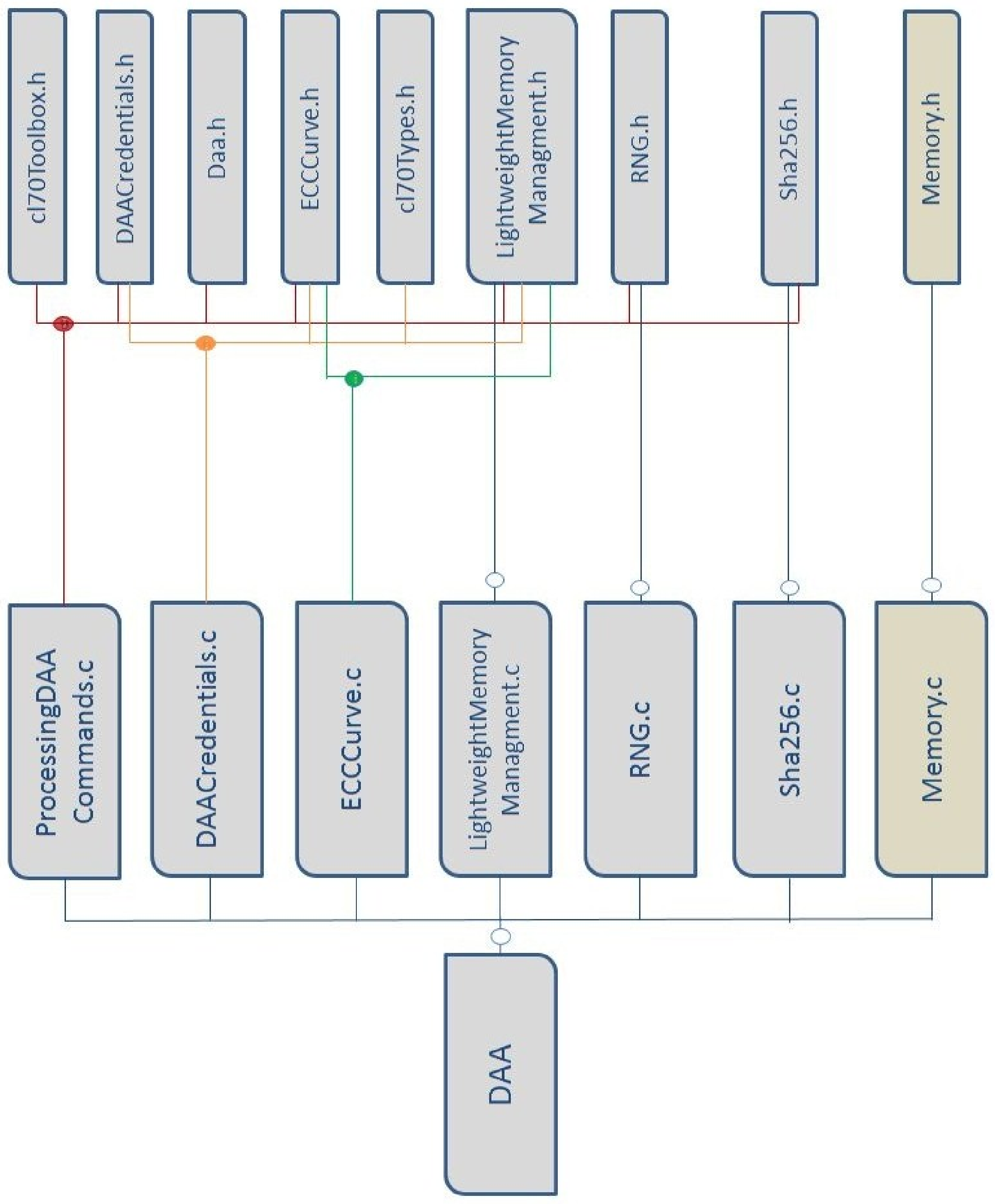

Figure 5.3: Implemented source code hierarchy

coefficients of the elliptic curve equation and `CLONG Modulus`, which is defining prime field number p or irreducible polynom $f(x)$ of degree n. **Class ECCCPARM** defines ECC curve parameters. All members of `ECCCPARM` will reflect a valid ECC curve which satisfies one of the equations:

ECC curves over $GF(p)$: $y^2 = x^3 + ax + b$, and $4a^3 + 27b^2 \neq 0$.

ECC curves over $GF(2^m)$: $y^2 + xy = x^3 + ax^2 + b$, and $b \neq 0$.

Inherited from CryptoLibrary structure `ECCCPARM`, `ECCCurve` structure is defined, which contains all parameters, so that selected pairing-friendly curve can be implemented.

Daa.h is a structure with pointers for `CLONG` where private signing key and affine points (`AFFINEPOINTS` credentials) lying on the curve must be stored. Private key and credentials build `DAAPARM`.

DAACredentials.h defines a superclass of `DAAPARM`. With this h-file, different issuer credentials can be used (e.g., for different verifiers).

memory.h is an optional h-file. Before creating `DAAPARM` superclass, memory containers were implemented, with fixed parameters from issuer. As explained before, `DAAPARM` is much better option, because not just one but more different credential sets issued from different or one issuer are used in one DAA scheme.

5.6.2 Source

In RNG.c is the implementation of a main true RNG register. This source code is tag-related and included in tag-software.

Sha256.c is Infineon SLE78CLxxxP SHA-256 function. In this file all hash calculations are implemented.

LightweightMemoryManagment.c provides needed memory depending on global variables. For every `CLONG, UNIT8, UNIT16, AFFINEPOINT` and other variables memory and pointers are assigned. During computation, a pointer of the memories where needed parameters are stored, can be called for processing.

ECCCurve.c calls ECC curve parameters. Sturucture `ECSET` sets pointer to structure containing ECC curve parameter set and over ArithmeticMode ($eModN = 0$ for $GF(p)$ and $eGF2$ for $GF(2^m)$) the type of the curve is defined. With `static const UNIT8` Curve parameters are defined, as shown in Appendix B.

Using this option, different curves can be implemented. Chosen curve parameters `ECCCurve` are then allocated in working memory. Bitlength values have to be corrected, so that `Modulus, Coefficient, Basepoint` and `BasepointOrder Bitlength` are valid and `PECCCurve GetEccCurve` returns pointer for memory containing valid ECC curve parameters.

DAACredentials.c defines DAA credentials structure. A structure `DAASET` contains pointer to structure containing `DAA` credentials and size of an array pointed to by `DAAPARM`. Arithmetic Mode definition has to be the same as in ECCCurve.c, because all operations on the tag have to be performed on the same curve, using same curve parameter. Using `static const UINT8 DAACred, Characteristic, Modulus` (elliptic curve based), Private key and Credentials ($D.X$, $D.Y$, $E.X$, $E.Y$, $F.X$, $F.Y$, $W.X$, $W.Y$) from issuer are assigned to `DAACREDPARM` set. More different DAA sets can be defined and used. After correcting a `Bitlength` value of parameters, it returns a pointer to `DAACredentials`. By implementing an issuer, this part should be redesigned.

memory.c is optional helper file for loading a `CLONG` structure with `UINT8 huge` containers. Before implementing DAACredentials.c, memory.c was used as test implementation.

ProcessDaaCommands.c is a main file. All Secure Element S_i and Host H_i computations are implanted here. In Appendix C the main parts of the source code are shown. Validity of all parameters have to be checked with a function `PreformEcc-SelfTest`, which calls all parameters pointers and memories. If the test fails, new curve parameters are called and set.

H_i stores credentials and computes all privacy-related operations. Using RNG host chooses a random prime number used for credential randomization, blinding. After choosing a random number, host calculates blinding of every affine point representing credentials using `ECC DH()` CryptLibrary function, with `TrueBitLength` of scalar and `TrueBitLength` of `PECCCPARM` as input. The necessary temporary working memory required in bytes, for passing `CLONG` of this function, can be calculated by using the `GET MIN TEMP MEM ECC DH` macro. This function is used to implement a scalar multiplication of a random number with an `AffinePoint`. ECC curves over prime field $GF(p)$ as well as over $GF(2^m)$ finite field are supported. New credential values $(D1(X,Y),\ E1(X,Y),\ F1(X,Y),\ W1(X,Y))$ are then hashed with SHA-256 and ready for Secure Element S_i computations. All H_i functions and computations are under `CLIB STAUTS performDaaHostCalculation` as a new CryptoLib function defined.

All Secure Element S_i computations are under `CLIB STAUTS performDaaSECalculation` as a new CryptoLib function defined. First, S_i gets random number from RNG and preforms scalar multiplication with its random number z and $E1(X,Y)$ received from the H_i with same `ECC DH()` function, described before. After calculating a hash digest from SHA-256, modular multiplication is used for signature calculation: $s = (z + v \cdot f)(mod\ q)$. This is a CryptoLibrary function `CryptoModMult` and its implementation is presented in Appendix C.

In this chapter, implementation of the signing protocol on the RFID tag is presented. In Appendix C are some of the main parts of the implemented source code.

Chapter 6

Results

The chapter illustrates the testing of the software implementation, which covers computational and performance testing. The testing results proved that the DAA algorithm is feasible and it can be a implemented on a RFID tag. The protocol between entities has to be computational, and a given result are indicative only. The implementation of the scheme was constructed using µKeil based on C. Both, Secure Element S_i and Host H_i, use SHA-256 as an underlying hash and RNG function, and implementation uses the Infineon SLE70 CryptoLibrary.

To evaluate the implemented DAA scheme, and compare it with lightweight DAA [28], some concrete experimental results are presented. The results do not include hidden costs such as parameters generation and verification, and that is why, the results are only indicative. Only signing protocol has been implemented and its results are discussed.

6.1 Implementation Result

The idea was to provide an anonymous authentication system based on the existing lightweight DAA scheme using CryptoLibrary and to include new DAA functions in the library.

`CLONG` class holds the large numbers such that it can be used in this implementation. The protocol view of the DAA scheme is shown in Figure 5.1. A new CryptoLibrary functions `CLIB STAUTS performDaaHostCalculation` and `CLIB STAUTS`

Host	Secure Element	Total
288 ms	0.85 ms	373 ms

Table 6.1: Timings of the implemented DAA Sign Protocol

Host	Secure Element
12154880	3524224

Table 6.2: System clock for the implemented DAA Sign Protocol

`performDaaSECalculation`, contain all computations of the implemented system. They depend on the elliptic curve and DAA parameters, which are used to define the elliptic curves and their related operations and the arithmetic of the finite field associated with them. The parameters will wrap all the essential components including: specific characteristic which defines the finite field over which the elliptic curve is defined ($GF(p)$ or $GF(2^m)$), Modulus, Coefficients a and b of the Equation 3.1, Basepoint, the smallest positive integer n such that n times Basepoint results in *the point-at-infinity*, and the Basepointorder.

This DAA scheme based on ECC only provides performance for the RFID tag, but the Issuer and Verifier are not implemented and evaluated. It is implemented with pairing-friendly elliptic curve $y^2 = x^3 + 3$ over prime field and affine coordinates. Signing protocol was computed with given issuer parameters: signing key and credentials. A produced signature was verified with Verifier V_j, which was disposed for this thesis. No revocation list and rogue tagging were activated.

Contactless measurements on a Infineon device with 48MHz were made in a strong field with external clock fixed at 1.7MHz. Average timings in milli-seconds are presented in Table 6.3.
System clocks are given in Table 6.2.

Although the total signing time is over 300 ms, the signature is computed under maximum execution time of 450ms. The results indicate that this DAA scheme is feasible for a RFID tag.

DAA Sign	Host	Secure Component	Total
ARM	94.8 ms	23.75 ms	118.75ms
ARM Thumb	92.57 ms	23.16 ms	115.73 ms

Table 6.3: Performance of the DAA Sign Protocol

6.1.1 Related Work

There exists many different RFID authentication protocols in the literature. To the best of my knowledge, this is the first scheme that proposes DAA signing protocol implementation on the RFID tag.

In [28] Dietrich implemented a lightweight anonymous authentication scheme on an ARM11 CPU of a mobile device, which runs at a clock speed of 600MHz. In Table 6.3 its average performance over 100 test-runs is presented. ARM Thumb in Table 6.3 represents a highly optimized performance implementation. This protocol implementation is represented in Section 4.3.

As it can be noticed, ARM processor has, as expected, much better execution time, due to its 600MHz clock speed, while SLE78CLxxxP's CPU can run at maximal 48MHz clock speed (just in a strong field). That is why the measurement were made in a strong field.

Chapter 7

Conclusion

RFID tags must continuously handle increasing capacities and process higher data rates, while at the same time, there is a rising demand for reduced size, cost and power consumption. In this thesis an anonymous authentication scheme based on pairing-friendly elliptic curve cryptography has been presented. A DAA sign protocol is implemented on a SLE78CLxxxP RFID tag, without formal security analysis and with no built-in support for tag-revocation. Security parameters and assumptions are based on [28], which is equivalent to a 128-bit AES or 256-bit ECC security level.

The computational performance of the implemented scheme has been evaluated using the µKeil software. This software incorporates a tool that enables the real-time simulation of Infineon SLE78CLxxxP. The µKeil also includes source code editor, project build environment and debugger. These tools enable users to produce an efficient code for their applications employing C.

RFID tags offer an ideal identification method by means of digital signatures. The anonymous authentication scheme based on elliptic curves is commonly used for achieving anonymity and authenticity. Elliptic curve cryptography allows the use of small key sizes which is necessary when targeting RFID tags. Within this thesis a co-processor was used for accelerating all operations involved in a DAA signature generation calculation. This was achieved by a fast software implementation of point and modular multiplication. Implemented scheme uses the standard includes support for large numbers and their addition, and point and modular multiplication.

It is now possible to perform a DAA signature generation calculation on a RFID tag in 373ms at a clock frequency of 48MHz. However, speed-up should be possible.

7.1 Future Work

Future work should be focused on implementing the complete lightweight anonymous authentication scheme for the RFID tags. The issuer and verifier are not implemented here yet, but outsorced for this thesis. On the one hand because of the lack of time and on the other hand, the pairing based cryptography and efficient algorithm for its implementation could be a work for itself, because of its complexity. A DAA scheme with pairing-based issuer and verifier, which allows a tag revocation, should be proposed as possible topic for future thesis. One could then implement the whole system and evaluate real-system performance with different verifiers and tags.

Another interesting work would be the evaltuation of the implementation using a $GF(2^m)$ arithmetics and comparing this within this thesis implemeted scheme based on pairing-friendly curve based on $GF(p)$ arithmetics. A research in construction of the pairing-friendly curves are still in progress. This and the comparison with real-system performance could give estimates for possible improvements.

On the next generations of RFID tags with reengineered co-processors, the scheme should be implemented and the resulting performance should be analyzed. Future software should be designed for speeding up implemented operations. One could try to implement these operations more efficiently, possibly by using more developed hardware and implementing even more state-of-the-art scheme.

Appendix A

Issuer and System Parameters

Issuer Source Code in C++

```
//brief Lightweight ECC-DAA basic issuer
//file ecdaa.cpp
/// New issuer initialization
sys params(bn)
ZZn2 frobenius X;
sys params.bn.setup(frobenius X);
// Construct generators
sys params.P1 = bn.generator();
sys params.P2 = bn.generator2();
// Fast multiplication of P2 with cofactor (p - 1 + t)
bn.cofactor(sys params.P2, frobenius X);
// Pick random x and y values (secret key)
x = rand(sys params.bn.q);
y = rand(sys params.bn.q);
// Generate public key
sys params.X = x* sys params.P2;
sys params.Y = y* sys params.P2;

/// Issuer initialization from parmeters.
issuer::issuer(const EC* P1, const ECn2* P2, const Big* x,
// Activate the issuer curve
sys params.bn.setup();
sys params.P1 = P1;
sys params.P2 = P2;
sys params.X = x * P2;
sys params.Y = y * P2;

/// Generate a new Mobile Device credential
void issuer::generate(Big* f, ECn* D, ECn* E, ECn* F, ECn* W) const
// Activate the issuer curve
sys params.bn.setup();
// Pick random f and r
f = rand(sys params.bn.q);
Big r = rand(sys params.bn.q);
D = r* sys params.P1;
E = y * D;
Big tmp = modmult(x, y, sys params.bn.q);
mad(tmp, f, x, sys params.bn.q, tmp);
F = tmp * D;
W = f * E;
generate(se-cred.f, host-cred.D, host-cred.E, host-cred.F, host-cred.W);
```

System Parameters

Public System Parameters:

```
P1 =
(919E34F0F01F364EC20E9DE76C8A819E7175762E5480A6653F09817EB831D94,
6FCA2CC3F9F2CBD6C9F10D6EF1EA84B129C864DAAE4A951D95FDD17F41FA68C)
P2 =
([5269AC04EB0CB657D4B7D4CE25018BC8803C776C4750624FC16E683C2CED9035,
2B499CDFFEAD4A348A9C713CCA7D1CDD7ABAF6E4A00198E30D7FB7B79A7F9F02],
[B43D94DE1D3B71F88F11472D2D8EDF922A6F3361AFD2AC3D0C39D45C687442CF,
848C3265BC0DCA81A8D7F90D27F7C18F7F1CCB9F22668AE43B88CD093DAD10BF])
X =
([1F6AD2365B025DEDCA56655D996745053AACEB4281C49305F1FE1EAF484CBF70,
23FAE00AB66FFC82D856E497ACA939B76CF1527427917EFFF74601A19D97C4B6],
[71EAD7A6B17AAD88100521668A753DF75A0E0B81AD54CAE9962F41D7A57FB596,
3C47CE6379895FEC2BCBB7DE65C34898605E0466224208535F9290FCE1C8B214])
Y =
([61902CEADE19BE2016F8640B696D94F3040A38358A12406552359ED4BEC06FB4,
39017B035EA836654E3BA72E9C8384891DC1C2518CA6121750273D36E2E841E8],
[594C22038DCC872FE56D8E1D6F3460A0BD8A7BEEEEA9C5A8A7D55C843514D270,
A1EAF8AA1D7EF712DD654EF7CCEE00032C62D36B0BDFCDC16A572CF051324EC6])
```

Curve: Baretto-Naehrig:

```
x = 600000000000219B
twist = sextic (D)
p =
B64000000000FF2F2200000085FD5480B0001F44B6B88BF142BC818F95E3E6AF
t =
D8000000000097398000000001A77F717
q =
B64000000000FF2F2200000085FD547FD8001F44B6B7F4B7C2BC818F7B6BEF99
cofactor =
B64000000000FF2F2200000085FD548188001F44B6B9232AC2BC818FB05BDDC5
```

Credentials

Private Parameters: (SE credential)

```
f =
587652595EAF8B82B72E2E246573A4252A86E5B94F16C2A441C7348C7659E1F4
```

Private Parameters: (Host credential)

```
D =
(9696A7F852BFCE2B557B974F0B34B522B8A3806AD9CCA5E83998B12FAABE3456,
31735DE41D1895339A9FB30567B6C455FE70C2D5C1967471FF0C364704D0DD3A)
```

```
E =
(4E8E71686BD2E45CABEA740ED3F72238EA2B2A61FC75DE5959928DE2E64BAAF5,
31AFB7EBE23FFC0DBF9E2FECDD15B0C9978C54690849ED112DFAB942F894A779)
F =
(175127A12A69A59EBE0ACB30B3D52A91F31318B8FA9D7119A4FE129B0ECCE448,
A10468F80397B4E4FB0B8EF1155A9F6C9B974BF7B046C86F665EC3794FCB66D0)
W =
(627CF3B64EE73ECF52E8C408D9F909AFCFB61756ABF4FCEF77C8507B983CA81F,
CE03DA5A4C7FEED604E1F7CD2E972066699237CC2D1A6FC14ED23466467658A)
```

Appendix B

Implementation of Pairing-Friendly Elliptic Curve

```
//@brief Pairing-friendly:  256-bit ECC with A = 0; B = 3:y^2 = x^3 + 3
  static const UINT8 N-256[] =
  // Characteristic //
  eModN,
  // Modulus p //
  0x00, 0x20, // BytesAllocated
  // Data field
  0xB6, 0x40, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x2F, 0x22, 0x00, 0x00, 0x00,
0x85, 0xFD, 0x54, 0x80,
  0xB0, 0x00, 0x1F, 0x44, 0xB6, 0xB8, 0x8B, 0xF1, 0x42, 0xBC, 0x81, 0x8F,
0x95, 0xE3, 0xE6, 0xAF,
  // CoefA a //
  0x00, 0x02, // BytesAllocated
  0x00, 0x00, // Data field
  // CoefB b //
  0x00, 0x02, // BytesAllocated
  0x00, 0x03, // Data field
  // Basepoint.X //
  0x00, 0x20, // BytesAllocated
   // Data field
  0x09, 0x19, 0xE3, 0x4F, 0x0F, 0x01, 0xF3, 0x64, 0xEC, 0x20, 0xE9, 0xDE,
0x76, 0xC8, 0xA8, 0x19,
  0xE7, 0x17, 0x57, 0x62, 0xE5, 0x48, 0x0A, 0x66, 0x53, 0xF0, 0x98, 0x17,
0xEB, 0x83, 0x1D, 0x94,
  // Basepoint.Y //
  0x00, 0x20, // BytesAllocated
  // Data field
   0x06, 0xFC, 0xA2, 0xCC, 0x3F, 0x9F, 0x2C, 0xBD, 0x6C, 0x9F, 0x10, 0xD6,
0xEF, 0x1E, 0xA8, 0x4B,
  0x12, 0x9C, 0x86, 0x4D, 0xAA, 0xE4, 0xA9, 0x51, 0xD9, 0x5F, 0xDD, 0x17,
0xF4, 0x1F, 0xA6, 0x8C,
  // BasepointOrder //
  0x00, 0x20, // BytesAllocated
  // Data field
  0xB6, 0x40, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x2F, 0x22, 0x00, 0x00, 0x00,
0x85, 0xFD, 0x54, 0x7F,
  0xD8, 0x00, 0x1F, 0x44, 0xB6, 0xB7, 0xF4, 0xB7, 0xC2, 0xBC, 0x81, 0x8F,
0x7B, 0x6B, 0xEF, 0x99,
  // szName[] of curve //
  "N-256"
```

Appendix C

Processing DAA Commands

In the first part of the presented code are global definitions and declarations. Particularly important is the global declaration `static const UNIT8 VERIFIER-N[]` which contains the expected values of a valid signature. Comparing this and computed values, it was possible to know if the tag executed all functions and if it computed a valid signature of knowledge on a random challenge N.

Implemented DAA Scheme on a RFID Tag

```
// global definitions
define PERFORM-SELFTESTS 0 // 0-no self tests; 1-perform self tests
define CURVE-INDEX 1 // Ellliptic Curve Index to be used
define CREDENTIAL-INDEX 0 // Credential Index to be used
define HASH-SIZE 32 // hash size in byte for SHA-256
// global declarations (verifier side)
static const UINT8 VERIFIER-N[] =
0x06, 0x40, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x2F, 0x22, 0x00,0x00,
0x00, 0x85, 0xFD, 0x54, 0x80, 0xB0, 0x00, 0x1F, 0x44, 0xB6,0xB8,
0x8B, 0xF1, 0x42, 0xBC, 0x81, 0x8F, 0x95, 0xE3, 0xE6, 0xAF
// global declarations
PAFFINEPOINT hostD1, hostE1, hostF1, hostW1; // blinded credentials
UINT8 host-hash[HASH-SIZE]; // hash value over cred'
PCLONG se-v; // hash generated by the secure element
PCLONG se-s;
```

```
UINT16 GetSignificantBitLength(PCLONG pcl) reentrant;
UINT8 cryptoDAATests();
void computeHash(UINT8 *hash, UINT8 *message, UINT8 message-length);
/// helper for initially loading a clong structure
define LOADCLONG(var, MyBYTES, MyBITLEN, MyPMEM)
var.BytesAllocated = (MyBYTES);
var.BitLength = (MyBITLEN);
var.Data = ((UINT8 *)(MyPMEM));
```

Implemented Credential Randomization

```
//@brief Perform DAA host calculation part
CLIB STATUS performDaaHostCalculation(PECURVEPARM pEccCurve,
PDAACREDPARM pDAACredentials, UINT16 ln, PUINT8 pHash)
PCLONG mem; // working memory
PCLONG r; // random number used for blinding
UINT8 hash buffer[size buffer]; // temporary hash buffer
CLIB STATUS ret val; // Status Information
UINT16 byteLen; // reserve memory space for calculations
byteLen = 2*((ln + datalength -1 ) / datalength);
r = LmmAllocAssignClong( byteLen );
mem = LmmAllocAssignClong( GET MIN TEMP MEM ECC DH
( GetSignificantBitLength( r ), ln ));
// generate random number for blinding
ret val = getRandomBytes(r->Data, r->BitLength/8);
if (ret val != 0x00)
return ret val;
// calculate blinding D1 = r*D
do
hostD1->X.BitLength = hostD1->X.BytesAllocated * 8;
ret val = ECC DH( ((PECCCPARM)pEccCurve), r, *
((PDAAPARM)pDAACredentials)->D, mem, hostD1);
return ret val;
// calculate blinding E1 = r*E
// calculate blinding F1 = r*F
// calculate blinding W1 = r*W
```

Implemented Hash Calculation

```
// calculate hash = SHA-256(D1, E1, F1, W1)
sha256init(hash buffer);
sha256update(hash buffer, hostD1->X.Data, byteLen);
sha256update(hash buffer, hostD1->Y.Data, byteLen);
sha256update(hash buffer, hostE1->X.Data, byteLen);
sha256update(hash buffer, hostE1->Y.Data, byteLen);
sha256update(hash buffer, hostF1->X.Data, byteLen);
sha256update(hash buffer, hostF1->Y.Data, byteLen);
sha256update(hash buffer, hostW1->X.Data, byteLen);
sha256update(hash buffer, hostW1->Y.Data, byteLen);
sha256final(hash buffer);
memcpy(pHash, hash buffer, HASH-SIZE);
return ret val;
```

Implemented Secure Element calculation

```
//@brief Perform DAA secure element calculation part
CLIB STATUS performDaaSECalculation
(PECURVEPARM pEccCurve, PDAACREDPARM pDAACredentials, UINT16 ln)
PCLONG mem; // working memory
PCLONG z; // random number
PCLONG tmp; // intermediate result
PAFFINEPOINT TAU;
UINT8 hash buffer[size buffer]; // temporary hash buffer
CLIB STATUS ret val; // Status Information
UINT16 byteLen; // reserve memory space for calculations
byteLen = 2*((ln + datalength -1 ) / datalength);
z = LmmAllocAssignClong( byteLen );
tmp = LmmAllocAssignClong( byteLen );
```

```
mem = LmmAllocAssignClong( GET MIN TEMP MEM ECC DH
( GetSignificantBitLength( z ), ln ));
TAU = LmmAllocAssignAffinePoint( byteLen, byteLen );
// generate random number for blinding/randomization
ret val = getRandomBytes(z->Data, z->BitLength/8);
if (ret val != 0x00)
return ret val;
// calculate TAU = z*E1
do
TAU->X.BitLength = TAU->X.BytesAllocated * 8; //right alligned
ret val = ECC DH( ((PECCCPARM)pEccCurve), z, hostE1, mem, TAU);
return ret val;
// calculate se v = SHA-256(host hash, TAU, verifier-n)
sha256init(hash buffer);
sha256update(hash buffer, host hash, sizeof(host hash));
sha256update(hash buffer, TAU->X.Data, byteLen);
sha256update(hash buffer, TAU->Y.Data, byteLen);
sha256update(hash buffer, VERIFIER-N, sizeof(VERIFIER-N));
sha256final(hash buffer);
memcpy(se-v->Data, hash buffer, HASH SIZE);
// calculate s = z + v*f mod q
ret val = CryptoModMult(se-v, *((PDAAPARM)pDAACredentials)->pf,
*((PDAAPARM)pDAACredentials)->Modulusq, mem, tmp);
if (ret val != CLIB STATUS SUCCESS UNSPECIFIC)
return ret val;
ret val = CryptoModMult(z, tmp,
*((PDAAPARM)pDAACredentials)->Modulusq, mem, se-s);
return ret val;
```

Appendix D

Glossary

Abbreviation	**Description**
AES	Advanced Encryption Standard
AIK	Attestation Identity Key
BN	Barreto-Naehrig Curve
CL	Camenisch Lysyanskaya credential system
CPU	Central Processor Unit
CAD	Computer Aided Design
DAA	Direct Anonymous Attestation
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
EEA	Extended Euclidean Algorithm
EEPROM	Electrically Erasable Programmable Read-Only Memory
EK	Endorsement Key
FIPS	Federal Information Processing Standard
GF	Galois (finite) field
LSB	Least Sgnificant Bit
MIRACLE	Multiprecisional Integer and Rational Arithmetic C/C++ Library
MSB	Most Sigificant Bit
NIST	National Institute of Standards and Technology
Privacy CA	Privacy Certification Authority

RAM	Random Access Memory
RFID	Radio Frequency IDentification
RNG	Random Number Generator
RSA	Rivest-Shamir-Adleman (public-key encription technology)
TCG	Trusted Computing Group
TPM	Trusted Platform Module

Bibliography

[1] P. S. L. M. Baretto and M. Naehring. *Pairing-friendly elliptic curves of prime order.* In Selected Areas in Cryptography - SAC 2005. Springer Verlag, 2006.

[2] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM conference on Computer and communications security*, CCS '04, pages 132–145, New York, NY, USA, 2004. ACM.

[3] Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. *ACM Trans. Inf. Syst. Secur.*, 15(1):4:1–4:30, March 2012.

[4] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2004.

[5] D. Chaum. *Security without identification: Transaction systems to make big brother obsolete*, volume 1030-1044 of *Communications of the ACM 28(10).* 1985.

[6] Liqun Chen, Dan Page, and Nigel P. Smart. On the design and implementation of an efficient daa scheme. In *Proceedings of the 9th IFIP WG 8.8/11.2 international conference on Smart Card Research and Advanced Application*, CARDIS'10, pages 223–237, Berlin, Heidelberg, 2010. Springer-Verlag.

[7] A.Menzes D. Hankerson and S.Vanstone. *Guide to Elliptic Curve Cryptography.* Springer Verlag, 2004.

[8] Kurt Dietrich. Anonymous credentials for java enabled platforms: A performance evaluation. In Liqun Chen and Moti Yung, editors, *INTRUST*, volume 6163 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 2009.

[9] J. Camenisch E. Brickell and L. Chen. *Direct anonymous attestation in context.* In C. Mitchell, Chapter 5 of Trusted Computing. IEEE London, 2005.

[10] I.B. Damgard E. F. Brickell, D. Chaum and J. Van der Graaf. *Gradual and verifaiable release of a secret.* Springer Verlag, 1988.

[11] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[12] Çetin Kaya Koc and Tolga Acar. Analyzing and comparing montgomery multiplication algorithms. *IEEE Micro*, 16:26–33, 1996.

[13] Trusted Computing Group. Tcg tpm specification.

[14] Trusted Computing Group. Tcg tpm specification.

[15] MIRACLE (Multiprecisional Integer and Rational Arithmetic C/C++ library).

[16] N. Koblitz. *Introduction to Elliptic Curves and Modular Forms, Graduate Texts in Mathematics*, volume 97. Springer Verlag, 1984.

[17] Neal Koblitz. *A Course in Number Theory and Cryptography.* Springer Verlag, 2nd edition, 1994.

[18] A. J. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian, editors. *Applications of Finite Fields.* Kluwer Academic Publishers, 1993.

[19] Alfred Menezes. *Elliptic Curve Public Key Cryptosystems.* The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, seventh printing edition, 1999.

[20] V. Miller. *Elliptic Curves and their use in Cryptography;.* 1997.

[21] A. Wander H. Eberle N. Gura, A. Patel and S. Schantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. 2004.

[22] L. B. Oliveira, D. F. Aranha, C. P. L. Gouvêa, M. Scott, D. F. Câmara, J. López, and R. Dahab. TinyPBC: Pairings for Authenticated Identity-Based Non-Interactive Key Distribution in Sensor Networks. *Computer Communications*, 34(3):485–493, 2011.

[23] A Certicom White Paper. The elliptic curve cryptosystem for smart cards.

[24] FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION. Digital signature standard (dss).

[25] R. L. Rivest, A. Shamir, and L. M. Adelman. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Massachusetts Institute of Technology, Laboratory for Computer Science, 1977.

[26] Masaaki Shirase. Barreto-naehrig curve with fixed coefficient - efficiently constructing pairing-friendly curves -. *IACR Cryptology ePrint Archive*, pages 134–134, 2010.

[27] W. Trappe and L. Washington. *Introduction to Cryptography with Coding Theory*. Prentice Hall, 2nd edition, 2004.

[28] Christian Wachsmann, Liqun Chen, Kurt Dietrich, Hans Löhr, Ahmad-Reza Sadeghi, and Johannes Winter. Lightweight anonymous authentication with tls and daa for embedded mobile devices. In *Proceedings of the 13th international conference on Information security*, ISC'10, pages 84–98, Berlin, Heidelberg, 2011. Springer-Verlag.