Graz University of Technology
Faculty of Informatics
Software Development and Business Management

Efficient Implementation of an Elliptic Curve Based Anonymous Credential System

Master's Thesis

Michael Kapfenberger B.Sc.

4th October 2012

Supervisor: Univ.-Prof. M.Sc. Ph.D. Roderick Paul Bloem
E-Mail: Roderick.Bloem@iaik.tugraz.at

Second Supervisor: Dipl.-Ing. Kurt Dietrich
E-Mail: Kurt.Dietrich@iaik.tugraz.at

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

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 entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ……. ……..
(Unterschrift)

Englische Fassung:

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)

Abstract

In this thesis we evaluate a new approach of an anonymous credential system (ACS) for low powered devices based on elliptic curve cryptography (ECC). The proposed algorithm was part of a research project at the Institute for Applied Information Processing and Communications at TU Graz. The target of this evaluation was the efficient implementation of the proposed scheme to gather useful information about the performance of the algorithm.
Nowadays schemes like Direct Anonymous Attestation (DAA) gaining more importance due to the fact of user profiling by different service providers or online fraud. Personal data, like localization data, or other characteristics are collected to obtain knowledge about a specific person. Therefore it is desirable to ensure anonymity and data integrity to the user keeping common use cases like online shopping or using the GPS for navigation as easy as possible. DAA provides such a mechanism but has its disadvantages compared to the proposed anonymous credential system. DAA is specified by the Trusted Computing Group (TCG) and uses RSA techniques for cryptographic processes. RSA needs long keys to provide a certain amount of security and powerful machines to reduce the calculation times. This is obviously not ideal for low powered mobile devices like Smart Cards, if fast execution is required. Furthermore, DAA specifies the usage of a crypto-token for sensitive calculations with the private key for example. Such secure elements are usually slower than a normal CPU which extends the processing times, of for example signatures, significantly. The proposed scheme abstracts the DAA scheme and therefore does not necessarily need a secure hardware element. It is possible to create two different environments by separating the usage of available resources, like CPU, RAM and Storage, to the normal and the secure world. The secure world is important for processing sensitive data which must not be revealed to the outside like secret keys. To prove this concept it was mandatory to test the performance of this signature scheme on a low powered device. Compared to DAA the test results showed a tremendous increase of performance in case of processing times and memory usage, but we also found some fundamental security flaws in the algorithm.

Kurzfassung

In dieser Arbeit wurde ein neuer Ansatz eines anonymen digitalen Signaturverfahrens evaluiert, welches mit Hilfe elliptischer Kurven-Kryptografie (ECC) für leistungsschwache Geräte optimiert ist. Der vorgestellte Algorithmus war Teil eines Forschungsprojekts am Institut für Angewandte Informationsverarbeitung und Kommunikation an der TU Graz. Das Ziel dieser Evaluierung war die effiziente Implementierung des vorgestellten Schemas um brauchbare Informationen über den Algorithmus und dessen Geschwindigkeit zu bekommen.
Heutzutage werden Verfahren wie Direct-Anonymous-Attestation (DAA) immer wichtiger aufgrund von Benutzer-Profiling oder Internetbetrug. Um an Informationen über einzelne Benutzer zu kommen, werden persönliche Daten (z.B. Aufenthaltsorte) oder andere Eigenschaften gesammelt. Daher ist es wünschenswert, dem Benutzer Anonymität und Datenintegrität zu versichern, wobei die Handhabung täglicher Abläufe wie Onlineshopping oder das Navigieren mit dem GPS kaum beeinflusst werden dürfen. DAA stellt einen solchen Mechanismus zur Verfügung, hat jedoch seine Nachteile im Vergleich zu dem in dieser Masterarbeit vorgestellten Verfahren - dem Anonymous-Credential-System (ACS). Beim DAA Verfahren, welches von der Trusted-Computing-Group (TCG) spezifiziert wurde, benutzt man RSA für kryptografische Prozesse. Dies erfordert jedoch lange Schlüssel (z.B. 2048 bits), um eine angemessene Stufe an Sicherheit zu erreichen, wobei man dafür hohe Rechnerlesitungen benötigt, um Berechnungszeiten zu verkürzen. Wie man nur unschwer erkennen kann, ist dies nicht gut geeignet, um leistungsschwache mobile Geräte wie Smart Cards mit dieser Technologie auszustatten. Weiters spezifiziert DAA die Benutzung eines sogenannten Crypto-Tokens, um sensible Berechnungen (z.B. mit dem privaten Schlüssel) durchzuführen. Solche Hardwareelemente sind im Normalfall langsamer als eine Standard-CPU, was die Berechnungszeiten für zum Beispiel die Erstellung einer digitalen Signatur drastisch erhöht. Das vorgestellte System abstrahiert das DAA-Schema und benötigt nicht mehr zwingend ein sicheres Hardwareelement. Mit dem neuen Verfahren ist es möglich, zwei getrennte Welten zu schaffen, indem man die Benutzung von vorhandenen Ressourcen (z.B. CPU, RAM, Speicher, etc.) aufteilt und eine normale Welt und eine sichere Welt schafft. Die sichere Welt ist wichtig für die Verarbeitung sensibler Daten, die dieses Areal niemals verlassen dürfen - wie zum Beispiel der private Schlüssel. Um dieses Konzept zu untermalen, war es wichtig, die Geschwindigkeit dieses Signaturverfahrens auf leistungsschwachen Geräten zu testen. Vergleicht man die Ergebnisse mit dem DAA-Schema, ergeben sich enorme Geschwindig-keitssteigerungen in Berechnungszeiten als auch ein viel geringerer Speicherbedarf. Jedoch wurden im vorgestellten Algorithmus gravierende Probleme im Bezug auf Sicherheit gefunden.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Background

Secure data handling, privacy and anonymity are gaining more and more importance in the world of fast growing data transfer. Companies and private users have a lot of sensitive data which is sent all over the world wide web. Methods to protect privacy of a user are mostly underrated and therefore not used in most of the applications we use every day. Personal data is collected by the different service providers to analyze their customers behavior and create user profiles. For example goal oriented advertising is widely used and deployed. Profiling could be misused to discriminate different classes of users as mentioned in [40].
Especially in the sector of mobile devices the use of private data in interaction with different applications is common. Smart phones, for instance, provide a wide range of applications like internet access or location-based services, with many opportunities to the user and also to the content service providers. Online shopping, recommendations about product offers or events are well-known. Location-based services using the GPS receiver on your mobile phone, like Clever Sense[1] showing restaurants near your actual location, or other software like Google Latitude[2] finds friends around you. This may sound harmless, but if your location is known you can be tracked, which is a big issue for privacy and anonymity. Mobile phones like the iPhone already collect data of the customers' location and send it regularly to Apple Inc. With this information and some other data which were produced and also collected by the user, it is possible to recreate an entire day or week of any performed movement and transaction as Audrey Watters described on ReadWriteWeb[3]. This is a big issue in terms of privacy. All these circumstances make the customer feel uncomfortable using such applications. Moreover, the user feels monitored when using any of those devices.
Nowadays privacy and anonymity gain more importance in protecting the user from things like tracking or profiling. In this project we try to gain results on usability and

[1] www.thecleversense.com
[2] www.google.com/latitude
[3] www.readwriteweb.com/archives/your_iphone_is_tracking_your_every_move.php

performance of an anonymous credential system based on ECC for embedded devices. To ensure high level security on low powered mobile devices it is mandatory to use fast cryptographic schemes. State of the art cryptographic schemes are mostly based on RSA technology. This widely used technique needs long keys, like 2048 bits, to provide a good amount of security. The consequences are time consuming procedures with high memory usage. On low powered devices like Smart Cards it is not possible to run such algorithms in a convenient amount of time. The lack of high level security in this area is a known issue and therefore needs more attention. This is why we are interested in elliptic curve cryptography (ECC). It needs smaller key sizes and therefore less resources. These facts give us enough reason to investigate ECC schemes for anonymous digital signatures which we put focus on. Digital signatures are important to ensure the authenticity of a document or any digital message to a recipient.
Direct Anonymous Attestation (DAA) which is specified by the Trusted Computing Group (TCG) represents such an anonymous digital signature scheme. It is based on RSA technology which has its disadvantages related to the requirements of fast processing times and low memory usage as mentioned before. Therefore it was a necessity to design and implement a new approach which is executable on low powered devices. This approach of a new anonymous credential system based on ECC is evaluated in this thesis.
The proposed algorithm was part of a research project led by Kurt Dietrich at the Institute for Applied Information Processing and Communications at TU Graz. This anonymous credential system, further denoted as KARIM-ACS, should provide fast execution times on low powered devices while hiding the user's identity. The expression KARIM does not have a special meaning or background, it was just the name of the proposed algorithm we implemented in this thesis. Therefore, we named the entire scheme KARIM-Anonymous Credential System. It aims to provide the properties of correctness, unforgeability, unclonability, unlinkability, revokability and practicability to the user as explained by Chen et al. [40] and listed as follows:

- correctness: A user with a valid credential is able to authenticate himself to a service provider.
- unforgeability: No authentication can be executed without a valid credential.
- unlinkability: Full anonymity must be provided. Two different signatures produced from the same entity must not be linked to each other.
- unclonability: No cloning of valid credentials is possible.
- revokability: A mechanism for revoking credentials or entire platforms must exist.
- practicability: An efficient implementation of all protocols should be possible. The provided algorithms should be fast and based on well-established standards.

With this properties the proposed scheme aims to provide total anonymity to the user. For this thesis we implemented and evaluated the KARIM-ACS to gather information about performance and memory usage. Furthermore, we investigated if the announced properties hold against attacks.

1.2 Motivation

A lot of effort is put into research about mobile anonymous authentication. The problem is that there is no scheme which is widely used, as described in the paper about a lightweight anonymous authentication system for embedded devices using TLS and DAA by Chen et al. [40]. In this paper they explain how to use the setup of a mobile network to implement anonymous authentication. The issued credential may then be used against service content providers to access arbitrary services via the world wide web. The goal is to create an authentication scheme to fulfill those requirements. However, not only mobile phones can be used for such operations. Other mobile devices like Smart Cards can also provide the necessary hardware to implement anonymous authentication schemes as mentioned before, even though the hardware components are not that powerful. Therefore, fast algorithms are required to achieve good performance values. Furthermore, it could be possible to use near field communication (NFC) as a data transfer channel between the signer and a verifier. NFC was introduced in 2004 and provides a standard for short range communication based on radio frequency [41]. Many new areas arise by using a cell phone or a Smart Card with integrated NFC technology. Direct interaction with posters, magazines or products to get related information in real-time or even the electronic wallet are possible use cases[4].
In state-of-the-art mobile phones we already have powerful CPUs, enough memory and RAM size for fast executions. In comparison, a Smart Card has a low powered CPU, much less RAM size and also constraints on memory size. Therefore, it is not as easy to provide satisfying processing times. Moreover, using NFC in combination with Smart Cards accumulates new complexity since the time in the NFC field is limited. All calculations need to be finished before the two entities are out of range and the communication distance of NFC is approximately 10cm [41]. This should not arise an issue with newer cell phones since they are fast enough for the used algorithms. Problems may occur when dealing with powerless devices since the computation times are much longer. If we want to use mobile anonymous authentication in various mobile devices we need to find a scheme which is portable, fast and still provides the necessary security.
As explained before the intention is to use mobile authentication systems on embedded devices. E-passports are gaining more importance, where biometric information and other sensitive characteristics are stored. To protect personal data at the border control we need secure protocols, which is discussed by Pasupathinathan et al. [32]. Fast and efficient algorithms could also be used to improve Smart Card security. This is a reason why we have chosen the MCB2130 evaluation board for testing purposes. The board comes with an ARMv4 architecture, which embeds an ARM7TDMI processor and 32kB of RAM size and is therefore comparable to a Smart Card. Bán gives an overview about the ARM7TDMI processor architecture in his master's thesis [4].
The research area of embedded devices including ECC is relatively new compared to widely used algorithms like RSA. But a powerless embedded system is usually not capable for fast execution of schemes which are based on RSA technology. Keys with

[4] http://java.sun.com/developer/technicalArticles/javame/nfc

length of 1024 or even 2048 bit are necessary to provide the required security. Due to these circumstances it is important to find new approaches based on smaller key sizes and faster computation times.
Hence, it is mandatory to protect the entire system against attacks. Properties like unlinkability or unforgeability are also investigated in this thesis. Furthermore, it gives us the opportunity to see how difficult it is to create a new anonymous credential system or group signature scheme. Not just technical constraints but also security issues need to be considered while developing such a new system for embedded devices.

1.3 Approach

The proposed scheme uses ECC as cryptographic mechanism. In contrast to RSA we do not need such long keys anymore to provide the same security standard. ECC uses much shorter keys in range of 192 bits up to 521 bits for high security applications which leads to the need of less computational power. Embedded devices usually have less computational power. When we talk about Smart Cards or RFID chips with small memory and slow processors, it is obvious that a signature calculated with RSA takes too long. The new mobile phone generation does not usually have those problems anymore since they are powerful with dual core processors up to 1,5 GHz and 512 MB of RAM or even more. Therefore we won't be able to receive meaningful results by testing the digital signature scheme with modern cell phones. For this thesis we have worked with very small powerless devices to get meaningful results about the chosen scheme and the included algorithms. This is the reason why we have chosen the MCB2130 evaluation board which is going to be explained in Section 5.5.1. It is a developer board which provides the necessary environment for our test purposes.
The entire system includes several entities, which will be explained in detail in Chapter 3. The issuer, which is part of the server, is responsible for creating a requested credential for a client who is allowed to receive one. The client side is divided into two parts, the normal and the secure world. The normal world talks via a monitor to the secure world, where all the sensitive data, like private keys, are stored. The monitor is operated by the secure world and handles the data exchange. It is important that sensitive data never leave the secure world to ensure trustworthiness of the entire system. Moreover, the normal world interacts with the issuer via SSL/TLS connection to transfer data packages in a secure way. Another important entity is the verifier which can be a service provider or any other third party. he verifies received signatures from a client. Therefore, a client is able to authenticate himself to the verifier while hiding his identity.
The advantage of introducing the separation of the client side is that the secure world uses parts of the resources on the client platform like CPU or RAM. This means the normal and the secure world share the same resources. Therefore, executions are not restricted to the resources of a secure hardware element like a TPM which are usually slower.
The approach of using ECC in combination with a faster execution environment for

sensitive calculations gives us the opportunity to gain good performance results for the KARIM-ACS. Our focus lies on the calculation times and memory usage of the secure world, since this is going to be the bottleneck of the entire system. All calculations including sensitive data and complex algorithms are going to be executed in this trusted area. The secure world will be implemented in C to gain the ability of easy cross compilation to other platform architectures. For further investigations this allows us to include different platforms in our evaluation process to achieve comparable results. Other parts of the system, like the server side, are going to run on powerful machines where performance does not play such a big role.
Furthermore, the KARIM-ACS provides two different solutions. We investigated both where the first is the basic scheme not using attributes as explained in Sectionr̃efsec:baseSolution. The second approach is to include attributes of a person like age, height or eye color where information like this is compulsory to gain access to certain services (e.g. cigarette vending machine) as described in Section 4.3.
For the evaluation of the proposed scheme we defined execution times which seem to be convenient for our purposes. The signature computation usually takes longest. Therefore, we define a computation time of around one second to create a signature. Additionally, the verifier also needs computational steps to verify such a signature. The verification of a received signature should not take longer than 300 ms as defined in [43]. This brings us to a processing time for an entire authentication procedure of maximum $1,5$ seconds.
Another important issue for this thesis was the security of the KARIM-ACS. We investigated the properties of the proposed scheme stated in Section 1.1 and found security flaws in the join and signature protocols.

1.4 Applications and Use Cases

The requirements for this project are diverse. Privacy, anonymity and faster processing times on low powered embedded devices are important facts. Furthermore, there are some uses cases which give a good reason to do research in this field like *e-Ticketing* or *Road Pricing*. ECC is a promising and already widely used scheme which is still under intense research.
To ensure privacy and anonymity, it is usually necessary to accept some limitations on usability and/or performance of an application. An RSA signature for example needs a lot of computational effort, which may result in very time consuming processes. This can lead to usability issues, because responding times of an application could be too long. To gain anonymity with DAA a secure element is needed which is usually not embedded in every mobile device. New ideas and approaches are required to keep up with the fast growing use of new technology. The proposed scheme could be an alternative way to use any embedded device which is set up properly, without the need to install further hardware. ECC gives us the opportunity to use short key sizes while providing a reasonable level of security and shorter computing times.

1.4.1 e-Ticketing

The purpose of an electronic ticket is to ease the purchasing procedure and it replaces the use of common tickets made of paper. e-Ticketing can be used with cell phones as well as with other mobile devices like Smart Cards. With a mobile phone you can directly buy the ticket online, which is then usually sent to you or downloadable as shown in Figure 1.1.
Using these new potentials seems to be quite convenient to the user. But privacy and moreover anonymity are typically not considered in systems like that. A user is easily identifiable by the content service provider and the ticket verifier, since personal data is exchanged between the entities.
To provide anonymity and hide personal data a anonymous credential system would be best practice. Additionally there are advantages for the ticket provider too. If a user is able to share his credentials and private keys among his friends everybody would be able to get access due to the ability of creating valid signatures. Therefore, these data is stored securely in the trusted environment where the user does not have direct access to. The goal is to protected the ticket provider against ticket fraud and the user against data acquisition. Due to these circumstances it is important to create user friendly systems ensuring privacy and security to the customer. Such mobile-commerce applications will be used more often if the application handling is secure and convenient for the customer.

1.4.2 Road Pricing

The maintenance of roads is a costly undertaking, which raises the topic of road pricing. There are a lot of options how to pay the toll. Sometimes there are toll booths where the driver has to stop, or a video surveillance system checks if the particular car has the permission to pass. A video system has the capability of tracking the car movements, since it saves the location where the car went through the toll station. This could be a violation of privacy, because the license plate usually corresponds to a particular person.
The introduction of a mobile anonymous authentication system for road pricing could be a convenient way to avoid tracking. A payment scenario could be to purchase a permission for a certain period. A trusted third party issues a credential along this permission. This credential is then used to produce a signature which can be verified by a checkpoint alongside the road. A signature may be precalculated which speeds up the entire process passing a checkpoint since only the verification is necessary anymore. The precalculation is omitted if we use revocation since a unique value has to be computed for each signature. This is explained in more detail later in this thesis.

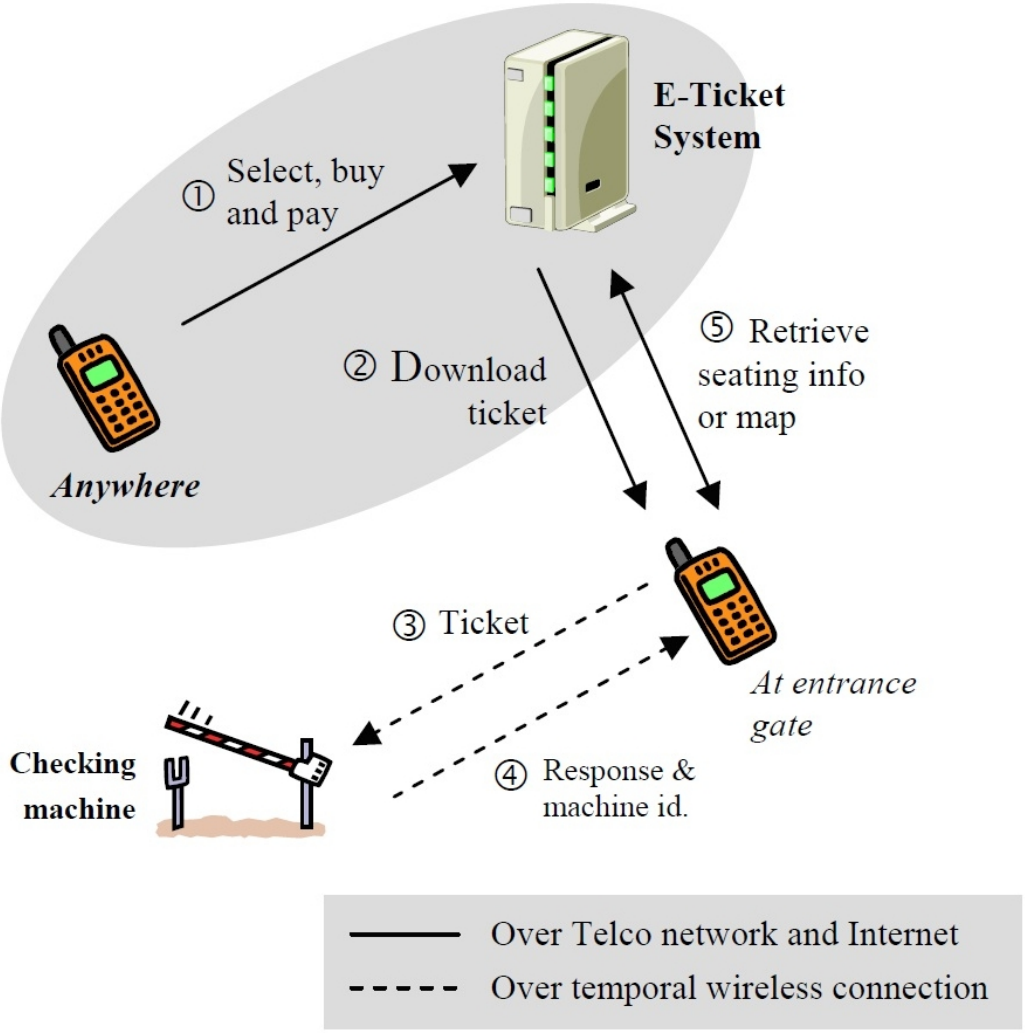

Figure 1.1: Flow of Ticketing using a Mobile Device [21]

1.5 Related Work

A lot of work in the area of anonymous credential systems in combination with elliptic curves has been done in the past. Smyth, Ryan and Chen [35] talk about user controlled anonymity dependent on a trusted platform module and show that this system is secure. They use the ECC Direct Anonymous Attestation scheme introduced by Brickell, Chen & Li [12]. Another approach is discussed by Sudarsono, Nakanishi & Funabiki [37] where a pairing-based anonymous credential system based on ECC is used to support the privacy-enhancing electronic ID (eID).
It is also from high interest to implement high level security on small devices like Smart Cards. Balasch's [3] master's thesis covers the topic of an RSA based anonymous credential system where he implements a simplified version of the DAA scheme. The goal is to run the protocol on an 8-bit AVR microcontroller using a Smart Card as a secure hardware element instead of a Trusted Platform Module (TPM). The use of RSA as a cryptographic basis on small devices may lead to time expensive computations due to long keys. Therefore, ECC based schemes are an alternative to RSA which uses much shorter key lengths. Gura et al.[19] implemented and compared ECC and RSA. They laid focus especially on elliptic curve point multiplication of 160, 192 and 224-bit NIST/SECG curves over $GF(p)$. The results were compared to RSA 1024/2048-bit operations on two 8-bit microcontrollers where they analyzed the viability of public key cryptography on small devices without using hardware acceleration.
More research for Smart Card security was done by Batina et al.[5] introducing a proof of possession of an anonymous credential implemented on Smart Cards using bilinear pairing based on ECC. They decided to use ECC because of the small key and certificate lengths which reduces the time and bandwidth needed for transmission. Another approach for an efficient implementation of an anonymous credential system on Smart Cards was introduced by Mostowski and Vullers [29]. They used Microsoft's U-Prove technology for a performance evaluation on a MULTOS Smart Card platform and compared this platform to a Java Card.

1.6 Outline

In Chapter 1 we introduce the project to get an overview of what we tried to accomplish, which other work has been done so far and what our motivation was to investigate this algorithm. Definitions, prerequisites and requirements for this project are discussed in Chapter 2 where we are defining some necessary components, talk about the mathematical background of anonymous credential systems and the difference between DAA and the proposed scheme. Furthermore we describe some use cases, and the derived requirements for this system directly lead to the proof of concept in Chapter 3. The important System architecture is explained, followed by discussion of the three main parts of this concept. The protocol definitions and algorithms used to produce such an anonymous credential system are shown in Chapter 4. Facts about the implemen-

tation and the test environment are discussed in Chapter 5. Furthermore, test results about the performance evaluation of the secure world are shown in Chapter 6 were we also discuss the security flaws found in the proposed algorithm. In Chapter 7 we summarize the scientific findings of the proposed scheme belonging the performance evaluation and the security issues.

Chapter 2

Prerequisites & Requirements

To fulfill all the necessary project requirements we need to combine different standards, tools and communication methods. To get an overview of the techniques used, the important definitions and prerequisites are discussed in this chapter. Firstly, it is necessary to define the difference between DAA which is also a anonymous credential system and the KARIM-ACS based on ECC since this is important for our investigations. Hence, we define different components we used for this work to achieve meaningful results. Secondly, it is important to get an overview about the mathematical background of elliptic curve cryptography and field arithmetic explained in Section 2.2. At the end of this chapter in Section 2.3 we talk about the derived requirements for our implementation.

2.1 Definitions

Some necessary definitions are explained in the next sections. Firstly, we define what a digital signature is and why we need it. Furthermore, the Direct Anonymous Attestation scheme, which is specified by the Trusted Computing Group[1] (TCG), is discussed in Section 2.1.2. Hence, it is important to point out the difference between the KARIM-ACS, as implemented and evaluated in this thesis, and DAA. Next, we define the ASN.1 standard[2] [23] which is described in Subsection 2.1.4 and the the X.509 certificate standard [25], which is explained in Section 2.1.5 and usually used in PKI systems for authentication purposes. We also used the Java Keytool to manage the certificates in our system which is explained in Section 2.1.6. SSL/TLS, which is necessary for a secure connection between the client and the server, is discussed in Section 2.1.7. For porting the implemented client part to other platform architectures, a cross-compilation Toolchain was required. Section 2.1.8 gives an overview on how such a Toolchain is used for cross-compilation to other platforms.

[1] http://www.trustedcomputinggroup.org
[2] http://www.itu.int/ITU-T/asn1

2.1.1 Digital Signature Scheme

First of all it is important to define what a digital signature scheme is and what it is good for. If two entities are digitally communicating with each other they exchange messages or any other kind of data. To ensure the integrity of this message the sender needs to sign it and the recipient should be able to prove the signature. This is accomplished by public key cryptography invented by Whitfield Diffie and Martin Hellman [16]. The sender computes a signature on a message m with his private key which is only known to him. A publicly know identity (public key) of the sender is available for the receiver which is able to verify the signature of the received message. No one is able to produce a valid signature on data signed by the sender. So the integrity of the data is ensured. Furthermore a trusted third party signs the public key so the recipient is also able to verify that the message comes from a trusted entity. The method of a digital signature scheme corresponds to the process of signing papers by hand.[28]

2.1.2 Direct Anonymous Attestation

Direct anonymous attestation (DAA) is a authentication or group signature scheme specified by the Trusted Computing Group. The purpose of DAA is to sign data while hiding the signers identity. This is achieved by altering the digital signature scheme. The public identity of the signer is represented by the group's public key which is distributed to any verifier. To verify a signature of a message m the recipient only needs the group's public key and some additional signing parameters from the signer. The sender is able to sign data with a credential and his private key on behalf of the group he joined beforehand. The procedure of the setup and join process is explained in more detail in [8]. The system works with a secure hardware element (SE) like a Trusted Platform Module[3] (TPM) which executes all sensitive calculations. The advantage of such a module is that calculations with sensitive data are only processed inside the secure element. Private keys or other sensitive data never leaves this environment. It is usually embedded in a notebook, mobile phone or any other device but it is very low powered and therefore complex computations are processed slowly.
A TPM has three different types of keys, including the Endorsement Key (EK), the Storage Root Key (SRK) and the Attestation Identity Key (AIK). The EK is a unique 2048 bit RSA key pair, whereas the private portion is never exposed to the outside. If a user takes ownership of a TPM, a new SRK is created. It is also a 2048 bit RSA key and is used to store additional private keys of the user, therefore it is considered the root key. The AIK is a second key pair and acts as a pseudonym for the EK which means that the EK signs the AIK and a trusted third party like a privacy certification authority (Privacy-CA) issues a credential for this key. The AIK is used to perform signatures, since the specification of the TCG does not allow signing data with the EK.[39]

[3] http://www.trustedcomputinggroup.org/resources/tpm_main_specification

Table 2.1: Mobile Device configuration [26]

Device	*Nokia E72*	*Nokia 6212 classic*
JVM	J2ME	J2ME
Execution Mode	interpreted	interpreted
Main CPU	ARM11	ARM11
CPU frequency	600 MHz	400 MHz
Operating System	Symbian OS v9.3	Nokia OS

Table 2.2: Runtimes on the Mobile Devices [26]

	`Nokia E72`			`Nokia 6212 classic`		
Scheme	Join	*Sign*	*Verify*	Join	*Sign*	*Verify*
HeGe	8,68s	5,01 s	3,84 s	38,92s	23,33 s	17,40 s
BCC	-	3,10 s	1,99 s	-	-	-

DAA was first introduced by the Trusted Computing Group within the TCG Software Stack Specification v1.2 [38]. The aim of DAA is to use the AIK to produce anonymous signatures on behalf of a group to hide the users identity. A verifier is able to verify a signature signed by a trusted entity not knowing anything about the signer. A detailed description of the protocol can be found in [8]. Further investigations on privacy and anonymity in the area of DAA are done by Jan Camenisch [9], Liqun Chen [11] and many others.
To get a feeling about processing times using a DAA scheme we are going to have a short look into a previous project we were working on. It implements and evaluates a DAA scheme of He Ge and Stephen R. Tate [18]. To get an overview about computation times on embedded devices using DAA, the paper with the title *A Direct Anonymous Attestation Scheme for Embedded Devices* [26] shows results using two different mobile phones as displayed in Table 2.1 and Table 2.2. We can see that these results are not usable for low powered devices, since executing a signature takes at best around 3 seconds with a 600 MHz processor, which is 10 times faster than the MCB2130 Evaluation Board we have used for the performance evaluation. Due to these circumstances it is very important to put more effort into research corresponding to ECC, because the key sizes are much smaller. Moreover, we are getting rid of the very time consuming modular exponentiations which are extensively used in such a DAA scheme to provide the necessary security.
The DAA scheme aims to hide the user's identity. The properties of correctness, unforgeability, unclonability, unlinkability and revokability as stated in Section 1.1 need to be addressed by such a scheme. Therefore, the anonymous user should be able to use services from different service providers without reveling private data.

2.1.3 KARIM-ACS vs. DAA

Secure elements, like a TPM, are not commonly integrated in state-of-the-art mobile devices. A DAA scheme therefore cannot be executed on such a device due to the

missing hardware component. It is necessary to find an alternative way to provide security and privacy for the user. The goal of the KARIM-ACS is to take existing hardware to protect running applications in a secure environment. It is important that the secure environment gets its own resources on the platform such as CPU, memory and registers to ensure a separate working area. We want to achieve a secure world which is a trusted area on the entire platform. The advantage of a secure world is faster execution because it has access to the standard platform resources unlike a TPM which is en extra module with hardware constraints.
A user is supposed to authenticate himself to the secure world in order to gain access for sensitive computational executions. This can be PIN authentication or password protected key usage. Sensitive data handling can include storage of secret keys, calculating signatures or other sensitive computing steps which use private parameters. It must be mentioned that private data like the client's private key or other generated secret keys may never leave the secure part of this system due to privacy concerns. If an attacker gets a hold of private data he can sign messages on behalf of that particular user. He is also able to alter messages sent by the user and renew the signature so the verifier does not notice the fraud. How to set up those two separated parts is described in Chapter 3. The KARIM-ACS also requires a trusted issuer which has the permission to issue credentials. The issuer usually resides on a server which is a trusted authority and has knowledge about the access rights of a particular platform. Such a credential for a given private key ensures the authenticity of a user to any verifier. Since the credentials are issued as certificates, we can use certificate revocation as a mechanism to filter compromised systems, which is explained in Section 4.4.
As a conclusion we are now able to differentiate between DAA and the KARIM-ACS. DAA needs a TPM which is not provided on every platform. Therefore it is not possible to integrate this scheme on arbitrary devices. The KARIM-ACS does not necessarily need a secure hardware element and has the ability to be deployable on various devices. Hence, the KARIM-ACS can provide a faster execution environment than DAA due to the fact of using the existing resources like CPU and RAM on the target platform. Furthermore it needs to be mentioned that the KARIM-ACS can be seen as an abstraction of the DAA scheme because it is also possible to use secure elements within the trusted environment which is indicated in Section 3.3.

2.1.4 ASN.1

Abstract Syntax Notation One or ASN.1 defines a standard for the collection of data types. It is platform independent and can be encoded with different encoding rules like BER (Basic Encoding Rules) or DER (Distinguished Encoding Rules) [24], whereas DER is a subset of BER and specifies a unique data description on the bit level. Therefore every data block has only one valid encoding, which is necessary for digital signatures and data transfer between different platforms. ASN.1 is also used in the X.509 certificate standard which is described in Section 2.1.5. More information about the ASN.1 standard can be found in [7] and [23].
For this project, the DER standard is used to ensure unique encoding for signatures

and other data which is transferred between platforms with different internal data representations. Also the Java library explained in Section 5.4.1 supports the data handling with DER.

2.1.5 X.509 Certificates

X.509 certificates are used in PKI systems to ensure a user's identity. The user's public key is sent to a trusted source like a certificate authority (CA), where it is certified to a corresponding private key. This means that the identity of the user is ensured through his public key certificate in combination with his private key. If data is signed by a particular user a verifier cannot just verify the data integrity of the received message, he is also able to ensure that the message comes from a trusted entity. The CA signs the certificate, including important information like a unique user ID and the user name along with the CA ID, the CA name, a serial number, the certificate version, the validity period and different optional extensions. Also the signing algorithm ID is included in the certificate. Furthermore it is unforgeable, which means that only the trusted CA is allowed to apply changes. Therefore any third party is able to verify a valid certificate and the corresponding user. The ASN.1 structure, as mentioned in Section 2.1.4, is used to represent the entries of such a certificate. [25] gives a more detailed explanation about the X.509 standard and its ASN.1 representation.

2.1.6 Java Keytool

The Keytool is used to manage the certificates and keys on the server and the client side in so-called key stores. Both the server and the client have to authenticate themselves to each other to ensure trustworthiness. Key stores are separated into a trust store and a normal key store to separate the certificates for authentication. The trust store holds the public certificates of all authorized entities, in order to be able to establish a connection. The key store includes the certificates which are sent to the other entity for verification purposes. If both sides verify the public certificates correctly, a SSL/TLS channel is created between them. How to create a key and a trust store can be found at http://www.coderanch.com/t/134353/Security/Importing-certificate-keystore.

2.1.7 SSL/TLS

For a secure connection between two entities, a server and a client side, we use SSL/TLS, which extends the normal TCP protocol stack by the SSL layer, as shown in Figure 2.1[4]. The book *SSL and TLS - Designing and Building Secure Systems* written by Eric Rescorla in the year of 2000 [33] gives a detailed explanation on SSL/TLS

[4] http://www.simple-talk.com/dotnet/.net-framework/tlsssl-and-.net-framework-4.0

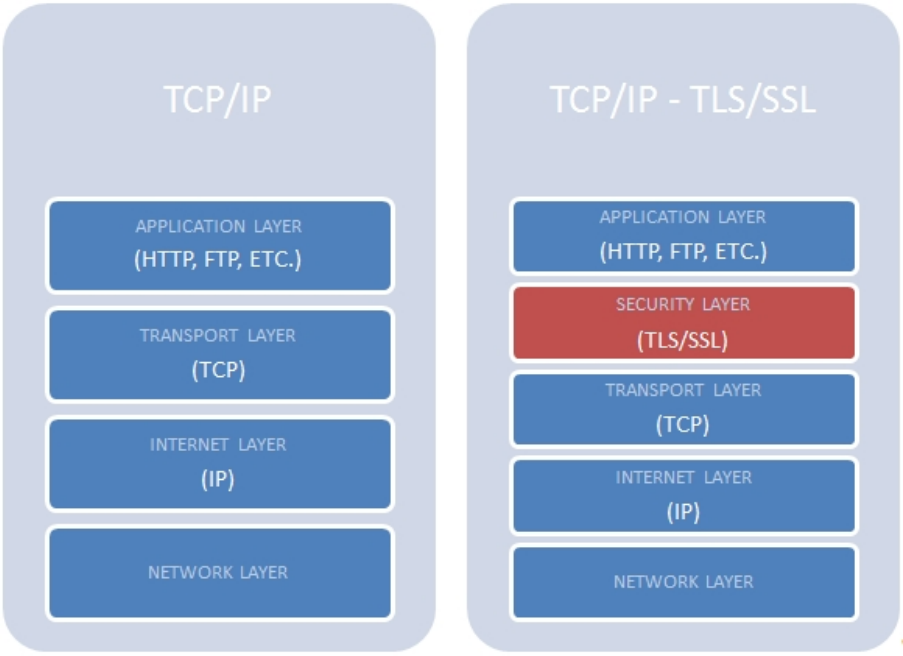

Figure 2.1: Standard TCP/IP Stack vs. SSL TCP/IP Stack

communication and their assets. Rescorla stated that the most common case is server-only authentication using RSA, but in our case we need both sides to authenticate themselves to each other. This is to ensure that the user on the client side who wants to join a group for hiding his identity is actually allowed to do that, and the server is the one that is trusted by this client application. The common way in SSL to authenticate both entities is the client authentication mechanism, as displayed in Figure 2.2 and explained by Rescorla, where the client is also requested to send his certificate and a certificate verification message to the server. This verification message is signed by the client's private key associated to the sent public certificate. After a successful handshake the pre-master secret is exchanged and the master secret will be derived from it to produce all the necessary keys to protect the data. Figure 2.3[5] shows the derivation from the pre-master secret to the encryption keys. A master secret is always used once for a session and must be kept secret. If it is compromised, the entire system is vulnerable for attacks. Furthermore, it also supports the X.509 certificate standard, which is required for the proposed system.

2.1.8 Cross Compiler Toolchain

The most important part of the proposed system, the secure world, is written in C. Since we are using the MCB2130 evaluation board for testing and performance evaluation, it is necessary to cross-compile the written code for this ARM platform. A more

[5] http://www.securitystuff.web.id/cryptography/mengenali-lebih-dalam-protokol-secure-socket-layer-ssl

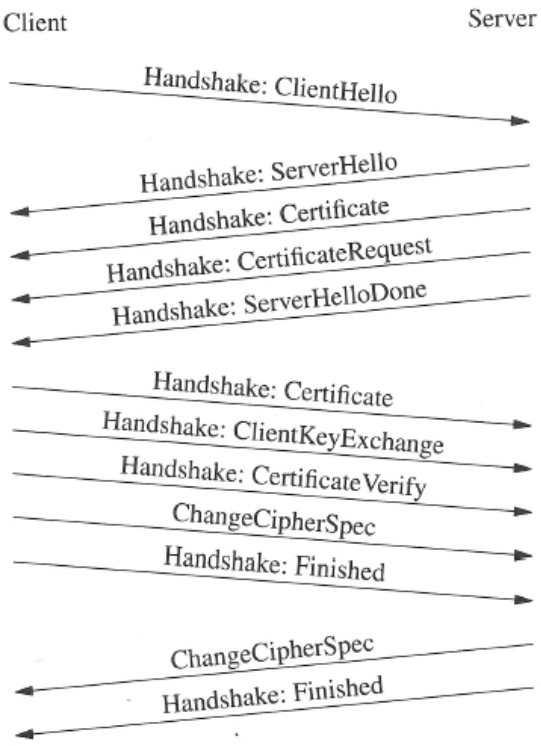

Figure 2.2: SSL Client Authentication Handshake [33]

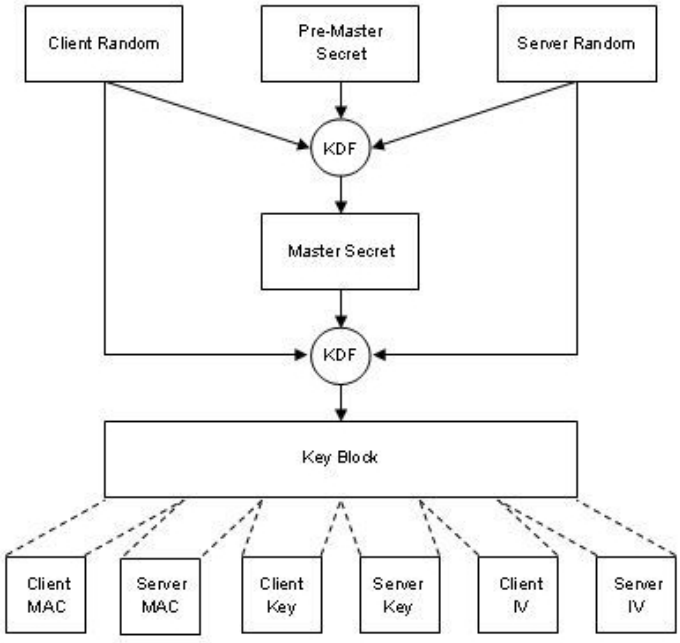

Figure 2.3: SSL Key Derivation

detailed description of the test environment is given in Section 5.5. The Code Sourcery[6] toolchain is sufficient for our needs. We used the μVision 4 IDE which supports the Code Sourcery toolchain out of the box. μVision 4 is a development platform for embedded applications which is provided by Keil[7] and used to generate code running on our evaluation board. It is easy to set up and work with. To compile the source code for the evaluation board we used the *ARM-NONE-EABI*[8] target alias where *ARM* defines the architecture, *NONE* means we are not building for a specific operating system and *EABI* says how binary files and libraries are stored and which calling conventions are used, and it tells us something about the register usage[9]. To load the program onto the device via COM port we used the Flash Magic Utility, which can be downloaded from http://www.flashmagictool.com. The combination of these tools made it possible to run and evaluate the implemented secure world.

2.2 ECC Mathematical Background

To understand elliptic curve cryptography and to get a better overview about this topic, some mathematical background is needed. This section informs about finite field and elliptic curve arithmetic and explains why the elliptic curve discrete logarithm problem (ECDLP) is essential for the security of an ECC scheme. Hence, the zero knowledge proof is explained which is very important for this anonymous digital signature scheme. The book *Guide to Elliptic Curve Cryptography* by Hankerson et al. [20] gives a detailed description about the mentioned topics and some very useful examples. It also includes different methods for attacking an ECC scheme and how to prevent those attacks. To get a quick overview about ECC, it is useful to go through the implementation guide of Anoop [2], which explains the major parts of ECC briefly. *Note: Points on the elliptic curve are always written as capitalized and integers as lowercase letters.*

2.2.1 Finite Field Arithmetic

A field $\mathbb{F}$ is a set of elements with two operations, addition (denoted by $+$) and multiplication (denoted by $\cdot$). A set of elements is called a field $\mathbb{F}$ if the field axioms are fulfilled, defined as follows: [20]

1. $(\mathbb{F}, +)$ forms an abelian group with identity denoted by 0

   Associativity: $(a + b) + c = a + (b + c)$

   Commutativity: $a + b = b + a$

   Identity: $a + 0 = a = 0 + a$

[6] http://www.codesourcery.com
[7] http://www.keil.com/
[8] http://www.codesourcery.com/sgpp/lite/arm/portal/subscription?@template=lite
[9] http://www.kunen.org/uC/gnu_tool.html

Inverses: $a + (-a) = 0 = (-a) + a$

2. $(\mathbb{F}\backslash\{0\}, \cdot)$ forms an abelian group with identity denoted by 1

   Associativity: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

   Commutativity: $a \cdot b = b \cdot a$

   Identity: $a \cdot 1 = a = 1 \cdot a$

   Inverse: $a \cdot a^{-1} = 1 = a^{-1} \cdot a$ for every non-zero element

3. Distributivity: $(a + b) \cdot c = a \cdot c + b \cdot c$ for $a, b, c \in \mathbb{F}$

Furthermore, a field contains at least two elements and every non-zero element in the field has its multiplicative inverse such as $x \cdot x^{-1} = 1$. A field with a limited number of elements is called a finite field or Galois Field (GF). For example the simplest finite field is $GF(2) \rightarrow [0, 1]$. The number of elements in the field is called field order. The order is denoted by $q = p^m$, where p is a prime number and called the characteristic of the finite field, and m is a positive integer. If $m = 1$ the field is called prime field usually denoted as $\mathbb{F}_p$ or if $m \geq 2$ it is called extension field denoted as $\mathbb{F}_{p^m}$. The prime field $GF(p)$ of order q contains elements from $[0, 1, 2, ..., p - 1]$ which results in $q = p^1 = p$.[20]

2.2.2 Elliptic Curve Arithmetic

With the underlying field arithmetic we can now define the elliptic curve arithmetic, which is, for cryptographic systems based on elliptic curves, the calculation of points on the curve. The general elliptic curve E over a field K is defined by the Weierstrass equation [20]

$$E : y^2 + a_1 \cdot x \cdot y + a_3 \cdot y = x^3 + a_2 \cdot x^2 + a_4 \cdot x + a_6 \tag{2.1}$$

where $a_1, a_2, a_3, a_4, a_6 \in K$. Due to the fact that the field characteristic $p \neq 2, 3$ the Weierstrass equation can be simplified to

$$E : y^2 = x^3 + a \cdot x + b \tag{2.2}$$

(derivation can be found in [20] - Section 3.1.1) where $a, b \in K$ and x, y are coordinates which satisfy the equation. Each point, with its coordinates (x,y), which fulfills the equation is a point on the curve. The set of points on the elliptic curve over K, further denoted by $E(K)$, forms an abelian group on the elliptic curve with group order n and the point at infinity ∞ as its identity. The discriminant of the elliptic curve is denoted by $\Delta = -16 \cdot (4a^3 + 27b^2)$. If $\Delta \neq 0$ the elliptic curve is nonsingular which means that there are no cusps or self-intersections (nodes) as shown in Figure 2.4[10]. If two branches meet at a point its called a cusp when the tangents of each branch are equal. Furthermore there are two basic operations on E which are the point addition and the

[10] http://mathworld.wolfram.com/EllipticDiscriminant.html

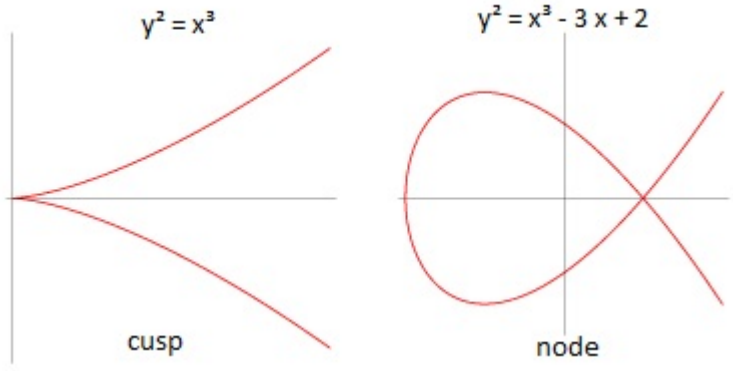

Figure 2.4: Cusp & Node

point doubling. Point addition is easily explained as a short example where $P(x_1, y_1)$ and $Q(x_2, y_2)$ are two points ($P \neq Q$) on the curve. First draw a line through P and Q: the line intersects the elliptic curve in a new point $-R$. To get the result $R = P + Q$, reflect $-R$ about the x-axis as shown in Figure 2.5(a). The point doubling is a special way to add a point $P(x_1, y_1)$ to itself. The calculation of $R = 2 * P$ or $R = P + P$ needs a tangent line to the elliptic curve in point P. This line intersects the curve in a new point $-R$, which is then reflected about the x-axis to the resulting point R shown in Figure 2.5(b).[20]

2.2.3 Elliptic Curve Discrete Logarithm Problem

As mentioned before, the ECDLP is essential for the security of ECC. Assuming an elliptic curve E over a prime field K with the base point $G \in E(K)$ and a point $P \in E(K)$ and a calculated point $Q \in \langle P \rangle$. G is the base point of E or the so-called generator point which is chosen from $E(K)$ where every point on E can be a generator point except the point at infinity. The generator point is publicly known and has to be defined prior to any calculations. The calculated point $Q \in \langle P \rangle$ where $\langle P \rangle$ denotes the generated group by P such as $\langle P \rangle = \{0, P, P + P, P + P + P, ...\}$. $Q = l * P$ is therefore a multiple of P where l is an integer in range of $[0, p - 1]$ and p is prime. p is also the generator of the underlying prime field. Therefore, l is called the discrete logarithm of Q to the base P ($l = log_P Q$) where it is hard to find l if the chosen p is sufficiently large. This is also called the ECDLP.[20]

2.2.4 ECDL Zero-Knowledge-Proof

The principle of a zero-knowledge-proof (ZKP) is to show a verifier that a signer has certain information without revealing it [17]. The main idea is to hide private data, like secret keys, and still be able to prove the possession of it as it is necessary for anonymous

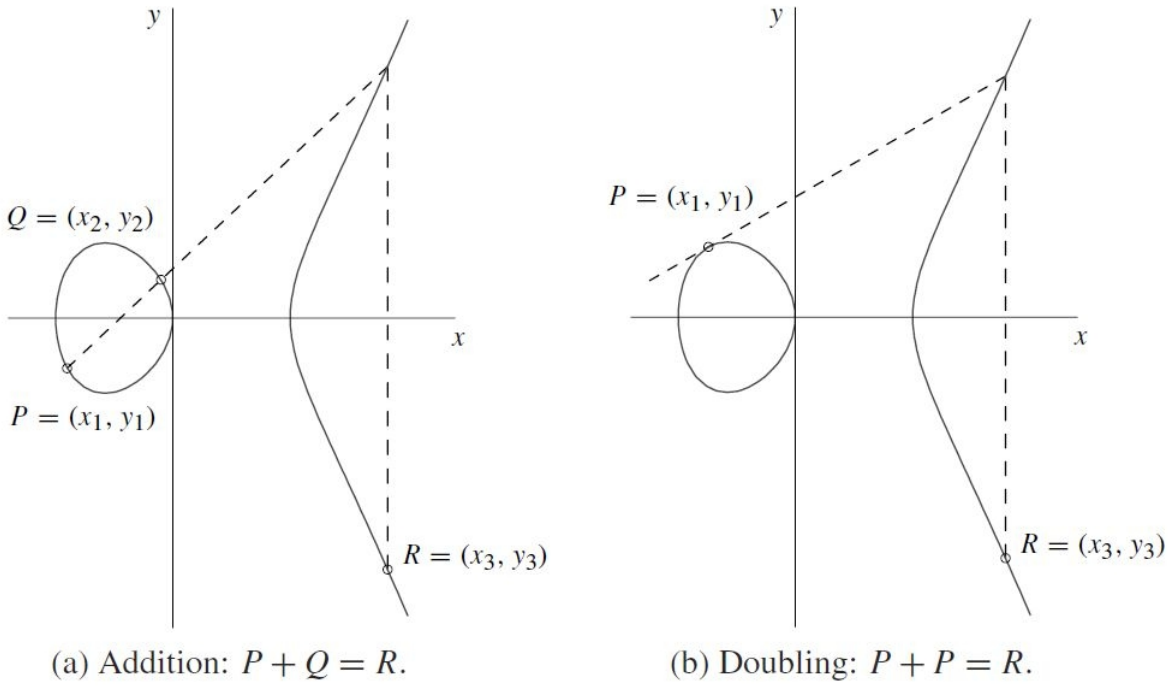

Figure 2.5: Geometric addition and doubling of elliptic curve points [20]

authentication schemes. Nobody should be able to recalculate those hidden values by obtaining the actual proof of knowledge. A lot of effort was put into research for ZKP based on RSA cryptography for example by Feige et al. [17] and by Carmer et al. [15]. For the anonymous credential system we used the ZKP mechanism on elliptic curves to prove the possession of a self-generated secret which is only known by the signer. Hence, we exploit the knowledge that it is computationally infeasible to calculate $s \in K$, the secret value, out of P where $P = s * G$ is a point on the elliptic curve E and G the generator point as mentioned in Section 2.2.3. Almuhammadi et al. [1] discuss this topic in more detail.

EC-based ZKP Example

A short explanation of the elliptic curve discrete logarithm zero-knowledge-proof (ECDL-ZKP) from [1] is given here.
There is a publicly known value $B = m*G$ where G is the generator point of an elliptic curve $E(K)$. The prover PR wants to convince the verifier VE that he knows a secret $m \in K$. PR generates a random $r \in K$, computes $A = r * G$ and sends A to VE. PR also computes $x = r + m$. The verifier chooses randomly if the prover should send him r or x. If VE receives r he computes $A = r * G$ and if he receives x he computes $x * G = A + B$. These steps are repeated until VE is convinced that PR knows the secret m.

EC-Schnorr Signature

For our purposes we used an adaption of the EC-Schnorr [13] signature scheme as a zero-knowledge-proof.
Assuming an elliptic curve $E(K)$, a public key where $P_k = f * G$ which is a point on the curve, a private key $f \in K$ and a message M to be signed. The signer chooses a random number $k \in K$ and calculates a point $Q = k * G$ where G is the generator point of $E(K)$. The next step is to compute a hash value $r = H(Q_x|M)$ where H is a collision resistant hash function like SHA-1 and Q_x is the x-coordinate of Q. Furthermore, the signature value $s = k - r \cdot f$ is computed. The signature parameters (r, s) over M are sent to the verifier.
The verifier knows the public key P_k, the message M and the signature parameters (r, s). First he verifies that (r, s) lie in the interval $[1, n-1]$. Furthermore, he computes the point $Q' = s * G + r * P_k$. With the point Q' it is possible to calculate the hash value $r' = H(Q'_x|M)$. If $r' = r$ the verifier is convinced that the signer knows a secret f which was used to sign the message M because Q and Q' have the same x-coordinate.

2.2.5 System Parameters

Additionally to the mathematical background we also need to define some system-wide parameters which are mandatory for the proposed scheme. He Ge and Steven R. Tate [18] already showed the importance of range checks for a secure system which were adopted and adjusted for the proposed scheme. Since we do not use an RSA based system, we have to find our own boundaries to convince a verifier that we produced a certain value which lies in a given range. These boundaries are lower and upper boundaries for randomly chosen or computed values as described later in Chapter 5.
Therefore, we define the constant integer c_X and the security parameters α, l_f, l_h and l_n. l_h describes the hash length which corresponds to a collision resistant hash function (e.g. $l_h = 160$ concludes to hash function SHA-1) and l_n is the length of the modulus n. $\alpha \geq 1$ and l_f need to be chosen in a way that $\alpha \cdot (l_f + l_h) + 1 < l_n$. The constant integer c_X is defined by the security parameters α, l_f and l_h such that $c_X > 2^{\alpha \cdot (l_f + l_h) + 2}$ which is directly derived from [18] and adjusted for our needs. Since we do not use RSA we need to alter the security parameters for the ECC based KARIM-ACS which uses much shorter key lengths and therefore smaller boundaries.

2.3 Requirements

As already mentioned before a lot of requirements are needed in the entire process to design a usable system. However, not just use cases and different applications define those needs. There are also technical facts which have to be considered. As we already know, RSA needs long key sizes, which leads to time consuming computations. Hence, ECC provides the necessary functionality and security with less computational effort.

Therefore we can, based on this technology, implement schemes on devices with memory constraints and less processing power like Smart Cards. The National Institute of Standards and Technology (NIST) [30] specifies several elliptic curves and their corresponding domain parameters. Curves over prime fields start from 160 bit up to 571 bit. The common standard which is widely used is the P-192 curve, which provides a satisfying, high security level and key sizes which are still computable in constrained embedded devices.
Furthermore we want to reduce the signature and verification times to produce a convenient system for the user. According to the Dedicated Short Range Communications (DSRC) broadcast protocol, which is widely used for traffic surveillance, the verification of a received signature should not take longer than 300 ms [43]. Also the signing process itself should be decreased to a minimum of time as stated in Section 1.3. A good example for a fast procedure could be e-Ticketing or road pricing because it is possible to prepare the signature beforehand and verify it on demand. If it is not possible to sign data in advance the entire signing and verification process needs to be executed which apparently takes more time.
Revocation is also important to have the ability of revoking compromised platforms and credentials. The verifier can check the revocation list before verifying signatures and is able to decline data signed by this entity.
Hence, it is mandatory to implement a trusted execution environment (e.g. secure world) to protect sensitive data like private keys or credentials. The secure world is separated from the normal world and secured against the user. This means the user can allow to trap into the trusted environment to execute computations including sensitive data but he does not have access to private keys or credentials directly. This protects the device not just from attackers but also from the user as he could share credentials among his friends to distribute valid e-Tickets as explained in Section 1.4.1 for example.
Furthermore, the algorithm needs to be secure against attacks and has to provide total anonymity for the client. We want to hide the user's identity by providing the properties of correctness, unforgeability and unlinkability.

Chapter 3

Proof of Concept

In this chapter we are going to discuss the entire system of the implemented prototype. To understand the different processes, the architecture is shown first. After getting an overview we can step into the server side explained in Section 3.2. Hence we are going to walk through the client side with its two different environments. To complete the overview of the prototype we will have a look at the external server and how a third party verifier is able to proof a signature without knowing anything about the signer. Furthermore the idea of a revocation mechanism is explained in Section 3.5.
The entire concept was introduced by Kurt Dietrich in a research project at the Institute for Applied Information Processing and Communications at TU Graz. The goal was to create a new approach for mobile anonymous authentication based on ECC, further named KARIM-ACS. The implementation of the entire system helped us gathering important information about the performance and the security of the system.

3.1 System Architecture

Figure 3.1 shows the system architecture for the different entities. The server is responsible to handle incoming requests which are distributed to the different issuers which are part of the server. Each issuer handles a group which may be joined by different clients. For each group, a group certificate or public issuer certificate is mandatory. To ensure the validity and trustworthiness of such a certificate, it has to be signed by a trusted third party (TTP). In a standard PKI system a privacy CA signs the certificate of the signer. For this approach we are not using a privacy CA to issue the credentials which are needed by the client to sign data. The reason for this is that for every authorization a new key would be necessary to avoid linkability between the signatures of one client. This is why we use a group signature scheme where every client is hidden behind the entire group and therefore must not be linkable to other signatures of its own. An external server obtains all those certificates for further distribution. The external server also needs to check the validity of a certificate by verifying it against the TTP. If verification fails, the certificate might be fake or already on the

revocation list. Leaving out verification may lead to fake certificate creation from an attacker. Anyone could send a self made certificate to the external server pretending he is a trustful entity. Due to these circumstances, a verifier could validate a wrong signature and therefore trust a fake issuer.
Figure 3.2 displays the data flow between the entities. The client sends a request to the issuer for joining a specified group. The issuer checks if the client is allowed to join this group and computes a credential with the received parameters from the join request. The credential and additional parameters are sent back to the client. A further calculation on the client side proves the correctness of the credential which is partly executed in the normal world and the secure world. The secure world is needed to protect private data from being revealed to the outside since an attacker is able to produce valid signatures by obtaining the credential along with the private key. It is also needed to protect private keys or credentials from the user if he intends to share private data among his friends as explained in Section 1.4.1. If all checks were correct on both sides the client is able to produce valid signatures on behalf of the joined group.
The communication between client and server is based on a network connection and protected by SSL/TLS to avoid man-in-the-middle attacks. To establish such a secure connection the server has to authenticate itself to the client and vice versa. The idea for this system is that each platform has a unique key pair which can be considered as the endorsement key EK like a TPM has it. It is possible to implement the EK into any trusted client platform beforehand, either at production time or before handing out a device. Therefore the server knows the public part of this key pair and is able to verify that a join request comes from a trusted client. The encrypted connection shields data from external influences and ensures to both entities the integrity of the sent parameters. Not using SSL/TLS to protect a connection could lead to system attacks. Anybody could send a credential request to receive a valid credential from a certain issuer. With this fault credential the attacker is able to sign data on behalf of that group, which can be successfully verified by any third party. To emphasize the usage of the EK, it is important to know that it is not possible to decrypt data with the EK due to the specification of DAA. The EK is only used for encryption purposes to ensure the trustworthiness of that platform. An EK is always bound to the platform and is never used to sign any data, as specified in [39].
Note: Also the client must verify the certificate of the server via the TTP to ensure a connection to the right instance.
The client itself is divided into two parts, which are explained in detail in Section 3.3. It is not necessary that the two different environments lie on the same device. A connection between the two worlds is sufficient. The idea of separated worlds is to distinguish between sensitive and normal data. Credentials or private keys are considered as sensitive data and must be protected in a secure environment to avoid attacks. Sensitive data is also protected from the user because he does not have direct access to private data as this is important to avoid fraud on intention (e.g. e-Ticketing). If the user has access to the secure world directly he could give his credential to others so they can sign data on behalf of the same group even though they never joined it. Figure 3.3 displays a more detailed view about the client architecture than Figure 3.1.

In Figure 3.1 we can see that all the important parts were mentioned so far, except the combination of the external server and the signature parameters for the verifier. The entire purpose of this system aims to verify a signature correctly. A user wants to stay anonymous while having the opportunity to authenticate himself to a service provider for example. Any verifier who receives the signature parameters from a client needs the corresponding public issuer certificate to verify a signature successfully. The external server provides the group certificate for verification purposes and also holds a revocation list of revoked platforms and credentials. Therefore, the verifier is able to verify the signature or reject it if the verification process fails or the client's platform or credential is revoked.
The entire architecture aims to achieve total anonymity for the client and therefore for the user. The user is able to sign data on behalf of the group he joined while hiding his identity. Furthermore the group signature scheme must avoid linkability between different signatures produced by one client using the same credential.

3.2 Server/Issuers

The server's challenge is to receive client requests and handle those in the right manner. The incoming query from a client has to go through some initial steps to ensure the right treatment of this request. As already mentioned before, the server knows its client public endorsement keys, which is necessary to filter requests. Every server has to know the public part of the client EK so he can verify the allowance to a specified group. Any request not coming from a valid client is rejected. Furthermore every known key corresponds to one or more issuers, so that a client request can be correctly mapped to the chosen group. A server may have more than one issuer to handle different groups. Every group is administrated by its own issuer and stores its own public group certificate. If the authentication succeeds, the client is able to get the public certificate and hence a credential. Such a credential is issued as a certificate and will be sent back to the requester. Now the client is able to sign data on behalf of that particular group. Every issued credential is assigned to the corresponding public key in a way that the issuer knows which credential was issued to what client. It also helps to revoke credentials for compromised platforms.
The server certificate is signed by a trusted third party which is then verified by the client. Every issuer has to sign its public group certificate as well so that no fraud can be committed on issuing fake credentials to a client. The server is also responsible for updating the external servers with all the group certificates. These servers are only allowed to accept and distribute those if the certificate verification succeeds. Due to this certificate distribution the actual server entity does not have to be online all the time. That means for a verification process of a signature it is sufficient that a verifier gains access to one of those external servers.
For revoked credentials or platforms a revocation list is maintained by the server which lists all the suspicious clients. The idea of revocation is not to revoke the private key directly, since this will not be the use case. The client does not have a mechanism to

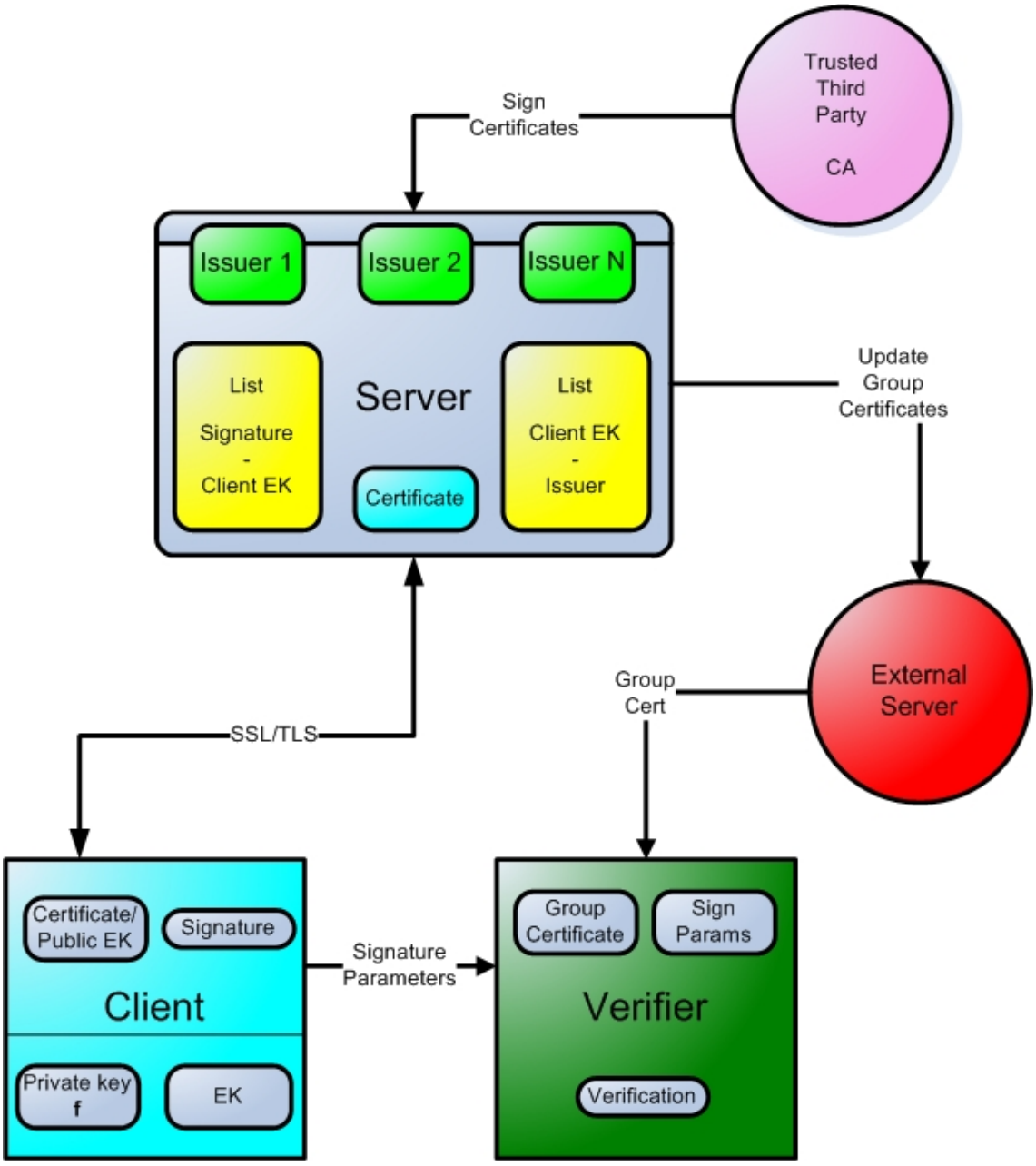

Figure 3.1: System Architecture

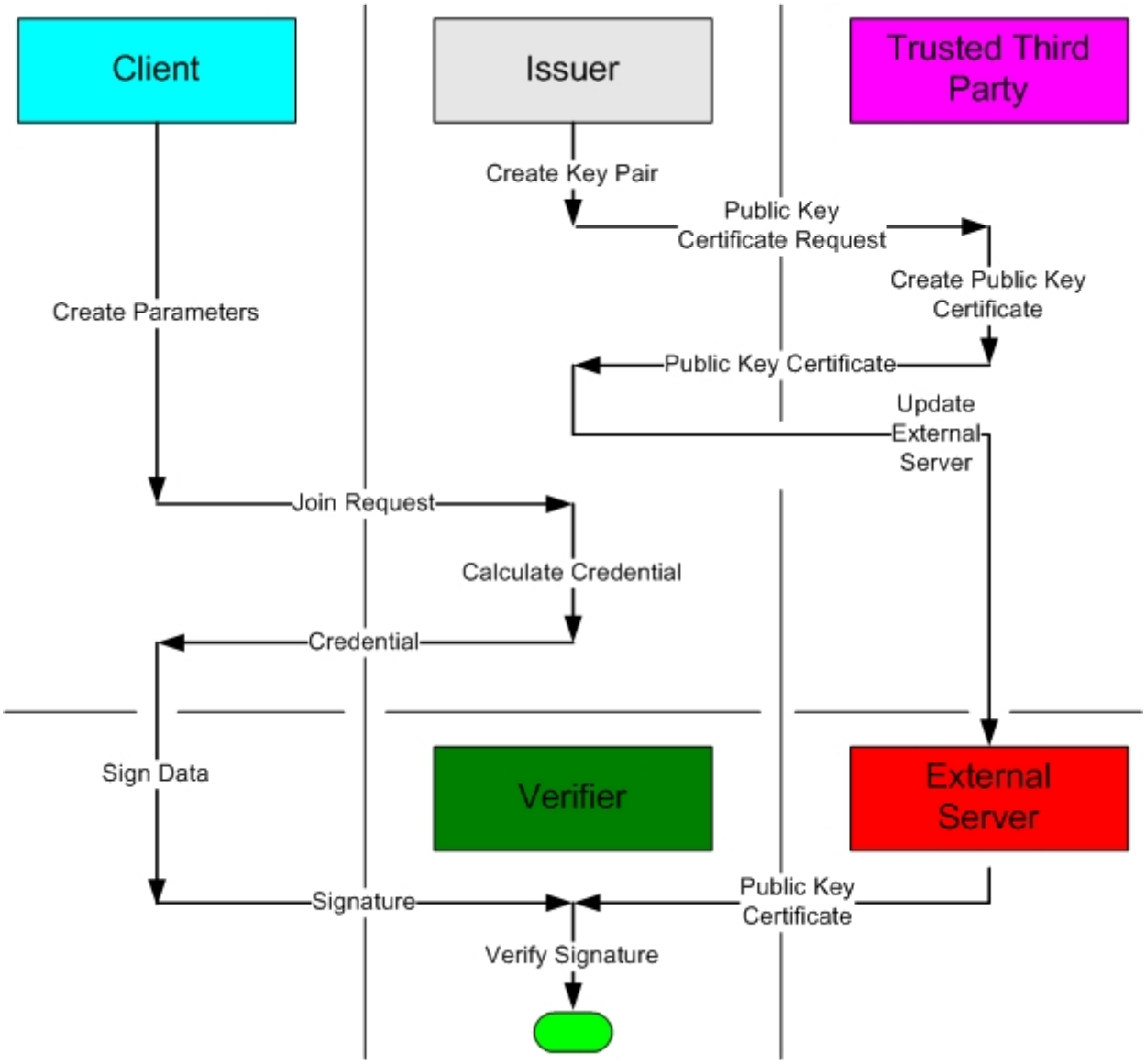

Figure 3.2: System Data Flow Diagram

revoke his own private key. It is easier to revoke the credential issued for the specific private key if any fraud is detected. If the entire platform is compromised it is also possible to revoke the client certificate so he cannot receive new credentials from an issuer. This revocation list for sure has to be updated to the external servers regularly. An additional possibility is to add attributes of a user to a signature. This means in order to verify the assets it is possible to claim a proof of the characteristics of this person while hiding his real identity. To check on properties like age, height, eye color and many more, it is feasible to prove to a verifier that the signer has these attributes. In order to use this kind of extension, the issuer has to provide more than just one generator point on the elliptic curve. How to provide additional generator points is explained in Chapter 5. To cover the diversity of attributes different generators need to be passed onto the client for further processing. Usually the server side knows the attributes and is able to create the appropriate amount of generator points. Additional generators are arbitrary points on the elliptic curve which are chosen on the server side.

Setup

The issuer needs to create its domain parameters which are distributed in form of a certificate to the clients. Firstly, an elliptic curve with an underlying prime field is chosen by the issuer. A random private key is then multiplied with the generator point of the elliptic curve to generate the group's public key. These parameters function as the domain parameters of the issuer. If we use the extended solution of the proposed algorithm including attributes, the issuer has to provide more than just the base generator point of the elliptic curve. As we already know, additional generator points are arbitrary points on the elliptic curve which can be chosen randomly. The last step is to sign the created domain parameters, which are wrapped into a certificate, by a trusted third party. Now the issuer is able to distribute the group's public key, represented by the signed certificate, to clients and external servers. The exact protocol definition of the issuer's setup phase is discussed in Section 4.1.1.

Join

In the join phase the issuer processes an incoming join request from a client. The client sends join parameters which are going to be part of the credential computation. The issuer takes the private key,the group's public key, chooses some random values and computes a credential including the client parameters. This credential is sent back to the requester which allows him to sign data on behalf of the joined group. Only clients which are known by the server and the issuer are allowed to request credentials for signing purposes. Any other client is rejected and cannot join the group. The join process of the issuer is explained in Section 4.2.1 and Section 4.3.1.

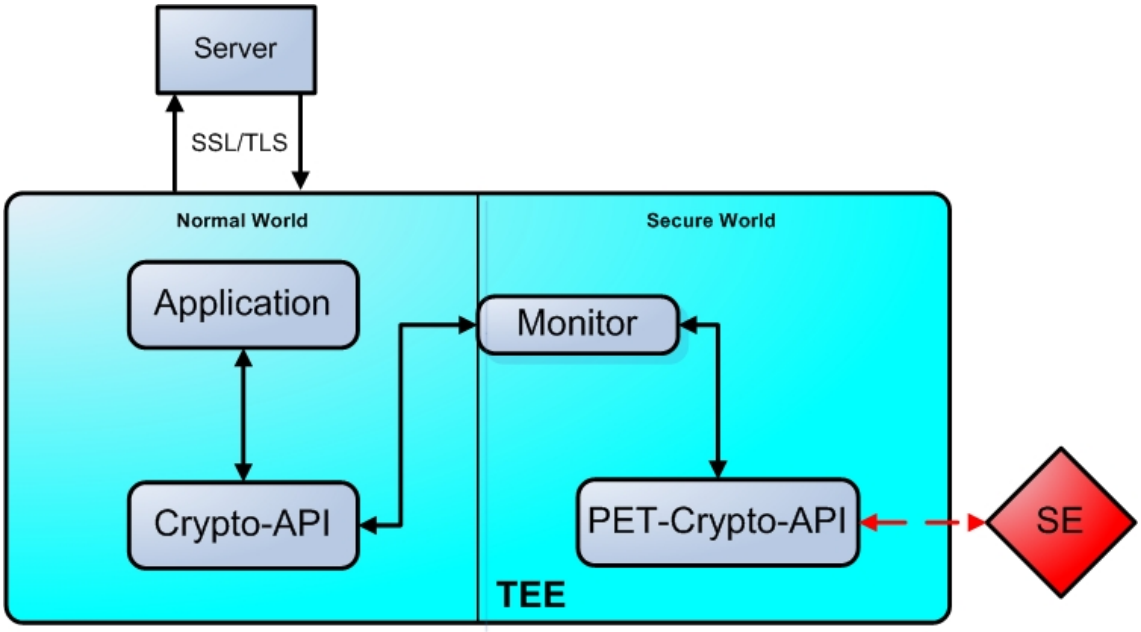

Figure 3.3: Client Architecture

3.3 Client

The client is divided into two parts, the normal world and the secure world. All operations which are not critical for security reasons may be computed in the normal world. However, sensitive data handling and cryptographical calculations must be operated in the secure area because private data must not be revealed to the outside. A monitor between those two separated worlds, as shown in Figure 3.3, is responsible for the right data treatment from the normal to the secure world and vise versa. A user only interacts with the normal world and does not have direct access to the secure environment. Only an authorization mechanism involves the user who allows the trap into the secure world by PIN code authorization for example. This means that data can be sent to the secure world if the user authorizes the program to do so. The data is then signed in the secure world and the signature parameters are given back to the caller. This not only protects the private data from being revealed to the outside. It also protects the system from fraud by intention of the user. If a rogue user gets direct access to the secure world he can share his credentials among his friends. Therefore, other persons are able to sign data on behalf of the group without ever joining it. This may be used to get multiple access with only one valid credential (e.g. e-Ticketing).
Secret key treatment, the join process with including the server/issuer, the important signing process and the verification of a signature have different severity in consideration of data security. The following section explains the coexistence of the normal and the secure world. The entire procedure of joining a group by receiving a credential from the issuer can not be executed separately. The join process depends on the cooperation of the issuer, the normal and the secure world. Only the signature and the verification process can be executed separately.

3.3.1 Normal & Secure World

The normal and the secure world are separated as already mentioned before. But to fulfill the requirements of total anonymity as mentioned in Section 2.3 it is necessary to interact between the two worlds. A monitor, as shown in Figure 3.3, handles the communication.
The normal world is responsible for data transmission between the server and the client. A standard application also runs in the normal world and has access to a Crypto-API which handles calls to the secure world over the monitor. The monitor resides in the secure world and requests permission for receiving data. The user's part is to authorize the trap into the secure world where data is signed for the caller application. This may be done by PIN code verification for example. The signature parameters are given back to the normal world for further processing. This can include data preparation like packing the signature parameters into an ASN.1 structure for sending purposes. After receiving the requested data, the opened channel is closed immediately. The normal world also handles the connection to the server for sending join requests to receive credentials. The connection is established via SSL/TLS as described in Section 2.1.7.
The secure world is a trusted execution environment (TEE) as shown in Figure 3.3. Data from the outside is received over the monitor which is the only way to communicate with the normal world. As already mentioned the secure world does net necessarily need a secure hardware element like the DAA scheme. This does not mean that it cannot work with such a secure element. Figure 3.3 shows that it is also possible to include a secure element into the TEE. This can be a Smart Card or a TPM for example. But the purpose of the KARIM-ACS is to exploit the fact that we can separate resources on the device directly. This gives us the opportunity of faster execution because we are using the same CPU, RAM and other shared resources. The secure world is responsible to hide sensitive data like private keys or credentials. All executions corresponding to sensitive data is executed in this environment to protect the user's identity. It also protects private data against a rogue user. The user does not have direct access to the secure world. He can only decide whether an application is allowed to enter the secure world or not. The important part is to understand that a trusted environment encapsulates secret keys, credentials and other sensitive data which must not be revealed to the outside.

Key Generation, Setup & Join

To execute the setup phase it is necessary to generate a private key. This can only be done in the secure world since the private key is considered as sensitive data and must not leave the secure area. After successful creation of a private key we can start the setup procedure. Firstly, we need to acquire the public group certificate from the issuer by sending a join request from the normal world. We receive the group's public key and other domain parameters like the generator point of the elliptic curve or the modulus of the underlying prime field. These parameters are going to be part of the

further execution. The normal world passes the parameters to the secure world. In the trusted area the join values are calculated including the private key and the issuer's public key. The computed values are shipped back to the normal world where they are wrapped into an ASN.1 structure. The next step is to send the join values back to the issuer for further processing. The issuer calculates the credential and sends it back to the client along with some additional parameters for a validation check. A validation check of the received credential is processed in the normal and the secure world. In the normal world a hash check of the received parameters is executed. After successful verification of the hash value the parameters are passed onto the secure world where a further check is executed including the private key. If all checks a verified correctly the client accepts the credential and stores it along with the private key and an alias name given by the normal world. The normal world needs a corresponding object to identify the secret key and the belonging credential in the secure world. This object could be a simple name for each produced private key. If a client joins more than one group it is necessary to distinguish the different private keys in the normal world which is done by using aliases. The exact protocols for the setup and join phase of the client are explained in Section 4.1.2, Section 4.2.1 and Section 4.3.1.
If the entire join procedure succeeds the client is able to sign data on behalf of the joined group. This is necessary to hide the user's identity.

Signature

The main purpose of the KARIM-ACS is signing data while hiding the signer's identity. The signing process is mainly executed in the secure world because sensitive data is included. For signing data the secure world receives a set of data (e.g. a simple message) from the normal world. As we know a client is able to join more than one group for different purposes. The normal world is responsible to set the right alias for the secure world. The alias is and indicator for taking the right private key and credential corresponding to the chosen group. The purpose of a signature is to ensure integrity and trustworthiness of the sent data to a verifier. The necessary signature parameters are computed and assembled in the secure world. Sensitive data is therefore never leaving the trusted environment. The signature procedure is the step with the highest computational effort since many parameters need to be calculated as shown in Section 4.2.2 and Section 4.3.2. If we are using attributes the signature process needs even more computational effort. For every shown attribute additional calculations are necessary. But the use of attributes makes it possible to satisfy verifiers which require certain information about the user even though the user is hiding his real identity. An easy example could be a cigarette vending machine which only needs to verify that the customer's age. Another use case could be a tram ticket which is valid for city A but not for city B. The customer needs to prove that he knows the attribute for city A to convince the verifier that he is in possession of a valid ticket. The same prove would not work in city B where he does not have the permission for using the tram because he is not able to show the right attribute.
A signature is usually sent to the verifier along with the set of data which was signed.

The verifier is then able to check the received signature and accepts the data only if the signature prove is valid.

Verification

The purpose of verification is to verify a signature to accept data received from a signer. The signature is sent along with the data and needs to be verified correctly. Otherwise the data is rejected or in other words a customer does not get access to a specific event for example. The whole verification process can be executed with the group certificate and the signature parameters from the client. The verifier checks if the hash value received within the signature parameters equals his computed hash value. Furthermore, the verifier checks that the signature parameters lie in a certain interval. This range check is used as an additional security feature. The target of an attacker could be to convince the verifier that his self generated set of data comes from a trusted entity. Therefore, he tries to alter the signature parameters in a way that the hash check succeeds. With the range check it is possible to detect such a fraud because it can happen that the additional calculations put the signature parameters out of the specified range. This is explained in more detail in Section 4.2.3 and Section 4.3.3. The verification procedure including attributes has the same assets as the base solution without attributes. The only difference is that the verifier requires additional information from the signer to verify a signature. The signature includes the attributes which are shown by the signer.
If the verification succeeds the received data is accepted by the verifier. A customer, for example, gets access to his booked event or is allowed to buy cigarettes since he is old enough.

3.4 Verifier & External Server

As shown in Figure 3.1, a verifier only needs the signature parameters from the client and the public group certificate from an external server. It is not necessary that the main server, representing the different issuers, stays continuously online. Each certificate from an issuer is distributed to the external server where they are claimable for the verifiers. The difference to using a privacy CA, as mentioned in Section 3.1, is that data is signed on behalf of the group and therefore the signature must no be linkable to another signature from the same signer. If we would use a privacy CA not hiding ourselves behind a group we have to generate a new key for each signature to avoid linkability. Certificate creation may also be executed off-line and manually stored onto a client device, as mentioned in Section 3.1. Just the public group certificates need to be put to an online entity. A signature is issued as a certificate as well and tells the verifier where to find the right public key. The verifier is then able to download the certificate to execute a successful verification process. The protocol is discussed in Section 4.2.3.

The external server receives regular updates from the server entity and provides necessary data to third party instances. It stores all the issuer certificates for distribution purposes and is in hold of the revocation list. It is not possible for another server to revoke anything because this is left to the main server, but for security reasons a verifier needs to be able to check on the trustworthiness of a client. The revocation list should be updated regularly to all distribution servers to fulfill the requirements of security.

3.5 Revocation

Revocation is important to make a system more secure against attacks. If an attacker compromises a platform it is possible to put this platform onto a revocation list. A common way used by DAA is to revoke the client's private key. The platform is not able to sign data on behalf of the joined group anymore because the signatures will not verify correctly. We adopt this approach and revoke the tuple private key and corresponding credential if a private key is publicly known. A revocation list holds the knowledge of all revoked tuples. Therefore, a verifier is able to check if the credential of a signer is already on the revocation list. The algorithm for the revocation mechanism is shown in Section 4.4.

Chapter 4

Protocol Definition

To get the entire system running we need protocols to control the different processes. In the last chapter, the theory of the implemented prototype was explained. In this chapter we are going into detail on calculation algorithms, system and domain parameter creation and the important *Setup, Join, Sign* and *Verify* protocols. Firstly, it is important to understand why we use system-wide parameters in our computations. Secondly, the *Setup* phase of both entities, server and client, which are are explained. Thirdly, the significance of the *Join* protocol between an issuer and a client is emphasized. The next two sections are about signing data and verifying a signature received by a verifier, which is the actual purpose of th proposed KARIM-ACS. Furthermore it is mandatory to differentiate between the base solution and the extended version including attributes. There is a slight difference in the setup phase, and the join process already involves the characteristics in the credential computation. We also need to distinguish the signature and verification procedures, which are discussed in the last sections of this chapter. Moreover, in Section 4.4 we talk about the revocation algorithm to revoke issued credentials.
For the proposed scheme we can use different elliptic curves over prime fields with domain parameters from P-192 to P-521 as shown in Table 4.1. A domain parameter P-xxx defines a set of parameters for the chosen elliptic curve. The NIST standard [30] provides a set of domain parameters for each curve over a prime field with modulus p and the order n. Figure 4.1 shows the data flow of the computed protocol parameters between server, client, verifier and external server.

Table 4.1: Domain Parameters Definition [22]

Key Size Parameter	Used Domain Parameter
keysize $\leq$ 192	P-192
192 $<$ keysize $\leq$ 224	P-224
224 $<$ keysize $\leq$ 256	P-256
256 $<$ keysize $\leq$ 384	P-384
384 $<$ keysize	P-521

4.1 Setup Protocol

As described briefly before there are two distinct protocols. One is executed on the issuer and the other on the client side. The issuer and client setup protocols are important to generate private and public parameters which need to be computed for further processing. The security parameters α, l_f, l_h, l_n and c_X needed for the protocols are defined in Section 2.2.5.

4.1.1 Issuer Setup Protocol

The issuer is responsible for creating the domain parameters for a chosen elliptic curve. The first step is to generate a public group certificate. The certificate is distributed to external servers and clients which request to join the group. The public group certificate is also needed by the verifier to verify signatures produced on behalf of that specific group. Every issuer certificate needs to be signed by a trusted third party so that it can be verified by anyone who receives this document. There is a slight difference in the setup protocol if we want to work with attributes. For each attribute the issuer has to provide an additional generator point on the elliptic curve. As we already know from Section 3.2 this can be an arbitrary point on the curve. The issuer knows the number of attributes of his trusted clients. The following description shows and explains how to set up the domain parameters (n, p, Ψ_I and Ψ_g):

1. Creation of a random value $\iota \in \{0,1\}^{l_n} \cap \mathbb{F}_p$ which is the issuer's private key.
2. Choose the elliptic curve to get the necessary parameters like the base point Ψ_g which is the group generator.
3. Attributes: Publish a set of generators G_i on the elliptic curve where each generator belongs to an attribute.
4. Calculate the issuers public key $\Psi_I = \iota * \Psi_g$.
5. The domain parameters are: n, p, Ψ_g and a set of G_i if attributes are used where n is the order of the generator point on the elliptic curve and p the modulus of the underlying prime field $\mathbb{F}_p$.
6. Publish the domain parameters and the issuer's public key Ψ_I as the group certificate.

4.1.2 Client Setup Protocol

To execute the client setup protocol we need the public certificate of the group we want to join. Therefore it is necessary to establish a connection to the server via SSL/TLS as discussed in Section 2.1.7. Important to know at this point is that both entities need to authenticate themselves to each other. Hence, the certificate sent by the client has to be extended with additional information like the group he wants to join and the

elliptic curve scheme (e.g. P-192). If the authentication was successful, the server sends back the group certificate to the client, where further processing is started to create the two setup values $\tilde{Y}$ and $\tilde{\omega}$. To initiate the join process described in Section 4.2.1, those parameters will be sent to the issuer. The steps of the setup phase are shown as follows:

1. Choose a random value $f \in [c_X - 2^{l_f}, c_X + 2^{l_f}] \cap \mathbb{F}_p$ which is the client's private key.
2. Choose a random value $\hat{\omega} \in \{0,1\}^{l_n} \cap \mathbb{F}_p$ for further calculations.
3. Compute the first parameter $\tilde{Y} = \hat{\omega} * \Psi_I$.
4. Compute the second parameter $\tilde{\omega} = f \cdot \hat{\omega} \bmod n$ which is a so-called blinding value for f.

The generated values $\tilde{Y}$ and $\tilde{\omega}$ are used by the issuer to produce a credential corresponding to the private key f. It is very important that the private key f is never revealed to the outside because this is sensitive data and supposed to be a secret value. It is also a necessity that the computational steps of this setup phase are handled in the trusted environment.

4.2 Base Solution

The base solution of the proposed scheme works without using attributes. Therefore fewer parameters are necessary which makes the execution faster compared to the extended solution explained in Section 4.3. After successful execution of the setup protocols the join phase is invoked. The join protocol is used to add a client to a group. Within this group the client is able to sign data on behalf of that group. The signature protocol is necessary to produce a valid signature using the issued credential $[U, P]$ for a chosen private key f. With the public group certificate and the signature parameters from a client a verifier is able to execute the verification protocol.

4.2.1 Join Protocol

As we know from the setup protocol the client has already received the public group certificate from the issuer. With this certificate the join parameters $\tilde{Y}$ and $\tilde{\omega}$ were created by the client. Those parameters are sent to the issuer where the credential is produced to add the client to the group. The issuer has to compute the parameter U which belongs to the private key f on the client side and P. Together they build the credential [U,P] issued to the client. The client receives the credential and verifies it. The verification of the credential is necessary to ensure that this credential comes from a trusted issuer. The exact protocol flow is shown as follows:

1. Choose random values $r_I \in \{0,1\}^{l_n} \cap \mathbb{F}_p$ and $\hat{r} \in \{0,1\}^{l_n} \cap \mathbb{F}_p$.

2. Compute the credential $U = (\tilde{\omega}^{-1} \cdot (\iota^{-1} + \hat{r})) * \tilde{Y} + \Psi_I$ and $P = \hat{r} * \Psi_I$ which correspond to the client's private key f because f is included in $\tilde{\omega}$. $\tilde{\omega}^{-1}$ and ι^{-1} are the multiplicative inverses in the field $\mathbb{F}_p$.
3. Calculate the proof $h = H(\Psi_I \| \Psi_g \| r_I * \Psi_g \| U)$ with a common hash function like SHA-1.
4. Furthermore the issuer computes the proof value $s = r_I + h \cdot \iota \bmod n$.
5. Send the credential values $[U, P]$ and the proof parameters (h, s) back to the client.
6. To verify the hash value h the client computes $h' = H(\Psi_I \| \Psi_g \| s * \Psi_g - h * \Psi_I \| U)$ using the same hash function as the issuer. This can be done in the normal world due to the fact that no private parameters are involved.
7. If and only if $h' = h$ and $f * U \equiv \Psi_g + P + f * \Psi_I$, which has to be checked in the secure world because the private key f is involved, the client accepts the credential $[U, P]$. Keep in mind that f never leaves the secure world when verifying these proofs. The two hash values h and h' are only equal if the calculation of $s * \Psi_g - h * \Psi_I$ equals $r_I * \Psi_g$ because the other hash parameters Ψ_I, Ψ_g and U stay the same.

As a last step of the join process the client stores the credential. Since a client may be part of more than one group he has to store the credential $[U, P]$ along with the private key f and a unique alias name chosen by the normal world as a data quadruple (f,U,P,Alias). Now a client is able to sign data on behalf of the joined group while not revealing the user's identity.
Note: Storing this quadruple is only possible in the secure world since the private key f resides in this area and must not leave it.

4.2.2 Signature Protocol

The signature protocol is the protocol with the highest computational effort. Many parameters are calculated within the signing procedure. As described in Section 3.3.1, we are operating in two different worlds where this process only takes place in the secure world. Only the step of signature assembling may be executed in the normal world since there is no sensitive data involved any more. The steps of producing a valid signature are shown as follows:

1. Choose random values $r \in [c_X - 2^{l_f}, c_X + 2^{l_f}] \cap \mathbb{F}_p$ and $r', r'' \in \pm\{0,1\}^{\alpha \cdot (l_f + l_h)} \cap \mathbb{F}_p$.
2. Compute the following parameters:

$$\Upsilon = r * U$$
$$\Phi = r * \Psi_g$$
$$\Gamma = r' * \Upsilon$$
$$\Delta = r'' * \Psi_g$$

$$\bar{r} = r \cdot f \bmod n$$
$$P' = r * P$$

3. Furthermore it is necessary to calculate the proof

$$h = H(\Psi_I \| \Psi_g \| \Upsilon \| \Phi \| \Gamma \| \Delta \| P' \| m)$$

with a common hash function like SHA-1 where m is the data to sign.

4. Also compute the proof values

$$\nu' = r' + h \cdot (f - c_X)$$
$$\nu'' = r'' + h \cdot (r - c_X)$$

which must lie in the field $\mathbb{F}_p$ due to the fact that r', $r'' \in \pm\{0,1\}^{\alpha \cdot (l_f + l_h)}$ and $\alpha \cdot (l_f + l_h) < l_n$ because $\alpha \cdot (l_f + l_h) + 1 < l_n$ as defined in Section 2.2.5.

5. For revocation purposes it is necessary to compute a pseudonym $PSN_S = HMAC(f, U \| r_S \| r_V)$ where r_S is a random number generated for each signature by the signer and r_V is received from the verifier.

6. To finalize the signing process assemble the signature parameters

$$\sigma = (h, \nu', \nu'', \Upsilon, \Phi, P', \bar{r}, PSN_S, r_S)$$

The signature values are packed into a certificate which is sent to a verifier.

4.2.3 Verification Protocol

Any third party is able to verify data integrity only by receiving the issuer's public parameters and the signature certificate. Before receiving a signature the verifier sends a randomly created value r_V to the signer. This is necessary for revocation purposes since the verifier does not accept signatures from revoked platforms. In Section 4.4 we talk about the revocation mechanism and how it is executed in the KARIM-ACS. Furthermore, the verifier extracts the necessary parameters from the certificates and computes the proof h' which is a hash value. Hence, the verifier executes a range check on the parameters $\nu' and \nu''$ as shown in the following verification steps:

1. First the verifier executes a range check on the parameters ν' and ν''. An honest signer knows the private key $f \in [c_X - 2^{l_f}, c_X + 2^{l_f}] \cap \mathbb{F}_p$ and the range of the randomly chosen parameters r, r' and r''. Therefore he is able to produce the values ν' and $\nu'' \in [-2^{\alpha \cdot (l_f + l_h) + 1}, 2^{\alpha \cdot (l_f + l_h) + 1}]$ which must lie in the given range.

2. Furthermore he computes the hash value

$$h' = H(\Psi_I \| \Psi_g \| \Upsilon \| \Phi \| (\nu' + h \cdot c_X) * \Upsilon - h * \Phi - (h \cdot \bar{r}) * \Psi_I - h * P' \| (\nu'' + h \cdot c_X) * \Psi_g - h * \Phi \| P' \| m)$$

using the same hash function as the signer.

3. The verifier accepts the message m if the range checks succeed and $h' = h$ which means that the hash values need to be equal.

Note: The verifier may receive the issuer parameters from an external server where different certificates are stored. The URL to this server is included in the signature certificate sent by the signer.

4.3 Extended Solution with Attributes

The extended version of this prototype is the use of attributes within a signature. Attributes can be any kind of additional data belonging to a person or maybe a mobile device. The goal of using such characteristics is to satisfy secondary requirements of a verifier. Such data can be the age, the height or the eye color of a person used for e-passport purposes. For using attributes it is necessary to include them into the join, signature and verification process. It demands some altering of the latter protocol definitions to fulfill those needs.
Note: Using attributes always reveals some information about the signer, even though he is anonymous.

4.3.1 Join Protocol

To produce a valid credential it is necessary that the issuer knows all the attributes or at least a product of all attributes. In the case of the KARIM-ACS the issuer knows all attributes of the client and creates a generator point for each attribute. Hence, the issuer computes $\bar{x} = H(x_1 * G_1 \| x_2 * G_2 \| ... \| x_n * G_n)$ which is a concatenation of all attributes multiplied with the corresponding generator points. Compared to the join protocol of the base solution the credential calculation is slightly altered as shown below.

$$U = (\tilde{\omega}^{-1} \cdot (\iota^{-1} + \bar{x} \cdot \hat{r})) * \tilde{Y} + \Psi_I \text{ and } P = \hat{r} * \Psi_I$$

Furthermore, the check of the credential inside the secure world on the client is adjusted for the extended solution as shown below.

$$f * U \equiv \Psi_g + \bar{x} * P + f * \Psi_I$$

The rest of the join protocol stays the same as described in Section 4.2.1.

4.3.2 Signature Protocol

The use of attributes is necessary if a verifier requests certain information about the signer to accept a signature. This can be the age of the signer for example. Such an attribute needs to be included into the signature process to satisfy the requirements of the verifier. All attributes which are not shown to the verifier remain hidden. In

the join phase the client also receives the parameter $\bar{x}$ which is the hash value over all attributes as shown in Section 4.3.1. The steps of the signature protocol are as follows:

1. Choose random values $r \in [c_X - 2^{l_f}, c_X + 2^{l_f}] \cap \mathbb{F}_p$ and $r', r'' \in \pm\{0,1\}^{\alpha \cdot (l_f + l_h)} \cap \mathbb{F}_p$.
2. Concatenate all shown attributes x_i multiplied with the corresponding generator point G_i and compute the hash value $\tilde{x}_{show} = H(x_{i_1} * G_{i_1} \| x_{i_2} * G_{i_2} \| ... \| x_{i_n} * G_{i_n})$.
3. Compute the following parameters

$$\begin{aligned} \Upsilon &= r * U \\ \Phi &= r * \Psi_g \\ \Gamma &= r' * \Upsilon \\ \Delta &= r'' * \Psi_g \\ \bar{r} &= r \cdot f \bmod n \\ P' &= (r \cdot \bar{x} \cdot \tilde{x}_{show}^{-1}) * P \end{aligned}$$

   where $\bar{x}$ is the set of all attributes.
4. Furthermore it is necessary to calculate the proof

$$h = H(\Psi_I \| \Psi_g \| \Upsilon \| \Phi \| \Gamma \| \Delta \| P' \| m)$$

   with a common hash function like SHA-1 where m is the data to be signed.
5. Also compute the proof values

$$\begin{aligned} \nu' &= r' + h \cdot (f - c_X) \\ \nu'' &= r'' + h \cdot (r - c_X) \end{aligned}$$

   which must lie in the field $\mathbb{F}_p$ due to the fact that r', $r'' \in \pm\{0,1\}^{\alpha \cdot (l_f + l_h)}$ and $\alpha \cdot (l_f + l_h) < l_n$ because $\alpha \cdot (l_f + l_h) + 1 < l_n$ as defined in Section 2.2.5.
6. For revocation purposes it is necessary to compute a pseudonym $PSN_S = HMAC(f, U \| r_S \| r_V)$ where r_S is a random number generated by the signer and r_V is received from the verifier.
7. To finalize the signing process assemble the signature parameters

$$\sigma = (h, \nu', \nu'', \Upsilon, \Phi, P', \bar{r}, SET(x_{i \in show}), PSN_S, r_S)$$

4.3.3 Verification Protocol

The verification process of the extended solution just needs an extra calculation of the hash value of all shown attributes, described as follows:

1. First the verifier executes a range check on the parameters ν' and ν''. An honest signer knows the private key $f \in [c_X - 2^{l_f}, c_X + 2^{l_f}] \cap \mathbb{F}_p$ and the range of the randomly chosen parameters r, r' and r''. Therefore he is able to produce the values ν' and $\nu'' \in [-2^{\alpha \cdot (l_f + l_h) + 1}, 2^{\alpha \cdot (l_f + l_h) + 1}]$ which must lie in the given range.

2. Compute $\tilde{x}_{show} = H(x_{i_1} * G_{i_1} \| x_{i_2} * G_{i_2} \| ... \| x_{i_n} * G_{i_n})$ where i are the indices of shown attributes and the corresponding generator points.

3. Compute

$$c' = H(\Psi_I \| \Psi_g \| \Upsilon \| \Phi \| (\nu' + h \cdot c_X) * \Upsilon - h * \Phi - (h \cdot \bar{r}) * \Psi_I - (h \cdot \tilde{x}_{show}) * P' \| (\nu'' + h \cdot c_X) * \Psi_g - h * \Phi \| P' \| m)$$

   using the same hash function as the signing process.

4. The verifier accepts the message m if the range checks succeed and $h' = h$ which means that the hash values need to be equal.

4.4 Revocation Protocol

As already described previously in Section 3.5 the revocation mechanism does not revoke the private key directly. Instead we try to gather information about compromised platforms and then revoke the corresponding credential if the private key gets publicly known. The revocation mechanism consists of a few simple calculation steps on the signer and the verifier side. As it was already mentioned in the signing process the signer receives a random value r_V from the verifier. The signer also generates a random value r_S each time he signs data. The pair (f, U) and the two generated numbers are calculated to the pseudonym $PSN_S = HMAC(f, U \| r_S \| r_V)$. PSN_S and r_S are sent to the verifier within the assembled signature.
To check if the platform or the credential has been revoked the verifier checks through the revocation list where he has access to. The revocation list contains all values of the pair (f, U) of all compromised platforms. Together with the received random number r_S from the signer and his own generated value r_V he is able to compute $PSN_V = HMAC(f, U \| r_S \| r_V)$. If and only if the received PSN_S equals the calculated PSN_V ($PSN_S = PSN_V$) the signature is refused by the verifier.

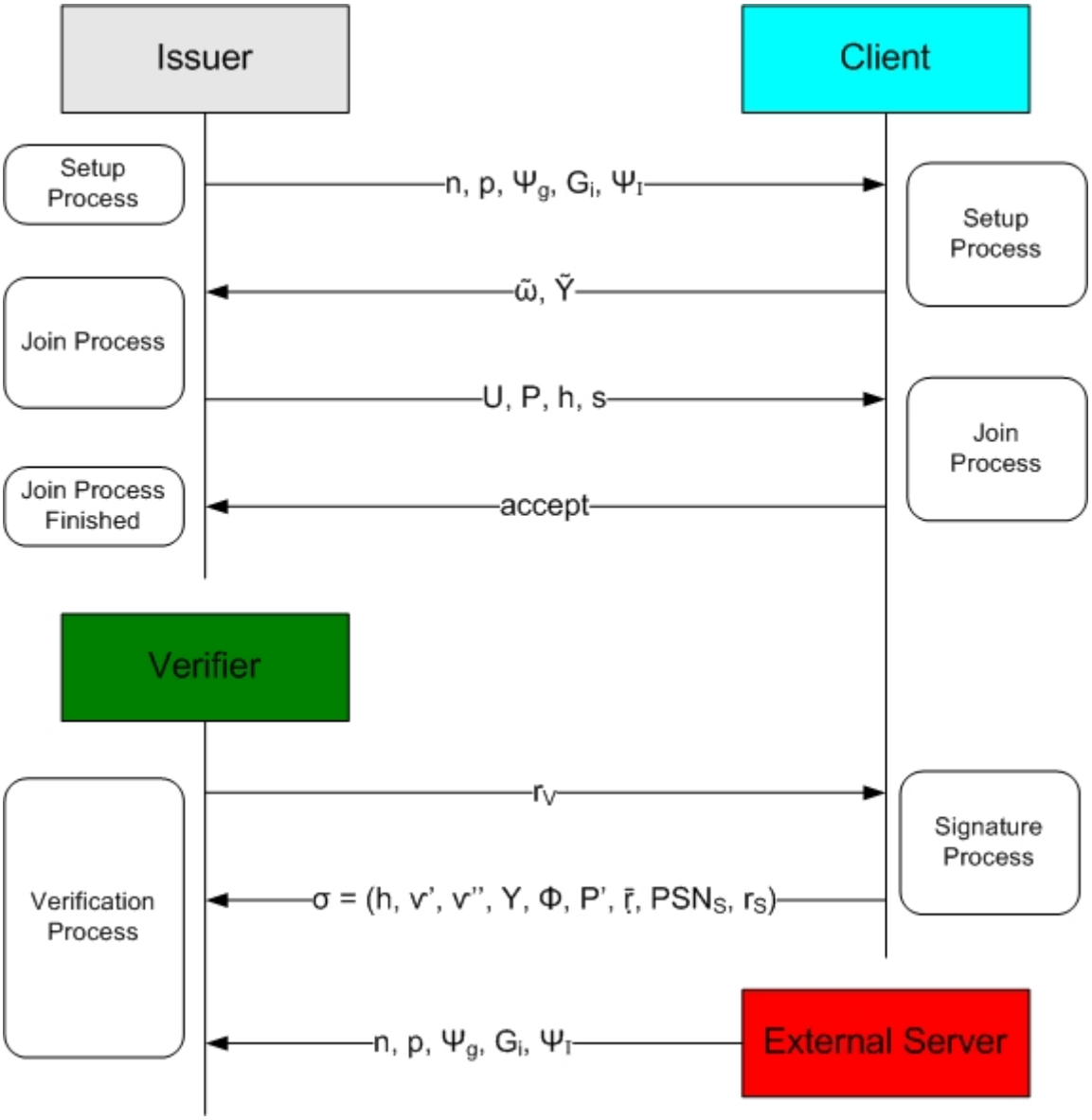

Figure 4.1: Data Flow of Protocol Parameters

Chapter 5

Implementation

After defining the different protocols for the proposed scheme we can now talk about the implementation. It was important to implement the protocols explained in Chapter 4 to gain results about the performance of the KARIM-ACS. The protocols for the server side and the normal world of the client are implemented in Java since they are running on a standard PC for test purposes. The protocols running inside the secure world are implemented in C. The focus of this implementation lies on the performance evaluation of the secure world. Therefore we used an external low powered device to gather meaningful results about processing times and memory usage. The device we used was the MCB2130 Evaluation Board explained in Section 5.5.1. The board was connected via serial port to the normal world to exchange data. For test purposes it was sufficient to use a connection via serial port which can be seen as the monitor of the client side as shown in Figure 3.3. The normal world and the server used a SSL/TLS connection for data transmission to establish a secure communication channel.
Furthermore it was important to us to implement the secure world in C. This gives us the opportunity to port the implementation easily to other devices with different platform architectures if further investigations are considered.
For this implementation we had to chose a domain parameter as shown in Table 4.1. We decided to take the P-192 domain parameter as this is used as a standard by the Austrian government for the citizen card [6]. The domain parameters n, p, Ψ_g are defined in P-192 and can be found in [30] on page 88.
As already mentioned before our focus lies on the secure world of the proposed scheme but for completeness we are going to introduce the implementation of the entire system.

5.1 Attributes

So far we were talking about attributes or characteristics of a person which may be shown to a verifier. We already defined these attributes as age, height, eye color or other information belonging to an individual. To involve these characteristics in our calculation algorithms, it is important to transform them into useful values. The ap-

proach we used was hashing the attributes from a string value which was then converted to a big integer for further computations. We used four attributes for the performance evaluation where we always showed two and the other two remained hidden.

5.2 Server/Issuer Implementation

As already explained in Section 3.2 the server provides a communication interface to the clients and holds one or more issuers. To establish a secure connection via SSL/TLS a server certificate is needed which is used to authenticate the server to the client. The server uses the Java Keytool explained in Section 2.1.6 to handle the incoming requests from a client. The server receives the client certificate and sends its own certificate back. If the client is not known by the server the request is rejected. If the certificate and the client request are valid a secure communication channel can be established. The client is now able to request credentials from a specified issuer using a secure connection during the join phase.

5.2.1 Setup Protocol

Before processing the join protocol every issuer needs to execute the setup protocol defined in Section 4.1.1. Every issuer needs to generate its own private key $\iota \in 0, 1^{l_n}$ which is used to compute the issuers public key $\Psi_I = \iota * \Psi_g$ where Ψ_g is the generator point of the elliptic curve defined in the domain parameters. The domain parameters and the public key are put into the issuer certificate which represents the public group certificate. It has to be signed by a trusted third party before distributing it to clients and external servers.
If we use attributes for the KARIM-ACS the issuer has to provide additional generator points on the elliptic curve. A generator point is chosen randomly on the elliptic curve and associated to one attribute. The issuer has knowledge about all attributes of its clients to produce valid credentials.

5.2.2 Join Protocol

The execution of the join protocol on the issuer side includes the creation of a credential $[U, P]$ for a corresponding private key f on the client side. The issuer receives the parameters $\tilde{\omega}$ and $\tilde{Y}$ which are included in the credential computation. Furthermore, the proofs (h, s) are calculated and also sent to the client along with the credential $[U, P]$. If the extended solution is used the issuer additionally needs to compute the hash value $\bar{x} = H(x_1 * G_1 \| x_2 * G_2 \| ... \| x_n * G_n)$ over all attributes which is also sent to the client along with the attributes. The attributes of a client are known by the issuer. Therefore, the issuer is able to compute $\bar{x}$ which is part of the credential $[U, P]$ using the extended solution explained in Section 4.3. An attribute may be the age of

a person or any other specified characteristics which are required by a verifier.

Short Example using Attributes
A customer buys a train ticket for the connections between the cities A and B. The client receives a credential including the attributes A_attt and B_attt. If the customer wants to take the train from city A to B he only shows the attributes A_attt and B_attt. The verifier is convinced that the customer is allowed to take this train. If the customer wants to take the train from city A to C he is only able to show the attribute A_attt. The verifier is not able to verify the customer's signature as he is not in possession of attribute C_attt.

5.3 Client Implementation

As we already mentioned at the beginning of this chapter the client is divided in two worlds. The normal world is implemented in Java and runs on a powerful machine whereas the secure world runs on the MCB2130 Evaluation Board. The two worlds are connected via a serial port to USB adapter since the evaluation board only provides a serial port interface for data transfer.

5.3.1 Setup Protocol

During the setup phase a private key f and the join parameters $\tilde{\omega} = f \cdot \hat{\omega} \bmod n$ and $\tilde{Y} = \hat{\omega} * \Psi_I$ are created as defined in Section 4.1.2. The setup protocol is executed in the secure world since sensitive data like the private key are included in the computations. The groups public key Ψ_I is received from the normal world. f and $\hat{\omega}$ are chosen randomly with a strong random number generator provided by the MIRACL library. The MIRACL library described in Section 5.4.2 was used for cryptographic calculations because it provides the necessary ECC and big number functions and also speeds up big number arithmetic. The normal world is usually responsible for packaging the produced parameters into an ASN.1 structure and sending it to the server.

5.3.2 Join Protocol

The join protocol with and without attributes explained in Section 4.3.1 and Section 4.2.1 is executed in both worlds and also on the issuer side. The parameters created in the setup phase $(\tilde{\omega}, \tilde{Y})$ are sent to the issuer which produces the credential $[U, P]$ and the proof values (h, s) and sends them back to the client. The entire credential parameters are packed into a certificate which is then received and extracted in the normal world. To validate the received credential the normal world executes a hash calculation and compares the result to the received hash value from the issuer. If the hash check succeeds the client invokes a trap into the secure world. A PIN code

authorization by the user is necessary to allow the trap into the secure world. This security feature prevents random applications for trapping into the secure world to avoid attacks from fraud applications. The credential $[U, P]$ is shipped to the secure world to execute the additional proof $f * U \equiv \Psi_g + P + f * \Psi_I$. If all checks succeed the client accepts the credential and stores it along with the private key and the alias name given by the normal world. This alias is used to identify the credential and the corresponding private key in the secure world since the normal world does not have access to sensitive data.

5.3.3 Signature Protocol

The credential stored received from the issuer gives us the ability to sign data on behalf of the group we joined in the previous steps. The signature protocol executed with and without attributes as explained in Section 4.3.2 and Section 4.2.2 is mostly processed in the secure world. To invoke a signature execution the normal world implements the Java security package which provides a signature interface for that purpose. We used the JCE/ECCelerate library explained in Section 5.4.1 which additionally provides the basic cryptographic functions for ECC. With the signature interface it is possible to choose different hash and signature algorithms by calling the desired service provider. We implemented such a new cryptographic service provider which provides the KARIM-ACS. Listing 5.1 shows how to work with such a new self-written provider and how to invoke a signature process. First we need to add our security provider to the list of supported providers. Now it is possible to get an instance to initiate and start our signature procedure. The *initSign* method needs the private key alias for initialization. The *update* routine is collecting data which is going to be signed. After initialization the signature execution is started and data is sent to the secure world. A PIN code authorization via the user is required for signing data in the secure world. The signature parameters are sent back to the normal world after execution. The normal world is usually responsible to pack the signature parameters into a signature certificate for further usage (e.g. sending it to a verifier).
Note: How to implement your own service provider for the Java Cryptography Architecture is shown in http://docs.oracle.com/javase/6/docs/technotes/guides/security/crypto/HowToImplAProvider.html .

5.3.4 Verification Protocol

The verification protocol with and without attributes as explained in Section 4.3.3 and Section 4.2.3 may be executed on any third party or on the client itself. The cryptographic service provider introduced in Section 5.3.3 also provides a mechanism to invoke a verification procedure as shown in Listing 5.1. The *initVerify* method takes the issuer's public key as parameter and the *update* routine collects the incoming data from a signer. To start the verification process the signature parameters are needed.

The normal world then executes the hash check and invokes the trap into the secure world for the parameter check including the private key. On success the verifier accepts the data sent by the signer.

```
/* Add the ACS provider to the list of supported providers */
iaik.security.ecc.acs.provider.ACS_Provider.addAsProvider();

//-------------------------------------------------------------------
//------------------------- SIGNATURE -------------------------------
//-------------------------------------------------------------------
KARIM_Signature signature =
                    (KARIM_Signature) Signature
                                .getInstance("KARIM", "IAIK_ACS");
signature.initSign(privateKey);
signature.update(test_bytes);
final byte[] sign_params = signature.sign();

//-------------------------------------------------------------------
//------------------------ VERIFICATION -----------------------------
//-------------------------------------------------------------------
KARIM_Signature verifier =
                    (KARIM_Signature) Signature
                                .getInstance("KARIM", "IAIK_ACS");
verifier.initVerify(issuerPublicKey);
verifier.update(test_bytes);
if(verifier.verify(sign_params))
  System.out.println("VERIFICATION_SUCCESSFUL");
else
  System.out.println("VERIFICATION_FAILED");
```

Listing 5.1: Client Signature/Verification - KARIM_Siganture.java

5.4 Libraries

To implement the entire system it is important to choose the appropriate environment and the according libraries. For the purpose of security we have chosen the JCE/ECCelerate library from the Institute of Applied Information Processing and Communications (IAIK) to provide the necessary cryptographic functionality for our implementations in Java. The MIRACL library supports a lot of cryptographical standards and also elliptic curve arithmetic, which was mandatory for this project. It is used to secure the TrustZone on our client side.

5.4.1 IAIK JCE/ECCelerate

The Java Cryptographic Extension[1] (JCE) implemented by the Institute of Applied Information Processing and Communications[2] (IAIK) provides the necessary basic cryptographic functions. It supports PKCS and X.509 standards, has a built-in ASN.1 library and implements LDAP search and random number generators. These services are easily included into any Java application via its own security provider. The current version, JCE 4.0, has been released on the 4th of November 2010. A documentation can be found online at http://javadoc.iaik.tugraz.at/iaik_jce/current/index.html.
For cryptographic elliptic curve operations the IAIK library ECCelerate[3] is used. This library is compliant with the standards ANSI X9.62-2005, ANSI X9.63, IEEE P1363a, FIPS 186-3, SEC1 v2.0, SEC2 v2.0 and RFC 5639. It supports fast finite field arithmetic for both prime and binary fields, ECDSA with SHA-1/SHA-256 according to ANSI X9.62-2005 and elliptic curve arithmetic with affine and several types of projective coordinates. Furthermore there are some optional speed-ups for special prime and binary fields. The current version, ECCelerate 1.0, has been released on the 1st of June 2011.[22]
Both libraries are entirely written in Java language[4].

5.4.2 MIRACL Library

Shamus Software Ltd.[5] provides a *M*ultiprecision *I*nteger and *R*ational *A*rithmetic *C*/C++ *L*ibrary (MIRACL) for cryptographic systems. It fully supports big number arithmetic, RSA public key cryptography, Diffie-Hellman key exchange, DSA digital signature and, since version 4.0, ECC over GF(p) and GF(2^m), AES, modes of operation and the hashing standards SHA-1/256/384/512. The MIRACL library is portable to almost any processor architecture to generate optimal configurations. Speeding up calculations may be of interest on small powerless platforms like we have used in this approach. How to set up and compile the perfect configuration for your environment is explained in the MIRACL user's manual [34].
For our small device, the MCB2130 evaluation board, we do not need the entire library because it would be too big. We have only chosen the necessary files for our special purposes shown in Listing 5.2. Furthermore, we have set the important header file *mirdef.h* as shown in Listing 5.3. *Note: This is just one option of the mirdef.h file. For testing purposes the uncommented lines where also included in different ways, as described in Section 6.2.*

```
mralloc.c
mrmath0.c
mrmath1.c
```

[1] http://jce.iaik.tugraz.at/sic/Products/Core-Crypto-Toolkits/JCA-JCE
[2] http://www.tugraz.at
[3] http://jce.iaik.tugraz.at/sic/Products/Core-Crypto-Toolkits/ECCelerate
[4] http://www.java.com
[5] http://www.shamus.ie/

```
mrmath2.c
mrmath3.c
mrbits.c
mrcore.c
mrcurve.c
mrgf2.c
mrjack.c
mrlucas.c
mrmonty.c
mrmuldv.c
mrshs.c
mrsroot.c
mrstrong.c
mrxgcd.c
mrio1.c
mrio2.c
// Only used if MR_COMBA is defined in mirdef.h
mrcomba.c
```

Listing 5.2: Optimized MIRACL Configuration

```
/*
 * MIRACL compiler/hardware definitions - mirdef.h
 * This version suitable for use with most 32-bit computers
 * e.g. 80386+ PC, VAX, ARM etc. Assembly language versions
 * f muldiv, muldvm, muldvd and muldvd2 will be necessary.
 * See mrmuldv.any
 *
 * Copyright (c) 1988-2006 Shamus Software Ltd.
 */
#define MIRACL 32
#define MR_LITTLE_ENDIAN
#define mr_utype int
#define MR_IBITS 32
#define MR_LBITS 32
#define mr_dltype long long
#define mr_unsign32 unsigned int
#define mr_unsign64 unsigned long long
#define MAXBASE ((mr_small)1<<(MIRACL-1))
#define MR_BITSINCHAR 8
#define MR_NO_FILE_IO
#define MR_NO_STANDARD_IO
#define MR_STRIPPED_DOWN
/* Include attributes */
#define ATTR
/* With Comba - Test Case 3
```

```
 * If Comba -> Monty is disabled automatically */
#define MR_SPECIAL
#define MR_COMBA 6 /* 6*32bit = 192bit */
#define MR_GENERALIZED_MERSENNE

/* No Comba but Monty - Test Case 2
 * Only comment out preprocessors from Test Case 3
 * Montgomery invoked automatically */

/* No comba, No Monty = Test Case 1 */
// #define MR_DISABLE_MONTGOMERY
```

Listing 5.3: mirdef.h - MIRACL

The following itemization explains the different preprocessors used in the *mirdef.h* file:

- MIRACL_32 - We are working on a 32bit processor
- MR_LITTLE_ENDIAN - Lowest address is least significant bit
- mr_utype int - The underlying type is int
- MR_IBITS 32 - Number of bits in an int
- MR_LBITS 32 - Number of bits in a long
- mr_dltype long long - Double length type
- mr_unsign32 unsigned int - Unsigned int type for 32bit
- mr_unsign64 unsigned long long - Unsigned long type for 32bit
- MAXBASE $((mr\_small)1 << (MIRACL - 1))$ - Maximum of an int
- MR_BITSINCHAR 8 - Number of bits in one character
- MR_NO_FILE_IO - Not using file I/O
- MR_NO_STANDARD_IO - No standard I/O from MIRACL
- MR_STRIPPED_DOWN - Minimize library (suppress all error messages)
- ATTR - Defines the use of attributes for the extended solution of the scheme
- MR_DISABLE_MONTGOMERY - Disable modular multiplication speed up by Montgomery arithmetic
- MR_COMBA 6 - Enable fast modular arithmetic with Comba method for the NIST P-192 domain parameter (6 * 32bit = 192bit)
- MR_SPECIAL - Enable fast modular arithmetic for special modulus lengths (e.g. 192bit)

- MR_GENERALIZED_MERSENNE - Special code for Mersenne primes like $2^{192} - 2^{64} - 1$ (assuming 32-bit processor). Usable in combination with MR_COMBA and MR_SPECIAL

5.5 Test Environment

The entire test environment is an easy set up with a PC and the MCB2130 evaluation board which are connected via a serial port to USB adapter, since the MCB2130 can only establish a connection via a serial port. For communication between server and client, we establish a SSL socket connection.
The PC test environment is a Suse Linux 11.3 64bit OS running on a Intel(R) Core(TM) i5 CPU with 4x2,67 GHz and 8 GB of RAM. For developing the server and client side in Java we used the Java Platform, Standard Edition 6 Development Kit[6].
For the secure world, which is written in C, we used the cross compiler toolchain (explained in Section 2.1.8) to make it runnable on an ARM architecture, since the MCB2130 has an ARM7TDMI processor. A more detailed description about the MCB2130 evaluation board is given in Section 5.5.1.

5.5.1 MCB2130 Evaluation Board

The MCB2130 evaluation board from Keil™, as shown in Figure 5.1, is a development board with a LPC2138 MCU from NXP[7]. The ARM7TDMI processor has a maximum CPU clock rate of 60 Mhz. The On-Chip RAM is 32kB and the On-Chip Flash provides 512kB of memory. Two serial ports provide an interface for the communication to the outside and a JTAG Connector helps to debug running programs. It also supports analog input and output but this is not used by our scheme. A detailed specification can be found at www.keil.com/mcb2130/specs.asp.

As shown at www.keil.com/support/man/docs/mcb2130/mcb2130_to_serial.htm we used the serial port COM0 for flash programming via the Flash Magic Utility[8]. Therefore it was necessary to set the jumpers J1 and J10 on the evaluation board. This allows a board reset and In-System Programming (ISP). The serial port COM1 was used to get console output for debugging and performance evaluation since no JTAG debugger was available.

[6] http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javase6-419409.html#jdk-6u23-oth-JPR
[7] www.nxp.com
[8] http://www.flashmagictool.com

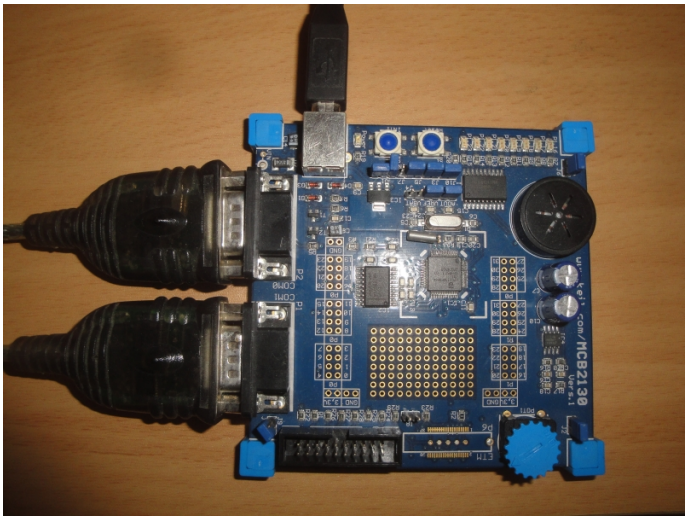

Figure 5.1: MCB2130 Evaluation Board from Keil™

5.6 Arithmetic Optimizations

For the implementation of the secure world we used the MIRACL library because it provides different mechanisms for fast modular multiplication. The library is implemented in a way that it is possible to include and exclude fast modular multiplication by defining preprocessors in the *mirdef.h* file as shown in Listing 5.3. This was important to gather information about execution times using different mechanisms provided by the library. We used two methods for faster modular multiplication on the KARIM algorithm. Additionally we executed the proposed algorithm without any acceleration routines and compared the results as discussed in Section 6.2. We have decided to use the Montgomery [27] and Comba [14] mechanisms for performance evaluation because the MIRACL library supports them out of the box.

5.6.1 Montgomery & Comba

Modular arithmetic usually takes high computational effort. Hence, it is mandatory to focus on speeding up these calculational steps for good performance values. This is important especially for low powered devices as we used in our test environment. By default the MIRACL library uses the Montgomery reduction when extensive modular arithmetic is required. We can turn off the use of Montgomery with the preprocessor command *MR_DISABLE_MONTGOMERY* in the *mirdef.h* file. This will execute the code without any modular multiplication acceleration.
To use the Comba mechanism it is necessary to include the preprocessor command *MR_COMBA*. In our case we used the acceleration for a 192 bit modulus on a 32 bit ARM processor. Therefore *MR_COMBA 6* stands for $6 * 32 = 192$ to optimize the reduction steps for the given modulus length. The Comba method is also suitable for

$GF(p)$ elliptic curve cryptography. As described in the MIRACL users manual [34], the Comba method is ideal for special small to medium sized moduli up to 512 bit. The special 192 bit modulus p ($p = 2^{192} - 2^{64} - 1$ which is a Mersenne prime [42]) we have used is perfect to exploit this mechanism. The preprocessor commands *MR_SPECIAL* and *MR_GENERALIZED_MERSENNE* need to be included into the *mirdef.h* file.

5.7 Memory Optimizations

The MCB2130 evaluation board only provides 512 kB of flash memory, therefore it was important to keep the program small enough to be stored onto the device. The GCC compiler provided by CodeSourcery made it possible to compile the written code with distinct optimization levels. We had the opportunity to chose between the following levels defined in [36]:

- No optimization: The code is compiled without any optimization
- Level 1: The compiler tries to reduce code size and execution time while keeping the compilation time short [GCC Flag: -O1]
- Level 2(speed): The compiler invokes all supported optimizations which do not involve a space-speed tradeoff and all optimizations from level 1 [GCC Flag: -O2]
- Level 2(size): The compiler optimizes the code for size including all optimizations from level 2(speed) that do not typically increase code size [GCC Flag: -Os]

Furthermore, it has to be mentioned that the code size varies from using different preprocessor commands as shown in Listing 5.3. The code size differs from using the Comba methods for modular multiplication accelerations to the usage of the Montogomery mechanism or no acceleration at all. The reason for this behavior is that the Comba mechanism needs additional methods from the MIRACL library. Therefore, we had to add the file *mrcomba.c* to the library of our program as shown in Listing 5.2. The code size also differs from using the base and the extended solution of the proposed algorithm. More code has to be compiled if we want to include attributes which increases the code size. Table 5.1 summarizes the different program sizes obtained from the different options. We have chosen to use the optimization level 2(speed) because the compiled code is small enough to fit into the flash memory of the MCB2130 evaluation board and our focus lies on fast execution times.

Table 5.1: Program Size with Various Compiler Optimizations

Case	**No Optimization**	**Level 1**	**Level 2 (speed)**	**Level 2 (size)**
No Attributes No Comba	522 kB	355 kB	410 kB	338 kB
No Attributes Comba	547 kB	374 kB	429 kB	357 kB
Attributes No Comba	529 kB	360 kB	415 kB	343 kB
Attributes Comba	554 kB	379 kB	434 kB	361 kB

Chapter 6

Evaluation

For an implementation like the KARIM-ACS, performance always plays a big role. The cryptographical part usually consumes the highest amount of system resources and therefore takes longest. For the proposed scheme, we are going to have a look at the memory usage and processing times of the different calculation steps in the secure world. For the performance evaluation we only focus on the secure world since this is going to be the bottleneck of the system. All necessary cryptographic algorithms are executed in the trusted environment. Summarized the secure environment is the critical and time consuming component of our system. The tests were executed on the MCB2130 Evaluation Board from our test environment as described in Section 5.5. The test results are displayed and discussed in Section 6.2.
During the implementation we found some security flaws in the KARIM-ACS which we analyze in Section 6.3.

6.1 System Parameters

The security parameters $\alpha \geq 1$, $l_f > 1$, $l_h > 1$ and $l_n > 1$ and the constant integer c_X are introduced for the proposed scheme. The size of the different values is chosen depending on the elliptic curve. The value l_n always represents the bit length of the modulus of the underlying field where P-192 concludes to $l_n = 192$. The security parameter $l_h = 160$ defines the collision resistant hash function $H : \{0,1\}^* \Rightarrow \{0,1\}^{l_h}$ which is SHA-1 [31] in our case. As defined in Section 2.2.5 we need to chose α and l_f in a way that $\alpha \cdot (l_f + l_h) + 1 < l_n$ which results in $\alpha = 1$ and $l_f = 30$. For the constant integer c_X we define $c_X > 2^{\alpha \cdot (l_f + l_h) + 2}$.

6.2 Performance Evaluation

The performance evaluation of the secure world includes calculation times of different steps and the memory usage in the proposed scheme. Since we are using the MIRACL library, we have the opportunity to use different approaches for faster modular multiplication. The modulus included in the domain parameter P-192 standardized by NIST has a fixed length of 192 bits. We can use this information to prepare the implemented C code for optimizations as described in Section 5.6. The *mirdef.h* file offers us the opportunity to control library intern method calls by setting different preprocessor instructions. We want to emphasize the most important preprocessors, described in Listing 5.3 for our tests in the following itemization:

- ATTR - Defines the use of attributes for our extended version of the scheme
- MR_DISABLE_MONTGOMERY - Disable modular multiplication speed up by Montgomery arithmetic
- MR_COMBA 6 - Enable fast modular arithmetic with Comba method for the NIST P-192 domain parameter (6 * 32bit = 192bit)
- MR_SPECIAL - Enable fast modular arithmetic for special modulus lengths (e.g. 192bit)
- MR_GENERALIZED_MERSENNE - Special code for Mersenne primes like $2^{192} - 2^{64} - 1$ (assuming 32-bit processor). Usable in combination with MR_COMBA and MR_SPECIAL

Terminology

Before we can start to analyze the scheme we need to give some important definitions on the used terms.
To generate a random number from the cryptographically strong random number generator provided by the MIRACL library we denote the term *strong-RNG*.
A multiplication of an integer i with an EC point P ($i * P$) is denoted as *curve multiplication* which is a repeated point addition. This is the most intensive operation because it breaks down to many modular multiplications and additions.
The operation of setting coordinates of an EC point is denoted as *set point* operation. The set point operation is considered to be a fast process, but if we break down the function call we can see that there is more than just a few operations. The set point method checks if the created point is a real point on the current elliptic curve. This needs some computation of the coordinates, which results in two modular additions, four modular multiplications and some comparing operations.
The term *normalization* denotes the conversion from projective coordinates to affine coordinates. Projective coordinates of an EC point are used for faster calculations inside the MIRACL library. To transmit an EC point we need to convert the coordinates (x,y,z) to (x,y) as described in [10]. We also denote the *point initializations* which ini-

Table 6.1: Test Cases

Test Case	No Attributes
1	No Comba, No Montgomery
2	No Comba, With Montgomery
3	With Comba
	With Attributes
4	No Comba, No Montgomery
5	No Comba, With Montgomery
6	With Comba

tializes an EC point before calling the set point operation. Due to the fact that these two function calls of normalization and point initialization are not really influencing the execution time we left them out in the code analysis.

6.2.1 Calculation Times

Calculation times are very important to get significant results on our performance evaluation. Because we are working on low powered devices, fast processing of the various protocols is mandatory. The fact that we have a basic and an extended solution of the KARIM-ACS leads us to six different test cases as shown in Table 6.1.

Code Analysis - Basic Solution

Corresponding to the introduced protocols in Section 4.2 we analyze the different computational steps executed in the secure world.

Setup
To produce the private key f and the integer $\hat{\omega}$ we need two strong-RNG function calls. Furthermore, we need one modular multiplication to compute $\tilde{\omega}$, one set point operation to set the group's public key Ψ_I and one curve multiplication to compute $\tilde{Y}$.

Join
The only part of the join protocol in the secure world is to execute the check $f * U \equiv \Psi_g + P + f * \Psi_I$. This involves four set point operations for U, P, Ψ_g and Ψ_I, two curve multiplications and three point additions. Furthermore, we just need to compare the two EC points for equality.

Table 6.2: P-192 / Base Solution

Test Case	Generate Key	Setup	Join	Sign	Total
No Monty - No Comba	**5ms**	**1,620s**	**1,611s**	**8,179s**	**11,291s**
With Monty		**0,867s**	**0,854s**	**4,279s**	**6,005s**
With Comba		**0,189s**	**0,180s**	**0,899s**	**1,273s**

Sign
Computing a signature is a mathematically intense operation which influences the processing time most as described in Section 4.2.2. To calculate the values $\Upsilon, \Phi, \Gamma, \Delta$ and P' we need five curve multiplications and one modular multiplication to get $\bar{r}$. Furthermore, it is necessary to calculate a hash value h which does not significantly influence the processing time. Hence, we need to compute the proof values ν' and ν'' which need two standard multiplications, two subtractions and two additions.
Note: The proof values ν' and ν'' need to lie in the interval of the underlying prime field $\mathbb{F}_p$ if the parameters f, r, r' and r'' are chosen with the right lengths.
Assembling the signature parameters can be done in the normal world but it does not influence the processing time much.

Time Analysis - Basic Solution

The first three tests were processed not using attributes as introduced in Section 4.2. The test results are shown in Table 6.2 where the first test was executed without using any modular multiplication accelerations, the second test with the Montgomery mechanism and the third test with the Comba methods. In the first test it is easy to see that not performing any accelerations takes a lot of time for computation. The entire process takes up to more than 11 seconds. For all tests it is important to know that the results come from an evaluation done directly on the MCB2130 board. No external systems where involved, neither is any operating system installed on the board itself. This means that there where no I/O operations involved, because we wanted to get more accurate results on the raw calculation times. For the second test we can see that he computation times are about two times faster compared to the first one, which is already a good performance enhancement. But signature times around 4 seconds are still too long. So we need to be as fast as possible in order to produce a valid signature in a convenient time. The third approach using the Comba methods fastens up the execution times tremendously. The entire process takes around 1,3 seconds, which is quite convenient. It is more than four times faster than using the Montgomery mechanism. The setup and join process takes about 200 ms, which does not influence the whole process substantially. Hence, the signing process with the highest computational effort of the entire procedure takes less than one second. Gaining such a result would make it worth to put more effort into this scheme and do further investigation on calculation accelerations and the algorithms especially for the signature protocol.

Code Analysis - Extended Solution

For the code analysis of the extended solution described in Section 4.3 we only need to focus on the signature algorithm, because the other protocols stay the same in consideration of executing them in the secure world. Since we are now using attributes, we have some additional calculations which consume more computational effort compared to the base solution.

Table 6.3: P-192 / Extended Solution

Test Case	Generate Key	Setup	Join	Sign	Total
No Monty - No Comba	**5ms**	**1,638**	**1,619s**	**9,814s**	**13,076s**
With Monty		**0,864s**	**0,856s**	**5,267s**	**6,991s**
With Comba		**0,189s**	**0,178s**	**1,097s**	**1,469s**

Sign
If we receive the hash value over all attributes $\bar{x}$ from the issuer we can spare one curve multiplication for each attribute and the execution of the hash function. However, for each attribute we want to show to the verifier one additional curve multiplication is required to further produce the hash value $\tilde{x}_show$. To produce the signature parameter P' it also needs more computational effort because the parameters $\bar{x}$ and $\tilde{x}_{show}$, as multiplicative inverse ($\tilde{x}^{-1}_{show}$), are involved in the calculation. This results in one additional computation of the multiplicative inverse and two additional modular multiplications compared to the signature algorithm from the base solution. The supplemental calculation steps apparently increase the processing time of the entire scheme. We used four attributes for our tests, two of which are shown to the verifier and two remain hidden as described in Section 5.1.

Time Analysis - Extended Solution

As we can see in Table 6.3 the use of attributes slows down the process. The Key generation, the setup and join phase result in the same processing times. Only the execution of the signature protocol needs more computational effort. This is easily explained, since the additional calculations for each attribute are executed here. The more attributes, the longer the calculation times. For each attribute we need one additional curve multiplication if we do not receive the hash value $\bar{x}$ from the issuer. If the attribute is shown another curve multiplication needs to be executed to produce the hash value over all shown attributes $\tilde{x}_{show}$.

6.2.2 Summarization

Modular arithmetic acceleration is important to reduce the execution times of the proposed algorithms tremendously. We only consider signature processing times for this summarization since the other protocols are usually executed beforehand. A signature procedure without using modular arithmetic acceleration is not considerable for further usage, since more than 8 to 10 seconds is not convenient for us. Also the use of Montgomery arithmetic does not provide satisfactory processing times (4 - 5 seconds). We need to consider that a user stands in front of a ticket reader (verifier) an wants to gain access to an event. Therefore signing a ticket needs to be executed as fast as

possible because the customer doe not want to wait. Only the Comba algorithm for fast modular arithmetic gives good performance values. The special prime modulus and the MIRACL library offered the opportunity to use this acceleration for our calculations. Execution times from around 900 ms not using attributes and 1,1 seconds with the extended solution are quite good for such a low powered device. It is necessary to know that we used four attributes, of which we decided to show two and hide the other two. This leads to the assumption that more attributes result in longer computation times, since every additional information needs some extra calculations. Total operation times all over the different steps are not really meaningful, since the processes of key generation and setup can be done anytime beforehand, and also the join procedure should be finished to be in possession of the important credential before signing any data.

6.2.3 Memory Usage

Memory usage is the second important point we were focusing on. The memory storage and the RAM size are usually small on embedded devices. For example, our evaluation board has a very limited size of RAM (32 kB), which forces the developer to take care of memory usage like allocating and freeing heap space. It has also a bounded flash storage module with only 512 kB of size. This needs special treatment as well, since a program which is too big is not able to run on this device. We separated our 32 kB of RAM into 8 kB for the stack and 24 kB as heap space for dynamic memory allocation. Because we did not have a JTAG debugger tool, we were forced to test the RAM usage of our system with Valgrind's[1] *Massif* tool. We also used Valgrind's *Memcheck* tool to detect memory-management problems and memory leaks. Therefor, we had to add the preprocessor command *MR_NOASM* in the *mirdef.h* file which defines that we are using C code only.

RAM Usage Analysis

Table 6.4 shows how much heap size was used in the different steps of our scheme. As we can see, the amount of memory usage does not change between executing the basic version and the extended version in *key generation*, *setup* and *join*. The reason for this is that the attributes are only used in processing the *signature* protocol. The occurrence of attributes raises the allocated space to 22,52 kB, which still fits in the provided heap space of 24 kB.

[1] http://valgrind.org

Table 6.4: RAM Size Usage

Allocated Heap Space				
Test Case	**Generate Key**	**Setup**	**Join**	**Sign**
1	7,95kB	14,73kB	15,44kB	20,12kB
2				
3				
4	7,95kB	14,73kB	15,44kB	22,52kB
5				
6				

Flash Memory Size Analysis

The MCB2130 evaluation board only provides 512 kB of flash memory, therefore it was important to keep the program size small. As explained in Section 5.7 we used the GCC compiler provided by CodeSourcery to achieve proper program sizes for the different tests. We used the compiler optimization level 2(speed) which was convenient for all test runs. The maximum program size of 434 kB as shown in Table 5.1 did not exceed the flash memory size of the evaluation board.

6.3 Security Concerns

During the implementation of the proposed prototype we found some security flaws in the protocols described in Chapter 4. We identified that the requirement of total anonymity is not given throughout this concept of an ECC-based anonymous credential system. Problems occur with the property of *unlinkability* which should assure total anonymity to the signer. Unlinkability must ensure that one signature should not be linkable to another signature from the same signer using the same credential $[U, P]$. Furthermore, we identified the issue of *self-joining* which gives an attacker the opportunity to join a group without the need of an issuer. This means an attacker can sign data on behalf of a group without executing the join procedure.

6.3.1 Linkability

By linkability it is meant that a signature coming from one specific signer can be linked to another signature by the same signer using the same credential $[U, P]$ issued by the issuer. The problem with linkability is that it harms the term of total anonymity which should be supplied by the proposed scheme. Unfortunately we found out that it is possible to link a signature from a signer S to any other signature from S assuming the same credentials $[U, P]$. This is feasible for both protocol variants, the base and the extended solution. For both we know that $P = \hat{r} * \Psi_I$ is a fixed value of the credential $[U, P]$ issued to the signer for his private key f. Therefore the pair $[f, P]$ can be considered as constant for each signature produced by the same signer holding the credential $[U, P]$.

Base Solution

The signature parameters received by the verifier are $\sigma = (h, \nu', \nu'', \Upsilon, \Phi, P', \bar{r}, PSN, r_P)$. We assume two received signatures σ_1, σ_2 from the same signer where we only need two parameters $(\bar{r}, P')$ to prove the linkability between the signatures. From the signature protocol we know that $P' = r * P$ and $\bar{r} = r \cdot f \bmod n$. Hence, it is easy to calculate the multiplicative inverse $\bar{r}^{-1}$ since we know the modulus n (n is the order of the underlying prime field). Now we can calculate L_1, L_2 the tokens for linkability of σ_1, σ_2 respectively as:

$$L_1 = \bar{r}_1^{-1} * P'_1 \tag{6.1}$$

$$L_2 = \bar{r}_2^{-1} * P'_2 \tag{6.2}$$

Hence, we use the knowledge about $\bar{r}^{-1}$ and P' in the next step as that

$$\begin{aligned} \bar{r}_1^{-1} &= (r_1 \cdot f)^{-1} \bmod n \\ &= r_1^{-1} \cdot f^{-1} \bmod n \\ \bar{r}_2^{-1} &= (r_2 \cdot f)^{-1} \bmod n \\ &= r_2^{-1} \cdot f^{-1} \bmod n \end{aligned}$$

which is substituted into equation 6.1 and 6.2.

$$L_1 = (r_1^{-1} \cdot f^{-1} \cdot r_1) * P$$

$$L_2 = (r_2^{-1} \cdot f^{-1} \cdot r_2) * P$$

It is obvious that the random values r_1, r_2 chosen by the signer cancel out. The linkability tokens L_1, L_2 remain as follows:

$$L_1 = f^{-1} * P$$

$$L_2 = f^{-1} * P$$

Now we can see that it is easy to link two signatures σ_1 and σ_2 from the same entity by checking on equality of L_1 and L_2. A signature from another entity with different values in $[f, P]$ would not be equal.

Extended Solution

Using attributes within the signature does not make it much harder to check if a signature comes from an already known signer. The signature parameters received by the verifier are $\sigma = (h, \nu', \nu'', \Upsilon, \Phi, P', \bar{r}, SET(x_{i \in show}), PSN, r_P)$. Again, we assume two signatures σ_1, σ_2 from the same signer where we need three parameters $(\bar{r}, P', SET(x_{i \in show}))$ to prove the linkability between the signatures. From the set of attributes we can calculate the hash value $\tilde{x}_{show}$ as shown in Section 4.3.3. From the signature protocol we know that $P' = (r \cdot \bar{x} \cdot \tilde{x}_{show}^{-1}) * P$ where $\bar{x}$ is the hash value of all attributes which is a fixed value for all signatures from the same signer. Furthermore, we know that $\bar{x}$ is part of the credential $[U, P]$ from the join protocol in Section 4.3.1. Now we can calculate L_1, L_2 the tokens for linkability of σ_1, σ_2 respectively as:

$$L_1 = (\bar{r}_1^{-1} \cdot \tilde{x}_{show1}) * P_1' \tag{6.3}$$

$$L_2 = (\bar{r}_2^{-1} \cdot \tilde{x}_{show2}) * P_2' \tag{6.4}$$

Hence, we use the knowledge about $\bar{r}^{-1}$ and P' in the next step and substitute these parameters into equation 6.3 and 6.4.

$$L_1 = (r_1^{-1} \cdot f^{-1} \cdot \tilde{x}_{1show} \cdot r_1 \cdot \bar{x} \cdot \tilde{x}_{show1}^{-1}) * P$$

$$L_2 = (r_2^{-1} \cdot f^{-1} \cdot \tilde{x}_{2show} \cdot r_2 \cdot \bar{x} \cdot \tilde{x}_{show2}^{-1}) * P$$

It is obvious that the random values r_1, r_2 chosen from the signer and the hash values of the shown attributes $\tilde{x}_{show1}, \tilde{x}_{show2}$ cancel out. The linkability tokens L_1, L_2 remain as follows.

$$L_1 = (f^{-1} \cdot \bar{x}) * P$$

$$L_2 = (f^{-1} \cdot \bar{x}) * P$$

Again, the equality of L_1, L_2 links the two different signatures of one signer.

6.3.2 Self-Joining

As already mentioned before, *self-joining* is possible within the proposed anonymous credential scheme. This is a tremendous problem for the entire system since it is not secure against attacks from the outside. Anyone who is able to receive the public parameters Ψ_g, Ψ_I is able to sign data on behalf of that particular group. The term of *Self-Joining* means that we do not need an issuer to produce a valid credential for signing purposes. How to join a group as an attacker is shown in the next paragraph.

Assuming that we are an attacker and we want to sign data on behalf of the group G with the public parameters Ψ_g, Ψ_I. It is possible to chose a random private key f and a random value $\hat{r}$ such that we are able to produce a valid signature σ.
We know that $U = (\tilde{\omega}^{-1} \cdot (\iota^{-1} + \hat{r})) * \tilde{Y} + \Psi_I$ and $P = \hat{r} * \Psi_I$ and the values ι, $\hat{r}$ are chosen randomly by the issuer where ι is the issuer's private key. Hence, the private key f is chosen randomly by the client. First we examine U to see of what variables we can get rid off. Therefore we need the knowledge about $\Psi_I = \iota * \Psi_g$ which is the group's public key, $\tilde{Y} = \hat{\omega} * \Psi_I$ and $\tilde{\omega} = f \cdot \hat{\omega} \bmod n$ which are calculated by the client. The following substitution reduces the variables to a minimum:

$$
\begin{aligned}
U &= (\tilde{\omega}^{-1} \cdot (\iota^{-1} + \hat{r})) * \tilde{Y} + \Psi_I = \\
&= ((f \cdot \hat{\omega})^{-1} \cdot (\iota^{-1} + \hat{r})) \cdot \hat{\omega} * \Psi_I + \Psi_I = \\
&= (f^{-1} \cdot \hat{\omega}^{-1} \cdot \iota^{-1} + f^{-1} \cdot \hat{\omega}^{-1} \cdot \hat{r}) \cdot \hat{\omega} \cdot \iota * \Psi_g + \Psi_I = \\
&= f^{-1} \cdot \hat{\omega}^{-1} \cdot \iota^{-1} \cdot \hat{\omega} \cdot \iota * \Psi_g + f^{-1} \cdot \hat{\omega}^{-1} \cdot \hat{r} \cdot \hat{\omega} \cdot \iota * \Psi_g + \Psi_I = \\
&= f^{-1} * \Psi_g + f^{-1} \cdot \hat{r} \cdot \iota * \Psi_g + \Psi_I = \\
&= f^{-1} * \Psi_g + f^{-1} \cdot \hat{r} * \Psi_I + \Psi_I
\end{aligned}
$$

We can now see that there are only two variables left which are chosen randomly for each credential. As an attacker we are able to chose our own private key f and the random value $\hat{r}$ to produce a valid U and also a valid P since $P = \hat{r} * \Psi_I$ which results in a new credential $[\tilde{U}, \tilde{P}]$. The only thing we need to take care of is the range check by the verifier. Therefore, we have to chose the variables f and $\hat{r}$ carefully. Now we are able to produce signatures on behalf the group G where we know the generator point Ψ_g of the curve and the public point Ψ_I.
The worst case for a group signature scheme is if anybody can sign data on behalf of a group without executing the join process. Due to the fact that this is possible it breaks the entire KARIM-ACS.

Chapter 7

Conclusion & Future Work

7.1 Conclusion

The system implemented and evaluated in this master's thesis is a new approach to anonymous authentication used on embedded devices developed as part of a research project led by Kurt Dietrich. The anonymous credential system based on elliptic curve cryptography provides a lot of opportunities for the researcher.
The idea of using anonymous credential systems arises from concerns about privacy and user profiling. A new architecture on the client side separating the normal world and the secure world, as shown in Chapter 3, makes it feasible to install this scheme on almost every mobile device. The secure world ensures secure data handling and hides private information about the user. A lot of effort was put into the algorithm development to fulfill those requirements. However, it has to be mentioned that creating a new group signature scheme has its hassles as we could see in Section 6.3 about security flaws. The protocols discussed in Chapter 4 are basically derived from a group signature scheme introduced by He Ge and Stephen R. Tate [18] based on RSA technology, but altered for our needs. To convince a verifier that data comes from a trusted entity while hiding the signers identity is not an everyday use case. A lot of computational steps need to be executed for a successful procedure. In some cases, a verification entity needs more information about the user than just the knowledge that the received signature is trustworthy. By using personal characteristics included in the signature process, the signer is able to show specific information. A set of attributes is assigned to a person, which makes it possible to satisfy different requirements of different verifiers. The extended version of the proposed protocol uses such attributes for this purpose.
The most important part of this thesis is the implementation of the proposed scheme and the subsequent performance evaluation of the secure world, since this is the bottleneck of the entire system. The MCB2130 evaluation board from Keil™ provided the test environment. We used the domain parameter P-192 standardized by NIST based on a special modulus p ($p = 2^{192} - 2^{64} - 1$ which is a Mersenne prime [42]), which made it possible to use modular multiplication accelerations as described in Section 5.6.

Moreover, the RAM size is limited to 32 kB, from which we used 24 kB of heap memory and the rest was reserved for the stack. As we can see in Table 6.4, the RAM usage of 22,52 kB at most did not exceed the reserved heap size. We have to consider that these tests were executed on bare metal, which means no running OS nor any other applications. Hence, there was no influence from other executions on RAM usage or processing times. This raises a few issues concerning mobile devices like Smart Cards, since we do not have the benefit of a stand alone application. To almost exhaust the heap size of 24 kB does not leave a lot of space for other applications which means that we would need more RAM if not running this program as a stand alone application. Also bigger key sizes up to 521 bit (P-521 domain parameter) could raise problems on such small RAM chips. Therefore, high security applications can run into troubles due to high memory usage. For our purposes we did not consider higher security levels, because the elliptic curve based on the NIST P-192 domain parameter used for the tests was sufficient for our needs. Another important issue gained from the tests is the processing times of the different protocols. Due to the fact that the signing procedure takes longest, it is worthwhile to analyze it more accurately as done in Section 6.2. As a conclusion we can say that it is not considerable to use the KARIM-ACS without any acceleration methods as introduced in Section 5.6. The Comba modular multiplication acceleration provides a fast mechanism which reduces the execution time of the protocols to a convenient level.
The security flaws discussed in Section 6.3 were detected in the review process after we finished the implementation part. As already mentioned it is not an easy task to develop a new group signature scheme. A lot of effort needs to be put into a project like this. However, the proposed scheme is not usable due to the fact that anybody is able to join a group without executing the join protocol. An attacker only needs to know the public parameters of the issuer to self-join the group as explained in Section 6.3. Moreover, the KARIM-ACS implements the linkability of signatures coming from the same signer. This would decrease the term of total anonymity to pseudo anonymity not considering the self-joining issue.
As a summarization we can say that using ECC is probably faster than a direct anonymous attestation scheme based on RSA because it uses shorter key lengths even though we did not compare the test results directly to an RSA based scheme. It is also possible to produce signatures in a satisfying amount of time on the MCB2130 Evaluation Board. Nevertheless, there are bugs in the algorithm so it could be possible that resolving those issues lead to higher computation times. The downside are the two security flaws found in the proposed anonymous credential system which makes the entire scheme unusable. However, we can say that the proposed scheme looks promising if the bugs are going to be investigated and resolved.

7.2 Future Work

Due to the fact that the proposed system has its security problems a first important step is to remove the bugs of the KARIM algorithm. The fact that anyone is able

to produce his own credential and that the signatures are linkable cancels the term of total anonymity for the proposed scheme. Further investigations need to focus on those security issues to achieve total anonymity. If the security issues are resolved a next step could be to accelerate the calculational steps in the signature algorithm with other methods than used in this evaluation process. Furthermore, the scheme could be tested on different platforms to get a better overview of performance.

References

[1] Almuhammadi, S., Sui, N. T., and McLeod, D. Better privacy and security in e-commerce: Using elliptic curve-based zero-knowledge proofs. In *Proceedings of the IEEE International Conference on E-Commerce Technology* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 299–302.

[2] Anoop, M. Elliptic curve cryptography: An implementation guide. Online, January 2007. http://www.infosecwriters.com/text_resources/pdf/Elliptic_Curve_AnnopMS.pdf.

[3] Balasch, J. Smart card implementation of anonymous credentials. Master's thesis, Katholieke Universiteit Leuven, 2008.

[4] Ban, J. Cryptographic library for ARM7TDMI processors. Master's thesis, Technical University of Kosice, 2007.

[5] Batina, L., Hoepman, J.-H., Jacobs, B., Mostowski, W., and Vullers, P. Developing efficient blinded attribute certificates on smart cards via pairings. In *Proceedings of the 9th IFIP WG 8.8/11.2 international conference on Smart Card Research and Advanced Application* (Berlin, Heidelberg, 2010), CARDIS'10, Springer-Verlag, pp. 209–222.

[6] Bauer, W. Kryptosysteme basierend auf Elliptischen Kurven, Einsatz und Verbreitung in Standardsoftware. Tech. rep., Graz, Technical University, 2008.

[7] Bonatti, C. D. ASN.1 Enhancements To Support Tactical Data Communications, June 1993. International Electronic Communications Analysts.

[8] Brickell, E., Camenisch, J., and Chen, L. Direct anonymous attestation. In *Proceedings of the 11th ACM conference on Computer and communications security* (New York, NY, USA, 2004), CCS '04, ACM, pp. 132–145.

[9] Camenisch, J. Better Privacy for Trusted Computing Platforms. In *Computer Security @ ESORICS 2004*, P. Samarati, P. Ryan, D. Gollmann, and R. Molva, Eds., vol. 3193 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2004, pp. 73–88.

[10] Certicom. SEC 1: Elliptic Curve Cryptography, Standard, May 2009.

[11] Chen, L. A DAA scheme requiring less TPM resources. In *Proceedings of the*

5th international conference on Information security and cryptology (Berlin, Heidelberg, 2010), Inscrypt'09, Springer-Verlag, pp. 350–365.

[12] Chen, X., and Feng, D. A new direct anonymous attestation scheme from bilinear maps. In *Proceedings of the 2008 The 9th International Conference for Young Computer Scientists* (Washington, DC, USA, 2008), ICYCS '08, IEEE Computer Society, pp. 2308–2313.

[13] Cheng, Z. Simple Tutorial on Elliptic Curve Cryptography. Online, December 2004.

[14] Comba, P. G. Exponentiation cryptosystems on the IBM PC. *IBM Syst. J. 29*, 4 (Oct. 1990), 526–538.

[15] Cramer, R., Damgård, I., and MacKenzie, P. D. Efficient Zero-Knowledge Proofs of Knowledge Without Intractability Assumptions. In *Proceedings of the Third International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography* (London, UK, UK, 2000), PKC '00, Springer-Verlag, pp. 354–372.

[16] Diffie, W., and Hellman, M. New directions in cryptography. *IEEE Trans. on Information Theory IT-22(6)* (November 1976), 644–654.

[17] Fiege, U., Fiat, A., and Shamir, A. Zero knowledge proofs of identity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing* (New York, NY, USA, 1987), STOC '87, ACM, pp. 210–217.

[18] Ge, H., and Tate, S. R. A direct anonymous attestation scheme for embedded devices. In *Proceedings of the 10th international conference on Practice and theory in public-key cryptography* (Berlin, Heidelberg, 2007), PKC'07, Springer-Verlag, pp. 16–30.

[19] Gura, N., Patel, A., W, A., Eberle, H., and Shantz, S. C. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In *CHES* (2004), pp. 119–132.

[20] Hankerson, D., Menezes, A. J., and Vanstone, S. *Guide to Elliptic Curve Cryptography*. Springer Berlin / Heidelberg, 2004.

[21] Hoepman, J.-H., Jacobs, B., and Vullers, P. Privacy and Security Issues in e-Ticketing – Optimisation of Smart Card-based Attribute-proving. In *Workshop on Foundations of Security and Privacy, FCS-PrivMod 2010, Edinburgh, UK, July 14-15, 2010. Proceedings* (July 2010), V. Cortier, M. Ryan, and V. Shmatikov, Eds. (informal).

[22] IAIK. *A Guide to the IAIK ECCelerate Library*, 1.0 ed. Graz University of Technology, Institute for Applied Information Processing and Communications Graz - University Of Technology Inffeldgasse 16a A-8010 Graz, June 2011.

[23] ITU-T. Information Technology: Abstract Syntax Notation One (ASN.1): Spec-

ification Of Basic Notation, July 2002. ITU-T Recommendation X.680, ISO/IEC 8824-1.

[24] ITU-T. Information Technology: ASN.1 Encoding Rules: Specification Of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) And Distinguished Encoding Rules (DER), July 2002. ITU-T Recommendation X.690.

[25] ITU-T. Information technology: Open systems interconnection: The Directory: Public-key and attribute certificate frameworks, November 2008. ITU-T Recommendation X.509.

[26] Kapfenberger, M. A Direct Anonymous Attestation Scheme for Embedded Devices: Implementation and Preformance Evaluation. Tech. rep., Graz, Technical University, April 2010.

[27] Koc, C. K., Acar, T., and Kaliski, B. Analyzing and Comparing Montgomery Multiplication Algorithms. *IEEE Micro 16(3)* (June 1996), 26–33.

[28] Lysyanskaya, A. *Signature Schemes and Applications to Cryptographic Protocol Design*. PhD thesis, MIT, September 2002.

[29] Mostowski, W., and Vullers, P. Efficient u-prove implementation for anonymous credentials on smart cards. In *7th International ICST Conference on Security and Privacy in Communication Networks* (September 2011), Springer, pp. 243–260.

[30] NIST. Digital Signature Standard, June 2009.

[31] O'Neill, M. Low-Cost SHA-1 Hash Function Architecture for RFID Tags. In *Proceedings of the 4th Workshop on RFID Security (RFIDSec08)* (Budapest, July 2008), pp. 41–51.

[32] Pasupathinathan, V., Pieprzyk, J., and Wang, H. An on-line secure e-passport protocol. In *Proceedings of the 4th international conference on Information security practice and experience* (Berlin, Heidelberg, 2008), ISPEC'08, Springer-Verlag, pp. 14–28.

[33] Rescorla, E. *SSL and TLS Designing and Building Secure Systems*. Addison-Wesley, 2000.

[34] Scott, M. *MIRACL Users's Manual*. Shamus Software Ltd., February 2007.

[35] Smyth, B., Ryan, M., and Chen, L. Formal analysis of anonymity in ecc-based direct anonymous attestation schemes. In *Formal Aspects in Security and Trust* (2011), G. Barthe, A. Datta, and S. Etalle, Eds., vol. 7140 of *Lecture Notes in Computer Science*, Springer, pp. 245–262.

[36] Stallman, R. M., and DeveloperCommunity, G. *Using The Gnu Compiler Collection: A Gnu Manual For Gcc Version 4.3.3*. CreateSpace, Paramount, CA, 2009.

[37] Sudarsono, A., Nakanishi, T., and Funabiki, N. Efficient proofs of attributes in pairing-based anonymous credential system. In *Proceedings of the 11th international conference on Privacy enhancing technologies* (Berlin, Heidelberg, 2011), PETS'11, Springer-Verlag, pp. 246–263.

[38] TCG. TCG Software Stack (TSS) Specification Version 1.2 Part1: Commands and Structures. Standard, January 2006.

[39] TCG. TPM Main Part 1 Design Principles. Standard, March 2011. Specification Version 1.2.

[40] Wachsmann, C., Chen, L., Dietrich, K., Löhr, H., Sadeghi, A.-R., and Winter, J. Lightweight anonymous authentication with TLS and DAA for embedded mobile devices. In *Proceedings of the 13th international conference on Information security* (Berlin, Heidelberg, 2011), ISC'10, Springer-Verlag, pp. 84–98.

[41] Want, R. Near Field Communication. *IEEE Pervasive Computing Vol. 10* (2011), 4–7.

[42] Yan, S. Y., and James, G. Testing Mersenne primes with elliptic curves. In *Proceedings of the 9th international conference on Computer Algebra in Scientific Computing* (Berlin, Heidelberg, 2006), CASC'06, Springer-Verlag, pp. 303–312.

[43] Zhang, C., Lu, R., Lin, X., Ho, P.-H., and Shen, X. An Efficient Identity-Based Batch Verification Scheme for Vehicular Sensor Networks. In *INFOCOM'08* (2008), pp. 246–250.

Glossary

$\cdot$ Defines the integer multiplication. 35

$*$ Defines the repeated addition of an elliptic curve point. 36

Direct Anonymous Attestation A digital signature scheme to hide the user's identity. 18

f The private key which is chosen randomly in the secure world ($f \in \mathbb{F}_p$). 55

G The generator point of the elliptic curve. 36

KARIM The name of the proposed scheme. 18

normal world Part of the client considered as the untrusted environment and seperated from the secure world. 20, 42

secure world Part of the client considered as the trusted environment and seperated from the normal world. 20, 42