Christian Gaisl, BSc

Privacy-friendly Payment Scheme from Group Signatures

Master's Thesis

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to
Graz University of Technology

Supervisor
Univ.-Prof. Dipl.-Ing. Dr.techn. Christian Rechberger
Dipl.-Ing. Dr.techn. Christian Hanser
Dipl.-Ing. Dr.techn. Sebastian Ramacher

Institute of Applied Information Processing and Communications

Faculty of Computer Science and Biomedical Engineering

Graz, February 2020

Affidavit

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 indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Abstract

Every day we are getting closer to a cashless society, a development not without its downsides. Digital payments allow for faster and easier payments in our everyday lives, but they also allow third parties to track every one of our purchases. In a typical card payment transaction at a store, there are multiple parties involved, including the store itself, the card issuer, and the bank. Each one of those parties could track the shopping habits and location of the customer and use this data for various purposes buried deep inside their terms of service. Cash, with all its shortcomings and inconveniences, has indisputable benefits for privacy-minded individuals who do not want to share their purchase history with a third party. However, this also makes cash an attractive option for unlawful transactions, especially for money laundering and organized crime. Concretely, the problem at hand is the creation of a digital payment system that combines the convenience of digital payments with the privacy advantages of cash while still maintaining full traceability for legal authorities.

This thesis proposes a payment system based on group signatures in which every party involved only has access to the information it "needs to know". It hides superfluous private customer data from parties that do not need it while still providing authenticity at every step of the transaction. Additionally, by combining data from multiple parties, full traceability of the transaction can be provided for a trusted third party (i.e., legal authorities). In particular, the bank does not get to know at which shop the customer spends his money, and the card provider and the shop do not gain any identifying information about the customer.

The proposed payment system is implemented in an Android App, simulating similar functionality as existing NFC based payment systems such as Apple Pay or Google Pay. This implementation demonstrates not only the feasibility of our payment system but also the feasibility of modern group signature schemes on mobile devices. The chosen group signature scheme behind our payment system is based on elliptic curve cryptography, meaning that a potential future quantum computer could break it. Because of this, this thesis also has a chapter on the present state of post-quantum secure group signature schemes and discusses their advantages and shortcomings.

Keywords: group signatures, privacy enhancing technologies, payment system, cashless society

Kurzfassung

Jeden Tag bewegen wir uns Schritt für Schritt in Richtung einer bargeldlosen Gesellschaft, eine Entwicklung nicht ohne ihre Nachteile. Digitale Zahlungssysteme ermöglichen uns schnellere und einfachere Zahlungsprozesse, aber sie ermöglichen auch Drittparteien wie Banken oder Kreditkartenaussteller jeden unserer Einkäufe zu protokollieren. Jede dieser Parteien kann unsere Einkaufsgewohnheiten und Standortdaten verfolgen und diese Daten für tief in den AGB versteckte Zwecke nutzen. Bargeld, mit all seinen Nachteilen und Unbequemlichkeiten, hat unbestreitbare Vorteile für Privatsphäre bewusste Konsumenten, welche nicht die Details von jedem ihrer Einkäufe mit ihrer Bank teilen möchten. Bargeld hat aber auch schwerwiegende Nachteile, vor allem in Bezug auf organisiertes Verbrechen, Geldwäsche und Steuerhinterziehung. Ein vollständig anonymes digitales Zahlungssystem würde Tür und Tor für illegale Aktivitäten öffnen. Das konkrete Problem, welches diese Arbeit zu lösen versucht, ist die Schaffung eines Zahlungssystems, welches die Bequemlichkeit von digitalen Zahlungssystemen mit den Privatsphäre schützenden Eigenschaften von Bargeld kombiniert und dabei trotzdem die vollständige Nachvollziehbarkeit jeder einzelnen Transaktion für die Behörden garantiert.

Dieses Werk beschreibt ein Zahlungssystem basierend auf Gruppensignaturen, in welchem jede involvierte Partei nur Zugriff auf die notwendigsten Daten hat. Es verbirgt unnötige private Kundendaten von involvierten Parteien während es gleichzeitig die Echtheit und Unverfälschtheit jeder Transaktion zu jedem Zeitpunkt garantiert. Zusätzlich können die Behörden Daten von mehreren Parteien kombinieren und gesamte Transaktion vollständig nachvollziehen. Im Speziellen erfährt die Bank nicht, wo der Kunde einkauft und der Shop und Kreditkartenanbieter erfahren keine identifizierenden Daten über den Kunden.

Das Zahlungssystem ist in einer Android Applikation implementiert und simuliert eine ähnliche Funktionalität wie bereits existierende Zahlungssysteme, wie zum Beispiel Apple Pay oder Google Pay. Die Implementation demonstriert nicht nur die Realisierbarkeit unseres Zahlungssystems, sondern auch die Machbarkeit von modernen Gruppensignaturverfahren auf mobilen Endgeräten. Das von uns gewählte Gruppensignatur Verfahren basiert auf Elliptische-Kurven-Kryptografie, was bedeutet, dass es von einem potentiellen zukünftigen Quantencomputer gebrochen werden könnte. Aus diesem Grund hat diese Arbeit auch ein Kapitel über den letzten Stand der Dinge von vor Quantencomputern sichere Gruppensignaturverfahren und geht tiefer in ihre Vor- und Nachteile, sowie deren Praktikalität auf modernen mobilen Endgeräten.

Acknowledgements

First of all, I would like to thank my friends and family, which enabled me to become who I am today. I would also like to give a special thanks to my parents and grandparents that supported me for the last few years. Special thanks also go to one of my supervisors Christian Hanser, who introduced me to this field and took the time to share his expansive knowledge with me. I would also like to mention Sebastian Ramacher, with whom I had many productive meetings that gave me a new sense of direction. Furthermore, I would thank Christian Rechberger for supporting the project and the insightful counseling and Daniel Kales for helping me with the finishing touches.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

Having privacy allows individuals to keep control of their own life. Personal data is used to determine whether we are investigated by the government, approved for a loan, or denied the ability to fly. Without having control over his data, an individual resigns control over a massive part of his life. Some personal data, such as location data, in the wrong hands, can even have adverse effects on the physical safety of individuals. Collections of personal data can also be used to infer additional information. Credit card companies, for instance, can infer whether an individual lost his job or not based on sudden spending changes.

Public perception of privacy is not the same as it was a decade ago. Over the last century, information technology has changed our way of life tremendously and has affected nearly every part of our daily lives. Possibly because of a lack of precedence, our society was mostly indifferent to companies and governments starting to collect vast amounts of data on customers and citizens [Wik20b]. With time and with the help of impactful media coverage of privacy advocates such as Edward Snowden, the importance of privacy in public perception has changed. In light of this, it is worth taking another look at the technologies we started using every day without taking a second look at it.

Digital payments are a double-edged sword. Every day, we are getting closer to a cashless society, allowing for faster and more convenient transactions, both in-store and online. This development leads to an ever more transparent consumer and potentially poses a huge privacy risk. In contrast, cash is practically untraceable and is inherently privacy-preserving. However, cash has its problems, namely money laundering and funding of organized crime. The untraceability of it makes it an attractive option for illegal activities.

A completely anonymous, untraceable digital payment system is not practical either. To make the funding of crime more complicated, the European Union (EU) started to phase out the 500€ banknote. An untraceable digital payment system that combines all the convenience of digital payments with the untraceability of cash would be a nightmare for the authorities, it would also pave the way for theft and fraud, without any way for the consumer to even know where his money went, let alone getting it back.

This thesis proposes a payment system that tries to combine the convenience of digital payments with the privacy-preserving aspects of cash while still maintaining full traceability for a third party (i.e., legal authorities). A crucial observation is that the transaction only needs to be traceable for a third party, which does not necessarily include the parties

directly involved in the transaction. The main principle behind the system is the "need to know" principle, meaning that for each interaction between each of the parties involved, we first look at the information that is necessary to complete the interaction and have it be traceable by a third party. The payment system is designed based on the findings of this process.

A reoccurring example for the payment system will be that of a transaction at a coffee shop. In the trivial case of a cash transaction, the shop does not need any personal information, but this transaction is also not fully traceable. The information that the shop "needs" to complete a digital transaction is an assurance that it will get its money and a piece of information that a third party can tie to the customer identity. For a digital payment, it is not obvious how much information the customer has to give up in order to convince the store that it will get the money from the bank account of the customer. Typically such information includes personal information such as identity and credit card number.

To evaluate its performance and demonstrate its practicality, we implemented the proposed payment system using Android apps and a backend written in Java. We implemented the payment system in such a way that it emulates the functionality of other popular NFC based payment systems such as Apple Pay [App19] or Google Pay [Goo19b]. To complete a transaction, the customer will have to hold his phone over the phone of the employee. Alternatively, the shop could also have a conventional payment terminal.

The main building block of the payment system is a modern group signature scheme [DS18]. The implemented payment system also serves as a test of the practicality of modern group signature schemes not only in a desktop environment but also in a mobile one. Since the chosen group signature scheme is based on elliptic curve cryptography, a potential future quantum computer could break it. Because of this, this thesis also has a chapter on the present state of post-quantum secure group signature schemes and discusses their advantages and shortcomings.

1.1 Background

This section will give a brief introduction into some of the regulatory background regarding the handling of personal data, as those regulations are a significant driving factor of privacy-enhancing technology. We will then go on to give an insight into the kind of data that is processed in existing typical digital payment systems. As the concept of digital signatures and group signatures is essential for our payment system, we also include a section on the intuition behind group signatures.

1.1.1 General Data Protection Regulation

The General Data Protection Regulation (GDPR) [EU16] is a regulation in EU law concerning data protection and privacy for individuals in the EU and the European

Economic Area (EEA). It was put into place as a response to increased public awareness of privacy and is one of the most impactful laws in recent history. The GDPR comes with rights and responsibilities both for end-users as well as service providers. Entities that collect personal data are called data controllers. They are required implement technical to uphold data protection principles. Businesses that handle personal data must be designed from the ground up with the principles of data protection in mind and must use the highest-possible privacy settings by default, except after receiving explicit consent from the data subject. All processing of personal data needs to have lawful basis specified by the regulation. The responsibilities for the data controller are numerous: the controller must disclose any instances of data collection and state the lawful basis and purpose for the data processing. Additionally, they have to declare how long this data is being retained for and if it is being shared with third parties, whether they are in or outside of the EEA. Businesses must also report any data breaches within 72 hours if they have an adverse effect on user privacy.

Unlike EU directives, the GDPR is a regulation that is directly binding, applicable, and enforceable. Non-compliance may result in a fine of up to €20 million or up to 4% of the annual worldwide turnover, whichever is higher. This has resulted in fines in the area of tens of millions of dollars for some very prominent companies operating in Europe, such as Google, British Airways, or Marriott International [Wik20a].

Particularly interesting for the topic of this thesis is the fine issued to the payment service provider MisterTango in Lithuania [Lit20]. It essentially stated that the company was processing more personal data than is necessary for effecting of the payment, exactly the kind of problem that this thesis trying to solve.

Impact on Businesses

Businesses must carefully consider any collection of personal data because there are many steps for collecting data in compliance with the GDPR. It starts with formulating a legal basis for collecting the data, getting the proper consent from the data subject, storing, and processing the data safely. It ends with providing a publicly available interface for the data subject to request retrieval or deletion of his data. The data controller also has to make sure to accurately identify the person requesting this data retrieval to avoid yet another costly disaster. All of those steps also need to be verifiable, and each step could have business threatening consequences if not implemented correctly.

The use of privacy-enhancing technology, such as our payment system, is strongly encouraged. In many cases, the use of this technology allows for the same service functionality, without collecting any personal information. By design, the GDPR discourages businesses from collecting personal data and makes them consider alternatives ways of doing business. After all, the simplest way of staying GDPR compliant is to avoid collecting personal data in the first place.

1.1.2 Payment Systems

In a conventional payment system, the customer reveals lots of personal information. In such a system, the customer often uses a debit/credit card or a tokenized version of that card inside a smartphone. By providing the credit card details online or by paying with the card inside a shop, the customer reveals uniquely identifying information to potentially untrusted third parties. This data can be roughly categorized into personal and non-personal data.

- **Personal data:** cardholders name, card number, and in the case of online-shops CVC number and billing address.
- **Non-personal data:** date & time of transaction, amount of transaction, and location of merchant.

While non-personal data cannot be used to identify an individual, it can be used in conjunction with personally identifiable data to extrapolate data about location history or shopping habits. Some payment systems even reveal the customer's bank account balance to the service provider, for instance, when using an ATM that displays the available bank account balance.

Usages of payment data

There are many different usages for payment data [Reg20], some of which may be in the consumer's interest, while others may not. Payment service providers (PSPs) use this data for purposes such as:

- **Providing personalized payment services**
  - This allows PSPs to offer better-tailored products and services that increase customer satisfaction. Some PSPs, for instance, share this data with their customers in order for them to get a better insight into their spending behavior and save money.
- **Tailoring their products to customer behavior**
  - Companies can analyze the data to get an insight into consumer behavior and develop new products based on that.
- **Identifying cross-selling opportunities**
  - A PSPs could share the payment data with its mortgage or insurance division, which could then try to sell products to the customer.
- **Preparing and selling statistical reports to third parties**
  - A major PSP in the UK compiled a report on the number of customers using their card to pay for train fares.
- **Meeting regulatory responsibilities**

  - Depending on local laws, data can be used to prevent money laundering or tax fraud.
- **Fighting fraud**
  - Data can be used to predict and prevent fraudulent transactions.

As can be seen with the example of personalized payment services, sometimes giving up personal data also comes with benefits to the consumer. What our proposed payment system tries to achieve is providing an alternative for the privacy-minded consumer, who might not even want any personalized services.

1.1.3 Digital Signatures & Group Signatures

A digital signature is a digital equivalent to regular signatures. It is used to give the recipient of a message a strong reason to believe that the message was created by a known sender [Wik19c]. Additionally, a digital signature also ensures that the message has been altered in any way during transit. They are used to sign e-mails, contracts, software, or in any other case where it is vital to detect tampering or forgery.

In digital payment systems, any financial transaction is digitally signed by the sender to ensure the authenticity of the transaction. For example, a typical credit card has a chip inside of it that allows the card to sign transactions in such a way that these signatures are uniquely attributable to this card only. However, by design, this reveals identifying information about the signer like name or credit card number.

Group signatures are an evolution on top of regular digital signatures. In 1991, Chaum and van Heyst [CH91] defined them as a signature created by an individual of a group having the following properties:

1. Only members of the group can sign messages.
2. The receiver can verify that it is a valid group signature, but cannot discover which group member created it.
3. If necessary, the signature can be "opened" to reveal the person who signed the message.

The word group in this context is to be understood in a classical sense, as in a group of people. The later parts of this thesis will make heavy use of group theory, in which the word group is to be understood in a mathematical sense.

Group signatures are being used as a credential/authentication scheme in which one person proves that he belongs to a certain group by creating a signature on behalf of the group. Anyone can verify a group signature, usually with a public key, and verify that it indeed originates from a valid (anonymous) member of the group. The larger the group, the more anonymous this kind of signature becomes. A use case example would be a public transport ticketing system: The operator of the transport system can issue group memberships to

everyone that has purchased a ticket. At the ticket gate, it is possible to create a proof of purchase by creating a group signature (on a non-identifying message).

One of the main distinguishing features of group signatures is that there exists an entity that can "open" signatures and re-identify the individual member that created the signature. Depending on the group signature scheme, this entity does not necessarily have to be the same as the one that is able to add new group members. In the example of the public transport ticketing system, the operator is unable to track the behavior of individuals. However, for instance, in case of a court order, an authorized third party (i.e., police) can re-identify individual users and use this information for an investigation.

The example use case for group signatures that the original paper uses is about a company with multiple departments, each with its own printer. Each printer should only be used by staff from the appropriate department, but at the same time, the company wants privacy: the user's name should not be revealed. However, if someone misuses the printer and prints an excessive amount of documents, the director must be able to identify the culprit, in order to send him a bill.
In this scenario, the employees from each department form a group. Each printer is only allowed to print signed documents from the group of its department. The director can then, if necessary, take a look at the log of the printer and re-identify the origin of each document.

More use cases for group signatures can be found in a wide array of research fields. One can also find use cases in the use of autonomous vehicles, where the data exchanged between vehicles must be authentic and anonymous. In case of an accident, all participants could still be identifiable with the help of an authorized entity. More use cases would be access control, toll systems, parking systems, e-voting, e-bidding, trusted computing, or in the case of this thesis: payment systems.

1.2 Thesis Outline

We will first go over the necessary preliminaries, including, but not limited to, group theory, elliptic curve theory, bilinear groups, pairings, computational hardness assumptions, and group signatures. We will then be able to define our payment system and go into detail about its design. Following that, we explain the implementations that resulted from this thesis and analyze their performance and put them into relation to other existing solutions. Lastly, we will give an overview of post-quantum secure group signature alternatives, and discuss their viability for our use case.

Chapter 2: Preliminaries

In this chapter, we explain the building blocks that we use in our payment system. We start with an introduction into what is considered a "hard" problem under certain circumstances. We then continue with a few basic cryptographic building blocks and will then go more in-depth into group theory and the foundations of elliptic curve cryptography (ECC). Finally, we will introduce the cryptographic primitives that we use in our payment system. Those primitives include, among others, digital signature schemes (DS), public-key encryption (PKE), proofs of knowledge (PoK), and group signatures (GS). The definitions, if not marked otherwise, either come from general knowledge in the field of cryptography or have been loosely based on other research [Han16].

2.1 Cryptographic Complexity Theory

Cryptographic complexity theory refers to the classification of computational problems according to their inherent difficulty under different circumstances.

2.1.1 Hard Problems

A hard problem is one that cannot be solved efficiently, where efficiently usually means in polynomial time. There exist some computational problems that are widely believed to be hard to solve. Prominent examples for such problems include the integer factorization problem, the discrete logarithm problem, or the Boolean satisfiability problem.

When talking about computational security, we assume that a potential adversary is computationally bounded (which all real-life adversaries are), meaning that he is unable to solve hard problems when using large enough parameters.

The way to show that a cryptographic scheme is computationally secure is by basing it on another well-known fundamental mathematical problem that is widely believed to be hard to solve. By showing that breaking the cryptographic scheme would also break the underlying mathematical problem, it is possible to prove the security of a scheme.

2.1.2 Computational Models

The requirements for a cryptographic scheme to be considered secure can change on additional assumptions besides the existence of hard problems. Each so-called computational model has different assumptions under which a cryptographic scheme is considered to be secure. The most commonly used models are the random-oracle model (ROM), the common reference string model(CRS), and the standard model.

Standard Model

The standard model does not rely on any idealized scenario and instead only relies on the raw computational complexity assumptions. For this reason, it is the strongest and most desirable model.

ROM - Random-Oracle Model

Cryptographic schemes that rely on random oracles are said to be in the Random Oracle Model (ROM). They are secure under the assumption that such an oracle exists. A random oracle is a function $\mathcal{O}$: $\{0, 1\}^* \mapsto \{0, 1\}^\kappa$. It responds to every unique input m with a truly random response with length κ, if m has not been queried before, and with the same answer as the first query for m otherwise.

In practice, random oracles are usually replaced by a cryptographic hash function (1.1.3). There are some artificial examples in which the replacement of random Oracles with hash functions turns the scheme insecure [CGH98] [GR04], indicating that the security of schemes in the ROM is of heuristical nature only. However, a proof in the ROM is generally accepted as proof for the practical security of a protocol and protocols based on it are widely used in practice.

Generic Group Model

The generic group model (GMM) is an idealized cryptographic model in which the adversary only has access to a randomly chosen encoding of a group ("group" is to be understood in a mathematical sense here). The adversary is thus forced to perform group operations using a group operation oracle. This oracle takes two group elements as an input, performs the group operations, and returns the resulting group element. The encodings of the resulting group elements are chosen at random, and just like the oracle from the ROM, already queried elements are being returned consistently to previous queries.

The GMM suffers from similar shortcomings as the ROM, as there exist constructions that have provable security in the GMM, but are trivially insecure [Den02] once the encoding is replaced by an efficient encoding used in practice, such as those used by the finite field or

elliptic curve groups. This indicates that a GMM security analysis only gives a heuristical indication about the security of the system.

CRS - Common Reference String Model

The CRS model, also known as the registered public-key model, assumes that a trusted third party (TTP) correctly performs a setup and outputs a common reference string, to which every participating party has access. In practice, the public parameters of the protocol at hand are commonly referred to as the CRS. In other words, schemes in the CRS are only secure given that every party has access to the same correct public parameters, which is a strong assumption.

2.2 One-Way Functions & Trapdoor Functions

A one-way function (OWF) is a function f: $\{0,1\}^* \rightarrow \{0,1\}^*$ that is easy to compute on every input, but it is hard to find the original input for a given output.

A trapdoor function is an OWF that becomes easy to invert when given some additional information, a secret key, for instance.

Easy and hard here are to be understood in the sense of computational complexity. OWFs are an essential building block of modern cryptography, yet their existence is still an open conjecture in theoretical computer science. In practice, the terms easy and hard take on the meaning of something along the lines of "cheap enough for legitimate users" and "prohibitively expensive for any malicious agents".

2.3 Hash Functions

A hash function is a function that maps an input of arbitrary length to a bitstring of fixed size κ, typically called *hash value* or *message digest*: $\{0,1\}^* \rightarrow \{0,1\}^\kappa$. Hash functions that are suitable for use in cryptography are called cryptographic hash functions. They are defined as functions $\mathsf{hash}(m)$ having the following properties [Wik19b]:

1. **Pre-image resistance:** Given a hash value h, it is hard to find a message m such that $\mathsf{hash}(m) = h$.
2. **Second pre-image resistance:** Given a message $m1$, it is hard to find a message $m2$ such that $\mathsf{hash}(m1) = \mathsf{hash}(m2)$.
3. **Collision resistance:** It is hard to find two messages $m1$ and $m2$ such that $\mathsf{hash}(m1) = \mathsf{hash}(m2)$.

Particularly proving collision resistance is problematic in theory because there is always a constant time algorithm that creates a collision: the one that creates a collision with hardcoded inputs. Hash functions in practice have a virtually limitless input length while at the same time having a fixed output length. This directly implies that it is impossible for them to fulfill any theoretical definition of collision resistance.

To formally define collision resistance in practice, one can imagine a probabilistic polynomial time (PPT) bounded Adversary $\mathcal{A}$ that is trying to find two sets of inputs x and x' that map to the same output. If the function is collision-resistant, then there is a negligible function $\epsilon(\cdot)$ such that:

$$\Pr\left[\ (x,\, x') \xleftarrow{R} \mathcal{A}(\mathsf{hash}(\cdot)) \ :\ \mathsf{hash}(x) = \mathsf{hash}(x')\ \right] \leq \epsilon(\kappa)$$

When we consider a hash function to be collision resistant in practice, we mean that it is hard to find one. For example, the cryptocurrency mining network behind bitcoin uses SHA-256 and had a period where it calculated over 100 quintillion ($10^{20} \approx 2^{66}$) hashes every second [Bit19], yet not a single collision for SHA256 has ever been found. Many different hash functions have withstood decades of intense scrutiny and are widely used in practice.

2.4 Groups

In mathematics, a group G is a set with a binary operation ($\circ$) that combines any two elements of the group to form a third element, while satisfying the four group axioms [Wik19e]:

1. **Closure:** $\forall a, b \in G$: $a \circ b$ is also in the group
2. **Associativity:** $\forall a, b, c \in G$: (a $\circ$ b) $\circ$ c = a $\circ$ (b $\circ$ c)
3. **Identitiy Element:** $\exists$e such that $\forall a \in G$: e $\circ$ a = a $\circ$ e = a
4. **Inverse Element:** $\forall a \in G$: $\exists a^{-1}$ such that $a \circ a^{-1} = a^{-1} \circ a = e$, where e is the identitiy element

Abelian groups are groups that are also commutative:

5. **Commutative:** $\forall a, b \in G$: $a \circ b = b \circ a$

For Abelian groups, there are two main notational conventions:

Convention	Operation	Identity	Powers	Inverse
Additive	x + y	0	n $\cdot$ x or nx	-x
Multiplicative	x $\cdot$ y or xy	e or 1	x^n	x^{-1}

Table 2.1: Notational conventions for abelian groups [Wik19a].

Additive groups are groups where the group operation is called addition and is denoted "+", whereas the group operation in multiplicative groups is called multiplication and is denoted by a "$\cdot$".

The literature uses both multiplicative notation and additive notation, depending on the context. In this thesis, we will mostly deal with additive abelian groups and will stick to additive notation for most of the time. However, during the explanation of some of the preliminaries, the operation "$\cdot$" might be ambiguous, which is why we sometimes use multiplicative notation.

An example of a group would be the set of integers $\mathbb{Z}_8 = \{0,...,7\}$ under the group operation of addition modulo 8 since it fulfills all group axioms. Following are a few examples to give an intuition to the group axioms in the case of this group.

1. **Closure:** $1 + 2 = 3 \in \mathbb{Z}_8$
2. **Associativity:** $(3 + 4) + 5 = 3 + (4 + 5) = 4 \pmod 8$
3. **Identitiy Element:** The identitiy element for this group is 0.
4. **Inverse Element:** The inverse element of every member in this group is: 8 - a. Example: $3 + (8 - 3) = 3 + 5 = 0 \pmod 8$

2.4.1 Cyclic Groups

A cyclic group is a group that is generated by a generator. A generator g is an element of the group from which every other element of the group can be obtained by repeatedly applying the group operation to itself. The set of elements generated by an element g in a cyclic group is denoted as $\langle g \rangle$.

Order of a Cyclic Group

In the context of cyclic groups, the concept of an order can mean one of two things:

- **Order of a group:** The order of a (cyclic) group is the number of elements in it. It is denoted by |G|.
- **Order of a group element:** The order of a group element g in a cyclic group is the smallest positive integer x such that $g^x = e$ with e being the identity element.

Not every element of a cyclic group necessarily generates the whole group. An element $a \in \mathbb{G}$ of order $x <$ |G| will generate a subgroup of order x, since by definition $a^x = e$.

Lagrange's theorem states that for every group G, the order of a subgroup H of G is a divisor of the order of G. From this, one can deduce that in a cyclic group of prime order every element ($\neq e$) generates the whole group.

2.5 Elliptic Curve Cryptography

Elliptic curve cryptography (ECC) is a public-key cryptography system based on the algebraic structure of elliptic curves over finite fields. It is widely used in technologies like TLS [Res18], PGP [Zim19], SSH [YL06], and in many other systems that make up the modern IT-world as we know it. Bitcoin [Nak+08] and other cryptocurrencies also make heavy use of it. One of the reasons for the popularity of ECC is that it provides dramatically smaller key sizes and more efficient computation than alternatives while providing the same amount of security. This section is intended to be an introduction to the basics of ECC, and its content and illustrations are based on a wonderful blog-post by Andrea Corbellini [And19a; And19b; And19c; And19d].

For the purposes of this thesis, we work with elliptic curve cryptography that uses points inside a finite field that satisfy the following equation:

$$y^2 = x^3 + ax + b$$

where $4a^3 + 27b^2 \neq 0$ (to exclude singular curves). This is called the Weierstrass normal form for elliptic curves. Depending on the choice of a and b they have different forms and shapes, as can be seen in figure 2.1.

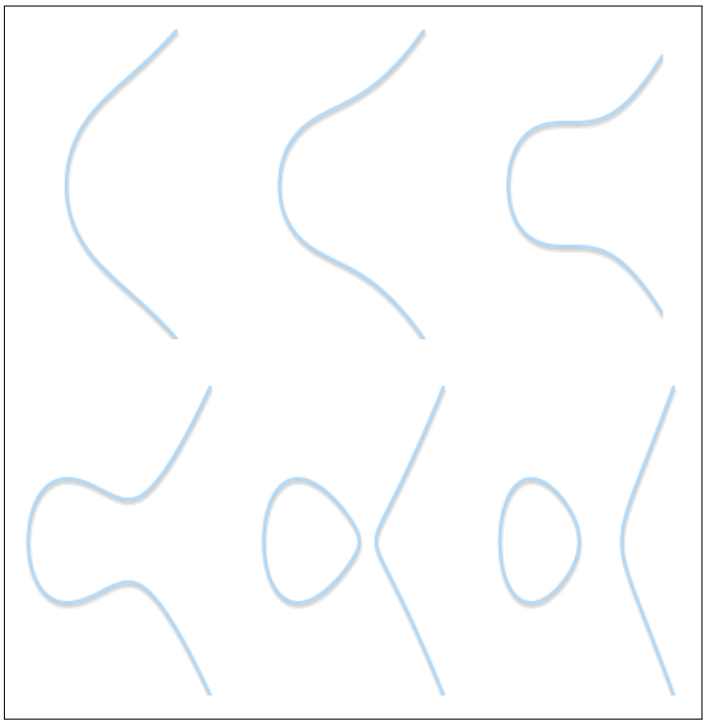


Figure 2.1: Different elliptic curves, with $\mathbf{b = 1}$ and $\mathbf{a = 2}$ **to -3**.
By Andrea Corbellini [And19a] licensed under CC BY 4.0.

2.5.1 Group Law for Elliptic Curves

The points defined by an elliptic curve can be used to define a group. To be able to do this, we need to add the point at infinity to the definition of the curve. From now on, we will call this point 0.

With this point, we can redefine our definition of an elliptic curve to be as follows:

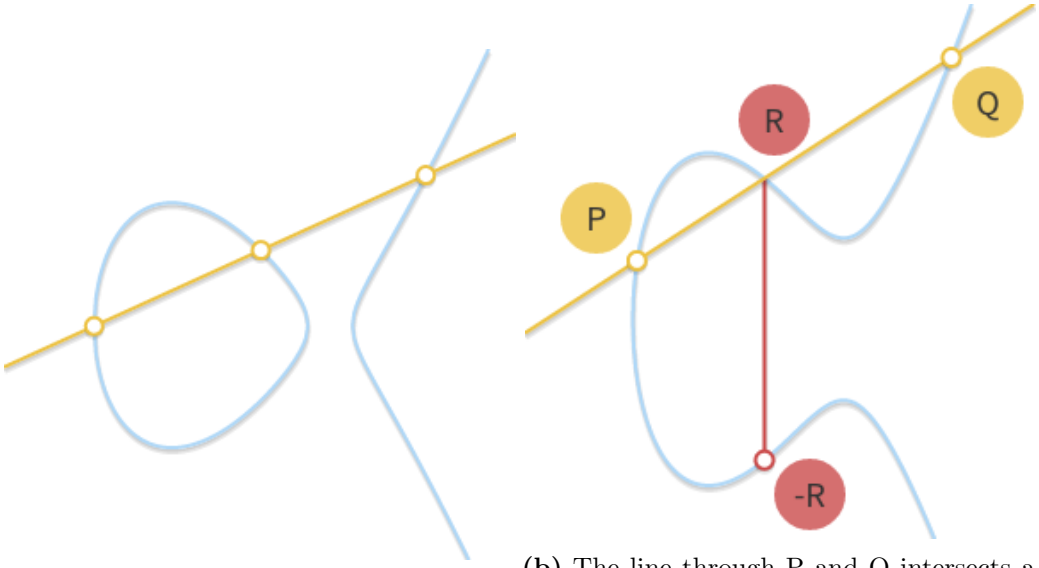
$$\{(x, y) \in \mathbb{R}^2 \mid y^2 = x^3 + ax + b, 4a^3 + 27b^2 \neq 0\} \cup \{0\}$$

Now we can define a group structure of elliptic curves:

- The elements of the group are the points on the curve.
- The identity element is the point at infinity 0.
- The inverse of a point P = (x, y) is the one symmetric about the x-axis $\mathrm{P}^{-1} = (x, -y)$.
- The group operation is defined to be addition and is given by the following rule: given three aligned (on one line), non-zero points **P**, **Q**, **R**, their sum is **P + Q + R = 0.**

Our definition of addition only requires 3 points to be aligned, without respect to order. This implies that P + (Q + R) = Q + (P + R) = R + (P + Q) = ... = 0, meaning that our group operation is both associative and commutative. Our group is therefore an abelian group.

Since we are in an abelian group, we can rewrite our group operation to be **P + Q = -R** (the inverse of R)(2.2).



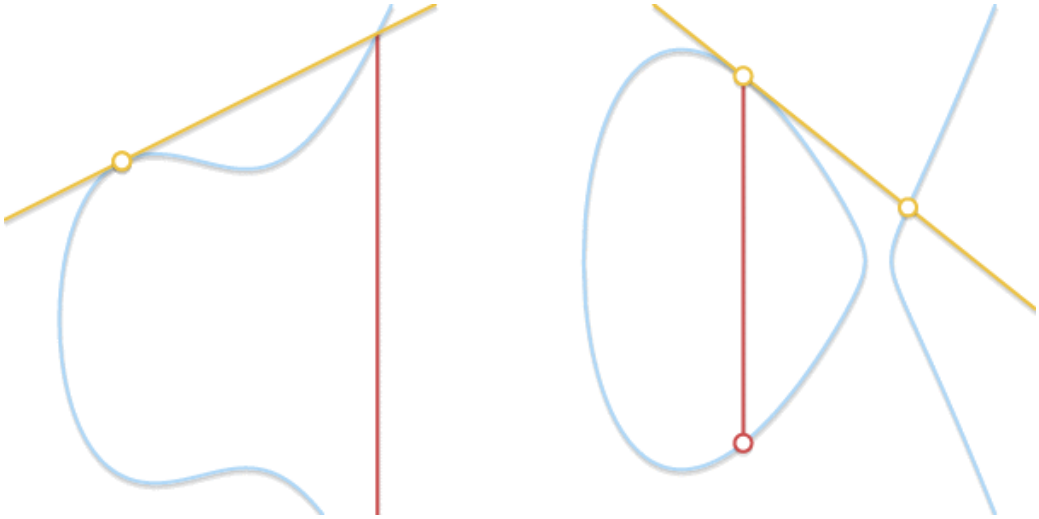
(a) The sum of three aligned points is 0.

(b) The line through P and Q intersects a third point R. The point symmetric to it (-R) is the result of P + Q.

Figure 2.2: ECC group operation.
By Andrea Corbellini [And19a] licensed under CC BY 4.0.

There are a few corner cases for this geometric group operation definition:

- **What if P = 0?** 0 is defined as identity element, therefore P + 0 = 0.
- **What if P = -Q?** If P is the inverse of Q, then P + Q = 0, by the definition of inverse.
- **What if P = Q?** If we think about it geometrically, the closer Q is to P on the curve, the more the line between P and Q will start to look like the tangent to the curve. This is why P + P is defined as -R, where R is the point of intersection between the tangent of P and the curve.
- **What if P ≠ Q but there is no third point R?** In this case, the line between two points is a tangent to the curve. If P is the tangency point of the line from P to Q, then P + Q = -P.



(a) Doubling of point P is defined as -R, where R is the point of intersection between the tangent of P and the curve.

(b) If the line between two points intersects just two points, it is tangent to the curve.

Figure 2.3: Corner-cases for the geometric definition of the elliptic curve group operation. By Andrea Corbellini [And19a] licensed under CC BY 4.0.

2.5.2 Finite Fields

A finite field is a finite set of elements that satisfy certain basic rules. An example of a finite field would be the set of integers modulo **p**, where p is a prime number. It is denoted in many ways, such as $\mathbb{Z}/p\mathbb{Z}$, GF(p), $\mathbb{Z}_p$ or $\mathbb{F}_p$. We will use the latter notation.

The properties of a field are as follows [Wik19d]:

- **Associativity of addition and multiplication:** $\forall a, b, c \in \mathbb{F}$: $(a+b)+c = a+(b+c)$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- **Commutative:** $\forall a, b \in \mathbb{F}$: $a + b = b + a$ and $a \cdot b = b \cdot a$
- **Additive and Multiplicative Identitiy Element:** $\exists$ e^+ and $e^\cdot$ such that $\forall a \in \mathbb{F}$: $e^+ + a = a + e^+ = a$ and $e^\cdot \cdot a = a \cdot e^\cdot = a$
- **Additive Inverse Element:** $\forall a \in \mathbb{F} : \exists(-a)$ such that $a + (-a) = (-a) + a = e^+$.
- **Multiplicative Inverse Element:** $\forall a \neq 0 \in \mathbb{F} : \exists\, a^{-1}$ such that $a \cdot a^{-1} = a^{-1} \cdot a = e^\cdot$
- **Distributivity:** of multiplication over addition: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.

In our case, we will mostly work with the field that is the set of integers mod p, where p is prime, with its notation: $\mathbb{F}_p$.

2.5.3 Elliptic Curves in Finite Fields

This section is based on the second part of Andrea Corbellini's [And19b] blog-series.

In ECC we restrict the set of elements of an elliptic curve over a field $\mathbb{F}_p$. With this, we can redefine our previous definition of elliptic curves to:

$$\{(x, y) \in (\mathbb{F}_p)^2 \mid y^2 = x^3 + ax + b \pmod{\text{p}}, 4a^3 + 27b^2 \not\equiv 0 \pmod{\text{p}}\} \cup \{0\}$$

where 0 is still the point at infinity, and a and b are integers $\in \mathbb{F}_p$.

Whereas before we had a continuous curve, we now have a set of disjoint points in the xy-plane. Still, it is proven that elliptic curves in $\mathbb{F}_p$ form an abelian group [Fri17].

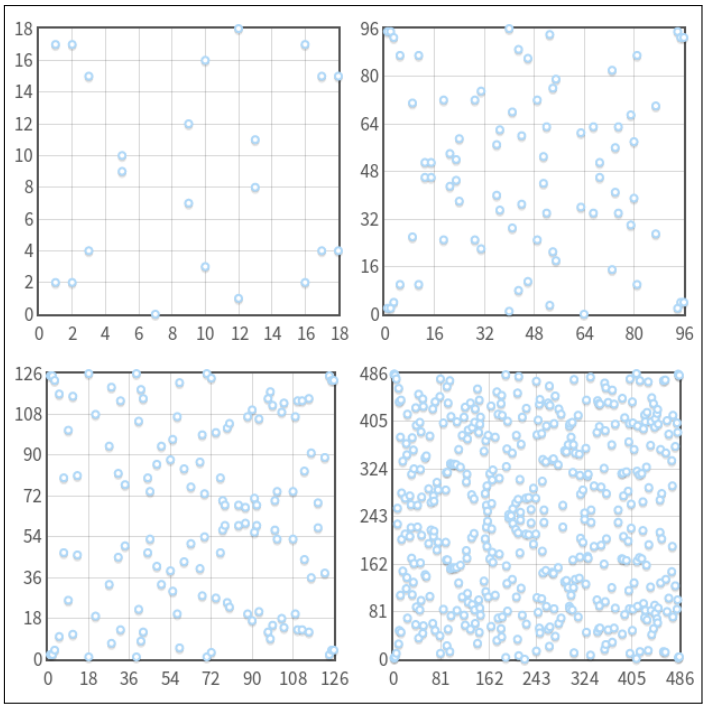


Figure 2.4: The curve $y^2 \equiv x^3 - 7x + 10 \pmod{\text{p}}$ with p = 19,97, 127, 487. Note that for every x, there are at most two points. Note the symmetry around y = p/2.
By Andrea Corbellini [And19b] licensed under CC BY 4.0.

Our definition of addition (3 aligned points, P + Q + R = 0) still holds. Three points are aligned if there is a line that connects them. However, lines in $\mathbb{F}_p$ are slightly different from the ones in $\mathbb{R}$. The new definition for a line is:

$$ax + by + c \equiv 0 \ (mod\ p)$$

which is the standard line equation, with the addition of **mod p**.

All of our group-arithmetic still works in this setting:

- P + 0 = 0 + P = P
- The inverse of P = (x, y) is still $P^{-1} = (x, -y)$, but -x will be taken mod p. (for instance $-3 \equiv 2 \pmod 5$)
- P + (-P) = 0, by the definition of an inverse

The geometric representation of calculating group operations gets a little bit more complicated in this setting and would not be practical for the purpose of demonstration. Nevertheless, the algebraic version of these operations can be directly applied to this setting by working (mod p).

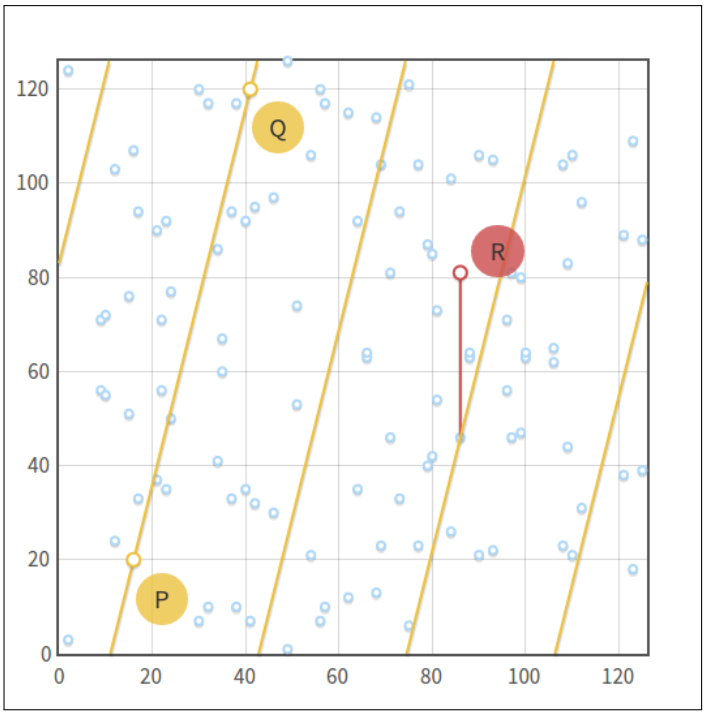


Figure 2.5: Point addition over the curve $y^2 \equiv x^3 - x + 3 \pmod{127}$, with P = (16,20) and Q = (41, 120). Note how the line $y \equiv 4x + 83 \pmod{127}$ that connects the points "repeats" itself on the plane.
By Andrea Corbellini [And19b] licensed under CC BY 4.0.

2.5.4 Order of an Elliptic Curve Group

The order of an elliptic curve group is the number of points defined by it. We defined elliptic curves over a finite field, which has a finite number of points. With that said, we need to find out how many points there are exactly. There are different methods to get

to this number, but for our purposes, it is sufficient to know that there exists an efficient algorithm that gives us this number, Schoof's algorithm [Sch85].

2.5.5 Scalar Multiplication and Cyclic Subgroups

This section is based on the second part of Andrea Corbellini's [And19b] blog-series.

A scalar multiplication operation can be defined by using the addition operation and adding a number n times to itself: $n \cdot P = P + P + ... + P$. In practice, this can be achieved efficiently by applying a double and add algorithm. Doubling a number can be done by adding it to itself. This reduces the number of operations to $\mathcal{O}$ (log n).

Scalar multiplication of points for elliptic curves in $\mathbb{F}_p$ has the property to create cyclic subgroups. Take for instance for the curve $y^2 \equiv x^3 + 2x + 3 \mod (97)$ and the point P = (3,6):

- 0P = $\underline{0}$
- 1P = (3,6)
- 2P = (80,10)
- 3P = (80,87)
- 4P = (3, 91)
- 5P = $\underline{0}$
- 6P = (3,6)
- 7P = (80,10)
- 8P = (80,87)
- 9P = (3, 91)
- ...

We see that the points start to repeat themselves and there are only 5 distinct multiples of P. They are repeating cyclically, meaning that for every integer k we can write:

- 5kP = 0
- (5k + 1) P = P
- (5k + 2) P = 2P
- (5k + 3) P = 3P
- (5k + 4) P = 4P

An even shorter notation would be: kP = (k mod 5)P

It can be proven that the set of multiples of P is a cyclic subgroup of the group formed by the elliptic curve. A subgroup is a group, which is a subset of another group. A cyclic subgroup contains a generator element that produces all elements of the subgroup when repeatedly applying the group operation to itself.

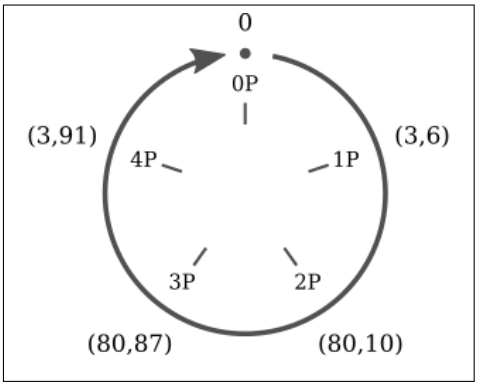


Figure 2.6: Multiples of P = (3,6) are just five distinct points and they are repeating cyclically. It is easy to spot the similarity between scalar multiplication on elliptic curves and addition in modular arithmetic.
By Andrea Corbellini [And19b] licensed under CC BY 4.0.

2.5.6 Subgroup Order and Finding a Generator

A question that arises is how to find the order(number of elements) of a subgroup generated by P (or equivalently, what is the order of P). The order of a group element P is the smallest integer x such that xP = 0. At this point, Lagrange's theorem is very useful. It states that for any finite group, **the order of a subgroup is a divisor of the order of parent group**.

This information leads us to the following algorithm:

1. Calculate order N of elliptic curve using Schoof's algorithm.
2. Factor N
3. For every divisor n of N, compute nP
4. The smallest n such that nP = 0 is the order of the subgroup.

Another consequence of Lagrange's theorem is, that if the order of a group is prime, there are no possible subgroups, since their order would have to divide the order of the original group.

From the previous section, we know that Schoof's algorithm is efficient. Looking at this algorithm, the obvious question that arises is how to factor N. We will give an explanation once we look at how the results of this algorithm are actually used.

A **cofactor** of a subgroup is the number h = N/n. As yet another consequence of Lagrange's theorem this number is always integer. Since N is a multiple of n we know that $\mathbf{N \cdot P = 0}$. Combining these two definitions leads to: $\mathbf{n(hP) = 0}$

This equation tells us, that a point G = hP generates a subgroup of order n (except when G = hP = 0).

In light of this, we can outline the following algorithm to find a generator for a group of order n:

1. Calculate the order N of the elliptic curve
2. Choose the order n (must be prime) of the subgroup from one of the divisors of N
3. Computer the cofactor h = N/n
4. Choose a random point P on the curve
5. Compute G = hP
6. if G = 0, go back to 4. Else we have found a generator of a subgroup of order n and cofactor h.

If n was not prime, the order of G could be one of the divisors of n.

In the algorithm for finding the order n of a subgroup, we factor the order of the whole elliptic curve group. Factoring large integer numbers efficiently, in general, is considered to be a hard problem, but our algorithm can still be efficient since we never have to factor general integers fully. ECC algorithms work in the subgroup that our generator generates, and we want this group to be only trivially smaller than the original group and have prime order. Our approach when factoring N is removing a few small factors and checking if the remainder is prime. If not, our subgroup is going to be too small, and we should start looking for new curve parameters.

2.5.7 Discrete Logarithm

The security of elliptic curves comes from the hardness of calculating a discrete logarithm (DLP). Given two points P and Q, the DLP is the problem of finding a scalar k, such that Q = kP. In terms of computational complexity, it is believed to be a hard problem.

It is called the elliptic-curve discrete logarithm because it is analogous to the discrete logarithm problem used in other cryptosystems. The original Diffie-Hellman key exchange protocol, for instance, uses the multiplicative group of integers modulo p, with a large prime p (also called $(\mathbb{Z}/p\mathbb{Z})^x$) instead of elliptic curves. In this setting, the discrete logarithm problem can be stated as follows: given the integers a,b; what is the integer k such that: $b \equiv a^k \mod p$?

Both of these problems are called discrete because they involve finite sets, and they are called logarithms because they are similar to regular logarithms.

The advantage that elliptic curve groups have over other groups like the $(\mathbb{Z}/p\mathbb{Z})^x$ is that the existing algorithms for calculating the discrete logarithm in this group are a lot less efficient. This allows for a smaller security parameter κ than in comparable cryptosystems with a similar level of security, leading to smaller keys and signatures and more efficient computation.

2.5.8 Random Curves

There exist special classes of "weak" or "insecure" curves for which an efficient algorithm for the discrete logarithm problem exists. One example of that would be curves where the order of the underlying finite field is the same as the order of the elliptic curve group. These curves are vulnerable to Smart's attack [Sma99], which has polynomial run-time on a classical computer.

Since there are known examples of curves that are "weak", people are rightfully suspicious if an untrusted party publishes curve parameters that everyone is supposed to use. After all, nobody can be sure if the publisher of those parameters might be aware of a secret algorithm that works specifically on this particular curve.

This is the reason why some descriptions of curves contain a seed value as one of their parameters. The argument behind this value is that **the curve parameters a and b are based on a cryptographic hash of this seed.** Since, by the definition of a cryptographic hash function, the output of a hash is supposed to be unpredictable (for different inputs), the curve parameters are also pseudo-random.

This trick should give some assurance that the curve has not been specifically designed to expose vulnerabilities known to the publisher. It is designed to show that the author was not free to arbitrarily choose the curve parameters a and b.

One question that remains is the selection of the seeds for these hash function. Some cryptographic systems use so-called "nothing up my sleeve numbers". Examples for that would be the first digits of pi, e, or irrational roots. However, the NIST standard, for instance, uses a seed of unexplained origin, leaving room for speculations about possible manipulations (the standard was created in conjunction with the NSA). However, even seeds based on nothing up my sleeve numbers can be used to generate potentially manipulated curves [Ber+15].

There exist many different standards, each with their own security guarantees. Also, there are different resources on the internet that compile resources on "safe" elliptic curve standards [BL19].

2.5.9 Domain Parameters

ECC algorithms work in a cyclic subgroup of an elliptic curve over a finite field. The curve description for these algorithms usually contains the following domain parameters:

- The prime **p** that specifies the size of the finite field.
- The coefficients **a** and **b** of the elliptic curve equation.
- The base point **G** that generates our subgroup.
- The order **n** of the subgroup.
- The cofactor **h** of the subgroup.

- Optional: The Seed to verify the randomness of the curve parameters & instructions on how to generate them.

As an example, here is the description of a NIST recommended curve (**cofactor** for this curve is **1**, **a** is always **-3**, these curve parameters come with an instruction on how to use the seed together with a hash function to generate b, and c is one of the results of this instructions.)

```
Curve P-256
p =
115792089210356248762697446949407573530086143415290314195533631308867097853951
n =
115792089210356248762697446949407573529996955224135760342422259061068512044369
SEED =  c49d3608 86e70493 6a6678e1 139d26b7 819f7e90
c =  7efba166 2985be94 03cb055c 75d4f7e0 ce8d84a9 c5114abc af317768 0104fa0d
b =  5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b
G_x =  6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945 d898c296
G_y =  4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece cbb64068 37bf51f5
```

2.6 Bilinear Groups

A bilinear map (also called pairing) maps elements from two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ to a target group $\mathbb{G}_T$. The two source groups are additive groups defined by two elliptic curve equation over a base field $\mathbb{F}$, and the target group is a multiplicative group over and extension field of $\mathbb{F}$. The elliptic curves equations don't have to be the same, but they need to have the same order. In practice they are often closely related. To better differentiate elements from the two groups, elements from $\mathbb{G}_2$ will be denoted by a hat ˆ (e.g. $\hat{P}$). Following this is a more formal definition of a bilinear group [Han16].

Let ($\mathbb{G}_1$,+), ($\mathbb{G}_2$,+), generated by P and $\hat{P}$, respectively, and ($\mathbb{G}_T$,·) be cyclic groups of prime order p. We call e : $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ a bilinear map or pairing if it is efficiently computable and the following properties hold:

1. **Bilinearity:** $e(aP, b\hat{P}) = e(P, \hat{P})^{ab} = e(bP, a\hat{P})\ \forall a, b \in \mathbb{Z}_p$.
2. **Non-degeneracy:** $e(P, \hat{P}) \neq 1_{\mathbb{G}_T}$; meaning $e(P, \hat{P})$ generates $\mathbb{G}_T$.

We distinguish between three types of pairings. If $\mathbb{G}_1 = \mathbb{G}_2$, then e is said to be a Type-1 (symmetric) pairing. The distinguishing factor between Type-2 and Type-3 pairings is the existence of an efficiently computable isomorphism ψ: $\mathbb{G}_2 \mapsto \mathbb{G}_1$, for Type-2 pairings. Such an isomorphism is not known for Type-3 pairings.

For our purposes, we will only work with Type-3 pairings using Barreto-Naehrig curves. In the pairings that we use, all groups have the same prime order p, meaning that $|\mathbb{G}_1|$ = $|\mathbb{G}_2|$ = $|\mathbb{G}_T|$ = p. This allows us to define a billinear group in a rather abstract way

by combining all related entities into a billinear group description BG = (p, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, e, P, $\hat{P}$). For Barreto-Naehrig curves, this allows us to define the following deterministic algorithm [Han16]:

BGGen(1^κ): An efficiently computable algorithm that takes a security parameter 1^κ and outputs a bilinear group BG = (p, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, e, P, $\hat{P}$) consisting of three prime-order p groups $\mathbb{G}_1 = \langle P \rangle$, $\mathbb{G}_2 = \langle \hat{P} \rangle$, and $\mathbb{G}_T$ with $\log_2 \text{p} = \lceil \kappa \rceil$ and a pairing e: $\mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$.

2.7 Hardness Assumptions in Bilinear Groups

In this section, we discuss some of the common complexity assumptions in the context of Type-3 bilinear pairings. We will begin each section by giving a formal definition, followed by a more verbose and intuitive explanation.

2.7.1 Discrete Logarithm Assumption - (DL)

Given a bilinear group generator BGGen that outputs BG = (p,$\mathbb{G}_1$,$\mathbb{G}_2$,$\mathbb{G}_T$,e,P,$\hat{P}$). The DL assumption holds for BGGen in $\mathbb{G}_i$ if for every probabilistic-polynomial time (PPT) Adversary $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that [DS18]:

$$\Pr\left[\ \text{BG} \xleftarrow{R} \text{BGGen}(1^\kappa),\ a \xleftarrow{R} \mathbb{Z}_p,\ a' \xleftarrow{R} \mathcal{A}(BG, aP)\ :\ a'P = aP\ \right] \leq \epsilon(\kappa)$$

In other words, given BG and aP it is hard to compute a. This assumption is the weakest assumption in bilinear-group-based cryptography, meaning it is the hardest problem. An efficient solution to this problem would enable us to solve any other problem defined in this section.

2.7.2 Computational Diffie-Hellman Assumption - (CDH)

Given a bilinear group generator BGGen that outputs BG = (p,$\mathbb{G}_1$,$\mathbb{G}_2$,$\mathbb{G}_T$,e,P,$\hat{P}$). The DL assumption holds for BGGen in $\mathbb{G}_i$ if for every PPT Adversary $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that [DS18]:

$$\Pr\left[\ \text{BG} \xleftarrow{R} \text{BGGen}(1^\kappa),\ \text{r,s} \xleftarrow{R} \mathbb{Z}_p,\ T \xleftarrow{R} \mathcal{A}(BG, rP, sP)\ :\ T = rsP\ \right] \leq \epsilon(\kappa)$$

Given BG, rP, and sP it is hard to compute rsP. This assumption turns out to be equivalent to the following Diffie-Hellman inversion assumption [BDZ03].

2.7.3 Diffie-Hellman Inversion Assumption - (DHI)

Given a bilinear group generator BGGen that outputs BG = (p,$\mathbb{G}_1$,$\mathbb{G}_2$,$\mathbb{G}_T$,e,P,$\hat{P}$). The DL assumption holds for BGGen in $\mathbb{G}_i$ if for every PPT Adversary $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that [DS18]:

$$\Pr\left[\ \text{BG} \xleftarrow{R} \text{BGGen}(1^\kappa),\ \text{a} \xleftarrow{R} \mathbb{Z}_p,\ T \xleftarrow{R} \mathcal{A}(BG, aP)\ :\ T = \tfrac{1}{a}P\ \right] \leq \epsilon(\kappa)$$

Alternatively, given BG and aP it is hard to find $\frac{1}{a}$. The following co-DHI also implies and encompasses this assumption.

2.7.4 Co-Diffie-Hellman Inversion Assumption - (co-DHI)

Given a bilinear group generator BGGen that outputs BG = (p,$\mathbb{G}_1$,$\mathbb{G}_2$,$\mathbb{G}_T$,e,P,$\hat{P}$). The DL assumption holds for BGGen in $\mathbb{G}_1$if for every PPT Adversary $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that [DS18]:

$$\Pr\left[\begin{array}{c} \text{BG} \leftarrow \text{BGGen}(1^\kappa),\ \text{a} \xleftarrow{R} \mathbb{Z}_p, \\ T \xleftarrow{R} \mathcal{A}(BG, aP, a\hat{P}) \end{array} : \begin{array}{c} T \in \mathbb{G}_1 \\ \wedge\ \text{e}(T, a\hat{P}) = \text{e}(P, \hat{P}) \end{array}\right] \leq \epsilon(\kappa)$$

Since the only way that e(T, $a\hat{P}$) can be equal to e($p, \hat{P}$) is for T to be $\in \mathbb{G}_1$ and to be comprised of T = $\frac{1}{a}P$, this assumption means that given BG and aP **and** $a\hat{P}$ it is hard to find $\frac{1}{a}$. This assumption can be defined similarly for $\mathbb{G}_2$. What differentiates this assumption from DHI is the availability of the instance value "a" in both groups, making this assumption a stronger one.

2.7.5 Decisional Diffie-Hellman Assumption - (DDH)

Given a bilinear group generator BGGen that outputs BG = (p,$\mathbb{G}_1$,$\mathbb{G}_2$,$\mathbb{G}_T$,e,P,$\hat{P}$). The DL assumption holds for BGGen in $\mathbb{G}_1$if for every PPT Adversary $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that [DS18]:

$$\Pr\left[\begin{array}{c} \text{BG} \xleftarrow{R} \text{BGGen}(1^\kappa),\ \text{b} \xleftarrow{R} \{0,1\},\ \text{r,s,t} \xleftarrow{R} \mathbb{Z}_p, \\ b^* \xleftarrow{R} \mathcal{A}(BG, rP, sP, (b \cdot rs + (1-b) \cdot t)P) \end{array} :\ b^* = \text{b}\ \right] - \frac{1}{2} \leq \epsilon(\kappa)$$

Intuitively, given BG, rP, sP, and xP it is hard to differentiate whether $x = \text{r} \cdot \text{s}$ or x is random. Assuming that given r and P computing rP can be done efficiently, this implies that given P and rP it is hard to compute r.

2.7.6 (Symmetric) External Diffie-Hellman assumption - (XDH and SXDH)

(Definition from [DS18]) Let BG be a bilinear group generated by BGGen. Then the XDH assumption states that the DDH assumption holds in $\mathbb{G}_1$. Similarly, the SXDH assumption states that the DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

2.8 Public-Key Encryption

Public-Key cryptosystems use pairs of keys, a public and a private one, of which the public key can be distributed, and the private key remains secret under the control of the owner. Up until the 70s, only symmetric key algorithms were known. That changed with the advent of cryptosystems like RSA [RSA78], ElGamal [ElG85] encryption, and the Diffie-Hellman key exchange [DH76]. This section will give a generic model of a public-key encryption scheme [Han16]:

- **KeyGen(1^κ):** A probabilistic algorithm that takes a security parameter 1^κ as input. It returns a keypair (sk, pk).
- **Enc(m, pk):** A (probabilistic) algorithm that takes a plaintext m and a public key pk as an input. Returns the ciphertext c of m under pk
- **Dec(c, sk):** A deterministic algorithm that takes a ciphertext c and a secret key sk as an input. Returns the plaintext m of c under sk.

A public-key cryptosystem needs to be correct and must at least provide indistinguishability against chosen-plaintext attacks.

- **Correctness:** A PKE scheme is correct if for all security parameters κ, all choices of key pairs (sk,pk) $\xleftarrow{R}$ KeyGen(1^κ), and all messages the following holds:

$$\Pr\left[\ \text{Dec(Enc(m,pk),sk)} = \text{m}\ \right] = 1$$

- **Indistinguishability against chosen-plaintext attacks (IND-CPA):** A PKE scheme is called IND-CPA secure, if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that:

$$\Pr\left[\begin{array}{c} (\text{sk, pk}) \xleftarrow{R} \text{KeyGen}(1^\kappa), \\ (m_0, m_1) \leftarrow \mathcal{A}\ (\text{pk}), \\ \text{b} \xleftarrow{R} \{0,1\},\ \text{c} \leftarrow \text{Enc}(m_b, pk), \\ \text{b}^* \leftarrow \mathcal{A}(\text{c},\ m_1, m_2,\ \text{pk}) \end{array} : b^* = \text{b} \right] - \frac{1}{2} \leq \epsilon(\kappa)$$

In other words, an adversary sends two plaintexts to an oracle, which in turn responds with the ciphertext of one of the plaintexts. The adversary must not be able to determine to which plaintext this ciphertext belongs. The attacker is allowed to do as many encryption operations on arbitrary plaintexts, including the ones he submits to the oracle, at all times.

- **Indistinguishability against chosen-ciphertext attacks (IND-CCA):** This is an extension to the oracle game of IND-CPA. In this case, the adversary is also allowed to do any number of **decryptions** on arbitrary ciphertexts **before** he submits his plaintexts to the oracle.

- **Indistinguishability against adaptive-ciphertext attacks (IND-CCA2):** This is an extension to IND-CCA. In this case, the adversary is allowed to do any number of **decryptions** on arbitrary ciphertexts **at all times** (excluding the challenge ciphertext he receives from the oracle).

2.9 ElGamal Encryption

ElGamal encryption [ElG85] is a PKE system invented by Taher Elgamal. It uses a cyclic group $\mathbb{G} = \langle P \rangle$ with prime order p and a generator P. The message space are members of $\mathbb{G}$. It is secure, assuming that the decisional Diffie-Hellman problem (2.7.5) in the chosen group is hard.

- **KeyGen():**
  - $q \xleftarrow{R} \mathbb{Z}_p$
  - $sk \leftarrow q$
  - $pk \leftarrow qP$
  - return (sk, pk).
- **Enc(M, pk):**
  - $r \xleftarrow{R} \mathbb{Z}_p$
  - $\mathrm{C}_1 \leftarrow \mathrm{M} + \mathrm{r} \cdot pk$
  - $\mathrm{C}_2 \leftarrow rP$
  - return $\mathrm{C} = (\mathrm{C}_1, \mathrm{C}_2)$
- **Dec(C, sk):**
  - $\mathrm{M} \leftarrow \mathrm{C}_1 - sk \cdot \mathrm{C}_2$ (note: $\mathrm{C}_1 - sk \cdot \mathrm{C}_2 \equiv M + r \cdot qP - \mathrm{q} \cdot rP$)
  - return M

2.10 Commitments

A cryptographic commitment scheme allows someone to commit to a value/statement, with the ability to later open and verify the commitment. They are the digital analog to sealed envelopes.
A commitment scheme consists of the following algorithms:

- **Setup(1^κ):** Takes a security parameter 1^κ. Outputs public parameters pp
- **Commit(pp,m):** Outputs a tuple (C,O) consisting of the commitment C and the opening O
- **Open(pp, C, O)** Outputs either the message m or $\perp$ to indicate success or failure.

In such a commitment scheme, someone can commit to a statement by sending a third party the commitment C, without revealing the actual statement. At a later point, one can send the opening O to that same third party to prove the commitment.

A commitment scheme is secure if it is correct, hiding, and binding [Han16]:

- **Correctness:** A commitment scheme is correct if for all security parameters κ, all choices of public parameters pp $\xleftarrow{R}$ Setup(1^κ), all messages m it holds that:

$$\Pr\left[\ \text{Open(pp, Commit(pp,m))} = \text{m}\ \right] = 1$$

- **Binding:** A commitment scheme is binding, if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that:

$$\Pr\left[\begin{array}{c} \text{pp} \xleftarrow{R} \text{Setup}(1^\kappa), \\ (\text{C,O},O') \xleftarrow{R} \mathcal{A}(\text{pp}), \\ \text{m} \leftarrow \text{Open(pp,C,O)}, \\ m' \leftarrow \text{Open(pp,C,}O') \end{array} : \text{m} \neq m' \wedge m, m' \neq \perp \right] \leq \epsilon(\kappa)$$

- **Hiding:** A commitment scheme is hiding, if for all PPT adversaries there is a negligible function $\epsilon(\cdot)$ such that:

$$\Pr\left[\begin{array}{l} \text{pp} \xleftarrow{R} \text{Setup}(1^\kappa), \text{b} \xleftarrow{R} \{0,1\}, \\ (\text{st},m_0,m_1) \xleftarrow{R} \mathcal{A}(\text{pp}), \\ (\text{C}, \cdot) \xleftarrow{R} \text{Commit(pp},m_b), \\ \text{b}^* \xleftarrow{R} \mathcal{A}(\text{st,C}) \end{array} : \text{b} = \text{b}^* \right] - \frac{1}{2} \leq \epsilon(\kappa)$$

If the game for binding and hiding holds for unbounded adversaries and $\epsilon = 0$, they are called perfectly binding and perfectly hiding, respectively. A commitment scheme cannot be perfectly hiding and perfectly hiding at the same time: In order for a commitment to be perfectly hiding, it must be the case that two different messages can create the same commitment. However, for a scheme to be perfectly binding, there must only be one message that can create a particular commitment.

Popular examples for commitments are discrete logarithm (DL) commitments and Pedersen commitments [Ped91]. While the former are perfectly binding, they are only computationally hiding under the DL assumption. Pederson commitments are perfectly hiding, but only computationally binding under the DL assumption. In addition to that, any IND-CPA-secure encryption scheme yields a perfectly binding, computationally hiding commitment scheme.

2.10.1 Discrete Logarithm Commitments and Pedersen Commitments

Both DL commitments and Pedersen commitments use a cyclic group $\mathbb{G} = \langle P \rangle$ with prime order p and a generator P. The message space one commits to are scalars $\in \mathbb{Z}_p$.

DL Commitment

The DL commitment is one of the simplest commitment schemes. The public parameters are the cyclic group with its generators.

- **Commit(pp, c):** Given public parameters $pp = (\mathbb{G}, p, P)$ and a message $m \in \mathbb{Z}_p$: Output (C $\leftarrow m$P , O $\leftarrow m$)
- **Open(*pp*, C, O):** Given public parameters $pp = (\mathbb{G}, p, P)$, a commitment C $\in \mathbb{G}$, and an opening O $\in \mathbb{Z}_p$: Output O if: O $\cdot$ P = C; else output $\perp$.

It is easy to see how this is perfectly binding, as there is only one value that can create the commitment. However, an adversary that could defeat the DL assumption would be able to reveal the commitment by calculating the discrete logarithm of C, meaning it is only computationally hiding.

Pedersen Commitment

The Pedersen commitment uses an additional public parameter in the form of a second generator Q = q $\cdot$ P. It is generated by a trusted third party and q is supposed to be thrown away after the creation of Q.

- **Commit(pp, m):** Given public parameters $pp = (\mathbb{G}, p, P, Q)$ and a message m: $r \xleftarrow{R} \mathbb{Z}_p$, Output (C $\leftarrow m$P + rQ , O $\leftarrow$ (m,r))
- **Open(*pp*, C, O):** Given public parameters $pp = (\mathbb{G}, p, P, Q)$, a commitment C $\in \mathbb{G}$, and an opening O = (m, r): Output m if: $m \cdot$ P + $r \cdot$ Q = C; else output $\perp$.

The Pedersen commitment is perfectly hiding because there are a lot of combinations of commitment values that could have produced C. Even if an adversary were able to break the DL assumption he would not be able to pin down the original message. However, it is only binding assuming that the scalar q is unknown to the creator of the commitment. If the value q were to be known by calculating the discrete logarithm of Q, it would be

possible to create openings for different messages m for a given commitment C, even after the commitment has been created. This leads to the Pedersen commitment only being computationally binding.

2.11 Zero-Knowledge Proofs of Knowledge

Before we go into the definition of zero-knowledge proofs of knowledge, we first need to define an interactive proof system is.

2.11.1 Interactive Proof Systems

An interactive proof system, also known as an IPS $(\mathcal{P}, \mathcal{V})$ for a language L is an interactive protocol between an unrestricted prover $\mathcal{P}$ and a PPT verifier $\mathcal{V}$ that has the following properties [Han16]:

- **Completeness:** $\forall x \in L : \Pr\left[(\mathcal{P}, \mathcal{V})(x) = 1\right] = 1$.
- **Soundness:** $\forall x \notin L \ \forall\ \mathcal{P}^*$: $\Pr\left[(\mathcal{P}^*, \mathcal{V})(x) = 1\right] \leq \frac{1}{2}$.

$\mathcal{P}^*$ represents any malicious prover and $(\mathcal{P}, \mathcal{V})(x) = 1$ represents that $\mathcal{V}$ accepts the interaction with $\mathcal{P}$ on a common input x.

A more verbose definition of completeness would be that if the statement is true, an honest verifier can be convinced of this fact by an untrusted entity.
Soundness means that if the statement is false, no (potentially malicious) prover can convince an honest verifier of this fact.

2.11.2 Zero-Knowledge Proofs

In Zero-knowledge proofs (ZKP) [GMR89] a prover $\mathbb{P}$ interacts with a verifier $\mathbb{V}$ on some common input x and is able to convince the verifier of x being contained in some formal language L without the verifier learning anything beyond the validity of the proven statement. They are a form of IPS. Their formalization is as follows [Han16]:

- An IPS $(\mathcal{P}, \mathcal{V})$ for a language L is ZK if for any (potentially malicious) verifier $\mathbb{V}^*$, there exists a PPT algorithm $\mathcal{S}$ (a simulator) such that:

$$\{\mathcal{S}(x)\}_{x \in L} \approx \{\langle (\mathcal{P}, \mathcal{V}^*)(x) \rangle\}_{x \in L}.$$

$\langle(\mathcal{P}, \mathcal{V}^*)(x)\rangle$ denotes the transcript of the interaction between $\mathcal{P}$ and $\mathcal{V}^*$ on a common input x. The idea behind this definition is that if for any verifier $\mathcal{V}$, there exists an efficient simulator $\mathcal{S}$ that can reproduce the conversation between $\mathcal{P}$ and $\mathcal{V}$ on any given input exists, then the system is zero-knowledge. An interaction produced by this simulator is indistinguishable from a real interaction. Since the whole interaction can be simulated without knowledge of the secret value, it is impossible that any secret information gets transferred during the interaction.

2.11.3 Proofs of Knowledge

Proofs of Knowledge (PoK) prove the knowledge of a so-called witness w instead of just proving the validity of a statement. If a machine $\mathcal{M}$ "knows" something, this knowledge can be extracted. This is formalized by another machine $\mathcal{E}$, the extractor, which can extract the knowledge of a given machine $\mathcal{M}$ [Han16].

- **Completeness:** Given the interaction between a verifier $\mathcal{V}$ and a prover $\mathcal{P}$, the probability that the verifier is convinced is 1.
- **Soundness:** The success probability of an extractor $\mathcal{E}$ extracting w from the machine of the prover must be at least as high as the probability that a (potentially malicious) prover $\mathcal{P}$ is able to convince a verifier $\mathcal{V}$ of having knowledge of a witness w.

With this definition, no prover that does not know the witness w can convince a verifier.

2.11.4 Schnorr Proof & Fiat-Shamir Heuristic

Zero-Knowledge Proofs of Knowledge (ZKPoK) combine PoKs with ZKPs. It is a special case of a zero-knowledge proof where the statement *only* consists of the assertion that the prover possesses the secret information. Such a protocol must necessarily require some form of interaction between the prover and the verifier [GO94]. If this were not the case, the verifier could record the protocol and convince someone else that he knows the secret.

However, if one is willing to make some assumptions, it is possible to turn many interactive protocols into non-interactive ones. The following protocol assumes the Random Oracle Model. In practice, this can be implemented by replacing the oracle with a fixed hash function.

Schnorr-proof: an interactive protocol, between a prover and a verifier to prove knowledge of a discrete logarithm of $\mathrm{Q} = \mathrm{x} \cdot \mathrm{P}$ (i.e., prove knowledge of the secret scalar x, with Q and P publicly known)

1. Prover creates $r \xleftarrow{R} \mathbb{Z}$ and $\mathrm{R} \leftarrow r \cdot P$ and sends R to verifier
2. Verifier replies with challenge value $c \xleftarrow{R} \mathbb{Z}$
3. Prover creates response $s \leftarrow r + c \cdot x$ and sends to verifier s
4. Verifier accepts, if: $s \cdot P \stackrel{?}{=} R + c \cdot Q \qquad (\equiv (r + c \cdot x) \cdot \mathrm{P} \stackrel{?}{=} r \cdot \mathrm{P} + c \cdot \mathrm{x} \cdot \mathrm{P})$

see figure 2.7 for a sequence diagram.

Fiat-Shamir Heuristic: a way to make an interactive protocol, like the aforementioned Schnorr-proof, non-interactive. It replaces the interactive step number 2 with a call to a Random Oracle $\mathcal{O}$. Instead of choosing c at random, the challenge value becomes c $\leftarrow \mathcal{O}$(*Transcript of interaction so far*). Since the output of the random oracle is, by definition, indistinguishable from random values, the challenge c value becomes essentially random. This removes the need for interaction, while still convincing the verifier of the randomness of the challenge value c.

1. Prover creates r $\overset{R}{\leftarrow} \mathbb{Z}$ and R $\leftarrow r \cdot P$
2. Prover creates challenge value c $\leftarrow \mathcal{O}(Q||P||R)$
3. Prover creates proof s $\leftarrow r + c \cdot x$ and sends to verifier (R, s)
4. Verifier accepts, if: s $\cdot$ P $\overset{?}{=}$ $R + \mathcal{O}(Q||P||R) \cdot Q$ (= (r + c $\cdot$ x) $\cdot$ P $\overset{?}{=}$ r $\cdot$ P + c $\cdot$ x $\cdot$ P)

2.11.5 Example Usage - Signature of Knowledge

In the implementation of our group signature scheme, we will make use of signatures of knowledge and the following example is from our group signature implementation.

Let BG = (p, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, e, P,$\hat{P}$) be publicly available to every participant. Let $\mu \in \mathbb{Z}_p$. Prove knowledge of $\boldsymbol{\mu}$ in: $\boldsymbol{P' = \mu \cdot P}$ and create a signature for message m.

1. Prover creates v $\overset{R}{\leftarrow} \mathbb{Z}_p$ and N $\leftarrow v \cdot P$
2. Prover creates challenge value c $\leftarrow \mathcal{O}(N||\sigma'||P||P'||m)$
3. Prover creates z $\leftarrow$ v + c $\cdot$ μ and sends signature $\sigma_2 \leftarrow (N, z)$
4. Verifier creates c $\leftarrow \mathcal{O}(N||\sigma'||P||P'||m)$
   Verifier accepts if: z $\cdot$ P $\overset{?}{=}$ $N + c \cdot P'$ (= (v + c $\cdot \mu$) $\cdot$ P $\overset{?}{=}$ v $\cdot$ P + $c \cdot \mu \cdot P$)

It is possible to reorder the equations in points 3 and 4 to reduce the number of group elements in the resulting signature:

3. Prover creates z $\leftarrow$ v + c $\cdot$ μ and sends signature $\sigma_2 \leftarrow$ (**c**, z)
4. Verifier creates N $\leftarrow$ z$\cdot$P - c$\cdot P'$ (= (v + c $\cdot$ μ) $\cdot$ P - c $\cdot \mu \cdot$ P)
   Verifier accepts if: c $\overset{?}{=}$ $\mathcal{O}(N||\sigma'||P||P'||m)$

2.12 Digital Signature Scheme

A digital signature scheme is a mathematical way to ensure the authenticity of a message or documents. Just like regular signatures, they give the recipient of a message a strong reason to believe that the message was created by a known sender [Wik19c]. Prominent examples for digital signature schemes are ECDSA [JMV01] or RSA [RSA78] based solutions.

Cryptographic schemes regularly use digital signature schemes as a building block, denoted as Σ.

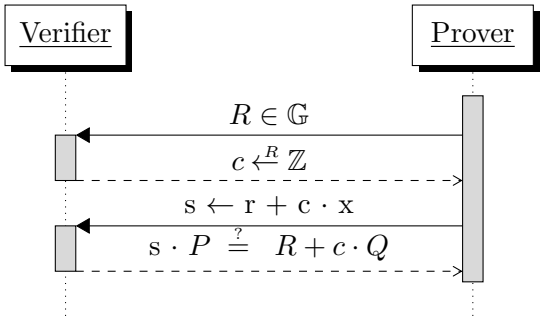


Figure 2.7: Schnorr Protocol to proof knowledge of the scalar x in Q = x · P.
R = r · P with r $\xleftarrow{R} \mathbb{Z}$
Also called a Σ protocol, because of the shape of the protocol in sequence diagram form.

Digital Signature: (definition from [CS18]) A digital signature scheme Σ is defined as follows:

- **Setup(1^λ):** on input a security parameter λ, this algorithm outputs the public parameters pp of the system;
- **Keygen(pp):** on input pp, this algorithm outputs a pair of signing and verification keys (sk, pk);
- **Sign(sk,m):** on input the signing key sk and a message m, this algorithm outputs a signature σ;
- **Verify(pk,m,σ):** on input the verification key pk, a message m and its alleged signature σ, this algorithm outputs 1 if σ is a valid signature on m under pk, and 0 otherwise.

2.13 ECDSA - Elliptic Curve Digital Signature Algorithm

ECDSA [JMV01] is a widely used signature algorithm. Its advantage over other signature algorithms, such as RSA [RSA78] based ones, is its small signature size. It uses a cryptographic hash function HASH, such as SHA-3.

- **Setup(1^κ):** on input a security parameter κ, this algorithm outputs the parameters of an elliptic curve, a generator P, and the order p of group $\mathbb{G}$ generated by P.
- **Keygen(pp):** on input pp, $q \xleftarrow{R} \mathbb{Z}_p$: Output ($sk \leftarrow q$, $pk \leftarrow q \cdot$ G)

- **Sign(sk,m):** parse the signing key sk as q, takes a message m:

  - $z \leftarrow$ HASH(m) mod p;

  - $k \xleftarrow{R} \mathbb{Z}_p$

  - $(x_1, y_1) \leftarrow k \cdot \mathrm{P}$
  - $r \leftarrow x_1 \bmod p$ (if $r = 0$ select new k)
  - $s \leftarrow \frac{z + r \cdot q}{k} \bmod p$ (if $s = 0$ select new k)
  - output (r, s) as signature

- **Verify(pk,m,σ):** parse the verification key pk as Q ($= q \cdot \mathrm{P}$), σ as (r, s), takes a message m:
  - $z \leftarrow \mathrm{HASH}(m) \bmod p$;
  - $u_1 \leftarrow \frac{z}{s} \bmod n$
  - $u_2 \leftarrow \frac{r}{s} \bmod n$
  - $(x_1, x_2) \leftarrow \mathrm{u}_1 \cdot \mathrm{P} + \mathrm{u}_2 \cdot \mathrm{Q}$

    $(\equiv \frac{z}{s} \cdot \mathrm{P} + \frac{r \cdot q}{s} \cdot \mathrm{P} = \frac{z + r \cdot q}{s} \cdot P = \frac{k}{z + r \cdot q} \cdot (z + r \cdot q) \cdot \mathrm{P} = k \cdot \mathrm{P})$
  - if r $= x_1$ return true; else return $\perp$

It is widely assumed that the size of an ECDSA public key should be twice the size of the desired security level: a 128bit would require a 256bit public key. The reason for that is that the most efficient algorithms to solve the elliptic-curve discrete logarithm problem, such as Pollard's rho algorithm [Pol78], have an exponential time complexity of $\mathcal{O}(\sqrt{n})$.

To add a little bit of trivia: ECDSA has been used to sign software for the Playstation 3, but their implementation was flawed. Instead of a random k, Sony used a fixed k, which lead to the following consequences: The parameter r of the signature of a signature of the form (r, s) is calculated by taking the x coordinate of the point $k \cdot P$ on the elliptic curve, and is therefore also fixed. An adversary could now take two signatures (r, s) and (r, s') for 2 messages m and m'. From the messages he could calculate z and z'. By reordering the equation $s - s' = \frac{z - z'}{k}$ to $k = \frac{z - z'}{s - s'}$ it is possible to calculate k. Since $s = \frac{z + r \cdot q}{k}$ one can reorder this equation to compute the secret key $q = \frac{s \cdot k - z}{r}$.

With this technique, a single reuse of the value k could compromise the private key. A modified version of this technique also applies to Fiat-Shamir and Schnorr signauture based schemes. This is why it is necessary to take measures to prevent that, either by making sure that one uses a secure random generator or by generating the parameter k deterministically with a pseudo-random generator.

2.14 Randomizeable Signatures

A randomizeable signature is one that allows anyone to derive a new valid signature σ' from an existing signature σ, sometimes without knowledge of a secret key [PS18]. Depending on the scheme, the new signature is valid on the original message or on some randomization of it. Let us consider that some entity has some certified data with a signature and wants to convince another party that this data is indeed certified while remaining anonymous. In some contexts, it is not possible to reveal the signature because of tracking/privacy concerns. In this case, it may be practical to show a randomized version of the signature that is still valid on the certified data.

One of the first signature schemes to implement this feature using bilinear groups was proposed by Camenisch and Lysyanskaya [CL04] (CL signatures). A CL signature σ can be randomized by raising each element of σ to the power of a random scalar t. The new signature is valid on the same message but is indistinguishable from the old signature under the DDH assumption.

2.15 Structure Preserving Signatures

Structure preserving signatures (SPS) are signatures defined in the context of bilinear groups BG = (p, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, e, P,$\hat{P}$) and allow for the signing of group-element vectors without requiring any prior encoding (their structure is preserved). The public keys and the signatures σ are also group elements, and verification is also done through group-membership tests and the conjunction of pairing-product equations (PPE). In case of fully structure-preserving signatures, even secret keys are group elements [Abe+15].

SPS have found numerous applications in public-key cryptography, such as blind signatures, group signatures, homomorphic signatures, delegatable anonymous credentials, compact verifiable shuffles, network encoding, oblivious transfer, and e-cash [KPW15]. SPS have originally been introduced in the context of Groth-Sahai [GS08] proofs as a new type of signature scheme compatible with this proof system. Using this as a basis, other types of structure-preserving signatures have been found [Lib+15][HS14][FHS15].

2.16 Group Signatures

Group signatures are a digital signature scheme where an individual of a group of people can sign messages on behalf of the whole group without revealing his identity [CH91]. With the use of a public key, anyone can check the validity of the signature and verify that this signature originates from this group. In case of a dispute, there is an opening authority that able to "open" a group signature and reveal the person that signed it. Depending on the scheme, this authority is the one that distributes group memberships, or it may be a

completely separate entity with its own secret key. In figure 2.8, a mock-up of the basic functions of a group signature can be seen.

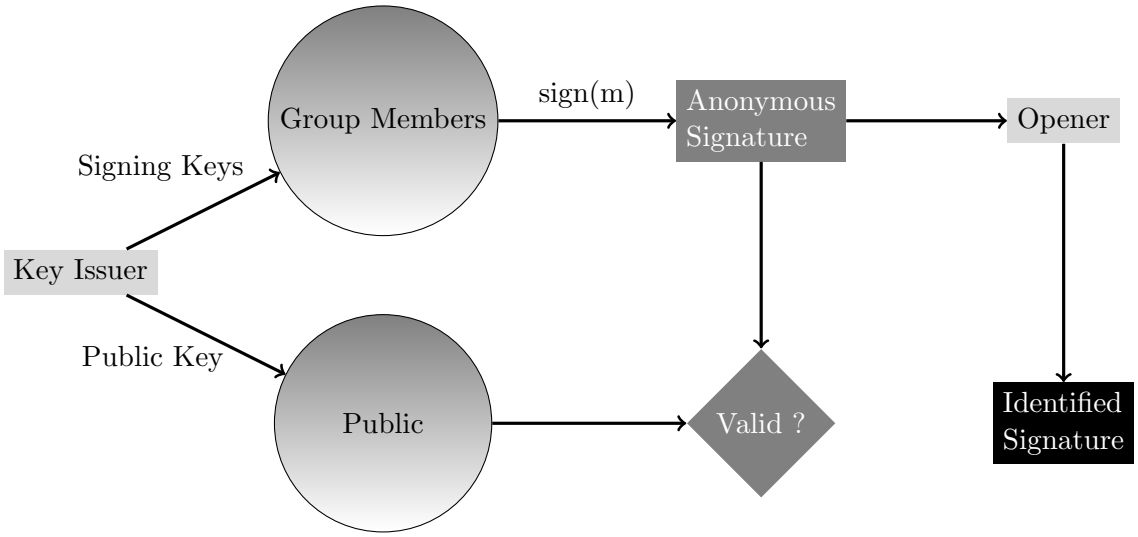


Figure 2.8: Group members are in possession of a signing key and can create signatures on behalf of the group. Anyone can verify the signature, but only the opening authority can identify the individual signer.

2.16.1 Dynamic Group Signature Scheme

In the description of our payment protocol, we will use the following definition of a generic dynamic group signature scheme, in which group membership is defined as owning a group signing key. Dynamic in this context means that it is possible to add new group members after the initial group has been established. Our model for dynamic group signatures is based on the model of Derler et al. [DS18].

Public Values:

- **gpk** (group public key)

Actors with Secret Values:

- Group Members: **gsk** (group signing key)
- Issuing Authority: **ik** (issuing key)
- Opening Authority: **ok** (opening key)

Operations:

- **GKGen(1^κ):** Takes a security parameter κ. Returns a triple (gpk, ik, ok) containing the group public key gpk, the issuing key ik, and the opening key ok.
- **UKGen(1^κ):** Takes a security parameter κ. Outputs a user key pair (usk_i, upk_i).

- **Join(gpk, $\mathbf{usk}_i$, $\mathbf{upk}_i$):** Takes the group public key gpk and the user keypair (usk_i, upk_i) as input. Interacts with the Issue algorithm and returns a group signing key gsk_i for user i.
- **Issue(gpk, ik, i, $\mathbf{upk}_i$, reg):** Takes the group public key gpk, the issuing key ik, the index of a user i, the upk_i of this user, and the registration table reg as input. Interacts with Join algorithm and adds and entry for user i in reg and returns reg.
- **Sign(gpk, $\mathbf{gsk}_i$, m):** Takes a group public key gpk, a group signing key gsk_i and a message m. Returns a group signature σ.
- **Verify(gpk, m, $\boldsymbol{\sigma}$):** Takes a group public key gpk, a message m, and a signature σ. Verifies the validity of the group signature on that message and outputs a bit $b \in \{0,1\}$.
- **Open(gpk, ok, reg, m, $\boldsymbol{\sigma}$):** Takes the group public key gpk, the opening key ok, the registration table reg, a message m, and a valid signature σ on m under gpk. Reveals the identity i of the group member that created σ and creates a proof that proves the opening τ. If successful, outputs (i, τ), returns $\perp$ otherwise.
- **Judge(gpk, m, $\boldsymbol{\sigma}$, i, $\mathbf{upk}_i$ $\boldsymbol{\tau}$):** Takes a group public key gpk, a message m, a valid signature on m under gpk σ, an index i, the public key of user i, and a proof τ. Verifies that user i created the signature on message m. Outputs a bit $b \in \{0,1\}$.

2.16.2 Security model

In this section, we will establish our security model for group signatures. We will explain a simplified version of the model of Derler et al. [DS18]. Their model requires computational correctness instead of perfect correctness, and it also addresses the possibility of corrupt protocol members. For the use case of our payment system, we require group signatures to be correct, CPA-Anonymous (2.16.3), traceable, and non-frameable. Due to traceability, every valid signature can be attributed to a group member, and due to non-frameability, this signature can only be created with the group signing key of that group member. This directly implies unforgeability.

Oracles

Our definition of our security model requires the definition of some oracles. To define the capabilities of an oracle $\mathcal{O}$ we add a function Func to it using the following notation: $\mathcal{O} \leftarrow \{\mathsf{Func}\}$. An adversary $\mathcal{A}$ having access to such an oracle $\mathcal{O}$ is denoted as $\mathcal{A}^{\mathcal{O}}$.

We assume an environment in which only our group of interest exists and that all keys (gpk, ik, ok) are implicitly available to every oracle. The environment also maintains the registry of every created group member and their public keys, which are available to every participant in the protocol.

Helper functions:

AddU(i): Takes an index i as input. Returns $\perp$ if user with index i already exists . Else it adds a new group member by running $(usk_i, upk_i) \leftarrow \mathsf{UKGen}(1^\kappa)$ and $(reg, gsk_i) \leftarrow \langle\ \mathsf{Join}(\text{gpk, usk}_i\text{, upk}_i)\text{: } \Leftrightarrow \mathsf{Issue}(\text{gpk, ik, i, upk}_i\text{, reg})\text{: }\rangle$ and returns the results to the caller.

Ch(b, $\mathbf{i}_0$, $\mathbf{i}_1$, m): Takes a bit b, two indexes i_0 i_1 and a message m as input. Computes $\sigma \leftarrow \mathsf{Sign}(gpk, gsk_{i_b}, m)$ and returns σ.

- **Correctness:** A GSS is correct, if everything works correctly if every participant is honest. More formally, a GSS is correct if for all PPT adversaries $\mathcal{A}$ there is exists a negligible function $\epsilon(\cdot)$ such that:

$$\Pr\left[\begin{array}{c} (gpk, ik, ok) \leftarrow \mathsf{GKGen}(1^\kappa), \\ \mathcal{O} \leftarrow \{\text{AddU}(\cdot)\}, \\ (i, m) \leftarrow \mathcal{A}^{\mathcal{O}}(gpk) \\ \sigma \leftarrow \mathsf{Sign}(gpk, gsk_i, m), \\ (j, \tau) \leftarrow \mathsf{Open}(gpk, ok, \\ reg, m, \sigma) \end{array} : \begin{array}{c} \\ \\ \\ \mathsf{Verifiy}(gpk, m, \sigma) = 1 \\ \wedge\ i = j, \\ \wedge\ \mathsf{Judge}(gpk, m, \sigma, \\ i, upk_i, \tau) = 1 \end{array}\right] \geq 1 - \epsilon(\kappa)$$

- **Traceability:** A GSS is traceable if the issuer is honest and it is infeasible to create a valid signature that does not belong to some member of the group. Note that a corrupt issuer can always create a dummy group signing key and create group signatures.
  For all PPT bounded adversaries there exists a negligible function $\epsilon(\cdot)$ such that:

$$\Pr\left[\begin{array}{c} (gpk, ik, ok) \leftarrow \mathsf{GKGen}(1^\kappa), \\ \mathcal{O} \leftarrow \{\text{AddU}(\cdot)\}, \\ (m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(gpk, ok) \\ (i, \tau) \leftarrow \mathsf{Open}(gpk, ok, \\ reg, m, \sigma) \end{array} : \begin{array}{c} \\ \\ \mathsf{Verifiy}(gpk, m, \sigma) = 1 \wedge \\ (i = \perp \vee \\ \mathsf{Judge}(gpk, m, \sigma, i, upk_i, \tau) = 0) \end{array}\right] \leq \epsilon(\kappa)$$

- **CPA - Anonymity:** Informally, a GSS is anonymous, if it is infeasible for an adversary to identify the signer of a message, without knowledge of the opening key. Different group signature schemes feature different levels of anonymity. For our purposes, we require CPA-anonymity, meaning that an even an adversary with access to all signing keys of the group can not distinguish the signers of two group signatures. Formally, for all PPT bounded adversaries there exists a negligible function $\epsilon(\cdot)$ such that:

$$\Pr\left[\begin{array}{c} (gpk, ik, ok) \leftarrow \mathsf{GKGen}(1^\kappa), \\ \mathcal{O} \leftarrow \{\mathsf{AddU}(\cdot), \mathsf{Ch}(\text{b}, \cdot, \cdot, \cdot)\}, \\ b \xleftarrow{R} \{0, 1\},\ b^* \leftarrow \mathcal{A}^{\mathcal{O}}(gpk, ik) \end{array} : b = b^* \right] \leq \epsilon(\kappa)$$

- **Non-Frameability:** Non-frameability encompasses the notion that no one can forge signatures for honest users. A GSS is non-frameable if it is infeasible to create a valid group signature that the Open algorithm identifies as being signed by user i,

and the Judge algorithm confirming it, without knowledge of the group signing key gsk_i. Non-Frameability holds, even if the issuer and the opener are controlled by the adversary. A GSS is non-frameabble if for all PPT bounded adversaries there exists a negligible function $\epsilon(\cdot)$ such that:

$$\Pr\left[\begin{array}{c} (gpk, ik, ok) \leftarrow \mathsf{GKGen}(1^\kappa), \\ i \xleftarrow{R} \mathbb{Z},\ \mathsf{AddU}(i), \\ \mathcal{O} \leftarrow \{\mathsf{AddU}(\cdot)\}, \\ (m, \sigma, i, \tau) \leftarrow \mathcal{A}^{\mathcal{O}}(gpk, ok, ik) \end{array} : \begin{array}{c} \mathsf{Verifiy}(gpk, m, \sigma) = 1 \\ \wedge\ \mathsf{Judge}(gpk, m, \sigma, i, upk_i, \tau) = 1 \end{array}\right] \leq \epsilon(\kappa)$$

2.16.3 Group Signature Features

Different group signature schemes may have additional features that extend the basic functionality of a generic group signature. This section will go into some of the most common features of group signature schemes.

Anonymity

There are already exist established definitions for security notions regarding ciphertext indistinguishability. These notions can be adapted to define anonymity notions of group signatures. In this context, the security notions are not about distinguishing ciphertexts, but about distinguishing group signatures made by different group members. For this task, an adversary has different tools at his disposal, depending on the security notion.

The anonymity notions are defined as a game in which the adversary receives a challenge which contains two group signatures, created by two different group members. The adversary wins the game if he is able to distinguish the signers of the group signatures with a probability of greater than $\frac{1}{2} + \epsilon(\kappa)$, where $\epsilon(\cdot)$ is a negligible function.

- **CPA**: The adversary has access to all signing keys of all members of the group and can create as many group signatures as he wants, before and after receiving the challenge.
- **CCA**: In addition to having access to all signing keys, the adversary also has access to an opening oracle that can reveal the signer of any group signature. He is only allowed to make calls to the opening oracle before receiving the challenge.
- **CCA2:** in addition to everything in CCA, the adversary is able to make calls to the opening oracle even after receiving the challenge, except on the signatures in the challenge itself.
- **Full-Anonymity vs. Selfless-Anonymity:** In the selfless-anonymity setting, as opposed to the full-anonymity setting, the adversary does not have access to the signing keys of the group members that created the challenge signatures. In this setting, signatures are only anonymous as long as the signing keys of the respective

signers do not leak.
The notation of selfless-anonymity in the literature is not fully consistent, but common examples would be: CCA-selfless-anonymity or CCA^{-1}.

Dynamic

Earlier examples of group signature schemes were static, meaning that the group members were fixed at the time of the creation of the group. Most real-world use cases require dynamic features, as opposed to static group signatures. In static group signatures, recreation of the whole group structure, including signing and verifying keys, is necessary upon adding a new member.

A fully dynamic group signature is one that has both of the following properties:

- **Dynamic User enrollment**: It is possible to add new group members to an existing group, and adding new group members to the group does not need interaction with existing group members.
- **Verifier Local Revocation** [BS04]: revocation messages are sent to verifiers, without the need for interaction with individual signers. This makes it possible to dynamically revoke group signing keys (i.e., leave the group).

Deniability

A deniable group signature is one in which the signer of a group signature can create a proof that states that he did NOT create a given signature. Ishida et al. [Ish+17] have introduced the concept of a deniable group signature. In some situations, it is only required to know that someone did not create a signature, and simply opening the signature might violate the privacy of the actual signer.

An example would be a building, for which entry and exit control is managed with a group signature system. With a group signature scheme that does not support deniability, if the authorities want to know if someone was in the building a particular point in time, they would have to open all signatures from that time frame. This would violate the privacy of innocent people, who just happened to be in the building at the same time. Using a deniable group signature scheme, that person would be able to prove that he was not in the building during this time frame.

Message dependent opening

Message dependent opening for group signatures means that signatures can only be opened by the opening authority if they have previously been approved by a separate admitting authority. Sakai et al. [Sak+12] introduced group signatures with message dependent opening. The issue they are trying to solve is the high level of trust that is put into the

opener. They introduce a new authority, called the admitter, which is not able to open any signatures on its own but can admit specified messages whose signatures should be opened. In such a scheme, the opener can only open signatures that were previously admitted by the admitter.

Verifiable opening

In the context of group signatures, verifiable opening refers to the ability of the opener to create a publicly verifiable proof that the identity revealed during the opening process is correct. Since the opener is the only one that has access to the opening mechanism, no one can stop him from falsely accusing someone of having produced a particular signature. In schemes with verifiable openings, the opening consists of the identity of the signer and some publicly verifiable proof that the identity is indeed correct. In the literature, this feature has sometimes become intrinsic to the definition of group signatures and is not even mentioned as a dedicated feature. It is usually described as the Judge function.

Group Signatures with Distributed Authorities

In group signatures with distributed authorities, the role of the issuer and the opener are completely separated and independent. In some of the earlier group signature schemes, these roles were combined into the same role, often called group manager (sometimes group manager only refers to the issuer). The issuer is the entity that is in possession of the issuing key, and the opener is the entity that is in possession of the opening key. In these earlier schemes, the issuing key and the opening key had some parameters in common that made it impossible to separate them.

Chapter 3: Payment System

In this chapter, we introduce our privacy-friendly payment scheme from group signatures. We will go into the design of the payment system protocol and discuss the requirements that a group signature scheme for our system would need to fulfill. Using these requirements as a basis, we will compare current group signature schemes and discuss our choice of group signature scheme. For now, we will work with our generic definition of a group signature scheme that we assume to be dynamic and efficient.

3.1 Motivation

Our motivation for the creation of this system was to create a payment system that would minimize the amount of private data exposed to every party involved. This leaves every party only with the minimum amount of information they "need to know" in order to complete the transaction, while still maintaining full traceability for a third party (i.e., governing authorities). The reason for maintaining full traceability is the need to prevent theft, fraud, money laundering, or the funding of organized crime.

Regular payment schemes use regular digital signature schemes combined with certificates to achieve authenticity. They do not take measures to ensure the privacy of the customer. In fact, this non-anonymity is actually a desirable property for many parties involved in a payment process. The signed transaction has information about the buyer, the shop, and the purchase. It goes through the shop, the payment processor, and in the end, the bank. The shopping habits, exact location (the shop's location is public knowledge) of an individual combined with the time of purchase, are all valuable personal data. All parties involved can track the buyers' purchases and use this information for various purposes, starting from loyalty point reward programs, to targeted marketing campaigns, or even just simply selling this information to third parties.

An excellent way to gain some understanding of the necessary data exchange is by analyzing a cash transaction. By paying in cash, no customer name, no credit card number, no bank account details get exchanged, and the bank does not get to know where the customer shops or what he bought. This system works because cash acts as a sort of certificate, an assurance for the merchant that he has received his part of the deal.

It is easy to see how a naive implementation of a completely anonymous untraceable payment system, using basic cryptographic primitives, would be able to emulate a cash transaction. However, an untraceable digital payment system would just exacerbate the problems that cash already has, mainly fraud and money laundering. Also, if the payment details of someone get stolen, this individual would like to have some way to get his money back, or at least know where it went. Likewise, the responsible government agency would like to be able to trace money transactions to investigate criminal activities.

The goal of this payment system is to combine the privacy benefits of cash with the convenience of digital payments while still maintaining full traceability. We achieve this by restricting the amount of information each actor in the system gains to a bare minimum, by diligently applying the "need to know" principle. For a full reconstruction of the transaction, the information of multiple actors needs to be combined.

The main information points our payment system tries to keep separated/anonymous are customer identity and shop identity. Only the bank knows the customer identity, while everyone else only works with fully anonymous payment data. At the same time, the bank does not get to know where the customer spends his money. Our use of group signatures allows for the authentication of legitimate transactions at every step of the protocol without compromising on privacy. For full traceability, the information that each entity has access to can be collected by a trusted third party (e.g., law enforcement) on demand (e.g., a warrant).

Another feature that our payment system provides is the ability for employees of a shop to bring their own devices and use them in the payment process. The shop can complete and authenticate a transaction without knowledge of the identity of the employee, meaning that the identities of the employees can also be made anonymous in our payment system.

The general data protection regulation (GDPR) [EU16] in the EU has implemented strict rules for the handling of personal data, with many responsibilities and hefty fines attached to them. Inevitably, storing data in a verifiable GPDR compliant way creates some overhead, which could turn payment data collected by conventional digital payment systems into a liability for small businesses. In our payment system, merchants do not gain any personal information on their customers in the first place, circumventing the regulation altogether.

3.2 Setting

The setting of our example use case is that of a typical coffee shop, and it serves as a benchmark for our payment system in practice. Through the prevalence of other payment apps, like Apple Pay or NFC enabled credit cards, consumers today have become used to the paradigm of paying via NFC. Therefore, we opted to use Host Card Emulation (HCE) with NFC in our payment system. The customer will pay for his coffee using an app on his Android phone, and an employee of the shop will receive the payment with his own Android device, with the transmission done through NFC. Additionally, our protocol will

be designed with the privacy concerns of the employee in mind, which enables the employee to use his personal device. During a transaction in this setting, there are 5 different actors involved:

- **Customer:** the customer of the shop
- **Employee:** the employee of the shop
- **Shop:** the shop itself
- **Payment Processor:** middle man that mediates between the shop and the bank
- **Bank:** the bank, of which the user is a customer

We chose this use case because it is representative of a lot of other payment use cases. In the case of a supermarket, the employee of the coffee shop could be replaced by a self-checkout machine. In public transport ticketing, one would have to replace the role of the employee by a ticket gate. It could also be applied to vending machines by replacing the employee with an NFC receiver on the front of the vending machine.

In all of those scenarios, the transaction between the end-user and the employee must happen rapidly. In the case of a public transport ticket gate, a processing/waiting time of 1s, 500ms, or 200ms, would make a huge difference in the usability of the system. Also, because NFC transmissions are highly dependent on the position of the antennas, a processing time of 1 second or more would be highly detrimental to usability, because users have to hold their devices in a particular position for this amount of time.

3.3 Payment Protocol

In this section, we will first go over the structure of our payment system and will then go into the details of an example transaction in our system. By dissecting a completed transaction, we will demonstrate what kind of information is revealed to each party and how it can be authenticated without compromising on privacy. Having laid out how our system is supposed to work, we then define our requirements for a group signature scheme and discuss potential candidates for our implementation.

3.3.1 Group Structures and Setup

For our payment system, we established three groups (3.1). Each member of a group is able to create anonymous group signatures on behalf of the group.

- The group of all customers of the bank
- The group of all shops that use our payment processor
- The group of all employees of the shop

In our case, both the issuing key and the opening key of each group are held by the respective entity, namely the bank, the payment processor and the shop. The group public keys of each group are assumed to be implicitly available to anyone.

It is assumed that the initial issuing of group memberships/group signing keys is done through a secure authentic communication channel.

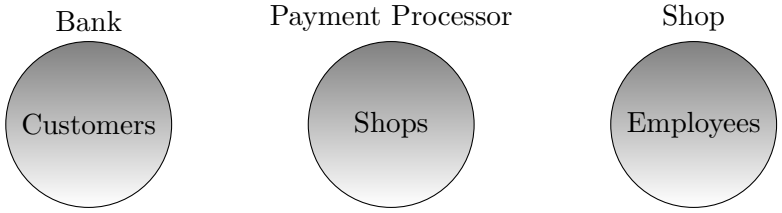


Figure 3.1: Group structure of payment system. The bank is the group manager of the group of all customers, the payment processor is the group manager of the group of all shops, and the shops are the group managers of the group of all of their employees.

3.3.2 Transaction

In our coffee shop example, a new transaction is always created by the employee. The tasks every participant has to fulfill in the protocol are described in this section, and the path that this transaction takes is also illustrated in figure 3.2.

Customer

1. The customer receives a transaction and a group signature from the employee and can verify the employees signature using the public key of the shop.
2. The customer signs the transaction on behalf of all customers of his bank and sends his signature back to the employee.

Employee

1. The employee creates the transaction, signs it on behalf of the shop, and sends it to the customer.
2. The employee receives the signature of the customer and can check if it is valid by using the public key of the bank.
3. The employee sends the transaction, his signature, and the signature of the customer to the shop.
4. The employee gets notified by the shop if the transaction was successful.

Shop

1. The shop receives the transaction, the signature of the customer, and employee from the employee and can check the validity of the signatures using the public keys of the bank and its own public key.
2. The shop signs the transaction on behalf of all shops and sends his signature, the signature of the customer, and the transaction to the payment processor.
3. The shop either receives the money or an error message from the payment processor.
4. The shop notifies the employee about the results of the transaction.

Payment Processor

1. The payment processor receives the transaction, the signature of the customer, and shop and can check the validity of the signatures using the public keys of the bank and its own public key.
2. The payment processor forwards the transaction and signatures to the bank.
3. The payment processor either receives the money or an error message from the bank.
4. Depending on the response of the bank, the payment processor notifies the shop about the results of the transaction and forwards the money.

Bank

1. The bank receives the transaction and the signature of the shop and customer and can check the validity of the signatures using the public keys of the bank and its own public key. Using its own opening key, it can open the signature and determine the identity of the customer.
2. Using the identity of the customer, the bank can subtract the money amount specified in the transaction from the customer's bank account and send it to the payment processor.

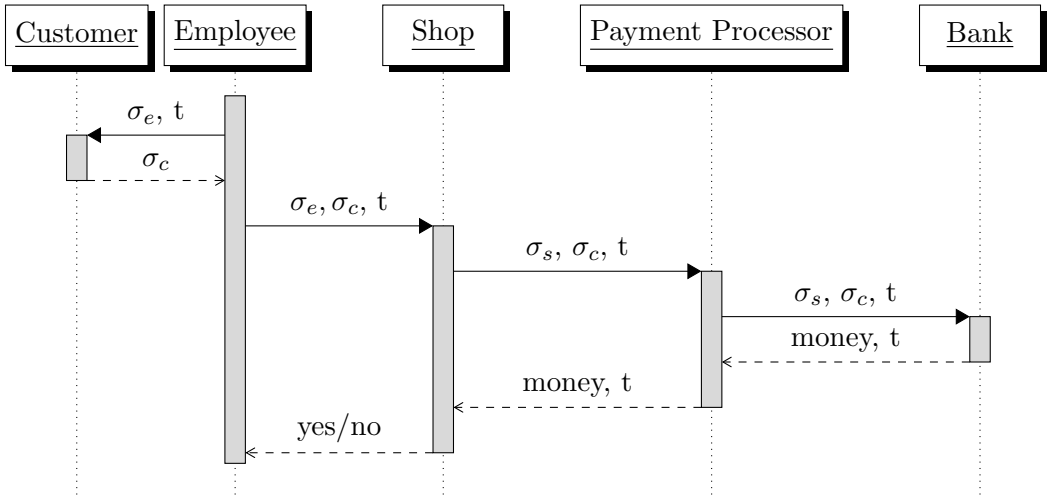


Figure 3.2: Payment process in detail. σ refers to a group signature on behalf of the group of employees, shops, or customers. Every actor can verify the authenticity of the transaction by verifying the group signatures attached to it without revealing the actual signer. The employee only learns whether the transaction was successful or not.

3.4 Security

In this section, we will explain some security properties in the context of our payment system and why we need them. Just like the requirements for our group signature scheme, we need our payment system to be correct, anonymous, traceable, and non-frameable.

An important point to mention at the beginning is that valid transactions in our payment system basically consist of group signatures on transactions. This means that the security of the payment system is based directly on the security of the underlying group signature scheme.

Correctness

Correctness means that if the user, the employee, the shop, the payment processor, and the bank are honest, then the payment protocol works as intended. The need for correctness of our system is obvious and is based directly on the correctness claims of the underlying GSS, meaning that our system is correct if and only if the GSS is correct.

Anonymity

By design, transactions in our payment system are only anonymous to parties that do not have access to the opening keys of the accompanying signatures. Given that the underlying

GSS is anonymous, the identity of the customer remains hidden from the employee, the shop, and the payment provider. The identity of the shop remains hidden to the bank.

Traceability

Traceability means that every transaction can be fully retraced by a third party. In our payment system, this is possible if the payment processor and the bank provide the opening of the signatures of the shop and the customer. If and only if the underlying group signature is non-frameable and traceable, one can be sure of the identity and purchase location of the customer. Traceability is a key component of our payment system since it provides the tools for law enforcement to prevent fraud, theft, or money laundering.

Non-Frameability

The payment system is non-frameable if it is infeasible to create or sign a transaction that either the shop, the payment processor, or the bank, would attribute to group member i, without knowledge of the exact group signing key gsk_i. This property is a direct consequence of the non-frameability of the GSS and makes it impossible to fraudulently sign a transaction on behalf of a customer or create fake transactions of behalf of another entity in our payment system.

3.5 Choice of Group Signature

There are multiple different group signature schemes in the literature, yet actual implementations are sparse. In order to not be constrained in our choice of group signature scheme by the availability of suitable implementations, we decided to look for the most suitable one for our use case and implement it ourselves. We first defined the requirements for our group signature by taking a look at the requirements of our coffee shop use case then chose the most efficient one that met these requirements.

3.5.1 Group Signature Requirements

Our choice of group signatures is restricted because part of our payment system runs on mobile devices, one Android app for the end-user, and another one for the employee of a business. In our use case, only the signing of transactions happens on mobile devices, while the verification can happen on server hardware. Having this in mind, we came up with the following requirements.

Time

For usability reasons, we set the computation time limit on smartphones to be 300ms, similar to other papers, that cite that number to be the limit for public transportation smartcards [PS16]. When using HCE on one phone and another phone as an NFC reader, the two phones must be aligned for the duration of the transmission. For usability reasons, it is imperative to minimize the time they have to be aligned, which implies minimizing computation time. For other use cases, like public transport ticketing, minimizing interaction time might be even more critical for the viability of such a system.

Size

We set the upper limit for signature size to be 2048 bytes or 2KB. When looking at the chain of devices that our payment system uses, the weakest devices in the chain are the smartphones. Nowadays, even smartphones have gigabytes of memory, which would imply that signature size should not be a big factor in our decision. However, the signature has to be transferred from the device of the user to the device of the employee. Our testing in table 4.3 demonstrates that the bandwidth that an NFC transmission between two Android devices provides is quite low, meaning that the size of the signature directly affects our time requirement. Looking at our testing in table 4.3 and our time requirements of 300ms, we set the upper limit for signature size to be 2KB.

Dynamic

For practicality reasons, our group signature scheme needs to have at least **dynamic user enrollment**. We want it to be possible for a user in our payment system to sign a transaction offline. Therefore, any interaction with existing group members during the enrollment of a new member would be impractical.

Verifier Local Revocation is a feature that is certainly nice to have, but it is not a mandatory feature for our payment system since revocation can be implemented in different stations in our protocol. For instance, since the bank opens all user signatures, it could blacklist users according to their ID.
Deniability would extend the functionality of our payment system, but it is not critical for the basic functionality of the system. Message dependent opening is also not something that our payment system relies depends on.

Security

For our payment system to be secure, we need our GSS to be correct, at least CPA-anonymous, traceable, and non-frameable. The definitions for these terms have already been covered in detail in previous sections (2.16).

3.5.2 Potential Group Signature Schemes

We compiled a table (3.1) of popular group signature schemes in the literature, and we came to the conclusion that the most efficient schemes all seem to utilize elliptic curve cryptography, pairing-based cryptography to be specific. In pairing-based cryptography, the operations of scalar multiplication and the calculation of pairings are the dominant operations in terms of computing cost. In fact, they are so dominant that it is reasonable to discard any other necessary computing operation when comparing the efficiency of these schemes. Incidentally, this makes comparing the efficiency of those schemes rather simple.

The most important performance metric we are looking for is fast signing times because customers in our payment system have to create signatures on their mobile devices. The group signature scheme from Derler and Slamanig [DS18] seems to take the least amount of computationally dominant operations.

While verification times are second to signing times on our priority list, the same scheme also seems to be ahead of the competition in this regard.

Signature size also effectively affects our interaction time because the signature has to be transferred via NFC. The scheme of Derler and Slamanig seems to have a competitive signature size. Depending on the implementation and the usage of point compression [Too20], it can have the smallest signature size compared to the alternatives.

The scheme by Derler and Slamanig also meets our security requirements for anonymity by having dynamic user enrollment and being fully CPA anonymous. Other group signature schemes, such as the one ones from Bichsel et al. [Bic+10] and Pointcheval et al. [PS16], use the relaxed anonymity notion called *selfless anonymity*, which in essence means that the anonymity of group members whose signing keys leaked is not guaranteed. This kind of relaxed anonymity notion is undesirable in our payment system.

After the creation of our payment system, we discovered a new group signature scheme that was proposed by Clarisse and Sanders [CS18]. It has many similarities to our chosen scheme [DS18], in the sense that it also builds on [FHS18], and combines them with the signature scheme of Pointcheval et al. [PS16]. It promises a slight increase in efficiency compared to [DS18] and it does so in the standard model, as opposed to the Random Oracle Model. However, at the time of writing this thesis, it does not seem to have been published anywhere else but on the Cryptology ePrint Archive [ePr19] and does not seem to have undergone any kind of formal review. In light of this, we opted to stick to our initial decision to use the scheme of Derler and Slamanig [DS18].

Scheme	Signature Size	Sign	Verify	Open
[DS18]	$1\mathbb{G}_2 + 4\mathbb{G}_1 + 2\ \mathbb{Z}_p$	$1\mathbb{G}_2 + 5\mathbb{G}_1$	$5\text{P} + 2\mathbb{G}_1$	$\mathcal{O}(\text{n})$
[BBS04]	$3\mathbb{G}_1 + 6\ \mathbb{Z}_p$	$3\mathbb{G}_T + 9\mathbb{G}_1$	$4\mathbb{G}_T + 8\mathbb{G}_1$	$\mathcal{O}(1)$
[DP06]	$4\mathbb{G}_1 + 5\ \mathbb{Z}_p$	$3\mathbb{G}_T + 8\mathbb{G}_1$	$1\text{P} + 3\mathbb{G}_T + 2\ \mathbb{G}_2 + 7\mathbb{G}_1$	$\mathcal{O}(1)$
[Lib+16c]	$7\mathbb{G}_1 + 3\ \mathbb{Z}_p$	$4\text{P} + 2\mathbb{G}_T + 16\mathbb{G}_1$	$8\text{P} + 3\mathbb{G}_T + 7\mathbb{G}_1$	$\mathcal{O}(1)$
[Bac+18]	-	$2\mathbb{G}_2 + 9\mathbb{G}_1$	-	$\mathcal{O}(\text{n})$
[CS18]	$1\mathbb{G}_2 + 4\mathbb{G}_1$	$1\mathbb{G}_2 + 5\mathbb{G}_1$	$5\text{P} + 1\mathbb{G}_2$	$\mathcal{O}(\text{n})$

Table 3.1: Comparison between Group Signature Schemes. In terms of computational cost, we only count the expensive operations in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$. Pairings are more expensive that $\mathbb{G}_2$ operations, which are more expensive than $\mathbb{G}_1$ operations.

3.6 FHS / SPS-EQ Signature Scheme

A big part of implementing our payment system was implementing the actual group signature scheme itself. Our chosen group signature scheme uses the [FHS18] signature scheme as a base (also known as SPS-EQ). To the best of our knowledge, there did not exist any pure implementations of this scheme, meaning that we had to implement it ourselves before implementing our actual group signature scheme. In this section, we will give a brief outline of the unique features of this signature scheme followed by our implementation details and the full construction.

3.6.1 Features

FHS signatures are a structure-preserving signature, meaning that the operations of signing and verifying are all done through group membership tests and pairing evaluations, without requiring any a-priori encoding. Signatures and public keys also consist purely of group elements. Structure-preserving signatures have the main advantage that they can be efficient and secure under standard assumptions without having to rely on Random Oracles.

The signature scheme is also randomizeable, which, in this case, allows for the randomization of message-signature pairs without knowledge of a secret key. Given a message-signature pair, anyone can randomize this pair into a new valid message-signature pair that is indistinguishable from a fresh signature on a random message. This new pair cannot be traced back to the original one under the DDH assumption. This randomizability is the main feature that our group signature scheme utilizes.

SPS-EQ work in a Type-3 bilinear groups setting BG = (p, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, e, P,$\hat{P}$). The space of possible messages consists of a vector of elements of $\mathbb{G}_1^\ell$ with $\ell \geq 2$. The idea behind these signatures is that, since $\mathbb{G}$ is of prime order, $\mathbb{G}^\ell$ contains an underlying vector space $\mathbb{Z}_p^\ell$. For an $\ell \geq 2$ we can partition this vector space into equivalence classes by defining a

equivalence relation. Defined on $\mathbb{G}^\ell$, this equivalence relation $\sim_{\mathcal{R}}$ is as follows:

$$M \in \mathbb{G}^\ell \sim_{\mathcal{R}} N \in \mathbb{G}^\ell \Leftrightarrow \exists \mu \in \mathbb{Z}_p^* : M = \mu N$$

In other words, given two vectors M and N $\in \mathbb{G}^\ell$, the vector N is considered equivalent to M if it is a scalar multiple of M. To give some intuition it might be easier to think of such an equivalence class in a 2-dimensional case as lines running through the origin. An equivalence class contains all elements on that line except for the all-zero vector itself.

This signature scheme uses this equivalence relation to enable the randomization of message-signature pairs by moving them inside their equivalence classes. Under the DDH assumption, those new message-signature pairs cannot be linked to the original pair and are indistinguishable from fresh signatures on random messages.

3.6.2 Implementation Details of FHS / SPS-EQ signature scheme

A complete construction of the signature scheme can be found in figure (3.3). For all of our bilinear group elliptic curve operations, we chose a Baretto-Naehrig [BN05] curve (BN) with a base field size of 448 bit. The functionality to work with such a curve, including the functionality to do elliptic curve operations and calculate pairings, is provided by the ECCelerate library [AC19], a cryptography library written entirely in Java.

3.7 Derler-Slamanig Group Signature Scheme

The implementation of the [DS18] group signature scheme was a big part of the implementation of our payment system since it is the foundation on which our system is built upon. In this section, we will start by outlining the general concept behind the operating mechanism of this signature scheme, followed by the full construction and our implementation details.

3.7.1 General Concept

The group signature scheme is based on the sign-randomize-proof (SRP) paradigm. In this setting, every group member has a secret group membership certificate that is signed by the issuer. To create a group signature, group member randomizes his certificate and then proofs that he is in possession of the original certificate without actually revealing it. The main building block of this signature scheme is the randomizability of the FHS signature scheme.

To add members to the group, the issuer creates and distributes message-signature pairs signed with the FHS scheme using his own secret issuing key. These act as membership certificates to prospective group members,

$\mathsf{BGGen}_{\mathcal{R}}(1^\kappa)$: Given a security parameter κ, output $\mathsf{BG} \leftarrow \mathsf{BGGen}\ (1^\kappa)$.

$\mathsf{KeyGen}_{\mathcal{R}}(\mathsf{BG}, \ell)$: Given a bilinear-group description BG and vector length $\ell > 1$, choose $(x_i)_{i\in[\ell]} \xleftarrow{R} (\mathbb{Z}_p^*)^\ell$, set the secret key as $\mathsf{sk} \leftarrow (x_i)_{i\in[\ell]}$, compute the public key $\mathsf{pk} \leftarrow (\hat{X}_i)_{i\in[\ell]} = (x_i\hat{P})_{i\in[\ell]}$ and output $(\mathsf{sk}, \mathsf{pk})$.

$\mathsf{Sign}_{\mathcal{R}}$ (M,sk): On input a representative $M = (M_i)_{i\in[\ell]} \in (\mathbb{G}_1^*)^\ell$ of equivalence class $[M]_{\mathcal{R}}$ and a secret key $\mathsf{sk} = (x_i)_{i\in[\ell]}$, choose $y \xleftarrow{R} \mathbb{Z}_p^*$ and output $\sigma = (Z, Y, \hat{Y})$ with

$$Z \leftarrow y \sum_{i\in[\ell]} x_i M_i, \qquad Y \leftarrow \tfrac{1}{y} P, \qquad \hat{Y} \leftarrow \tfrac{1}{y}\hat{P}$$

$\mathsf{Verify}_{\mathcal{R}}(\mathsf{M},\sigma,\mathsf{pk})$: Given a representative $M = (M_i)_{i\in[\ell]} \in (\mathbb{G}_1^*)^\ell$ of equivalence class $[M]_{\mathcal{R}}$, a signature $\sigma = (Z, Y, \hat{Y}) \in \mathbb{G}_1 \times \mathbb{G}_1^* \times \mathbb{G}_2^*$ and public key $\mathsf{pk} = (\hat{X}_i)_{i\in[\ell]}$, check whether

$$\prod_{i\in[\ell]} e(M_i, \hat{X}_i) \stackrel{?}{=} e(Z, \hat{Y}) \qquad \wedge \qquad e(Y, \hat{P}) \stackrel{?}{=} e(P, \hat{Y}),$$

and if this holds, output true and false otherwise.

$\mathsf{ChgRep}_{\mathcal{R}}(\mathsf{M}, \sigma, \mu, \mathsf{pk})$: On input a representative $M = (M_i)_{i\in[\ell]} \in (\mathbb{G}_1^*)^\ell$ of equivalence class $[M]_{\mathcal{R}}$, a signature $\sigma = (Z, Y, \hat{Y})$, a randomization factor $\mu \in \mathbb{Z}_p^*$ and public key pk, return $\perp$ if $\mathsf{Verify}_{\mathcal{R}}(M, \sigma, \mathsf{pk}) =$ false. Otherwise pick $\psi \xleftarrow{R} \mathbb{Z}_p^*$ and return $(\mu \cdot M, \sigma')$ with $\sigma' \leftarrow (\psi\mu Z, \frac{1}{\psi} Y, \frac{1}{\psi}\hat{Y})$.

$\mathsf{VKey}_{\mathcal{R}}(\mathsf{sk}, \mathsf{pk})$: Given $\mathsf{sk} = (x_i)_{i\in[\ell]} \in (\mathbb{Z}_p^*)^\ell$ and $\mathsf{pk} = (\hat{X}_i)_{i\in[\ell]} \in (\mathbb{G}_2^*)^\ell$, output true if $x_i\hat{P} \stackrel{?}{=} \hat{X}_i\ \forall i \in [\ell]$ and false otherwise.

Figure 3.3: Construction of an SPS-EQ / FHS Scheme [FHS18]. Members of $\mathbb{G}_2$ are marked with a hat (i.e. $\hat{P}$).
BGGen: takes a security parameter 1^κ, outputs a bilinear group BG = (p,$\mathbb{G}_1$,$\mathbb{G}_2$,$\mathbb{G}_T$,e,P,$\hat{P}$) consisting of three prime-order p groups $\mathbb{G}_1 = \langle P \rangle$, $\mathbb{G}_2 = \langle \hat{P} \rangle$ and $\mathbb{G}_T$ with $\log_2 \mathrm{p} = \lceil \kappa \rceil$ and a pairing e: $\mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$.

To create a group signature, members randomize this certificate and create a zero-knowledge proof of knowledge (ZKPoK) that proofs knowledge of the used randomization factor. This randomized membership certificate is still valid under the public key of the issuing authority but is indistinguishable from a random signature-message pair. The ZKPoK proves that the signer is in possession of a non-randomized membership certificate. By using a Fiat-Shamir heuristic (2.11.4) for creating the proof, it is possible to turn it into a signature of knowledge on an arbitrary message.

3.7.2 Implementation Details

Similar to our implementation of the [FHS18] scheme, we created a Java implementation of the group signature scheme utilizing ECCelerate [AC19]. We used the same BN curve with a base field size of 448bit to create our bilinear group environment.

The original paper gives two different versions of the group signature scheme. One is CCA2 anonymous, and the other one is a simpler, more efficient CPA anonymous version. We decided that for our use case, the CPA version would be more suitable because of performance reasons.

To give a better understanding of the group signature scheme, we decided to explain how some of the notation used in the full construction relates to our instantiation.

Notation

- Members of $\mathbb{G}_2$ are marked with a hat (i.e. $\hat{P}$).
- BG = (p,$\mathbb{G}_1$,$\mathbb{G}_2$,$\mathbb{G}_T$,e,P,$\hat{P}$)
- κ refers to the security parameter (Base field size of the curve in bits).
- $\mathcal{R}$ refers to operations related to the FHS signature scheme.
- $_O$ and Ω refers to operations related to El-Gamal encryption (2.9).
- Σ refers to operations related to ECDSA (2.13).
- Π and τ refers to operations related to proofs of knowledge (2.11).

The meaning of σ sometimes changes depending on the context. In the context of a regular signature scheme (Σ) it refers to the signature. In the context of SPS-EQ ($\mathcal{R}$) signatures, it refers to only the signature part of a message-signature tuple. However, in the context of randomizing group signing keys, it refers to the whole message-signature tuple.

The meaning of crs in the function GKGen() is "common reference string" (2.1.2), which in this context means doing a protocol that ensures that every participant in this scheme has access to the correct public curve parameters and public keys.

To further improve understanding of the scheme, we will explain the operations Sign, Vrf, Open, and Judge in the following paragraphs in more detail.

Sign

To sign a message, the group member randomizes his *gsk* of the form (rP, P) with a random scalar p as an argument. This results in a message-signature pair in which the message part has the form (prP, pP). He then creates a ZKPoK to prove that he knows the factor p in pP. Using a Fiat-Shamir heuristic (2.11.4) he puts the message (the one that is to be signed, not the message of the membership certificate message-signature pair) into the hash of the proof, therefore creating a group signature on this message.

Verify

Verifying a signed message is simply verifying that the membership certificate verifies against the public key of the issuing authority and verifying the validity of the accompanying ZKPoK together with the message.

Open

To open a signature, for every group member the opening authority decrypts the ciphertext, that the member initially submitted during the join procedure, in order to get access to $\hat{R} \equiv r\hat{P}$. Remember that the message part of a randomized membership certificate has the form (prP, pP). The opener uses a pairing (2.6) to check whether e(prP, $\hat{P}$) $\equiv$ e(pP, $\hat{R}$). If that is the case, the identity of the group member has been found.

Judge

In the use case of our payment system, we did not have the need for verifiable openings. Therefore we opted to omit the implementation of the Judge function for performance reasons. If one were to implement it, one would have to make the following changes to our implementation of the group signature:

- During the Join procedure: add another ciphertext encrypting e(P, $\hat{P}$)^r using a **billinear version of ElGamal** encryption. It works in $\mathbb{G}_T$ which is a multiplicative group, hence the multiplicative notation:
  - **KeyGen:** x $\xleftarrow{R}$ $\mathbb{Z}_p$, pk = e(P,$\hat{P}$)^x , $sk = \hat{P}^x$
  - **Enc(M** $\in \mathbb{G}_\mathbb{T}$**):** k $\xleftarrow{R}$ $\mathbb{Z}_p$, C = (C$_1$, C$_2$) = (P^k, M $\cdot$ e(P^k, $\hat{P}$) $\cdot$ pk)
  - **Dec(C$_1$, C$_2$):** M $\leftarrow$ C$_2$ $\cdot$ e(C$_1$, sk)$^{-1}$

- During the open procedure one would have create a Groth-Sahai proof [GS08] of the following statements. Keep in mind that σ is the randomized membership certificate and $\sigma[1]$ is of the form (prP, pP). Underlined values are secret values of which knowledge of is being proven.
  - C$_2$ $\cdot$ e(C$_1$, $\underline{sk}$)$^{-1}$ $\cdot$ e(P, $\underline{\hat{R}}$)$^{-1}$ = 1
  - e($\sigma[1][1], P$)$\cdot$ e($\sigma[1][2], \underline{\hat{R}}$))$^{-1}$ = 1
  - e(P, $\underline{sk}$) = pk

The implementation of the Judge function would then need to verify this proof.

GKGen(1^κ): Run $\mathsf{BG} \leftarrow \mathsf{BGGen}_{\mathcal{R}}(1^\kappa)$, $(\mathsf{sk}_{\mathcal{R}}, \mathsf{pk}_{\mathcal{R}}) \leftarrow \mathsf{KGen}_{\mathcal{R}}(\mathsf{BG},2)$, $(\mathsf{sk}_O, \mathsf{pk}_O) \leftarrow \Omega.\mathsf{KGen}(1^\kappa)$, $\mathsf{crs}_J \leftarrow \Pi.\mathsf{Setup}(1^\kappa)$, $\mathsf{crs}_O \leftarrow \Pi.\mathsf{Setup}(1^\kappa)$, $\mathsf{crs}_S \leftarrow \mathsf{SoK.Setup}(1^\kappa)$, set $\mathsf{gpk} \leftarrow (\mathsf{pk}_{\mathcal{R}}, \mathsf{pk}_O, \mathsf{crs}_J, \mathsf{crs}_O, \mathsf{crs}_S)$, $\mathsf{ik} \leftarrow \mathsf{sk}_{\mathcal{R}}$, $\mathsf{ok} \leftarrow \mathsf{sk}_O$ and return $(\mathsf{gpk}, \mathsf{ik}, \mathsf{ok})$.
UKGen(1^κ): Return $(\mathsf{usk}_i, \mathsf{upk}_i) \leftarrow \Sigma.\mathsf{KGen}(1^\kappa)$.

Join$^{(1)}$(gpk, usk$_i$, upk$_i$): Choose $\mathsf{q},\mathsf{r} \xleftarrow{R} \mathbb{Z}_p^*$, set $(U_i, Q) \leftarrow (r \cdot qP, qP)$, and output $\mathsf{M}_J \leftarrow ((U_i, Q), \hat{C}_{J_i}, \sigma_{J_i}, \pi_{J_i})$ and $\mathsf{st} \leftarrow (\mathsf{gpk}, q, U_i, Q)$, where

$$\hat{C}_{J_i} \leftarrow \Omega.\mathsf{Enc}(\mathsf{pk}_O, r\hat{P}; \omega),$$
$$\sigma_{J_i} \leftarrow \Sigma.\mathsf{Sign}(\mathsf{usk}_i, \hat{C}_{J_i}),$$
$$\pi_{J_i} \leftarrow \Pi.\mathsf{Proof}(\mathsf{crs}_J, (\mathsf{U}_i, Q, \hat{C}_{J_i}, pk_O), (\mathsf{r}, \omega)).$$

Issue(gpk, ik, i, upk$_i$, reg): Receive $\mathsf{M}_J = ((U_i, Q), \hat{C}_{J_i}, \sigma_{J_i}, \pi_{J_i}) = 1$, and return reg and send σ' to user i, where:

$$\mathsf{reg}_i \leftarrow (\hat{C}_{J_i}, \sigma_{J_i}), \sigma' \leftarrow \mathsf{Sign}_{\mathcal{R}}((U_i, Q), sk_{\mathcal{R}}),$$

if $\Pi.\mathsf{Vrf}(\mathsf{crs}_J, (U_i, Q, \hat{C}_{J_i}, pk_O), \pi_{J_i}) = 1 \wedge \Sigma.\mathsf{Vrf}(\mathsf{upk}_i, \hat{C}_{J_i}, \sigma_{J_i}) = 1$, and return $\perp$ otherwise.

Join$^{(2)}$(st, σ'): Parse st as $(\mathsf{gpk}, \mathsf{q}, \mathsf{U}_i, \mathsf{Q})$ and return gsk_i, where

$$\mathsf{gsk}_i = ((\mathsf{rP}, \mathsf{P}), \sigma), \leftarrow \mathsf{ChgRep}_{\mathcal{R}}((\mathsf{U}_i,\mathsf{Q}), \sigma', q^{-1}, pk_{\mathcal{R}}),$$

if $\mathsf{Vrf}_{\mathcal{R}}((\mathsf{U}_i, \mathsf{Q}), \sigma', \mathsf{pk}_{\mathcal{R}}) = 1$, and return $\perp$ otherwise.

Sign(gpk, gsk$_i$, m): Choose $\mathsf{p} \xleftarrow{R} \mathbb{Z}_p^*$, and return $\sigma \leftarrow (\sigma_1, \sigma_2)$. where

$$\sigma_1 \leftarrow \mathsf{ChgRep}_{\mathcal{R}}(gsk_i, p, pk_{\mathcal{R}}), \sigma_2 \leftarrow SoK.\mathsf{Sign}(crs_S, (P, \sigma_1[1][2]), p, \sigma_1||m).$$

Vrf(gpk, m, σ): Return 1 if the following holds, and 0 otherwise:

$$\mathsf{Vrf}_{\mathcal{R}}(\sigma_1, pk_{\mathcal{R}}) = 1 \wedge SoK.\mathsf{Vrf}(crs_S, (P, \sigma_1[1][2]), \sigma_1||m, \sigma_2) = 1.$$

Open (gpk, ok, reg, m, σ): Parse σ $as(\sigma_1, \sigma_2)$, and ok as sk_O. Obtain the lowest index i, so that it holds for $(\hat{C}_{J_i}, \sigma_{J_i}) \leftarrow reg_i$ that $\hat{R} \leftarrow \Omega.\mathsf{Dec}(sk_O, \hat{C}_{J_i})$ and $e(\sigma_1[1][1], \hat{P}) = e(\sigma_1[1][2], \hat{R})$. Return (i, τ) and $\perp$ if no such entry exists, where

$$\tau \leftarrow (\pi_O, \hat{C}_{J_i}, \sigma_{J_i}), \text{ and } \pi_O \leftarrow \Pi.\mathsf{Proof}(crs_O, (\hat{C}_{J_i}, pk_O.\sigma), (sk_O, \hat{R})).$$

Judge(gpk, m, σ, i, upk$_i$, τ): Parse τ as $(\pi_O, \hat{C}_{J_i}, \sigma_{J_i})$, and return 1 if the following holds and 0 otherwise:

$$\Sigma.\mathsf{Vrf}(\mathsf{upk}_i, \hat{C}_{J_i}, \sigma_{J_i}) = 1 \wedge \Pi.\mathsf{Vrf}(\mathsf{crs}_O, (\hat{C}_{J_i}, pk_O, \sigma), \pi_O) = 1.$$

Figure 3.4: Derler-Slamanig group signature scheme [DS18]. Depending on context the meaning of σ changes. In the context of a regular signature scheme (Σ) it refers to the signature. In the context of SPS-EQ ($\mathcal{R}$) signatures it refers to only the signature of the message-signature tuple, however, in the context of randomizing group signing keys it refers to the whole message-signature tuple.

Chapter 4: Implementation & Performance Results

In this chapter, we will give an overview of the implementations that were created as part of this thesis. We will also discuss our results regarding the feasibility of our proposed payment system when using mobile devices.

The way our signature scheme implementations are designed to be used is modeled after the Java Cryptography Architecture [Ora19]. In the following listing we give an example from the Android cryptography manual [Goo19a]:

```
byte[] message = ...;
PrivateKey key = ...;
Signature s = Signature.getInstance("SHA256withECDSA");
s.initSign(key);
s.update(message);
byte[] signature = s.sign();
//-------------------------------------------------------
byte[] message = ...;
byte[] signature = ...;
PublicKey key = ...;
Signature s = Signature.getInstance("SHA256withECDSA");
s.initVerify(key);
s.update(message);
boolean valid = s.verify(signature);
```

4.1 SPS-EQ / FHS Signature Scheme

Our chosen group signature scheme uses the [FHS18] signature scheme as a base (also known as SPS-EQ). For all of our bilinear group elliptic curve operations, we chose a Baretto-Naehrig curve [BN05] (BN) with a base field size of 448 bit. We used the ECCelerate library [AC19], a cryptography library written entirely in Java.

Since we had to implement this signature scheme for our group signature in any case, we also decided to implement some functionality that allows this signature scheme to be used as a regular signature scheme on messages of arbitrary form. This is accomplished by hashing the message to an EC-point Q and using that point to create a new message vector $M = (Q, P) \in \mathbb{G}_1^2$ and signing it. In practice, there is not much utility in doing that though, because other regular signature schemes like ECDSA (2.13) are much simpler and more efficient.

4.1.1 Usage

Using the scheme to sign messages in the form of ECPoint vectors

```
//signing an ECPoint message with vector length l = 2
SpseqKeyGenerator kpg = new SpseqKeyGenerator(448);
SpseqKeyPair kp = kpg.generateKeyPair();

SpseqSignature s = new SpseqSignature();
s.initSign(kp);
ECPoint[] message = ...; //message vector with length 2
SignedEqClass message_signature_pair = s.sign(message);

s.initVerify(kp.getPublicKey());
boolean result = s.verify(message_signature_pair);
System.out.println("Result : " + result);

//alternatively
ECPoint[] ecSignature = message_signature_pair.getSignature();
s.update(message);
result = s.verify(ecSignature);
System.out.println("Result : " + result);
```

Using it as a regular signature scheme

```
SpseqKeyGenerator kpg = new SpseqKeyGenerator(448);
SpseqKeyPair kp = kpg.generateKeyPair();

SpseqSignature s = new SpseqSignature();
s.initSign(kp);
s.update("This is a test".getBytes());
byte[] signature = s.sign();
```

```
s.initVerify(kp.getPublicKey());
s.update("This is a test".getBytes());
boolean result = s.verify(signature);
```

4.2 Derler-Slamanig Group Signature Scheme

Similar to our implementation of the [FHS18] scheme, we created a Java implementation of the group signature scheme utilizing ECCelerate [AC19]. We used the same BN curve with a base field size of 448bit to create our bilinear group environment. We implemented the CPA version of the scheme, as described in the original paper [DS18].

4.2.1 Usage

For the usage of our group signature implementation, we tried to keep the style and conventions of the Java Cryptography Architecture [Ora19] and adapted it to group signatures.

Creating a group

```
GroupKeyGenerator keyGenerator = new
   GroupKeyGenerator(sizeOfEllipticCurveBaseField);

//Contains group public key, issuing key, opening key
GroupKeyChain groupKeychain = keyGenerator.generateKeychain();
```

Joining a group (interactive protocol)

User perspective:

```
MessageJoin messageJoin = new MessageJoin(groupPublicKey);
byte[] groupSigningKeyBlank =
   issuingAuthority.issue(messageJoin);
GroupKeyGenerator keyGenerator = new
   GroupKeyGenerator(groupPublicKey);
GroupSigningKey gsk =
    keyGenerator.createGroupSigningKey(groupSigningKeyBlank,
       messageJoin);
```

Issuing authority perspective:

```
byte[] issue(MessageJoin messageJoin){
    GroupKeyGenerator keyGenerator = new
       GroupKeyGenerator(groupPublicKey, issuingKey);
    IssueResult issueResult = keyGenerator.issue(messageJoin);
    registrations.add(issueResult.registration);
    return issueResult.groupSigningKeyBlank;
}
```

Signing a message on behalf of the group

```
GroupSignature gs = new GroupSignature();
gs.initSign(groupSigningKey, groupPublicKey);
gs.update("message".getBytes());
byte[] signature = gs.sign();
```

Verifying a group signature

```
GroupSignature gs_verify = new GroupSignature();
gs_verify.initVerify(groupPublicKey);
gs_verify.update("message".getBytes());
boolean result = gs_verify.verify(signature);
```

Opening a group signature

```
GroupSignature gs_open = new GroupSignature();
gs_open.initOpen(openingKey, registrations, groupPublicKey);

//contains user ID and proof of opening
Opening opening = gs_open.open(signature);
```

4.3 Payment System

The implementation of our payment system consists of a Java backend, complete with a customer and employee Android app. The backend can simulate a shop, the payment processor, and the bank. It is also capable of simulating a customer and an employee for testing purposes. Communication between phones is done through NFC and between other entities through an MQTT [209] server.

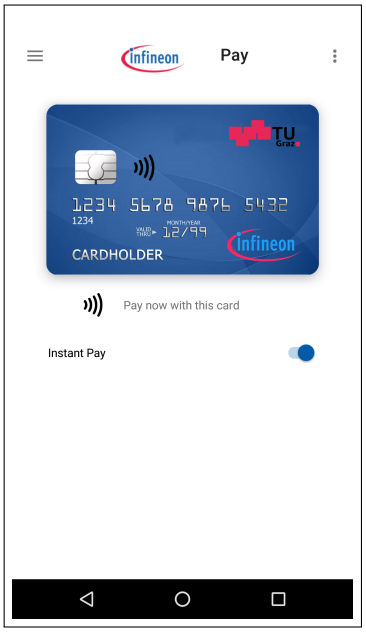


Figure 4.1: Customer Android App

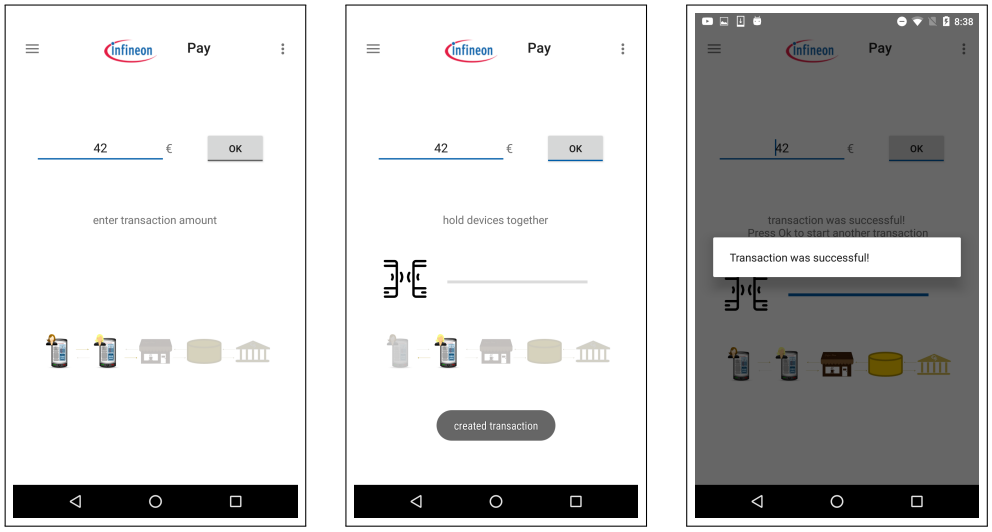
4.3.1 Customer App

A payment app that stores the group signing keys of the user and is able to create new ones using the join/issue protocol with the bank from the backend. It signs transactions received via NFC from the employee and responds with the signed transaction. It is also capable of simulating different users, visualized as different credit cards. There is also an "instant pay" toggle to switch between different modes. In the instant pay mode, the app acts like an NFC enabled credit card and will instantly respond to any NFC request with the signed transaction. If instant pay is turned off, a dialog that asks the user to confirm the transaction will pop up when a transaction is received.

4.3.2 Employee App

An app that stores the signing keys of the employee, allows him to create new transactions and send them to the device of the customer via NFC. After having received the response, it sends the signed transaction to the backend, which will respond with the transaction status information.

The app also includes a transaction visualizer, which takes real-time data from the backend and displays the current state of the transaction, including the exact actor that is currently processing the transaction.



(a) Creating a transaction **(b)** Transmitting signature **(c)** Transaction completed

Figure 4.2: Employee Android app

Coffee Shop

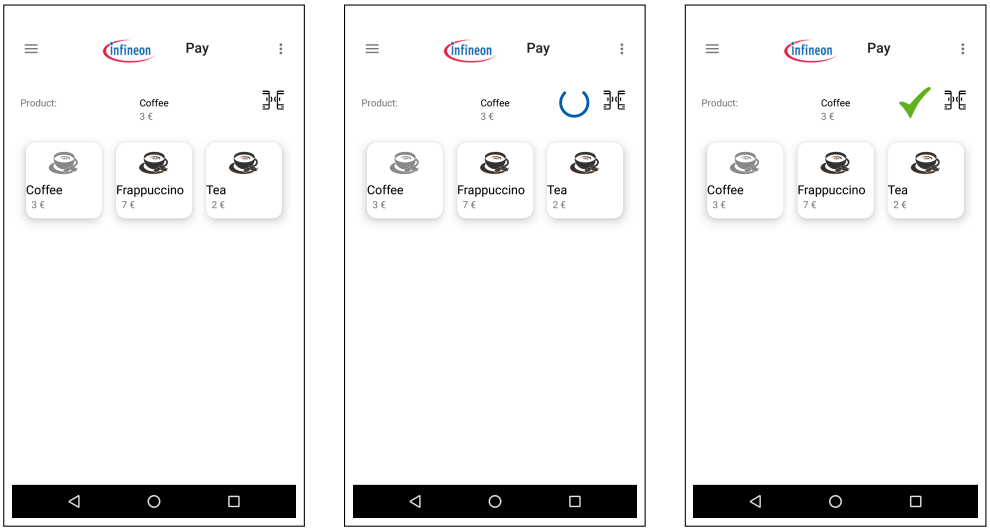
The employee app also includes a section that emulates a simplified payment device an employee in a coffee shop would use. The employee creates a transaction by choosing a product. The customer pays for the product by signing this transaction using the customer app via NFC.

4.3.3 Backend

The backend simulates the shop, the payment processor, and the bank, all in one program. Separation is achieved by having a distinct Java class for each entity and having those classes communicate with each other as if they were on different machines, through the MQTT broker. This allows for the simulation of the customer and the employee for testing purposes. The Android apps use the same classes and put an app around them. The actual payment system functionality is portable and platform-independent.

4.4 Performance of ECCelerate

The performance of our payment system can be estimated by looking at the computationally expensive ECC operations provided by the ECCelerate library. In the group signature scheme, as well as in many other schemes that use elliptic curve cryptography, the operations of point multiplication and pairings dominate the runtime of the whole scheme. This is



(a) Ordering a coffee **(b)** Transmitting **(c)** Completed

Figure 4.3: Example employee app for a coffee shop using our payment system.

Version	Codename	API	Distribution
4.4	KitKat	19	2.36%
5.0	Lollipop	21	2.78%
5.1		22	6.84%
6.0	Marshmallow	23	13.32%
7.0	Nougat	24	8.06%
7.1		25	5.53%
8.0	Oreo	26	9.98%
8.1		27	14.58%
9	Pie	28	35.25%

Table 4.1: Android version distribution October, 2019 [Sta19]

the reason why we created an ECCelerate benchmark (4.1) in Java and built an Android app 4.4 around it. This allows for performance comparisons between a wide range of devices, including both server-class hardware and mobile devices.

We ran our benchmark on a variety of devices 4.2, including two different Nexus 5 running a Snapdragon 800 processor. We upgraded one of them to Android 9.0 and compared it to the stock Nexus 5, which is still at Android 6.0 because it does not receive any updates anymore 4.1. In our results, we saw major performance improvement on the same hardware, simply from a software update. We attribute these efficiency gains to the improvements of the garbage collector in newer Android run time versions. Although this is just speculation, we suspect that the amount of BigInt allocations in ECCelerate might be a bottleneck.

256 bit / 384 bit	Nexus 5 Andr. 6.0	Nexus 5 Andr. 9.0	Nexus 5x Andr. 6.0	LG G4 Andr. 6.0	Pixel 3XL Andr. 10.0	i5-6300U Wind. 10
Mult. $\mathbb{G}_1$	152	150	73	86	46	4
Mult. $\mathbb{G}_2$	368	308	167	236	99	7
Pairing	946	666	400	615	222	16
Mult. $\mathbb{G}_1$	396	329	191	262	110	7
Mult. $\mathbb{G}_2$	957	759	479	694	271	22
Pairing	1928	1461	984	1533	543	28

Table 4.2: Values in [ms]. Benchmark for elliptic curve operations using the ECCelerate library for two Barreto-Naehrig curves with different base field sizes

Listing 4.1: Example benchmarking code

```
Pairing pairing = AtePairingOverBarretoNaehrigCurveFactory.
    getPairing(PairingTypes.TYPE_3, securityParameter);
ECPoint testG1 = pair.getGroup1().hashToPoint(*Random*);
ECPoint testG2 = pair.getGroup2().hashToPoint(*Random*);

long startTime = System.currentTimeMillis();
    for(int i = 0; i < runs; i++){
        pairing.pair(testG1, testG2);
    }
long elapsedTime = System.currentTimeMillis() - startTime;

return elapsedTime / runs;
```

4.5 Performance of Payment System

The performance analysis of our payment system can be split into the interactive user interaction part and the total transaction time. The user interaction part is much more time-critical, and the devices involved in this part are resource-constrained. We focused our efforts on optimizing the performance of this part of the transaction.

In order to get an insight into the performance of the user interaction, we broke it down into the most expensive operations:

- employee signs transaction

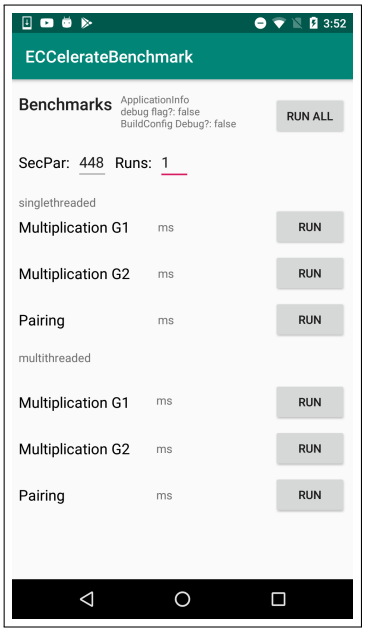


Figure 4.4: ECCelerate benchmark Android app

- transaction and employee signature get transferred over NFC
- user verifies employee signature
- user signs transaction
- transaction and user signature get transferred over NFC

In total, during the interactive part of our payment system, we have to create 2 signatures, verify 1 signature and transfer 2 signatures worth of data + transaction information over NFC. The dominant operations of creating a signature can be prepared beforehand, while the app is idling in the background, for instance, and can be left out of the interactive performance calculation.

What is left are 5 pairing calculations and 2 multiplications in $\mathbb{G}_1$ for the verification of the employee's signature, plus the time it takes to send this data over NFC. From table 4.5 we can see the size of our signatures and from table 4.3 we can see the performance of NFC transmissions. In table 4.2 we can see the performance of the ECC operations using the ECCelerate [AC19] library on various devices.

Even though almost all operations can be done concurrently, the time it takes to verify a signature on a mobile device will take a few seconds but is still practical in certain use cases. In the case of our user app, the user could get prompted with a confirm dialog as soon as the transaction is transferred. In the time it takes for the user to confirm the transaction, the signature of the employee is already verified in the background.

There are also use cases for our payment system where the authenticity of the transaction source is not an issue for the user, or it could be established using different means. Examples

Transmission size [B]	LG G4 to Nexus 5 [ms]	Pixel 3XL to Nexus 5 [ms]	Pixel 3XL to LG G4 [ms]
20	20	18	20
512	90	114	84
1024	143	140	133
2048	250	285	221
65200	6501	6980	6095

Table 4.3: Speed of NFC transmissions between two Android phones, with the target phone acting as an NFC tag using Host Card Emulation

for that would be ticket gates for public transport or proper payment terminals in shops. We believe that this is a reasonable assumption and improves performance significantly. If the transaction source authentication step could be omitted, the **total computational cost during the interactive part of the payment protocol becomes essentially negligible.** The most expensive operation that would be left is the transfer of the signature over NFC.

In our testing, the interactive part of our payment system takes about **200ms**. This includes transmitting the transaction over NFC to the customer's device and sending the signature back (this omits sending and verifying the employee's signature). We used a security parameter of 448bit. The performance of the system as a whole depends on different factors, including computing power of backend and the physical distance between the servers. In our experimental setup, the shop, the payment provider, as well as the bank, were simulated using a Java program running on our test laptop (Intel i5-6300U). To simulate real-life network delays, we limited all actors to only communicating with each other using an external MQTT [209] server, which was located 700km from our test site. Our final performance measurements resulted in a total transaction time of around **1500ms**, an acceptable time for many use-cases. This performance is close to a worst case scenario and we are confident that this performance can be improved significantly with faster backend hardware and a more efficient network setup.

4.5.1 Signature Size

The size of our group signatures has a significant effect on our payment system performance. Signatures in the [DS18] scheme consist of elements in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{Z}_p$. Using the point encoding and compression feature of ECCelerate, we already get group element sizes that are close to the theoretical minimum size. The method that we used to pack them into a portable package was Concise Binary Object Representation (CBOR). Table 4.4 shows the compressed size of different group elements at different curve sizes. Calculating the signature size by adding up those element sizes gives us the minimum achievable signature

Curve Size / Base field size [b]	e $\in \mathbb{G}_1$[B]	e $\in \mathbb{G}_2$[B]	e $\in \mathbb{Z}_P$ [B]
128	17	33	16
256	33	65	33
384	49	97	49
448	57	113	56

Table 4.4: Element size in bytes using ECCelerate point compression

Curve Size [b]	Elements	Total	CBOR package size = signature size
128	$4 \cdot 17 + 1 \cdot 33 + 2 \cdot 16$	133	143
256	$4 \cdot 33 + 1 \cdot 65 + 2 \cdot 33$	263	279
384	$4 \cdot 49 + 1 \cdot 97 + 2 \cdot 49$	391	407
448	$4 \cdot 57 + 1 \cdot 113 + 2 \cdot 56$	453	469

Table 4.5: CBOR overhead. All numbers in bytes. Signatures have the size: $4 \cdot \mathbb{G}_1 + 1 \cdot \mathbb{G}_2 + 2 \cdot \mathbb{Z}_P$

size using ECCelerate point compression. Packing all elements into a portable CBOR object only bears a small overhead, as can be seen in table 4.5.

4.6 Performance Discussion

The viability of a digital payment system such as ours is highly dependent on its performance. With our implementation, we have demonstrated that even a platform-independent Java version already achieves acceptable performance on modern smartphones. We believe that we have shown that group signatures are a viable privacy friendly alternative to traditional authentication schemes.

Optimizing elliptic curve operations for specific devices might lead to a significant increase in performance. A large benefit of using the ECCelerate [AC19] library, is that it is written entirely in Java. This allows for rapid prototype development. Because of the portable nature of Java, it also allows for easy deployment in various different devices, especially Android smartphones. However, this approach also has its drawbacks, mainly performance.

Another popular pairing library that is often cited in the literature is the Pairing-Based Cryptography Library (PBC), which is written entirely in C and is built on the GMP library. It is older and more established than ECCelerate, and it is free under GNU Lesser General Public License (ECCelerate is not free for general use). For this reason, we evaluated pairing times on our test PC, equipped with an i5-6300U, and had the following results:

Base Field Size [b]	PBC [ms]	ECCelerate [ms]
256	93	16
384	196	28
448	268	42

Table 4.6: Pairing performance of ECCelerate and PBC library on a i5-6300U on Windows 10. We used Barretto-Naehrig curves with the size of the base field given in bits.

As it turns out, even though PBC was written in C, ECCelerate was much faster. Currently, we are not aware of another publicly available pairing library with similar functionality that outperforms ECCelerate. We have strong reasons to believe that given enough resources it is possible to implement a commercial version of our payment system with even better performance by optimizing the pairing library that it uses.

Smartcards

We believe that it is feasible to implement the functionality of our customer app on a smartcard. Smartcards using ECC, like the ECDSA algorithm, have been around in Austria the form of ID-cards or health insurance cards since at least 2005. They use a special accelerator chip designed explicitly for ECC, which makes ECC arithmetic possible even on such a low powered device. The most expensive operation during the creation of an ECDSA signature is a single scalar multiplication. Given that the creation of a group signature using the [DS18] scheme takes 5 multiplications in $\mathbb{G}_1$ and one in $\mathbb{G}_2$ (which is roughly equivalent to 3 multiplications in $\mathbb{G}_1$) it is not unreasonable to assume, that creating such a group signature is at most 10x slower than creating an ECDSA signature. This leads us to believe that it would be possible to replace the customer Android app in our payment system with a smartcard and still stay within our time requirements.

Chapter 5: Post-Quantum Group Signatures

It is widely believed that a potential future quantum computer will be able to solve some of the hard problems that our current cryptography relies on efficiently, most notably integer factorization and the discrete logarithm problem. This chapter will give a brief introduction into post-quantum cryptography, the field of cryptography that assumes the existence of such a machine. We then delve into an analysis of post-quantum secure cryptosystems and their application to group signatures. The goal of this chapter is to give an overview of the current state of post-quantum groups signature schemes and their advantages, disadvantages, and possible applications. We also tried to find an answer to the question of whether there exists a post-quantum secure group signature scheme that we could use in our payment system. Unfortunately, because of the strict performance requirements of our use case, we had to answer this question in the negative.

5.1 Post-Quantum Cryptography

In 1994 Peter Shor invented an algorithm [Sho94] that could efficiently solve among others the integer-factorization problem, the discrete logarithm problem, and the ECC-discrete logarithm problem. Almost all widely used public-key cryptosystems at the time were relying on those fundamental mathematical problems for their security, with prominent examples being RSA, DSA, and in later years ECC. From that point in time, all of those cryptosystems had a publicly available, mathematically sound algorithm that would break all of their security promises. The only catch was that this algorithm needed to run on a powerful, not yet existing, quantum-computer.

The road to the completion of this machine is a long one, and quantum computers of today are still experimental and far away from posing an actual threat to modern cryptosystems. At the present day, the claim for the largest integer to have ever been factored on a quantum computer using Shor's algorithm seems to be 21 [Mar+12]. However, nobody can predict the speed of advancements in quantum-computers. Always having this in the back of one's mind is a state of existence that is unacceptable for many cryptographers and thus, a new

field in cryptography was born: post-quantum cryptography, the study of cryptographic algorithms that are secure against a hypothetical quantum computer.

5.2 Arguments for the Development of Post-Quantum Cryptography

Given that there already exist cryptosystems that are resistant against quantum computer attacks, why do we need to worry about the threat of quantum computers? After all, it is not even guaranteed that a powerful quantum computer will ever be built, and even on the off-chance that someone manages to build one, why not simply switch cryptosystems at that point in time?

The cryptographer Daniel J. Bernstein [Ber09] gave three compelling reasons for developing post-quantum cryptography sooner rather than later:

1. We need time to improve the efficiency of post-quantum cryptography.
2. We need time to build confidence in post-quantum cryptography.
3. We need time to improve the usability of post-quantum cryptography.

Maybe all of this preparation is unnecessary and maybe we can use RSA, DSA, and ECC indefinitely. However, if it suddenly turns out that users *do* need post-quantum cryptography, years of crucial research time will have been lost.

5.2.1 Efficiency

Currently, existing post-quantum cryptography is not yet competitive with regular cryptosystems in terms of signature size, signing + verification performance, and encryption + decryption performance. Suddenly switching to post-quantum cryptography would not be possible in terms of available computation power in toady's systems. Upgrading large servers that handle millions of requests would be prohibitively expensive, and small mobile systems, like smartcards and mobile phones, might not even have the necessary computing power in the first place.

5.2.2 Confidence

A familiar risk in cryptography is the possible existence of an unknown algorithm for a system that could either increase the efficiency of an attack or simply break it outright. The community invests a huge amount of resources in cryptanalysis, and sometimes cryptanalysts find devastating attacks, rendering a system completely useless. Prominent examples for that would be the Merkle-Hellman knapsack cryptosystem or the MD5 hash function. Other times, attacks are found that do not break a system but require them to

update their key-size. For example, improvements in integer factoring algorithms forced RSA to use larger key-sizes.

There are times when years of cryptanalysis of a cryptosystem do not bring siginficant improvements for any attack. The cryptographic community starts to gain confidence that the best possible attack has already been found or that a possible attacker will not be able to come up with anything better. This process takes time, and the sooner we start this process for post-quantum cryptosystems, the better.

5.2.3 Usability

Even if a cryptosystem is theoretically sound, finding the correct way to use it is sometimes a process that takes years to develop, with many potential pitfalls along the way. The RSA public-key cryptosystem, for instance, cannot simply be used as is, but instead, it needs proper randomization and padding standards. These standards also need to be analyzed separately in order to prevent attacks like the one on the "PKCS#1 v1.5" padding standard by Bleichenbacher [Ble98]. Since RSA is inefficient for longer messages, it is often used to encrypt a small key for a symmetric cipher, with which the original message is encrypted. Infrastructure and protocols like this need a long time to develop and standardize.

In addition to that, secure software implementations also need time to develop and years of scrutiny to build up trust. Standardized hardware implementations, like the ones that already exists for AES in Intel CPUs or specialized chips on smartphones, are also sometimes necessary to ensure efficient computation in many use cases. Post-quantum cryptography, just like the rest of cryptography, needs complete hybrid-systems, standards, and high-performance leak-resistant implementations.

5.3 Post-Quantum Algorithms

While Shor's algorithm breaks most of the prevalent public-key algorithms, there exist examples for public-key algorithms that are post-quantum as far back as 1978, with the McEliece public-key cryptosystem [McE78]. Most secret key algorithms are already thought to be secure against quantum-computer attacks [PC09], and there are whole classes of cryptosystems that rely on fundamental problems, which are unaffected by them. Daniel J. Bernstein briefly summarized them in the book "Post-Quantum Cryptography" [Ber09]:

- **Hash-based cryptography:** So far, limited to digital signatures. The classic example is Merkle's hash tree public-key signature system (1979), which builds upon one-time signatures like those by Lamport and Diffie.
- **Code-based cryptography:** Examples include the McEliece hidden Goppa-code public-key encryption system [McE78].

- **Lattice-based cryptography:** The example that gave traction to Lattice based cryptography is the Hoffstein-Pipher-Silverman "NTRU" public-key encryption system [HPS98].
- **Multivariate-quadaratic-equations cryptography:**
  One example would be Patarin's "HFE" public-key signature system [Pat96].
- **Secret-key cryptography:** The standard example is the Rijndael cipher [DR99], which was chosen to become the "AES" - advanced encryption standard.

All of these systems are believed to be resistant to attacks from classical computers and quantum computers alike. As far as we can tell, nobody has figured out a way to apply Shor's algorithm to any of these systems. There exists another quantum computer algorithm that can be applied to some of those systems: Grover's algorithm [Gro96]. However, the speedups from this algorithm are only quadratic, as opposed to the promised exponential speedup from Shor's algorithm. It is believed that simply doubling the key size in the applicable cryptosystems is sufficient to protect against this algorithm.

5.4 Post-Quantum Group Signatures

To the best of our knowledge, all group signature schemes known to date that are truly practical in terms of signature size, speed, and features, rely on hard mathematical problems, which are thought to be efficiently solvable with a quantum computer.

The study of group signatures resistant against quantum computer attacks has been very active in the last years, with more efficient schemes being proposed at a steady pace. The most promising approaches seem to be lattice-based, but code-based group signature schemes have also been proposed.

5.4.1 Lattices

At this point in time, the most promising research into post-quantum group signatures is lattice-based. Lattice-based cryptography is a generic term for constructions of cryptographic primitives using lattices. Many lattice-based constructions are thought to be resistant against attacks from both classical and quantum computers.

In table 5.1 we have compiled an overview of existing lattice-based group signature schemes. The earlier group signature constructions were limited to static groups with large signature and key sizes. Their signature sizes were also dependant on the size of the group, making it challenging to use them on a large scale. To the best of our knowledge, the scheme by Ling et al. [Lin+18] is the only lattice-based group signature scheme with a constant signature- and key size, i.e., a signature size that does not grow with the size of the group. It also supports dynamic enrollment. This makes it the most viable post-quantum group signature scheme for our payment system in terms of features. However, it still does not meet our size and speed requirements: The paper does not give any concrete numbers on size and

runtime, and it acknowledges that, as all known lattice-based group signatures, it is not truly practical yet. Even though the signatures are of constant size $\mathcal{O}(\lambda)$, due to a large poly-logarithmic factor contained in the $\mathcal{O}$ notation, the size is still too big to be useful in practice [Lin+18], according to the authors of the original paper. There do not seem to be any actual publicly available implementations to asses their claims and provide concrete numbers.

5.4.2 Code-Based

To the best of our knowledge, there currently exist two group signature schemes using code-based assumption: [Eze+15][Ala+17]. Both of their works were researched concurrently and independently, and [Ala+17] claims to have proposed the first code-based group signature. [Eze+15], acknowledging the work of [Ala+17] in the paper, claims to have proposed the first *provably* secure group signature scheme that relies *solely* on code-based assumptions. The [Ala+17] scheme also gives some concrete numbers for signature sizes, stating 2,5 MBytes large public keys and signature sizes of about 20 MBytes. Currently, this signature size makes them impractical for most use-cases.

5.4.3 Secret-key Cryptography - Symmetric Primitives

Most symmetric ciphers are already resistant against quantum computer attacks, provided one uses sufficiently large key sizes. Therefore, a group signature consisting of only symmetric primitives would also be a post-quantum group signature. However, at this point in time, no such construction is known. The construction that came the closest was [BEF19], of which an early version with the title "Post-Quantum Group Signatures from Symmetric Primitives" exists. Unfortunately, their definition of a group signature was missing an opening operation, and they subsequently changed the title to "Post-Quantum EPID Signatures from Symmetric Primitives.".

In fact, the existence of a group signature based on symmetric primitives might not be possible at all. Emura et. al [Emu+14] showed that a dynamic group signature with opening soundness implies public-key encryption, which by definition cannot be achieved with symmetric primitives alone.

Scheme	Sig Size	Group PK size	Signer SK size	functionality
[GKV10]	$\mathcal{O}(\lambda^2 \cdot N)$	$\mathcal{O}(\lambda^2 \cdot N)$	$\mathcal{O}(\lambda^2)$	static
[CNR12]	$\mathcal{O}(\lambda^2 \cdot N)$	$\mathcal{O}(\lambda^2)$	$\mathcal{O}(\lambda^2)$	static
[Lag+13]	$\mathcal{O}(\lambda \cdot \ell)$	$\mathcal{O}(\lambda^2 \cdot \ell)$	$\mathcal{O}(\lambda^2)$	static
[Lan+14]	$\mathcal{O}(\lambda \cdot \ell)$	$\mathcal{O}(\lambda^2 \cdot \ell)$	$\mathcal{O}(\lambda \cdot \ell)$	VLR
[NZZ15]	$\mathcal{O}(\lambda + \ell^2)$	$\mathcal{O}(\lambda^2 \cdot \ell^2)$	$\mathcal{O}(\lambda^2)$	static
[LNW15]	$\mathcal{O}(\lambda \cdot \ell)$	$\mathcal{O}(\lambda^2 \cdot \ell)$	$\mathcal{O}(\lambda)$	static
[Lib+16b]	$\mathcal{O}(\lambda \cdot \ell)$	$\mathcal{O}(\lambda^2 + \lambda \cdot \ell)$	$\mathcal{O}(\lambda \cdot \ell)$	static
[Lib+16a]	$\mathcal{O}(\lambda \cdot \ell)$	$\mathcal{O}(\lambda^2 \cdot \ell)$	$\mathcal{O}(\lambda)$	dynamic enrollment
[LMN16]	$\mathcal{O}(\lambda \cdot \ell)$	$\mathcal{O}(\lambda^2 \cdot \ell)$	$\mathcal{O}(\lambda)$	MDO
[Lin+19]	$\mathcal{O}(\lambda \cdot \ell)$	$\mathcal{O}(\lambda^2 + \lambda \cdot \ell)$	$\mathcal{O}(\lambda) + \ell$	fully dynamic
[Lin+18]	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	dynamic enrollment

Table 5.1: Comparison of lattice-based group signatures, in terms of asymptotic efficiency and functionality [Lin+18]. $\boldsymbol{\lambda}$ represents the security parameter and $\mathbf{N = 2^{\ell}}$ the maximum expected number of group users.
VLR = Verifier Local Revocation; **MDO** = Message Dependent Opening

Chapter 6: Conclusion

In this thesis, we presented a novel privacy-friendly payment scheme from group signatures and demonstrated its practicality. We created a Java implementation of the [FHS18] signature scheme and used it as a base for our Java implementation of the [DS18] group signature scheme. We also looked into post-quantum group signatures and have concluded that even the most efficient post-quantum group signature schemes are not yet practical in terms of speed and signature size, including those that don't even have the features our payment system requires.

We demonstrated that modern group signature schemes are efficient enough to be used in practice on mobile devices by instantiating our payment system using it. This also demonstrates that group signatures, in general, are not only a theoretical concept but can be applied in practice for use cases like access control or public transport ticketing today. Our implementation has a potentially significant impact on the implementation and choice of technology of future privacy-enhancing technologies.

6.1 Open Issues and Future Work

In our performance analysis, we briefly mentioned the possibility of implementing the customer part of our payment system on a smartcard. Based on the prevalence of other smartcards using ECC, we believe that such an implementation would be feasible and would serve as an excellent demonstration for the potential of future PETs.

If and when post-quantum group signature schemes were to become efficient enough to be used on smartphones (or if smartphones become powerful enough), then using such a scheme would be a drop-in upgrade for our payment-system and other privacy enhancing technologies like it.

Development in conventional, non-post quantum group signatures is also still ongoing, and we have yet to see the efficient do-it-all group signature that combines all desirable group signature features. In fact, researchers are still coming up with new features to solve more usability problems in different use cases. Nevertheless, even the basic concept of group signatures on its own has huge potential for future PETs. Note that the newest, most efficient group signature schemes have only first been published a few years ago and are

yet to gain trust or visibility of the public. Real-world implementations are still few and far between, and the awareness of the potential of group signatures for PETs needs to increase. Our work demonstrated the potential and practicality of group signatures today by implementing a common real-world use case and showed that a lot of other existing use-cases could be implemented in a privacy-preserving way without compromising on usability.

Bibliography

[209] I. 20922:2016. *Information technology — Message Queuing Telemetry Transport (MQTT) v3.1.1*. ISO, Geneva, Switzerland (cit. on pp. 60, 66).

[Abe+15] M. Abe, M. Kohlweiss, M. Ohkubo, and M. Tibouchi. "Fully structure-preserving signatures and shrinking commitments." In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2015, pp. 35–65 (cit. on p. 34).

[AC19] I. of Applied Information Processing and Communications. *ECCelerate*. 2019. URL: https://jce.iaik.tugraz.at/sic/Products/Core_Crypto_Toolkits/ECCelerate (visited on 08/01/2019) (cit. on pp. 51, 53, 57, 59, 65, 67).

[Ala+17] Q. Alamélou, O. Blazy, S. Cauchie, and P. Gaborit. "A code-based group signature scheme." In: *Designs, Codes and Cryptography* 82.1-2 (2017), pp. 469–493 (cit. on p. 73).

[And19a] C. Andrea. *Elliptic Curve Cryptography: a gentle introduction*. Creative Commons Attribution 4.0 International License: https://creativecommons.org/licenses/by/4.0/. 2019. URL: https://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/ (visited on 08/29/2019) (cit. on pp. 12, 14, 15).

[And19b] C. Andrea. *Elliptic Curve Cryptography: a gentle introduction*. Creative Commons Attribution 4.0 International License: https://creativecommons.org/licenses/by/4.0/. 2019. URL: https://andrea.corbellini.name/2015/05/23/elliptic-curve-cryptography-finite-fields-and-discrete-logarithms/ (visited on 08/29/2019) (cit. on pp. 12, 16–19).

[And19c] C. Andrea. *Elliptic Curve Cryptography: a gentle introduction*. Creative Commons Attribution 4.0 International License: https://creativecommons.org/licenses/by/4.0/. 2019. URL: https://andrea.corbellini.name/2015/05/30/elliptic-curve-cryptography-ecdh-and-ecdsa/ (visited on 08/29/2019) (cit. on p. 12).

[And19d] C. Andrea. *Elliptic Curve Cryptography: a gentle introduction*. Creative Commons Attribution 4.0 International License: https://creativecommons.org/licenses/by/4.0/. 2019. URL: https://andrea.corbellini.name/2015/06/08/elliptic-curve-cryptography-breaking-security-and-a-comparison-with-rsa/ (visited on 08/29/2019) (cit. on p. 12).

[App19] Apple. *Apple Pay*. 2019. URL: https://www.apple.com/at/apple-pay/ (visited on 11/07/2019) (cit. on p. 2).

[Bac+18] M. Backes, L. Hanzlik, K. Kluczniak, and J. Schneider. "Signatures with Flexible Public Key: Introducing Equivalence Classes for Public Keys." In: *ASIACRYPT (2)*. Vol. 11273. Lecture Notes in Computer Science. Springer, 2018, pp. 405–434 (cit. on p. 50).

[BBS04] D. Boneh, X. Boyen, and H. Shacham. "Short group signatures." In: *Annual International Cryptology Conference*. Springer. 2004, pp. 41–55 (cit. on p. 50).

[BDZ03] F. Bao, R. H. Deng, and H. Zhu. "Variations of diffie-hellman problem." In: *International conference on information and communications security*. Springer. 2003, pp. 301–312 (cit. on p. 23).

[BEF19] D. Boneh, S. Eskandarian, and B. Fisch. "Post-quantum EPID Signatures from Symmetric Primitives." In: *CT-RSA*. Vol. 11405. Lecture Notes in Computer Science. Springer, 2019, pp. 251–271 (cit. on p. 73).

[Ber+15] D. J. Bernstein, T. Chou, C. Chuengsatiansup, A. Hülsing, E. Lambooij, T. Lange, R. Niederhagen, and C. Van Vredendaal. "How to Manipulate Curve Standards: A White Paper for the Black Hat http://bada55. cr. yp. to." In: *International Conference on Research in Security Standardisation*. Springer. 2015, pp. 109–139 (cit. on p. 21).

[Ber09] D. J. Bernstein. "Introduction to post-quantum cryptography." In: *Post-quantum cryptography*. Springer, 2009, pp. 1–14 (cit. on pp. 70, 71).

[Bic+10] P. Bichsel, J. Camenisch, G. Neven, N. P. Smart, and B. Warinschi. "Get shorty via group signatures without encryption." In: *International Conference on Security and Cryptography for Networks*. Springer. 2010, pp. 381–398 (cit. on p. 49).

[Bit19] Bitcoin.com. *Bitcoin Hashrate*. 2019. URL: https://www.blockchain.com/de/charts/hash-rate (visited on 11/10/2019) (cit. on p. 10).

[BL19] D. J. Bernstein and T. Lange. *SafeCurves: choosing safe curves for elliptic-curve cryptography*. 2019. URL: https://safecurves.cr.yp.to (visited on 08/30/2019) (cit. on p. 21).

[Ble98] D. Bleichenbacher. "Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS# 1." In: *Annual International Cryptology Conference*. Springer. 1998, pp. 1–12 (cit. on p. 71).

[BN05] P. S. Barreto and M. Naehrig. "Pairing-friendly elliptic curves of prime order." In: *International Workshop on Selected Areas in Cryptography*. Springer. 2005, pp. 319–331 (cit. on pp. 51, 57).

[BS04] D. Boneh and H. Shacham. "Group signatures with verifier-local revocation." In: *Proceedings of the 11th ACM conference on Computer and communications security*. ACM. 2004, pp. 168–177 (cit. on p. 39).

[CGH98] R. Cannetti, O. Goldreich, and S. Halevi. "The random oracle methodology, revisited (preliminary version)." In: *Proc. 30th Annual ACM Symp. On Theory of Computing, Perugia, Italy, ACM Press*. 1998, pp. 209–218 (cit. on p. 8).

[CH91] D. Chaum and E. van Heyst. "Group Signatures." In: *Advances in Cryptology — EUROCRYPT '91*. Ed. by D. W. Davies. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 257–265. ISBN: 978-3-540-46416-7 (cit. on pp. 5, 34).

[CL04] J. Camenisch and A. Lysyanskaya. "Signature schemes and anonymous credentials from bilinear maps." In: *Annual International Cryptology Conference*. Springer. 2004, pp. 56–72 (cit. on p. 34).

[CNR12] J. Camenisch, G. Neven, and M. Rückert. "Fully anonymous attribute tokens from lattices." In: *International Conference on Security and Cryptography for Networks*. Springer. 2012, pp. 57–75 (cit. on p. 74).

[CS18] R. Clarisse and O. Sanders. *Short Group Signature in the Standard Model*. Cryptology ePrint Archive, Report 2018/1115. https://eprint.iacr.org/2018/1115. 2018 (cit. on pp. 32, 49, 50).

[Den02] A. W. Dent. "Adapting the weaknesses of the random oracle model to the generic group model." In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2002, pp. 100–109 (cit. on p. 8).

[DH76] W. Diffie and M. Hellman. "New directions in cryptography." In: *IEEE transactions on Information Theory* 22.6 (1976), pp. 644–654 (cit. on p. 25).

[DP06] C. Delerablée and D. Pointcheval. "Dynamic fully anonymous short group signatures." In: *International Conference on Cryptology in Vietnam*. Springer. 2006, pp. 193–210 (cit. on p. 50).

[DR99] J. Daemen and V. Rijmen. "AES proposal: Rijndael." In: (1999) (cit. on p. 72).

[DS18] D. Derler and D. Slamanig. "Highly-Efficient Fully-Anonymous Dynamic Group Signatures." In: May 2018, pp. 551–565. DOI: 10.1145/3196494.3196507 (cit. on pp. 2, 23–25, 35, 36, 49–51, 56, 59, 66, 68, 75).

[ElG85] T. ElGamal. "A public key cryptosystem and a signature scheme based on discrete logarithms." In: *IEEE transactions on information theory* 31.4 (1985), pp. 469–472 (cit. on pp. 25, 26).

[Emu+14] K. Emura, G. Hanaoka, Y. Sakai, and J. C. Schuldt. "Group signature implies public-key encryption with non-interactive opening." In: *International journal of information security* 13.1 (2014), pp. 51–62 (cit. on p. 73).

[ePr19] ePrint. *Cryptology ePrint archive*. 2019. URL: https://eprint.iacr.org/ (visited on 11/10/2019) (cit. on p. 49).

[EU16] EU. *General Data Protection Regulation*. 2016. URL: http://data.europa.eu/eli/reg/2016/679/oj (visited on 06/18/2019) (cit. on pp. 2, 42).

[Eze+15] M. F. Ezerman, H. T. Lee, S. Ling, K. Nguyen, and H. Wang. "A provably secure group signature scheme from code-based assumptions." In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2015, pp. 260–285 (cit. on p. 73).

[FHS15] G. Fuchsbauer, C. Hanser, and D. Slamanig. "Practical round-optimal blind signatures in the standard model." In: *Annual Cryptology Conference*. Springer. 2015, pp. 233–253 (cit. on p. 34).

[FHS18] G. Fuchsbauer, C. Hanser, and D. Slamanig. "Structure-preserving signatures on equivalence classes and constant-size anonymous credentials." In: *Journal of Cryptology* (2018), pp. 1–49 (cit. on pp. 49, 50, 52, 53, 57, 59, 75).

[Fri17] S. Friedl. "An elementary proof of the group law for elliptic curves." In: *Groups Complexity Cryptology* 9.2 (2017), pp. 117–123 (cit. on p. 16).

[GKV10] S. D. Gordon, J. Katz, and V. Vaikuntanathan. "A group signature scheme from lattice assumptions." In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2010, pp. 395–412 (cit. on p. 74).

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. "The knowledge complexity of interactive proof systems." In: *SIAM Journal on computing* 18.1 (1989), pp. 186–208 (cit. on p. 29).

[GO94] O. Goldreich and Y. Oren. "Definitions and properties of zero-knowledge proof systems." In: *Journal of Cryptology* 7.1 (1994), pp. 1–32 (cit. on p. 30).

[Goo19a] Google. *Android Cryptography Manual*. 2019. URL: `https://developer.android.com/guide/topics/security/cryptography` (visited on 08/01/2019) (cit. on p. 57).

[Goo19b] Google. *Google Pay*. 2019. URL: `https://pay.google.com/intl/de_de/about/` (visited on 11/07/2019) (cit. on p. 2).

[GR04] C. Gentry and Z. Ramzan. "Eliminating random permutation oracles in the Even-Mansour cipher." In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2004, pp. 32–47 (cit. on p. 8).

[Gro96] L. K. Grover. "A fast quantum mechanical algorithm for database search." In: *arXiv preprint quant-ph/9605043* (1996) (cit. on p. 72).

[GS08] J. Groth and A. Sahai. "Efficient non-interactive proof systems for bilinear groups." In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2008, pp. 415–432 (cit. on pp. 34, 55).

[Han16] C. Hanser. "Signatures on Equivalence Classes: A New Tool for Privacy-Enhancing Cryptography." PhD thesis. Graz University of Technology, 2016 (cit. on pp. 7, 22, 23, 25, 27, 29, 30).

[HPS98] J. Hoffstein, J. Pipher, and J. H. Silverman. "NTRU: A ring-based public key cryptosystem." In: *International Algorithmic Number Theory Symposium*. Springer. 1998, pp. 267–288 (cit. on p. 72).

[HS14] C. Hanser and D. Slamanig. "Structure-preserving signatures on equivalence classes and their application to anonymous credentials." In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2014, pp. 491–511 (cit. on p. 34).

[Ish+17] A. Ishida, K. Emura, G. Hanaoka, Y. Sakai, and K. Tanaka. "Group signature with deniability: how to disavow a signature." In: *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences* 100.9 (2017), pp. 1825–1837 (cit. on p. 39).

[JMV01] D. Johnson, A. Menezes, and S. Vanstone. "The elliptic curve digital signature algorithm (ECDSA)." In: *International journal of information security* 1.1 (2001), pp. 36–63 (cit. on pp. 31, 32).

[KPW15] E. Kiltz, J. Pan, and H. Wee. "Structure-preserving signatures from standard assumptions, revisited." In: *Annual Cryptology Conference*. Springer. 2015, pp. 275–295 (cit. on p. 34).

[Lag+13] F. Laguillaumie, A. Langlois, B. Libert, and D. Stehlé. "Lattice-based group signatures with logarithmic signature size." In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2013, pp. 41–61 (cit. on p. 74).

[Lan+14] A. Langlois, S. Ling, K. Nguyen, and H. Wang. "Lattice-based group signature scheme with verifier-local revocation." In: *International Workshop on Public Key Cryptography*. Springer. 2014, pp. 345–361 (cit. on p. 74).

[Lib+15] B. Libert, T. Peters, M. Joye, and M. Yung. "Linearly homomorphic structure-preserving signatures and their applications." In: *Designs, Codes and Cryptography* 77.2-3 (2015), pp. 441–477 (cit. on p. 34).

[Lib+16a] B. Libert, S. Ling, F. Mouhartem, K. Nguyen, and H. Wang. "Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions." In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2016, pp. 373–403 (cit. on p. 74).

[Lib+16b] B. Libert, S. Ling, K. Nguyen, and H. Wang. "Zero-knowledge arguments for lattice-based accumulators: logarithmic-size ring signatures and group signatures without trapdoors." In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2016, pp. 1–31 (cit. on p. 74).

[Lib+16c] B. Libert, F. Mouhartem, T. Peters, and M. Yung. "Practical Signatures with Efficient Protocols from Simple Assumptions." In: *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM. 2016, pp. 511–522 (cit. on p. 50).

[Lin+18] S. Ling, K. Nguyen, H. Wang, and Y. Xu. *Constant-size Group Signatures from Lattices*. Cryptology ePrint Archive, Report 2018/034. https://eprint.iacr.org/2018/034. 2018 (cit. on pp. 72–74).

[Lin+19] S. Ling, K. Nguyen, H. Wang, and Y. Xu. "Lattice-based group signatures: Achieving full dynamicity (and deniability) with ease." In: *Theor. Comput. Sci.* 783 (2019), pp. 71–94 (cit. on p. 74).

[Lit20] S. data protection inspectorate Lithuania. "MisterTango Payment Service GDPR fine." In: (2020). URL: https://www.ada.lt/go.php/eng/First-significant-fine-was-imposed-for-the-breaches-of-the-general-data-protection-regulation-in-lithuania/1 (visited on 01/16/2020) (cit. on p. 3).

[LMN16] B. Libert, F. Mouhartem, and K. Nguyen. "A lattice-based group signature scheme with message-dependent opening." In: *International Conference on Applied Cryptography and Network Security*. Springer. 2016, pp. 137–155 (cit. on p. 74).

[LNW15] S. Ling, K. Nguyen, and H. Wang. "Group signatures from lattices: simpler, tighter, shorter, ring-based." In: *IACR International Workshop on Public Key Cryptography*. Springer. 2015, pp. 427–449 (cit. on p. 74).

[Mar+12] E. Martin-Lopez, A. Laing, T. Lawson, R. Alvarez, X.-Q. Zhou, and J. L. O'brien. "Experimental realization of Shor's quantum factoring algorithm using qubit recycling." In: *Nature Photonics* 6.11 (2012), p. 773 (cit. on p. 69).

[McE78] R. J. McEliece. "A public-key cryptosystem based on algebraic." In: *Coding Thv* 4244 (1978), pp. 114–116 (cit. on p. 71).

[Nak+08] S. Nakamoto et al. "Bitcoin: A peer-to-peer electronic cash system." In: (2008) (cit. on p. 12).

[NZZ15] P. Q. Nguyen, J. Zhang, and Z. Zhang. "Simpler efficient group signatures from lattices." In: *IACR International Workshop on Public Key Cryptography*. Springer. 2015, pp. 401–426 (cit. on p. 74).

[Ora19] Oracle. *Java Cryptography Architecture (JCA) Reference Guide*. 2019. URL: https://docs.oracle.com/javase/8/docs%5C%2Ftechnotes%5C%2Fguides%5C%2Fsecurity%5C%2Fcrypto%5C%2FCryptoSpec.html (visited on 11/13/2019) (cit. on pp. 57, 59).

[Pat96] J. Patarin. "Hidden fields equations (HFE) and isomorphisms of polynomials (IP): Two new families of asymmetric algorithms." In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1996, pp. 33–48 (cit. on p. 72).

[PC09] R. A. Perlner and D. A. Cooper. "Quantum resistant public key cryptography: a survey." In: *Proceedings of the 8th Symposium on Identity and Trust on the Internet*. ACM. 2009, pp. 85–93 (cit. on p. 71).

[Ped91] T. P. Pedersen. “Non-interactive and information-theoretic secure verifiable secret sharing.” In: *Annual International Cryptology Conference*. Springer. 1991, pp. 129–140 (cit. on p. 28).

[Pol78] J. M. Pollard. “Monte Carlo methods for index computation (mod p).” In: *Mathematics of computation* 32.143 (1978), pp. 918–924 (cit. on p. 33).

[PS16] D. Pointcheval and O. Sanders. “Short randomizable signatures.” In: *Cryptographers’ Track at the RSA Conference*. Springer. 2016, pp. 111–126 (cit. on pp. 48, 49).

[PS18] D. Pointcheval and O. Sanders. “Reassessing security of randomizable signatures.” In: *Cryptographers’ Track at the RSA Conference*. Springer. 2018, pp. 319–338 (cit. on p. 34).

[Reg20] P. S. Regulator. “Discussion paper: Data in the payments industry.” In: (2020). URL: https://www.psr.org.uk/sites/default/files/media/PDF/PSR-Discussion-paper-Data-in-the-payments-industry-June-2018.pdf (visited on 01/16/2020) (cit. on p. 4).

[Res18] E. Rescorla. “The Transport Layer Security (TLS) Protocol Version 1.3.” In: *RFC* 8446 (2018), pp. 1–160. DOI: 10.17487/RFC8446. URL: https://doi.org/10.17487/RFC8446 (cit. on p. 12).

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. “A method for obtaining digital signatures and public-key cryptosystems.” In: *Communications of the ACM* 21.2 (1978), pp. 120–126 (cit. on pp. 25, 31, 32).

[Sak+12] Y. Sakai, K. Emura, G. Hanaoka, Y. Kawai, T. Matsuda, and K. Omote. “Group signatures with message-dependent opening.” In: *International Conference on Pairing-Based Cryptography*. Springer. 2012, pp. 270–294 (cit. on p. 39).

[Sch85] R. Schoof. “Elliptic curves over finite fields and the computation of square roots mod.” In: *Mathematics of computation* 44.170 (1985), pp. 483–494 (cit. on p. 18).

[Sho94] P. W. Shor. “Algorithms for quantum computation: Discrete logarithms and factoring.” In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134 (cit. on p. 69).

[Sma99] N. P. Smart. “The discrete logarithm problem on elliptic curves of trace one.” In: *Journal of cryptology* 12.3 (1999), pp. 193–196 (cit. on p. 21).

[Sta19] Statcounter.com. *Statcounter*. 2019. URL: https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide (visited on 11/07/2019) (cit. on p. 63).

[Too20] I. Tools. *Compact representation of an elliptic curve point*. 2020. URL: https://tools.ietf.org/id/draft-jivsov-ecc-compact-05.html#Miller (visited on 01/17/2020) (cit. on p. 49).

[Wik19a] Wikipedia. *Abelian Group*. 2019. URL: https://en.wikipedia.org/wiki/Abelian_group (visited on 08/22/2019) (cit. on p. 10).

[Wik19b] Wikipedia. *Cryptographic Hash Function*. 2019. URL: https://en.wikipedia.org/wiki/Cryptographic_hash_function (visited on 08/22/2019) (cit. on p. 9).

[Wik19c] Wikipedia. *Digital Signature*. 2019. URL: https://en.wikipedia.org/wiki/Digital_signature (visited on 08/22/2019) (cit. on pp. 5, 31).

[Wik19d] Wikipedia. *Field*. 2019. URL: https://en.wikipedia.org/wiki/Field_(mathematics) (visited on 08/22/2019) (cit. on p. 15).

[Wik19e] Wikipedia. *Group*. 2019. URL: https://en.wikipedia.org/wiki/Group_(mathematics) (visited on 08/22/2019) (cit. on p. 10).

[Wik20a] Wikipedia. "GPDR Fines." In: (2020). URL: https://en.wikipedia.org/wiki/GDPR_fines_and_notices (visited on 01/16/2020) (cit. on p. 3).

[Wik20b] Wikipedia. "PRISM surveilance program." In: (2020). URL: https://en.wikipedia.org/wiki/PRISM_(surveillance_program) (visited on 01/16/2020) (cit. on p. 1).

[YL06] T. Ylönen and C. Lonvick. "The Secure Shell (SSH) Protocol Architecture." In: *RFC* 4251 (2006), pp. 1–30. DOI: 10.17487/RFC4251. URL: https://doi.org/10.17487/RFC4251 (cit. on p. 12).

[Zim19] P. Zimmermann. *Pretty Good Privacy*. 2019. URL: https://www.pgp.com (visited on 11/10/2019) (cit. on p. 12).