

# On the Optimization of two Recent Proxy-Type Digital Signature Schemes and their Efficient Implementation in Java

David Derler  
david@derler.info

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



Master's Thesis

*Supervisors:*

Dipl.-Ing. Christian Hanser  
Dipl.-Ing. (FH) Dr.techn. Daniel Slamanig

*Assessor:*

Ass.Prof. Dipl.-Ing. Dr.techn. Peter Lipp

August, 2013


## Statutory Declaration

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

## Eidesstattliche Erklärung<sup>1</sup>

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

Graz, am 28. August 2013

<b>Signature Value</b>	daRIu0RHlgtecTR/SLTohqvjgc0qm2psDzIQIFypX74ebONMoXkTQY2ZbrsF+IckfqI83UxAB9FYBaixbp/vHQ==	
	<b>Signatory</b>	David Derler
	<b>Issuer-Certificate</b>	CN=a-sign-premium-mobile-03,OU=a-sign-premium-mobile-03,O=A-Trust Ges. f. Sicherheitssysteme im elektr. Datenverkehr GmbH,C=AT
	<b>Serial-No.</b>	953707
	<b>Method</b>	urn:pdfsigfilter:bka.gv.at:binaer:v1.1.0
	<b>Parameter</b>	etsi-bka-atrust-1.0:ecdsa-sha256:sha256:sha256:sha1
<b>Verification</b>	Signature verification at: <a href="http://www.signature-verification.gv.at">http://www.signature-verification.gv.at</a>	
<b>Note</b>	This document is signed with a qualified electronic signature. According to § 4 art. 1 of the Signature Act it in principle is legally equivalent to a handwritten signature.	
<b>Date/Time-UTC</b>	2013-08-28T12:45:59Z	

<sup>1</sup>Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008 Genehmigung des Senates am 1.12.2008

# Acknowledgements

Firstly, I would like to thank Peter Lipp, the assessor of this thesis, for giving me the opportunity to do this work in the Java group of the Institute for Applied Information Processing and Communications at the Graz University of Technology.

Secondly, I would like to thank Christian Hanser and Daniel Slamanig, my supervisors, for their outstanding support throughout my work on this thesis and for their constructive feedback concerning every aspect of this thesis. Moreover, I would like to thank them for always being available for my questions and the interesting discussions related to this thesis. I have learned a lot during this time.

Finally, I would like to thank my parents Erik and Hermine Derler for supporting me in everything I ever did.

# Abstract

Proxy-type digital signatures enable an *originator* to delegate the signing rights for a set of messages to a *proxy*. The *proxy* is then able to choose messages out of this set and issue signatures on behalf of the originator on it. In this thesis we particularly concentrate on two recent proxy-type digital signature schemes, namely the (extended) Blank Digital Signature scheme (BDSS) [38–40] and the Warrant-Hiding Proxy Signature scheme (WHPSS) [37]. After giving a comprehensive overview of the schemes, we show how they can be modified in order to use much more efficient Type-3 pairings instead of the originally proposed Type-1 pairings. Furthermore, several optimizations, significantly improving the performance of the schemes are being introduced. Then, we present a library, compatible with PKIX, integrating the optimized versions of the BDSS and the WHPSS respectively, within the Java Cryptography Architecture (JCA), and the keying material into X.509 certificates. In order to illustrate the flexibility of the proposed library, three signature formats, building up on XML and PDF, are introduced. Finally, we give a detailed insight in the performance of the protocols and our implementation.

**Keywords:** Proxy-Type Digital Signatures, Bilinear Pairings, BN Curves, Pedersen Commitments, Polynomial Commitments, Vector Commitments, PKIX, X.509, Java, Implementation, Optimization, Performance Evaluation

# Kurzfassung

Varianten von Proxy Signaturen erlauben es einem *Originator* die Zeichnungsrechte für eine Menge von Nachrichten an einen *Proxy* zu übertragen. Der *Proxy* ist dann in der Lage Nachrichten aus dieser Menge auszuwählen und im Namen des Originators zu signieren. Diese Arbeit konzentriert sich auf zwei aktuelle Varianten von Proxy Signaturen, nämlich das (erweiterte) Blank Digital Signature Schema (BDSS) [38–40] und das Warrant-Hiding Proxy Signature Schema (WHPSS) [37]. Nach einem umfassenden Überblick über die Schemata zeigen wir welche Modifikationen notwendig sind um anstatt der im Original vorgeschlagenen Type-1 Pairings viel effizientere Type-3 Pairings zu verwenden. Weiters stellen wir einige Optimierungen vor, die die Berechnungseffizienz beider Schemata signifikant verbessern. Darauf aufbauend präsentieren wir eine mit PKIX kompatible Library, welche die optimierten Versionen des BDSS und des WHPSS in die Java Cryptography Architecture (JCA), bzw. das Schlüsselmaterial in X.509, integriert. Um die Flexibilität der präsentierten Library zu illustrieren werden drei auf PDF bzw. XML aufbauende Signaturformate vorgestellt. Am Ende wird ein detaillierter Überblick über die Berechnungseffizienz der Protokolle und unserer Implementierung gegeben.

**Stichwörter:** Varianten von Proxy Signaturen, Bilineare Pairings, BN Kurven, Pedersen Commitments, Commitments zu Polynomen, Vektor Commitments, PKIX, X.509, Java, Implementierung, Optimierung, Evaluierung der Berechnungseffizienz

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Related Work . . . . .	2
1.2	Applications . . . . .	2
1.3	Outline . . . . .	3
<b>I</b>	<b>Background</b>	<b>4</b>
<b>2</b>	<b>Mathematical Background</b>	<b>5</b>
2.1	Complexity Theory . . . . .	5
2.1.1	Polynomial Reduction . . . . .	7
2.1.2	Negligible Functions . . . . .	8
2.1.3	One-way Functions . . . . .	8
2.1.4	Reductionist Security . . . . .	8
2.2	A Brief Overview over Number Theory . . . . .	9
2.2.1	Elementary Number Theory . . . . .	9
2.2.2	Groups . . . . .	11
2.2.3	Rings . . . . .	12
2.2.4	Fields . . . . .	14
2.2.5	The Discrete Logarithm Problem . . . . .	15
2.2.6	The Diffie-Hellman Problems . . . . .	16
2.3	Introduction to Elliptic Curves . . . . .	17
2.3.1	Elliptic Curve Arithmetic . . . . .	17
2.3.2	The Elliptic Curve Discrete Logarithm Problem . . . . .	20
2.3.3	Diffie-Hellman-Type Problems on Elliptic Curves . . . . .	20
2.4	Bilinear Pairings . . . . .	21
2.4.1	The Bilinear Diffie-Hellman Problem and Related Problems . . . . .	22
2.4.2	Bilinear Pairings over BN Curves . . . . .	23
<b>3</b>	<b>Selected Topics of Cryptography</b>	<b>24</b>
3.1	Basic Notion . . . . .	24
3.1.1	Symmetric Encryption . . . . .	25
3.1.2	Public Key Encryption . . . . .	27
3.1.3	Hybrid Encryption . . . . .	29
3.1.4	Key Agreement . . . . .	30
3.1.5	Digital Signatures . . . . .	31
3.1.6	Public Key Infrastructures . . . . .	33
3.1.7	Cryptographic Hash Functions . . . . .	34

3.2	Proxy Signatures . . . . .	35
3.2.1	Delegation-by-Certificate . . . . .	35
3.3	Commitments . . . . .	36
3.3.1	Discrete Logarithm and Pedersen Commitments . . . . .	36
3.3.2	Chameleon Hash Functions . . . . .	37
3.3.3	Polynomial Commitments . . . . .	39
3.3.4	Merkle Hash Trees . . . . .	40
3.3.5	Vector Commitments . . . . .	43
<b>II Proxy-Type Digital Signature Schemes</b>		<b>44</b>
<b>4</b>	<b>Blank Digital Signatures</b>	<b>45</b>
4.1	Basic Notion . . . . .	45
4.2	Setup . . . . .	46
4.3	Template and Message Encoding . . . . .	47
4.3.1	Extending the Encoding for Blank Elements . . . . .	48
4.4	Scheme . . . . .	48
4.4.1	Construction . . . . .	48
4.4.2	Security . . . . .	49
4.5	Tweaks and Optimizations . . . . .	50
4.5.1	Using Type-3 Pairings . . . . .	50
4.5.2	Aggregating Elements . . . . .	52
<b>5</b>	<b>Warrant-Hiding Proxy Signatures</b>	<b>53</b>
5.1	Basic Notion . . . . .	53
5.2	Setup . . . . .	53
5.3	Message and Message Space Encoding . . . . .	54
5.4	Schemes . . . . .	55
5.4.1	Constructions . . . . .	55
5.4.2	Security . . . . .	55
5.5	Tweaks and Optimizations . . . . .	56
5.5.1	Using Type-3 Pairings . . . . .	57
5.5.2	Hashing the Commitment . . . . .	57
<b>6</b>	<b>Delegation of Signing Rights to Multiple Proxies</b>	<b>59</b>
6.1	Hybrid Encryption of the Proxy Signing Keys . . . . .	59
6.2	Proxy Hiding . . . . .	60
6.2.1	Randomized Merkle Hash Trees . . . . .	60
6.2.2	Polynomial Commitments . . . . .	61
6.3	Abstract Description of the Delegation . . . . .	62
<b>III Implementation Aspects</b>		<b>63</b>
<b>7</b>	<b>Implementation in Java and Integration into JCA</b>	<b>64</b>
7.1	Background . . . . .	64
7.1.1	Java Cryptography Architecture . . . . .	64
7.1.2	BNpairings Library . . . . .	65

7.2	Overview . . . . .	65
7.3	Encoding and Key Representation in X.509 . . . . .	68
7.4	Proposed Signature Formats . . . . .	68
7.4.1	BDSS Signature Formats . . . . .	69
7.4.2	WHPSS XML Signature Format . . . . .	71
<b>8</b>	<b>Performance Evaluation</b>	<b>72</b>
8.1	Test Setup . . . . .	72
8.2	Results . . . . .	72
8.2.1	Proxy-Type Signature Schemes . . . . .	73
8.2.2	Comparison of the two Proxy Hiding Approaches . . . . .	75
<b>9</b>	<b>Conclusion</b>	<b>77</b>
<b>A</b>	<b>Definitions</b>	<b>78</b>
A.1	Abbreviations . . . . .	78
A.2	Used Symbols . . . . .	79
<b>B</b>	<b>Implementational Details</b>	<b>80</b>
B.1	Polynomial Arithmetic . . . . .	80
B.2	Used Object Identifiers . . . . .	81
<b>C</b>	<b>Proofs</b>	<b>82</b>
C.1	Security Analysis of the (E)BDSS Modifications . . . . .	82
C.1.1	Aggregating Fixed Elements . . . . .	82
C.1.2	Aggregating Blank Elements . . . . .	82
C.2	Composition of Secure Chameleon Hash and Secure Hash . . . . .	83
	<b>Bibliography</b>	<b>84</b>



# Chapter 1

## Introduction

Digital signatures basically allow a *signer* to issue a signature on a message and, given the message and the signature, any *verifier* is able to detect whether the message has been altered after the signature issuance (integrity) and to verify whether the signer has actually issued the given signature (authenticity). Nowadays, digital signature schemes are heavily used in many practical applications such as for issuing advanced electronic signatures, being legally equivalent to handwritten signatures, as defined in *Directive 1999/93/EC of the European Parliament and the Council of 13 December 1999 on a Community framework for electronic signatures* [81] or for ensuring integrity and authenticity of messages exchanged amongst arbitrary communication parties.

However, some applications require to delegate the signing rights to another party instead of directly signing the messages, which can be realized using proxy-type digital signatures. Here, in contrast to conventional digital signatures, three parties are involved, namely an *originator*, a *proxy* and a *verifier*. In such schemes, the originator delegates the signing power (for some particular well defined set of messages) to a proxy. This proxy can then produce a signature for a message and any verifier, given the message and the signature, can check whether the proxy has produced the signature on behalf of the originator (authenticity), the integrity of the message and, in case the originator has restricted the message space, it can also be verified whether the given message is one of the "allowed" messages.

Blank Digital Signatures (BDS) [38, 39] are a special instance of proxy-type digital signatures, allowing an originator to define and issue a signature on a *template*, containing *fixed* and *exchangeable elements* and allowing a designated proxy to produce signatures for *instantiations* of this template (messages). Given such a template signature, the proxy creates a message by choosing one of the predefined values for each of the exchangeable elements and issues a signature w.r.t. the template signature. When verifying this signature, only the message and the corresponding signature are required and, thus, the verifier does not learn anything about the unused choices in the exchangeable elements in the template (privacy property). The extended Blank Digital Signature scheme (EBDSS) [40] further extends this approach with the possibility to include a third type of elements, to be replaced by arbitrary strings of predefined length (*blank elements*), into templates, and, thus, enables a wider range of practical applications.

Similarly, the Warrant-Hiding Proxy Signature scheme (WHPSS) [37] serves for related purposes. In contrast to the BDSS, it allows an originator to define a set of messages and delegate the signing rights for it to a proxy. The proxy then chooses one message out of this set and issues a signature based on this delegation. Again, only the chosen

message gets revealed on verification of such a signature, whereas the rest of the message space stays private (privacy property). Due to their similarity, the BDSS could be used to achieve a similar functionality as the WHPSS by defining a template containing just one exchangeable element.

Accordingly, the question arises how the BDSS and the WHPSS would perform in a practical implementation, and to which extent they can be integrated into off-the-shelf cryptographic frameworks such as the Java Cryptography Architecture [71] and key infrastructures such as PKIX [22].

## 1.1 Related Work

The basic concepts for building schemes, which allow an originator to delegate signing rights to a proxy were introduced in [69, 82]. Both papers discuss, amongst others, an approach where the delegation is represented by an originator-issued certificate describing the delegation, which can then be verified by simply verifying the certificate and its contents. In [60], Mambo et al. firstly introduced a proxy signature scheme. They refer to the previously mentioned delegation-by-certificate approach as *delegation-by-warrant* approach. This approach is of particular interest for this thesis, since the BDSS as well as the WHPSS make use of it. Moreover, in [11, 12] more recent proxy signature schemes are discussed and a security model for proxy signatures is proposed. This security model builds the basis for the WHPSS construction.

Compared to the basic idea of proxy signatures, the WHPSS and the BDSS allow the originator to explicitly restrict the proxy's signing rights to sets of messages (WHPSS) or templates (BDSS), respectively, whilst only the proxy-chosen message or the filled in version of the template gets revealed to the verifier, enabling new applications which can not be realized using only the aforementioned concepts.

Another, somewhat related, paradigm is the concept of sanitizable and redactable signatures [4, 16, 45, 66, 78]. Sanitizable/redactable signature schemes allow for issuing a signature on a piece of data, and certain (previously defined) parts of this data can then be sanitized whilst the sanitizer is still able to provide a valid signature (without having access to the secret signing key). Thus, to a certain extent, a similar behavior can be achieved using the BDSS by quite simple modifications. Though, the BDSS does not allow for the modification of the originally signed message and requires to compute a new signature on the "sanitized" message in such a case.

## 1.2 Applications

Basically, the (E)BDSS enables an originator to hand over a signed form (template), containing fixed elements, exchangeable elements (and blank elements), to a proxy being designated to sign an arbitrary instance of this form, i.e., a filled in form, on behalf of the originator. Figure 1.1 illustrates the basic idea of the (E)BDSS by illustrating a protocol execution. As shown in this figure, it is also possible to encode yes-/no-choices within a template by simply encoding yes and no in an exchangeable element.

The (E)BDSS is applicable to any form, which requires to leave a few choices open to an intermediary party, whilst the rest of the content is fixed and the unused choices of the exchangeable elements do not get revealed on verification of an instance of this form. For instance, a valid scenario would be, that an attorney makes a business deal

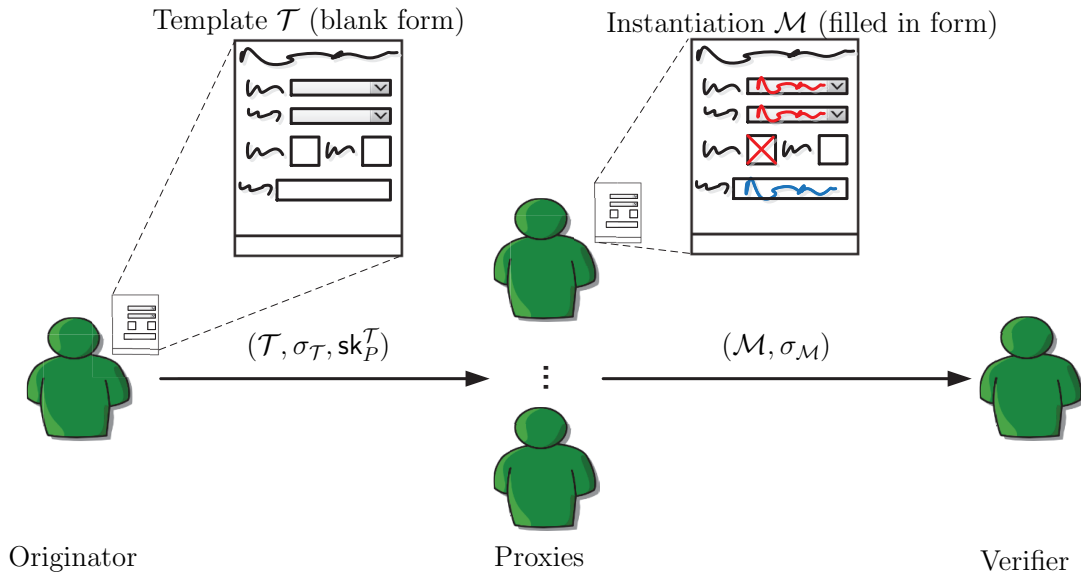


Figure 1.1: Schematic (E)BDSS execution [40]

on behalf of a client, using a template, previously defined and signed by the client [39]. Furthermore, governmental organizations could publish or distribute signed forms, which are intended to be completed and signed by any member of a specific group, e.g., any citizen of a country [40]. In the former case, the privacy property of the (E)BDSS could be of interest as well, whereas in the latter case this would, of course, be of no interest.

The WHPSS enables similar applications, with the difference that the originator can define a set of messages and the proxy is then able to choose one message instead of filling out a whole form. Thus, to a certain extent, the WHPSS is applicable to a subset of the (E)BDSS use cases.

Finally, due to the similarity to sanitizable/redactable signatures, also use cases applying to sanitizable/redactable signatures could be realized using the (E)BDSS or the WHPSS.

### 1.3 Outline

This thesis is basically composed of three parts building up on each other. Part I gives an overview over the mathematical background (Chapter 2) as well as the cryptographic building blocks (Chapter 3), being necessary to understand this thesis. On top of this, Part II introduces the BDSS in Chapter 4 and the WHPSS in Chapter 5 respectively, and discusses our optimizations regarding the performance of both schemes. The part concludes with the discussion of the implications of the delegation of the signing rights to multiple proxies in Chapter 6. Then, Part III gives a comprehensive overview of our Java implementation of the aforementioned schemes in Chapter 7 and a detailed overview of their performance in Chapter 8. Finally, conclusions are drawn in Chapter 9.

**Part I**  
**Background**

## Chapter 2

# Mathematical Background

This chapter is intended to give a brief overview of the mathematical foundations, being necessary to understand the contents of this thesis. Consequently, we recommend skipping it for readers who are already familiar with the mathematical paradigms discussed here. The whole chapter is composed of four sections. The first section gives a brief overview of some complexity theory related topics in order to provide some background for the security proofs of certain algorithms, whereas the remaining three sections build the mathematical basis for the schemes used in this thesis. Firstly, number theoretic basics and the discrete logarithm problem are discussed in Section 2.2. Secondly, elliptic curves are introduced in Section 2.3 and, finally, Section 2.4 introduces bilinear pairings and presents a concrete instantiation of bilinear pairings making use of elliptic curve groups.

The content of this chapter is a collection of definitions, theorems and examples, adopted from diverse books [9, 34, 35, 52, 57, 63, 72, 74, 77, 84] as well as lecture notes [59, 75] to certain courses held at the Graz University of Technology. Furthermore, miscellaneous other resources were used [6, 31, 36, 48, 55, 58, 61]. Additional sources are explicitly marked by in-text citations. Moreover, in some cases the proofs are omitted for a smooth readability. Hence, we refer the reader to the aforementioned sources for a more detailed overview.

### 2.1 Complexity Theory

In complexity theory, a *computational problem* is a problem where, when given an input, it is required to produce an output which conforms to certain properties. We will have a closer look at computational problems in the following paragraphs.

**Definition 2.1** (Formal language). *Let  $\Sigma$  be a finite alphabet. Furthermore, we denote  $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$  as the set of all strings resulting from arbitrarily adjoining elements out of  $\Sigma$ . A formal language  $L$  is a subset of  $\Sigma^*$ .*

Furthermore, we need to introduce the notion of (universal) Turing machines. A Turing machine is a machine, which operates on an infinite sequence of cells, each containing a single symbol out of an alphabet  $\Sigma$ . The machine can read the value of one cell at a time, can write the value of one cell at a time and can move one step left and one step right on the sequence in each step. Furthermore it stores an internal state  $q$ . The set of all possible symbol and state pairs together with a left or right movement transition represents the "instruction set" of the machine, e.g., when reading 1 in state 0 write 0, move left and set the state  $q$  to 1. One such transition according to the transition function is further referred to as a *computational step*. More formally this means:

**Definition 2.2** (Turing machine [77]). *A Turing machine is a 7-tuple,  $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$  where  $Q, \Sigma, \Gamma$  are all finite sets and*

1.  $Q$  is the set of states,
2.  $\Sigma$  is the input alphabet not containing the blank symbol  $\#$ ,
3.  $\Gamma$  is the tape alphabet, where  $\# \in \Gamma$  and  $\Sigma \subseteq \Gamma$ ,
4.  $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$  is the transition function,
5.  $q_0 \in Q$  is the start state,
6.  $q_{\text{accept}}$  is the accept state, and
7.  $q_{\text{reject}}$  is the reject state, where  $q_{\text{reject}} \neq q_{\text{accept}}$ .

A *universal Turing machine* takes a description of a Turing machine  $M$  and an input  $x$  and outputs  $M(x)$  if  $M$  terminates on the input of  $x$  and does not terminate otherwise. An *oracle Turing machine* is a machine, which can write a sequence  $x$  on an additional band. The oracle runs on input  $x$  and writes the output onto the band in one computational step such that the oracle Turing machine can read it.

Although the Turing machine model is quite simple, the *Church-Turing* thesis basically states that Turing machines can compute all functions that can be computed with modern computers.

**Definition 2.3** (Algorithm). *An algorithm  $A$  is a sequence of computational steps for a universal Turing machine (Definition 2.2), which transforms an input state to an output state.*

For the rest of this thesis, we assume an *oracle* to be a blackbox, which executes an algorithm in one computational step but does not allow for any analysis or change of this algorithm. Thereby, other algorithms can make use an oracle for their computations. An algorithm  $A$  making use of an oracle  $\mathcal{O}$  is further denoted as  $A^{\mathcal{O}}$ .

As one can derive from the definition above, one can make use of an algorithm for solving a computational problem. For our further explanations we only consider algorithms terminating on each input, i.e., after a finite number of steps.

**Definition 2.4** (Time complexity). *The number of steps an algorithm  $A$  takes on each possible input  $x$  is referred to as the time complexity  $t_A$  of  $A$  and is a function  $t_A : \Sigma^* \rightarrow \mathbb{N}$ . The time complexity of  $A$  depending on the input length  $n$ , denoted by  $T_A$ , is defined as:*

$$T_A(n) = \max_{x \in \Sigma^n} \{t_A(x)\}.$$

If for algorithm  $A$  there exists a polynomial  $p(x)$  such that there is an  $n_0 \in \mathbb{N}$  and for every  $n \geq n_0$  the time complexity  $T_A(n)$  is bounded by  $p(n)$ , then  $A$  is said to be a polynomial-time algorithm.

**Definition 2.5** (Probabilistic polynomial-time algorithm [33]). *A probabilistic polynomial-time algorithm is a polynomial-time algorithm (polynomially bounded Turing machine), which allows for making random steps during the execution. With the number of such random steps for input  $x$  being denoted by  $t_M(x)$  and  $M_r(x)$  being the output of the machine*

for a fixed choice of random steps  $r$ , the output of such a machine  $M$  is a probability distribution:

$$\Pr[M(x) = y] = \frac{|\{r \in \{0, 1\}^{t_M(x)} : M_r(x) = y\}|}{2^{t_M(x)}}.$$

Computational problems can either be *decision problems* or *search problems*, which are introduced in the following definitions.

**Definition 2.6** (Decision problem). *Let  $L \subseteq \Sigma^*$  be a language. A decision problem requires to decide whether a given input  $x \in \Sigma^*$  is contained in  $L$ .*

**Example 2.1.** *Let  $L$  be the set of all composite integers. A decision problem would be, given an integer  $x$ , to decide whether  $x$  is composite or not.*

**Definition 2.7** (Search problem). *Let  $R \subseteq \Sigma^* \times \Sigma^*$  be a relation between inputs and outputs. A search problem requires to find a  $y \in \Sigma^*$  for a given  $x \in \Sigma^*$  such that  $(x, y) \in R$ .*

**Example 2.2.** *Let  $R$  be a relation between the composite integers and their factors. A search problem would be, given an integer  $x$ , to find the factors of  $x$ .*

Complexity theory provides measures to classify computational problems according to the difficulty of solving them in terms of required time and space. The most prominent open question in this field is whether the class of polynomial-time algorithms  $\mathcal{P}$  equals the class of non-deterministic polynomial-time algorithms  $\mathcal{NP}$ , i.e., whether  $\mathcal{P} = \mathcal{NP}$  holds. Loosely speaking,  $\mathcal{P}$  contains the problems being solvable in polynomial-time and  $\mathcal{NP}$  contains the problems which, when given a solution, the solution can be verified in polynomial-time. Consequently, one is interested in establishing an ordering relation between different (classes of) problems. Such a relation between two distinct problems can be established by reducing one problem to another problem, which is discussed in the following section.

### 2.1.1 Polynomial Reduction

A computational problem  $P_1$  reduces to a computational problem  $P_2$ , if there is an algorithm, that solves instances of  $P_1$  by using the algorithm for  $P_2$  as an oracle, i.e., transforms each instance of  $P_1$  to an instance of  $P_2$  and the other way round for the solutions of  $P_2$  (see Figure 2.1).

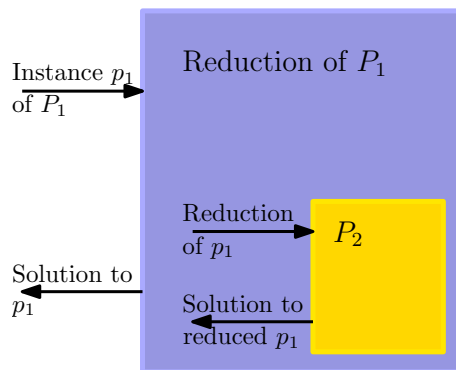


Figure 2.1: Schematic view of a reduction [52]

**Definition 2.8** (Polynomial reduction [63]). *If a reduction of a computational problem  $P_1$  to a computational problem  $P_2$  can be performed in polynomial-time,  $P_1$  is said to be polynomial-time reducible to  $P_2$ :*

$$P_1 \leq_P P_2.$$

This, in turn, means that  $P_2$  is at least as difficult as  $P_1$  when  $P_1 \leq_P P_2$  holds true.

Basically, one distinguishes between *Turing* and *Karp* reductions [67]. Karp reductions map single instances of a problem  $P_1$  to single instances of  $P_2$ , whereas Turing reductions use an oracle for solving a problem  $P_2$ , which can be used a polynomial number of times, as a subroutine for solving an instance of a problem  $P_1$ .

### 2.1.2 Negligible Functions

Cryptographic primitives which provide information theoretic security are rare and often not very practical. Most cryptographic schemes rely on computational problems which are assumed to be hard and the respective security proofs are in a complexity theoretic framework. Therefore, we need to define *negligible functions*.

**Definition 2.9** (Negligible function [52]). *A function  $\epsilon$  is called negligible, if for every polynomial  $p(\cdot)$  there is an  $n_0 \in \mathbb{N}$  such that for all  $n \geq n_0$*

$$\epsilon(n) \leq \frac{1}{p(n)}$$

*holds true. For the rest of this thesis, we denote a negligible functions as  $\epsilon(\cdot)$ .*

Using negligible functions, the security of a scheme is proved by showing that for any *probabilistic polynomial-time adversary* the probability of breaking the scheme is bounded by a negligible function  $\epsilon$  w.r.t. a security parameter  $\kappa$ , i.e., the breaking probability is exponentially small.

### 2.1.3 One-way Functions

Based on the definition of negligible functions, one-way functions can now be introduced.

**Definition 2.10.** *A one-way function  $f : X \rightarrow Y$  is a function that can be evaluated in polynomial time, but its inverse can not be efficiently computed. More precisely this means that there exists a negligible function such that for every probabilistic polynomial-time adversary  $\mathcal{A}$*

$$\Pr [\mathcal{A}(f(x)) = x] \leq \epsilon(\kappa).$$

#### Trapdoor One-way Functions

Trapdoor one-way functions are one-way functions, which allow the efficient inversion of the function when some additional information, i.e., the trapdoor information, is available.

### 2.1.4 Reductionist Security

Cryptographic paradigms often rely on mathematical problems  $P$ , for which it is assumed that it is computationally infeasible (w.r.t. a security parameter  $\kappa$ ) to find a solution  $s$  in



polynomial time. This means that for all *probabilistic polynomial-time adversaries*  $\mathcal{A}$  there exists a negligible function  $\epsilon$  such that

$$\Pr [\mathcal{A}(P) = s] \leq \epsilon(\kappa).$$

We further refer to these problems as hard problems. When one now wants to prove security of a scheme  $\mathcal{S}$ , he can simply reduce a hard problem  $P$  to it such that

$$P \leq_P \mathcal{S},$$

which shows that  $\mathcal{S}$  is at least as hard as  $P$ . In other words, this means that if  $\mathcal{S}$  could be broken efficiently it could also be misused to solve the problem  $P$ , which proves the security of  $\mathcal{S}$  under the assumption that  $P$  is hard.

## 2.2 A Brief Overview over Number Theory

Number theory constitutes the very basic building block of cryptography and provides the mathematical foundations for most state-of-the-art cryptographic protocols. Before we are able to introduce more sophisticated concepts, we are going to introduce the basic arithmetic rules for working with congruences in the following section.

### 2.2.1 Elementary Number Theory

For the rest of this thesis, we assume all numbers to be integers. The set of integers is denoted by  $\mathbb{Z}$ . Furthermore, all integers are treated to be equal when the remainder on division by another integer is identical, i.e., all integers having the same remainder modulo  $n$  are in one *equivalence class*.

#### Divisibility and Congruences

In order to define those *equivalence classes* and the arithmetic rules on representatives of those *equivalence classes*, we start with the definitions for divisibility and congruences.

**Definition 2.11.** *Given two elements  $a, n \in \mathbb{Z}$  and an element  $q \in \mathbb{Z}$  such that  $a \cdot q = n$  then  $a$  divides  $n$ . This relation is denoted as  $a \mid n$ .*

Using the definition above, we are able to introduce the congruence relationship of two integers  $a, b$  w.r.t. an integer  $n$ .

**Definition 2.12.** *An integer  $a \in \mathbb{Z}$  is congruent modulo  $n$  to another integer  $b \in \mathbb{Z}$  if  $n \mid (a - b)$ . The congruence relation is denoted as  $a \equiv b \pmod{n}$ .*

When taking a closer look at the congruence relation, we can derive that  $a \equiv b \pmod{n}$  implies that  $a$  and  $b$  have the same remainder when divided by  $n$ .

**Example 2.3.**  $17 \equiv 12 \pmod{5} \Leftrightarrow 5 \mid (17 - 12)$ .

**Definition 2.13.** *The congruence relation is an equivalence relation ((i) - (iii)) with the additional property (iv):*

(i) *Reflexivity:*  $a \equiv a \pmod{n}$

(ii) *Symmetry:*  $a \equiv b \pmod{n} \Leftrightarrow b \equiv a \pmod{n}$

(iii) *Transitivity:*  $a \equiv b \pmod{n} \wedge b \equiv c \pmod{n} \Rightarrow a \equiv c \pmod{n}$

(iv)  $a \equiv a_1 \pmod{n} \wedge b \equiv b_1 \pmod{n} \Rightarrow a + b \equiv a_1 + b_1 \pmod{n} \wedge a \cdot b \equiv a_1 \cdot b_1 \pmod{n}$

**Definition 2.14.** *The integers modulo  $n$  are denoted as  $\mathbb{Z}_n = \{x \pmod{n} : x \in \mathbb{Z}\}$ . This set contains  $n$  equivalence classes  $\{[0], \dots, [n-1]\}$ , where the equivalence class  $[i]$  for  $i$  would be  $\{i, i \pm n, i \pm 2n, \dots\}$ .*

Usually, one chooses the integers  $\{x \in \mathbb{Z} : 0 \leq x < n\}$  as representatives for the equivalence classes in  $\mathbb{Z}_n$ .

**Example 2.4.**  $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$ .  $5 \equiv 0 \pmod{5}$ ,  $7 \equiv 2 \pmod{5}$ .

### Primality

**Definition 2.15.** *A prime number  $p$  is an integer  $\geq 2$  such that for every integer  $k$  it holds that*

$$k \mid p \Rightarrow k = 1 \vee k = p.$$

The set of prime numbers is denoted by  $\mathbb{P}$ .

**Definition 2.16.** *The greatest common divisor of two integers  $a, b \in \mathbb{Z}$  is the greatest positive integer  $c \in \mathbb{N}$  such that  $c \mid a \wedge c \mid b$ :*

$$\gcd(a, b) = \max\{c \in \mathbb{N} : c \mid a \wedge c \mid b\}.$$

The greatest common divisor of two integers  $a, b \in \mathbb{Z}$  can be computed using the following recursive formula:  $\gcd(a, b) = \gcd(a, b \pmod{a}) \wedge \gcd(a, 0) = a$ .

**Definition 2.17.** *If the greatest common divisor of two integers  $a$  and  $b$  is 1,  $a$  and  $b$  are called co-prime or relatively prime.*

Hence, the question if it is possible to count the numbers being co-prime to an integer  $n$  arises. Defining a function for counting them, requires the following theorem.

**Theorem 2.1** (Fundamental theorem of arithmetic). *Every integer  $n \geq 2$  can be uniquely (up to the ordering of terms) represented as a product of prime powers*

$$n = \prod_{p_i \in \mathbb{P}} p_i^{e_i}$$

such that  $e_i \in \mathbb{N}_0$  and there are only finitely many  $e_i > 0$ .

Based on this factorization, Euler defined the so-called  $\varphi$ -function (see Definition 2.18).  $\varphi(n)$  counts the integers smaller than  $n$  being coprime to  $n$ . In other words, it counts the invertible elements in  $\mathbb{Z}_n$ :

$$\varphi(n) = |\{k : 1 \leq k \leq n \wedge \gcd(k, n) = 1\}|.$$

**Definition 2.18** (Euler's phi). *The  $\varphi$ -function of an integer  $n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$  is computed as*

$$\varphi(n) = \prod_{i=1}^k p_i^{e_i-1} (p_i - 1).$$

The  $\varphi$ -function is multiplicative, i.e.,  $\varphi(pq) = \varphi(p) \cdot \varphi(q)$  whenever  $\gcd(p, q) = 1$ .

Note that if  $n$  is prime, we have  $\varphi(n) = n - 1$ , as all integers in  $]0, n[$  are relatively prime to  $n$ . Furthermore, the following theorem holds true for any two coprime integers.

**Theorem 2.2** (Euler-Fermat). *For any two coprime integers  $a, n$  the congruence  $a^{\varphi(n)} \equiv 1 \pmod n$  holds true.*

### 2.2.2 Groups

Based on the number theoretical basics introduced in the previous sections of this chapter, we are now able to define the notion of groups.

**Definition 2.19.** *A group  $(G, \cdot)$  is a set  $G$  together with a binary operation  $\cdot$ . For a group  $(G, \cdot)$  the following properties hold:*

- (i) *Associativity:  $a \cdot (b \cdot c) = (a \cdot b) \cdot c$  for all  $a, b, c \in G$ .*
- (ii) *Neutral element: There is an element  $e \in G$  such that  $a \cdot e = e \cdot a = a \quad \forall \quad a \in G$ .*
- (iii) *Inverse element: There is an element  $a^{-1} \in G$  for each element  $a \in G$  such that  $a \cdot a^{-1} = a^{-1} \cdot a = e$ .*

*In addition to that, for Abelian groups also the commutativity property holds:*

- (iv) *Commutativity:  $a \cdot b = b \cdot a \quad \forall \quad a, b \in G$ .*

**Definition 2.20.** *The order  $\text{ord}(G)$  of a group  $G$  is equal to the number of elements in  $G$ . If this number is finite, the group is called finite.*

#### Example 2.5.

- $(\mathbb{Z}_n, +)$  with  $n \in \mathbb{N}$  is an Abelian group. The neutral element w.r.t. the group operation  $+$  is 0, whereas the inverse element of an element  $a \in \mathbb{Z}_n$  is  $-a$ .
- $(\mathbb{Z}_p \setminus \{0\}, \cdot)$  with  $p \in \mathbb{P}$  is also an Abelian group. Note that it is crucial in this case that the modulus  $p$  is a prime number, since taking an arbitrary modulus might violate the inverse element property (see Section 2.2.3). The neutral element regarding the group operation  $\cdot$  is 1.
- $(\mathbb{Z}_n \setminus \{0\}, \cdot)$  with  $n \notin \mathbb{P}$  is not a group, since  $(\mathbb{Z}_n \setminus \{0\}, \cdot)$  contains elements which are not invertible (see Section 2.2.3).
- $(\mathbb{Z}_5, +)$  with  $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$  is a finite group of order 5.

For the rest of this thesis we denote  $\mathbb{Z}_n^*$  as the set of invertible elements modulo  $n$

$$\mathbb{Z}_n^* = \{k \in \mathbb{Z}_n : \text{gcd}(k, n) = 1\}.$$

$(\mathbb{Z}_n^*, \cdot)$  is a group and the order of  $\mathbb{Z}_n^*$  is  $\varphi(n)$ .

Finally, some definitions related to the order of a group and some other related properties are needed.

**Definition 2.21.** *The order of an element  $a \in G$  is the smallest integer  $k$ , such that  $a^k = e$ . When no such integer exists,  $a$  is of infinite order.*

The order of an element  $a \in G$  is denoted as  $\text{ord}_G(a)$ .

**Definition 2.22.** A generator  $g$  of a group allows for expressing all elements in the group as power of the generator:

$$\forall a \in G \exists i \in \mathbb{Z} : a = g^i.$$

The order of a generator is always equal to the group order. If such a generator exists for a group, the group is called cyclic.

**Example 2.6.** The order of 3 in  $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$  is 4. Thus, 3 is a generator of  $\mathbb{Z}_5^*$  and  $\mathbb{Z}_5^*$  is cyclic.

$$\begin{aligned} 1 : 3^1 &\equiv 3 \pmod{\mathbb{Z}_5^*} \\ 2 : 3^2 &\equiv 4 \pmod{\mathbb{Z}_5^*} \\ 3 : 3^3 &\equiv 2 \pmod{\mathbb{Z}_5^*} \\ 4 : 3^4 &\equiv 1 \pmod{\mathbb{Z}_5^*} \end{aligned}$$

For the rest of this thesis, a generator  $g$  of a group  $G$  is denoted as  $g \in_g G$ .

**Definition 2.23** (Subgroup). Let  $(G, \cdot)$  be a group. A non-empty subset  $H$  of  $G$  is called a subgroup of  $(G, \cdot)$  when  $H$  is again a group w.r.t. the group operation  $\cdot$ .

**Definition 2.24** (Torsion subgroup). An  $r$ -torsion element  $a$  in a group  $G$  is an element satisfying  $a^r = e$ . The  $r$ -torsion subgroup of  $G$  is the set of all  $r$ -torsion points:

$$G[r] = \{a \in G : a^r = e\}.$$

**Theorem 2.3** (Lagrange). For any finite group  $(G, \cdot)$ , the order of any subgroup  $(H, \cdot)$  divides the order of  $(G, \cdot)$ .

**Theorem 2.4** (Sylow). Let  $G$  be a finite group with order  $\text{ord}(G) = m \cdot p^r$  with  $p \in \mathbb{P}$ ,  $\text{gcd}(m, p) = 1$  and  $r > 0$ . There exists a subgroup of order  $p^r$  of  $G$ .

**Theorem 2.5.** The multiplicative group  $\mathbb{Z}_n^*$ , i.e., the multiplicative group of invertible integers modulo  $n$ , is cyclic if  $n = 4$ ,  $n = p^\alpha$  or  $n = 2p^\alpha$ , with  $p$  being an odd prime and  $\alpha$  being a positive integer.

**Theorem 2.6.** All subgroups of cyclic groups are cyclic.

### 2.2.3 Rings

A ring  $(R, +, \cdot)$  further extends the group definition by adding a second binary operation  $+$ .

**Definition 2.25.** Let  $R$  be a set together with the two binary operations  $+$  and  $\cdot$ , respectively. The following properties apply to a ring:

- (i)  $(R, +)$  is an Abelian group with 0 as neutral element.
- (ii) Associativity w.r.t. multiplication:  $\forall a, b, c \in R : a \cdot (b \cdot c) = (a \cdot b) \cdot c$ .
- (iii) Distributivity of the  $\cdot$  operation over the  $+$  operation:  $\forall a, b, c \in R :$ 
  - $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$  (left-distributivity)
  - $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$  (right-distributivity).

Moreover, the ring is commutative if  $\forall a, b \in R : a \cdot b = b \cdot a$ .

**Definition 2.26.** A ring  $(R, +, \cdot)$  with 1 is a ring containing an element 1 such that  $a \cdot 1 = 1 \cdot a = a$  for all  $a \in R$ .

**Example 2.7.**  $(\mathbb{Z}, +, \cdot)$  and  $(\mathbb{Z}_n, +, \cdot)$  are both commutative rings with 1.

### The Ring of Integers Modulo $n$

The integers modulo  $n$ , together with  $+$  and  $\cdot$  constitute a ring. Obviously,  $(\mathbb{Z}_n, +)$  is an Abelian group and also the associativity and the distributivity properties hold for the integers modulo  $n$ .

However, we need to take a closer look at the inversion of integers and the properties of invertible integers w.r.t. the multiplication operation in order to be able to define fields.

### Modular Inverses

In order to be able to introduce modular inverses in  $(\mathbb{Z}_n, +, \cdot)$ , we need to introduce a method for efficiently computing the greatest common divisor of two integers  $a$  and  $b$ . The Euclidean algorithm provides such a method (see Algorithm 2.1).

---

#### Algorithm 2.1 The Euclidean Algorithm

---

```

1: Input:  $a, b \in \mathbb{Z}$  with  $a, b > 0$ , and w.l.o.g.  $a > b$ 
2: while  $b \neq 0$  do
3:   Find  $q, r$ , such that  $a = b \cdot q + r$ 
4:    $a = b, b = r$ 
5: end while
6: return  $a$ 

```

---

**Example 2.8.** We compute the greatest common divisor of 931 and 147.

$$\begin{aligned} 931 &= 6 \cdot 147 + 49 \\ 147 &= 3 \cdot \boxed{49} + 0 \end{aligned} \rightarrow \gcd(931, 147) = 49$$

The additive inverse of an element  $a \in \mathbb{Z}_n$  is  $-a$  since  $a - a \equiv 0 \pmod{n}$ . However, this is not that simple for the multiplicative inverse  $a^{-1}$ , which must fulfill  $a \cdot a^{-1} \equiv 1 \pmod{n}$ . Such an inverse only exists if  $\gcd(a, n) = 1$ . The following theorem is the basis for computing multiplicative inverse elements:

**Theorem 2.7 (Bezout).** For each pair of integers  $a, b$ , there exist two integers  $s, t$  such that  $\gcd(a, b) = s \cdot a + t \cdot b$ .

Now, an extension of the Euclidean algorithm can be exploited for efficiently computing the multiplicative inverse of a given integer, which only exists if the greatest common divisor of the modulus and the element to be inverted is 1. The extended Euclidean algorithm is executed in the same way as the Euclidean algorithm and then uses the intermediate values  $q$  and  $r$  for substituting intermediate results and computing the integers  $s$  and  $t$  in the way shown in the following example.

**Example 2.9.** We compute the modular inverse of  $146 \pmod{931}$ , i.e., an integer  $x$  such that  $146 \cdot x \equiv 1 \pmod{931}$ . Firstly, the Euclidean algorithm is executed:

$$\begin{aligned} 931 &= 6 \cdot 146 + 55 \\ 146 &= 2 \cdot 55 + 36 \\ 55 &= 1 \cdot 36 + 19 \\ 36 &= 1 \cdot 19 + 17 \rightarrow \gcd(931, 146) = 1 \\ 19 &= 1 \cdot 17 + 2 \\ 17 &= 8 \cdot 2 + 1 \\ 2 &= 2 \cdot \boxed{1} + 0 \end{aligned}$$

Secondly we start from  $1 = 17 - 8 \cdot 2$  and substitute back:

$$\begin{aligned} 1 &= 17 - 8 \cdot 2 = 17 - 8 \cdot (19 - 1 \cdot 17) = 9 \cdot 17 - 8 \cdot 19 = -8 \cdot 19 + 9 \cdot (36 - 1 \cdot 19) = \\ &= 9 \cdot 36 - 17 \cdot 19 = 9 \cdot 36 - 17 \cdot (55 - 1 \cdot 36) = -17 \cdot 55 + 26 \cdot 36 = \\ &= -17 \cdot 55 + 26 \cdot (146 - 2 \cdot 55) = 26 \cdot 146 - 69 \cdot 55 = 26 \cdot 146 - 69 \cdot (931 - 6 \cdot 146) = \\ &= 440 \cdot 146 - 69 \cdot 931 \end{aligned}$$

If we have a look at  $440 \cdot 146 - 69 \cdot 931 \equiv 1 \pmod{931}$ , we can immediately see that  $69 \cdot 931 \equiv 0 \pmod{931}$ . Thus, we have  $\boxed{440} \cdot 146 \equiv 1 \pmod{931}$  and 440 is our desired multiplicative inverse  $x$  of  $146 \pmod{931}$ .

Note that there is also a recursive version of the extended Euclidean algorithm which is omitted here for the sake of simplicity.

## 2.2.4 Fields

Finally, based on the definition of rings with 1 and multiplicative inverses in the previous section, the notion of fields can be introduced.

**Definition 2.27.** A field  $(F, +, \cdot)$  is a commutative ring with 1, with the additional property that each element is invertible w.r.t. multiplication. In other words, the following properties hold true for a field:

- (i)  $(F, +)$  is an (additive) Abelian group with neutral element 0.
- (ii)  $(F \setminus \{0\}, \cdot)$  is a (multiplicative) Abelian group with neutral element 1.
- (iii) The distributivity property (see Definition 2.25 property (iii)) holds for  $(F, +, \cdot)$ .

**Definition 2.28** (Extension field). A field  $(\overline{F}, +, \cdot)$  is called an extension field of  $(F, +, \cdot)$ , if  $F \subset \overline{F}$ .  $F$  is called a subfield of  $\overline{F}$  in this case.

**Definition 2.29.** If the number of elements in  $F$  is finite,  $F$  is called a finite field.

In the following we use  $\mathbb{F}$  to denote a finite field. As in the group case, the number of elements in a field  $F$  is referred to as the order of the field. Note that a finite field  $\mathbb{F}$  of order  $q$  only exists if  $q$  is a prime power, i.e.,  $q = p^m$  for  $p \in \mathbb{P}$  and  $m$  being a positive integer.

**Theorem 2.8.** For each integer  $m$  and prime  $p$ , there is exactly one (up to isomorphisms) field  $\mathbb{F}$  of order  $q = p^m$ . The field of order  $q$  is referred to as  $\mathbb{F}_q$ .

**Corollary 2.1.** Given a finite field  $\mathbb{F}_{p^m}$ , for every positive divisor  $l$  of  $m$  exactly one (up to isomorphisms) finite subfield of order  $p^l$  exists.

**Example 2.10.**  $(\mathbb{Z}_p, +, \cdot)$  with  $p \in \mathbb{P}$  is a finite field of prime order  $p$ .

**Definition 2.30.** The characteristic of a field  $F$  is the smallest integer  $n \geq 1$  such that  $\sum_{i=1}^n 1 = 0$ . If no such integer exists, the characteristic is 0.

Thus, the characteristic of a field  $\mathbb{F}$  with order  $q = p^m$  is  $p$  and  $m$  is called the extension degree, which can be any positive integer. Depending on the extension degree  $m$ , we distinguish between prime fields and extension fields. While prime fields have an extension degree  $m = 1$ ,  $m$  is greater than 1 for extension fields.

### Prime Fields

As already mentioned before, the integers modulo a prime  $p$  together with the addition and the multiplication form a finite field. Accordingly, elements in a prime field  $\mathbb{F}_p$  are reduced modulo a prime  $p$ .

### Extension Fields

Extension fields are constructed by reducing the elements in the polynomial ring  $\mathbb{F}_p[X]$  modulo a monic, irreducible polynomial  $f \in \mathbb{F}_p[X]$  of degree  $m$ . An *irreducible polynomial*  $f(X)$  is a polynomial which cannot be factored into other polynomials, each of a lower degree than  $f(X)$ . Formally, an irreducible polynomial can be defined as

$$f \in \mathbb{F}_p[X] \text{ irreducible} \Leftrightarrow \forall g \mid f \Rightarrow g \in \mathbb{F}_p^* \vee g = f.$$

Note that constructing two extension fields from two different monic, irreducible polynomials of degree  $m$  leads to two isomorphic field extensions. Formally, the extension field  $\mathbb{F}_{p^m}$  is constructed as follows:

$$\mathbb{F}_{p^m} = \mathbb{F}_p[X]/(f(X)) = \{g \bmod f : g \in \mathbb{F}_p[X]\}.$$

With  $\alpha$  being a root of  $f(X)$ , the elements of  $\mathbb{F}_{p^m}$  can be written in additive representation w.r.t. the generator  $\alpha$ :

$$\mathbb{F}_{p^m} \cong \{a_{m-1} \cdot \alpha^{m-1} + a_{m-2} \cdot \alpha^{m-2} + \dots + a_2 \cdot \alpha^2 + a_1 \cdot \alpha + a_0 \mid a_i \in \mathbb{F}_p\}.$$

The addition of two field elements can then be performed in the same way as a usual polynomial addition, whereas the multiplication is performed as a conventional polynomial multiplication followed by a reduction modulo the chosen irreducible polynomial of degree  $m$ . The reduction is, thereby, performed using a conventional polynomial division.

A special case of extension fields are the binary extension fields  $\mathbb{F}_{2^m}$ , which allow for very efficient field arithmetic due to the coefficients being in  $\mathbb{F}_2 = \{0, 1\}$ .

### Towering Field Extensions

In certain implementations, it may be beneficial to make use of multiple different field extensions, which, in turn, leads to a quite vast amount of implementation work, since each field extension requires its own arithmetic. Thus, it is useful to make use of field extension towers for odd extension degrees  $m$ , e.g., in the case of  $\mathbb{F}_{p^6}$  it is beneficial to make use of  $\mathbb{F}_{p^{3^2}}$ . When mapping this to the aforementioned polynomial representation, there is a polynomial of degree two, having its coefficients in  $\mathbb{F}_{p^3}$ .

#### 2.2.5 The Discrete Logarithm Problem

The security of some of the schemes discussed at a later point in this thesis, relies on the assumption of the hardness of the *discrete logarithm problem* (DLP). The discrete logarithm problem for multiplicative groups is defined as follows. Note that the following problem can also be applied to appropriately chosen fields.

**Definition 2.31.** *Let  $(\mathbb{G}, \cdot)$  be a finite cyclic group of prime order  $p$  which is generated by  $g$  and let  $x$  be an element out of  $\mathbb{Z}_p$ . Furthermore, let  $\text{DLP}^A$  be a game, which, on input of*

$g^x$ , requires the adversary  $\mathcal{A}$  to output  $x$ . If the output of  $\mathcal{A}$  is correct, the game outputs 1 and 0 otherwise. The DLP is for  $\mathcal{A}$  to win this game. The DL assumption in  $\mathbb{G}$  states, that for all probabilistic polynomial-time adversaries  $\mathcal{A}$  there exists a negligible function  $\epsilon$  such that

$$\Pr [\text{DLP}^{\mathcal{A}}(g^x) = 1] \leq \frac{1}{2} + \epsilon(\kappa)$$

holds true for all appropriately chosen  $p$  w.r.t. the security parameter  $\kappa$  (see Table 2.1).

The integer  $x$  is referred to as the discrete logarithm of  $g^x$  to the base  $g$ .

### 2.2.6 The Diffie-Hellman Problems

Finally, the knowledge of the *computational Diffie-Hellman* (CDH) problem and the *decisional Diffie-Hellman* (DDH) problem are crucial for understanding some of the schemes this thesis builds up on. In the context of multiplicative groups, these problems are defined as follows (see e.g. [70]). Again, the following problems can be applied to appropriately chosen fields as well.

*CDH* Let  $(\mathbb{G}, \cdot)$  be a finite cyclic group of prime order  $p$  which is generated by  $g$ . Moreover we have  $g^x, g^y \in \mathbb{G}$  with  $x, y \in \mathbb{Z}_p$ . Furthermore, let  $\text{CDH}^{\mathcal{A}}$  be a game, which, on input of  $g^x, g^y$ , requires the adversary  $\mathcal{A}$  to compute  $g^{xy}$ . If the output of  $\mathcal{A}$  is correct, the game outputs 1 and 0 otherwise. The CDHP is for  $\mathcal{A}$  to win this game. The CDH assumption in  $\mathbb{G}$  states, that for all probabilistic polynomial-time adversaries  $\mathcal{A}$  there exists a negligible function such that

$$\Pr [\text{CDH}^{\mathcal{A}}(g^x, g^y) = 1] \leq \frac{1}{2} + \epsilon(\kappa)$$

holds true for all appropriately chosen  $p$  w.r.t. the security parameter  $\kappa$  (see Table 2.1).

*DDH* Let  $(\mathbb{G}, \cdot)$  be a finite cyclic group of prime order  $p$  which is generated by  $g$ . Moreover we have  $g^x, g^y, g^z \in \mathbb{G}$  with  $x, y, z \in \mathbb{Z}_p$ . Furthermore, let  $\text{DDH}^{\mathcal{A}}$  be a game, which, on input of  $g^x, g^y, g^z$ , requires the adversary  $\mathcal{A}$  to tell whether  $x \cdot y \equiv z$  or not. If  $\mathcal{A}$ 's output is correct, the game outputs 1 and 0 otherwise. The DDHP is for  $\mathcal{A}$  to win this game. The DDH assumption in  $\mathbb{G}$  states, that for all probabilistic polynomial-time adversaries  $\mathcal{A}$  there exists a negligible function such that

$$\Pr [\text{DDH}^{\mathcal{A}}(g^x, g^y, g^z) = 1] \leq \frac{1}{2} + \epsilon(\kappa)$$

holds true for all appropriately chosen  $p$  w.r.t. the security parameter  $\kappa$  (see Table 2.1).

The above problems can be ordered according to their hardness as follows:

$$\text{DDHP} \leq_P \text{CDHP} \leq_P \text{DLP}.$$

Finally, we introduce the notion of *gap Diffie-Hellman* (GDH) groups, as well as the gap Diffie-Hellman problem.

**Definition 2.32** (Gap Diffie-Hellman group). *Groups, in which the DDHP is easy, whilst the CDHP is hard are called gap Diffie-Hellman groups.*



*GDH* Let  $(\mathbb{G}, \cdot)$  be a finite cyclic group of prime order  $p$  which is generated by  $g$ . Furthermore, we assume that an adversary  $\mathcal{A}$  has access to an oracle  $O^{DDH}$  for solving the DDH problem. The *GDHP* is for  $\mathcal{A}$  to win the CDH game, whilst having access to  $O^{DDH}$ . The GDH assumption in  $\mathbb{G}$  states, that for all probabilistic polynomial-time adversaries  $\mathcal{A}$  there still exists a negligible function such that

$$\Pr \left[ \text{CDH}^{\mathcal{A}, O^{DDH}}(g^x, g^y) = 1 \right] \leq \frac{1}{2} + \epsilon(\kappa)$$

holds true for all appropriately chosen  $p$  w.r.t. the security parameter  $\kappa$  (see Table 2.1).

## 2.3 Introduction to Elliptic Curves

In the mid of the 1980s, the use of elliptic curve groups in public key cryptography was proposed independently by Victor S. Miller [65] and Neal Koblitz [54]. Since then, this topic attracted quite a lot of interest and, consequently, a lot of research work has been put in it.

The security of elliptic curve cryptography (ECC) is based on the hardness of the elliptic curve discrete logarithm problem (ECDLP). For properly chosen elliptic curves there is no known sub-exponential time algorithm for solving the ECDLP. In contrast, for finite fields sub-exponential time algorithms for solving the DLP are known. Thus, properly chosen elliptic curve groups allow for using smaller security parameters which, in turn, allows for faster computations. Table 2.1 gives an overview over the comparable strengths proposed by NIST [6].

Bits of Security	FFC (e.g., DSA, DH)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
80	$L = 1024$ $N = 160$	$k = 1024$	$f = 160 - 223$
112	$L = 2048$ $N = 224$	$k = 2048$	$f = 224 - 255$
128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256 - 383$
192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384 - 511$
256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

Table 2.1: Comparable strengths proposed by NIST [6] with  $L$  being the size of the public key and  $N$  being the size of the private key in finite field cryptography (FFC) setups and  $k$  and  $f$  being the key sizes in integer factorization cryptography (IFC) and elliptic curve cryptography (ECC) setups, respectively.

### 2.3.1 Elliptic Curve Arithmetic

An elliptic curve  $E$  with underlying field  $F$ , further referred to as  $E/F$ , is defined by the so called (long) Weierstrass equation:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6. \quad (2.1)$$

The coefficients  $a_1, \dots, a_6$  of (2.1) are elements of  $F$ , and are chosen in such a way that the discriminant  $\Delta \neq 0$ . A nonzero discriminant ensures that  $E$  is smooth, i.e., every point on the curve has exactly one tangent line. It is defined as follows:

$$\left. \begin{aligned} \Delta &= -d_2^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2 d_4 d_6 \\ d_2 &= a_1^2 + 4a_2 \\ d_4 &= 2a_4 + a_1 a_3 \\ d_6 &= a_3^2 + 4a_6 \\ d_8 &= a_1^2 a_6 + 4a_2 a_6 - a_1 a_3 a_4 + a_2 a_3^2 - a_4^2 \end{aligned} \right\} \quad (2.2)$$

All points  $(x, y) \in F \times F$  fulfilling (2.1), together with the point at infinity  $\infty$  is further referred to as  $E(F)$ .

**Example 2.11.** *Figure 2.2 shows the points on the curve  $E : y^2 = x^3 + 2x + 5$  for two different underlying fields. Figure 2.2a shows  $E(\mathbb{R})$ , whereas Figure 2.2b shows  $E(\mathbb{Z}_{809})$ .*

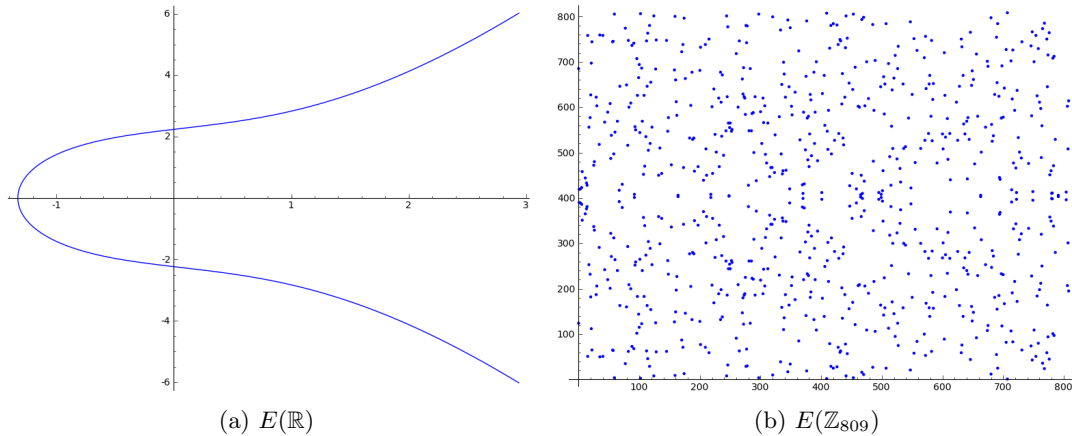


Figure 2.2: Plots of  $E : y^2 = x^3 + 2x + 5$  over different fields

### The Group Law

In order to make use of elliptic curves in cryptography, an Abelian group structure is required. Using the so called *chord-and-tangent* rule, the points on an elliptic curve with underlying field  $F$  form an Abelian group  $(E(F), +)$ . Thereby, the point at infinity  $\infty$  serves as neutral element. Figure 2.3 illustrates the point addition and doubling rules as defined in Definition 2.33 and Definition 2.34.

For the following definitions of the group law it is crucial that a line through two points on the elliptic curve intersects the curve at exactly one third point. As we will see, this holds true for  $E(F)$ .

**Definition 2.33** (Chord rule). *The addition of two points  $P, Q \in E(F)$  is performed by firstly drawing a line through  $P$  and  $Q$ . This line intersects the curve at a third point  $\bar{R}$ , which is reflected over the  $x$ -axis to obtain the point  $R$ .  $R$  then equals  $P + Q$ . See also Figure 2.3a.*

This operation is undefined when  $Q = P$ . Nevertheless, the point doubling can be performed by using the following operation.

**Definition 2.34** (Tangent rule). *When doubling a point  $P$ , one lays a tangent line through the point, which results in an intersection of the curve at the point  $\bar{R}$ . Reflecting  $\bar{R}$  over the  $x$ -axis leads to the point  $R$  which is equal to  $P + P = 2P$ . See also Figure 2.3a.*

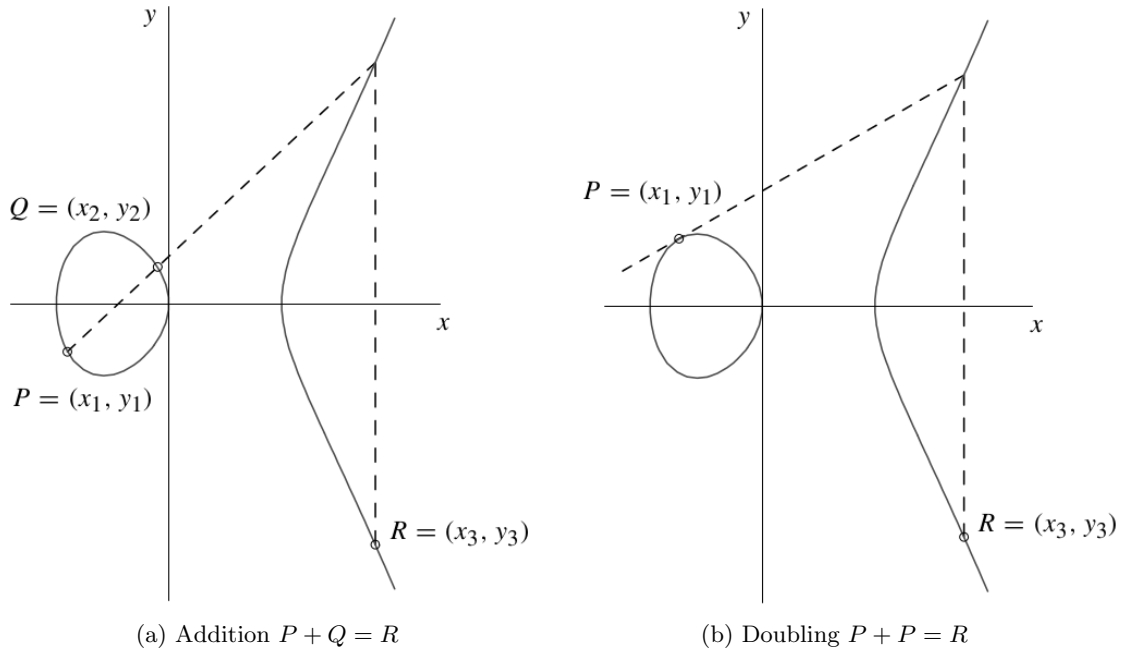


Figure 2.3: Geometric addition and doubling of elliptic curve points [35]

There are also algebraic formulas for point addition and point doubling, which are omitted here for the sake of simplicity. For more details we refer the reader to [35].

**Definition 2.35.** *The number of points on an elliptic curve with underlying finite field  $\mathbb{F}_q$  is called the order of  $E$  over  $\mathbb{F}_q$  and is denoted as  $\#E(\mathbb{F}_q)$ .*

A rough upper bound for this order can be found using the Weierstrass equation. It provides at most two solutions for each  $x \in \mathbb{F}_q$ . Thus,  $\#E(\mathbb{F}_q) \in [1, 2q + 1]$ . Hasse's Theorem provides a tighter upper bound as well as a lower bound for the order of an elliptic curve over  $\mathbb{F}_q$ .

**Theorem 2.9** (Hasse). *Let  $E$  be an elliptic curve defined over  $\mathbb{F}_q$ . Then*

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

Practical setups most likely make use of prime order subgroups of  $E(\mathbb{F}_q)$ , i.e.,  $E(\mathbb{F}_q)[p]$ . In this context, the cofactor  $h$  is defined as follows.

**Definition 2.36.** *Let  $E(\mathbb{F}_q)$  be an elliptic curve group and let  $E(\mathbb{F}_q)[p]$  be a  $p$ -torsion subgroup of  $E(\mathbb{F}_q)$ . Then, the cofactor  $h$  of  $E(\mathbb{F}_q)$  w.r.t. to the generator  $P \in E(\mathbb{F}_q)[p]$  is computed as*

$$h = \frac{\#E(\mathbb{F}_q)}{p}.$$

A group is said to have almost prime order when  $h \leq 4$ .

### Point Representation

When using the previously introduced affine coordinates, i.e., each point is represented by a tuple  $(x, y) \in F \times F$ , one inversion in the underlying field is necessary in order to double a point and to add two points respectively. As inversions are the most expensive operation in finite fields, they are circumvented using the notion of projective coordinates. The set of projective points  $\mathbb{P}(F)$  is composed of

$$\mathbb{P}(F)^* = \{(X : Y : Z) : X, Y, Z \in K, Z \neq 0\}$$

together with the so-called line at infinity

$$\mathbb{P}(F)^0 = \{(X : Y : Z) : X, Y, Z \in K, Z = 0\}.$$

An equivalence relation between projective and affine points can be established in the following way:

$$(X : Y : Z) \equiv \left( \frac{X}{Z} : \frac{Y}{Z} \right) = (x, y) \text{ for } z \neq 0.$$

All projective points on the line of infinity are mapped to the point at infinity  $\infty$  in the affine coordinate space. The projective (homogenized) version of the Weierstrass equation looks the following way:

$$E : Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \quad (2.3)$$

Again, the algebraic addition formulas are omitted here, and we refer the reader to [35] for further details.

### 2.3.2 The Elliptic Curve Discrete Logarithm Problem

The security of elliptic curve cryptosystems relies on the assumption of the hardness of the elliptic curve discrete logarithm problem (ECDLP). Similar to the discrete logarithm problem (see Section 2.2.5), the ECDLP is defined as follows.

**Definition 2.37.** *Let  $E$  be an elliptic curve defined over a field  $\mathbb{F}_q$ ,  $P \in E(\mathbb{F}_q)$  a point of prime order  $p$  and  $x \cdot P \in E(\mathbb{F}_q)$ . Furthermore, let  $\text{ECDLP}^{\mathcal{A}}$  be a game, which, on input of  $x \cdot P$ , requires the adversary  $\mathcal{A}$  to output  $x$ . If the output of  $\mathcal{A}$  is correct, the game outputs 1 and 0 otherwise. The ECDLP is for  $\mathcal{A}$  to win this game. The ECDL assumption in  $E(\mathbb{F}_q)$  states, that for all probabilistic polynomial-time adversaries  $\mathcal{A}$  there exists a negligible function such that*

$$\Pr [\text{ECDLP}^{\mathcal{A}}(x \cdot P) = 1] \leq \frac{1}{2} + \epsilon(\kappa)$$

*holds true for all appropriately chosen  $p$  w.r.t. the security parameter  $\kappa$  (see Table 2.1).*

### 2.3.3 Diffie-Hellman-Type Problems on Elliptic Curves

The Diffie-Hellman problems defined in Section 2.2.6 also apply to elliptic curve groups. Since they are, except for the additive notation in the elliptic curve case, exactly the same as the ones presented in Section 2.2.6 we do not restate them here.

## 2.4 Bilinear Pairings

Bilinear pairings were firstly used for attacking the ECDLP in certain elliptic curve groups by transforming it to an instance of the DLP (cf., [29,30,62,76]). However, the properties of bilinear pairings are also useful for constructing cryptographic protocols. Joux was the first one who discovered this. He proposed a scheme [46], based on pairings, allowing for a three party key agreement. Subsequently, an identity based encryption scheme [14] and a short signature scheme [15], both making use of pairings, were proposed. Henceforth, bilinear pairings received quite a lot of attention, were used to construct various cryptographic primitives which only work with pairings, such as functional encryption schemes, and a reasonable number of protocols employing pairings exist today. A pairing is defined as follows:

**Definition 2.38.** *Let  $\mathbb{G}_1$  and  $\mathbb{G}_2$  be cyclic additive groups and  $\mathbb{G}_T$  be a cyclic multiplicative group, all of prime order  $p$ : A bilinear pairing is a map of the form  $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$  and is called symmetric if  $\mathbb{G}_1 = \mathbb{G}_2$  and asymmetric otherwise. Additionally, the following properties must hold true for a bilinear pairing:*

- (i) **Bilinearity:**  $\forall P, Q \in \mathbb{G}_1, R' \in \mathbb{G}_2 : e(P + Q, R') = e(P, R') \cdot e(Q, R')$  and  $\forall P \in \mathbb{G}_1, Q', R' \in \mathbb{G}_2 : e(P, Q' + R') = e(P, Q') \cdot e(P, R')$ .
- (ii) **Non-degeneracy:**  $e(P, P') \neq 1$ , for  $P$  and  $P'$  being generators of  $\mathbb{G}_1$  and  $\mathbb{G}_2$  respectively.
- (iii)  $e$  needs to be efficiently computable.

While symmetric pairings are also known as Type-1 pairings, asymmetric pairings can be either Type-2 or Type-3 pairings. The difference between Type-2 and Type-3 pairings is that an efficiently computable isomorphism  $\Psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$  exists for Type-2 pairings, whilst for Type-3 pairings such an isomorphism is not known to exist.

For more information we refer the reader to [18,19].

**Definition 2.39** (Embedding degree). *Given  $E/\mathbb{F}_q$  and  $E(\mathbb{F}_q)[r]$  of order  $r$  with  $\gcd(q, r) = 1$ , the embedding degree of  $E(\mathbb{F}_q)$  is the smallest  $k$  such that  $r \mid q^k - 1$ .*

When bilinear pairings are implemented using elliptic curves, the ECDLP in  $\mathbb{G}_1$  can be transferred to the DLP in  $\mathbb{G}_T$ , which is an order  $p$  subgroup of the multiplicative group  $\mathbb{F}_{q^k}^*$ . This means that for reasonable small embedding degrees  $k$ , it can be easier to solve the DLP in  $\mathbb{G}_T$  than to solve the ECDLP in  $\mathbb{G}_1$  (see, e.g., [29,30,62,76]).

Furthermore, in a quite recent attack<sup>1</sup>, the discrete logarithm on a curve over  $\mathbb{F}_{(2^{257})^{24}}$  was broken within 550 hours, which renders binary extension fields unusable for pairing based cryptography. The method used for this attack was presented in [47], whereas an optimized algorithm for this attack is shown in [5].

Elliptic curves which are suitable for the use with bilinear pairings are referred to as *pairing-friendly curves*. In order for a curve to be suitable for the use with bilinear pairings it is required to contain a subgroup of large prime order  $p$  which ensures that the ECDLP is hard in  $\mathbb{G}_1$  and  $\mathbb{G}_2$ , respectively. Further a pairing-friendly curve is required to have an embedding degree small enough that computations in  $\mathbb{G}_T$  are feasible and large enough that it is still intractable to solve the DLP in  $\mathbb{G}_T$  when appropriate security parameters are chosen.

<sup>1</sup><https://listserv.nodak.edu/cgi-bin/wa.exe?A2=NMBRTHRY;49bb494e.1305>

### 2.4.1 The Bilinear Diffie-Hellman Problem and Related Problems

Building up on the definitions in Section 2.2.6, the Diffie-Hellman related problems can also be defined for bilinear pairings. For the sake of simplicity, we use Type-1 pairings  $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$  for our explanations. However, these problems can also be transferred to Type-2 and Type-3 pairings [18].

**Definition 2.40** (Bilinear Diffie-Hellman problem (BDHP)). *Let  $\mathbb{G}, \mathbb{G}_T$  be finite cyclic groups of prime order  $p$  and  $P \in_g \mathbb{G}$ . Furthermore let  $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$  be a bilinear pairing. We are given  $(P, x \cdot P, y \cdot P, z \cdot P)$  and let  $\text{BDHP}^{\mathcal{A}}$  be a game, which, on input of  $P, x \cdot P, y \cdot P, z \cdot P$ , requires the adversary  $\mathcal{A}$  to output  $e(P, P)^{xyz}$ . If the output of  $\mathcal{A}$  is correct, the game outputs 1 and 0 otherwise. The BDHP is for  $\mathcal{A}$  to win this game. The BDH assumption states, that for all probabilistic polynomial-time adversaries  $\mathcal{A}$  there exists a negligible function such that*

$$\Pr [\text{BDHP}^{\mathcal{A}}(P, x \cdot P, y \cdot P, z \cdot P) = 1] \leq \frac{1}{2} + \epsilon(\kappa)$$

holds true for all appropriately chosen  $p$  w.r.t. the security parameter  $\kappa$  (see Table 2.1).

**Definition 2.41** (Decisional bilinear Diffie-Hellman problem (DBDHP)). *Let  $\mathbb{G}, \mathbb{G}_T$  be finite cyclic groups of prime order  $p$  and  $P \in_g \mathbb{G}$  and  $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$  be a bilinear pairing. Furthermore, let  $\text{DBDHP}^{\mathcal{A}}$  be a game, played by the adversary  $\mathcal{A}$ . Thereby,  $\mathcal{A}$  gets  $(P, x \cdot P, y \cdot P, z \cdot P)$  and  $e(P, P)^w$ , and determines whether  $w \equiv xyz \pmod{p}$ . The game outputs 1 if  $\mathcal{A}$ 's output is correct and 0 otherwise. The DBDHP is for  $\mathcal{A}$  to win this game. The DBDH assumption states, that for all probabilistic polynomial-time adversaries  $\mathcal{A}$  there exists a negligible function such that*

$$\Pr [\text{DBDHP}^{\mathcal{A}}((P, x \cdot P, y \cdot P, z \cdot P), e(P, P)^w) = 1] \leq \frac{1}{2} + \epsilon(\kappa)$$

holds true for all appropriately chosen  $p$  w.r.t. the security parameter  $\kappa$  (see Table 2.1).

**Definition 2.42** ( $t$ -strong Diffie-Hellman problem ( $t$ -SDHP) [13, 51]). *Let  $\mathbb{G}, \mathbb{G}_T$  be finite cyclic groups of prime order  $p$  and  $P \in_g \mathbb{G}$  and  $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$  be a bilinear pairing. Further, let  $\alpha \in_r \mathbb{Z}_p^*$  and let  $t\text{-SDH}^{\mathcal{A}}$  be a game, which on input of the tuple  $(P, \alpha P, \dots, \alpha^t P) \in \mathbb{G}^{t+1}$  for some  $t > 0$  requires the adversary  $\mathcal{A}$  to output  $(c, \frac{1}{\alpha+c}P)$  for any  $c \in \mathbb{Z}_p \setminus \{-\alpha\}$ . If the output of  $\mathcal{A}$  is correct, the game outputs 1 and 0 otherwise. The  $t$ -SDHP is for  $\mathcal{A}$  to win this game. The  $t$ -SDH assumption states, that for all probabilistic polynomial-time adversaries  $\mathcal{A}$  there exists a negligible function such that*

$$\Pr [t\text{-SDH}^{\mathcal{A}}(P, \alpha P, \dots, \alpha^t P) = 1] \leq \frac{1}{2} + \epsilon(\kappa)$$

holds true for all appropriately chosen  $p$  w.r.t. the security parameter  $\kappa$  (see Table 2.1).

Note that pairing friendly curves represent GDH groups. Thereby, one can use the pairing as a DDH oracle, i.e., given an instance  $(P, x \cdot P, y \cdot P, z \cdot P)$  of the DDHP in  $\mathbb{G}$  one simply checks whether

$$e(x \cdot P, y \cdot P) \stackrel{?}{=} e(z \cdot P, P) \tag{2.4}$$

holds true, and, thus, implicitly verifies if  $x \cdot y \equiv z \pmod{p}$  holds. However, it is assumed that this DDH oracle does not give an advantage in solving the CDHP in  $\mathbb{G}$ .

In our case, we make use of Barreto-Naehrig (BN) curves which are briefly introduced in the following section.

### 2.4.2 Bilinear Pairings over BN Curves

BN curves are pairing friendly curves supporting Type-3 pairings. They were originally proposed by Barreto and Naehrig [8] and the curve equation looks as follows:

$$E : y^2 = x^3 + b.$$

where  $E$  is defined over some  $\mathbb{F}_p$  with  $p \equiv 1 \pmod{3}$ . The pairing  $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$  looks as follows for BN curves:

$$e : E(\mathbb{F}_p)[r] \times E(\mathbb{F}_{p^2})[r] \rightarrow \mathbb{F}_{p^{12}}[r].$$

As it can already be derived from the previous equation, BN curves have an embedding degree of 12. Thus, elements in  $\mathbb{G}_T$  have a bitlength of  $12 \cdot \kappa$ , with  $\kappa$  being the bitlength of  $\mathbb{G}_1$ . Currently a bitlength of 256bit in  $\mathbb{G}_1$  provides an appropriate level of security, which leads to a bitlength of 3072bit in  $\mathbb{G}_T$ . This choice is ideal at the 128bit security level w.r.t. the comparable strengths proposed by NIST [6], since the discrete logarithm problem should be equally hard in the additive groups  $\mathbb{G}_1$ ,  $\mathbb{G}_2$  and in the multiplicative group  $\mathbb{G}_T$  (see also Table 2.1).

Since the implementation of pairings is not part of this thesis, details regarding the computation of pairings are omitted here and we refer the reader to [58, 61] for further details.

## Chapter 3

# Selected Topics of Cryptography

Based on the mathematical foundations introduced in the previous section, this chapter is intended to present the cryptographic background needed for understanding this thesis. Again, it may be skipped by readers being already familiar with the concepts discussed in this section.

This chapter is organized as follows. Section 3.1 constitutes an introduction to the cryptographic basics. Section 3.2 then introduces the basic idea behind proxy signatures, which are the basis for the schemes introduced later on. Finally, Section 3.3 gives an overview over certain types of commitments being used by the aforementioned schemes.

### 3.1 Basic Notion

In this section we are going to introduce the basics of data encryption, as well as key agreements, digital signatures and cryptographic hash functions. Similar to the previous chapter, this section is based on various books [9, 35, 52, 63] as well as lecture notes [75] to certain courses held at the Graz University of Technology.

In virtually any introductory literature to cryptography, the goals of cryptography are introduced by the example of Alice and Bob, wanting to communicate over a potentially insecure channel. This is modeled by an eavesdropper Eve, who listens on this channel and potentially modifies transmitted messages (see Figure 3.1).

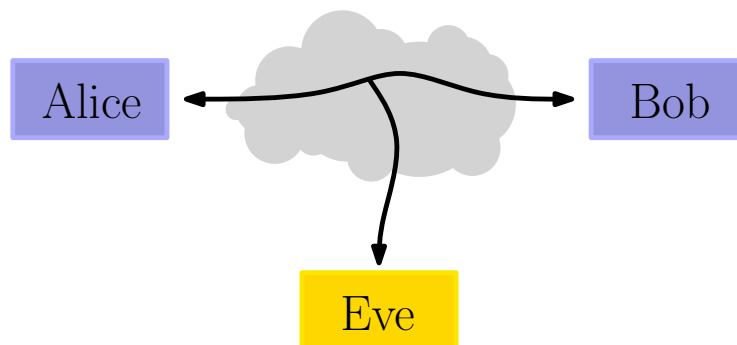


Figure 3.1: Alice and Bob, communicating over a potentially insecure channel

Based on this example, the following basic cryptographic goals can be derived (see, e.g., [63]):

- **Confidentiality:** No one, except the desired recipient(s), should be able to read the



contents of a given message. When, for example, Alice sends a message to Bob only Bob should be able to read the contents of the message. This goal can, for instance, be reached by using data encryption (see Section 3.1.1, Section 3.1.2, Section 3.1.3).

- **Integrity:** Given a message, any unauthorized modification of this message should be detectable, e.g., Bob is able to detect any modifications of a message originating from Alice. Integrity can, e.g., be ensured using message authentication codes (see, e.g., [3]) or digital signatures (see Section 3.1.5).
- **Authentication:** This goal is concerned with the verification of the identity of the communication parties (entity authentication) and the origin of messages (data origin authentication). Note that data origin authentication implicitly provides data integrity. One can reach this goal by using message authentication codes (see, e.g., [3]) or digital signatures (see Section 3.1.5).
- **Non-repudiation:** Given a message sent by a certain party, this party should not be able to disclaim to be the sender of the message. For instance, using digital signatures (see Section 3.1.5) would be a way for ensuring non-repudiation.

Depending on the used cryptographic primitive, or the combination of cryptographic primitives respectively, one or more of the aforementioned goals can be met.

Before we are able to start over with our explanations, we firstly need to introduce Kerckhoff's principle.

**Kerckhoff's Principle** Kerckhoffs [53] stated that the security of a cipher should only depend on keeping the key secret, whilst all involved algorithms can be publicly known. Today, this principle is applied to every cryptographic primitive involving secret information.

In the following sections we introduce encryption primitives, which provide confidentiality of data. Thereby, one basically can distinguish between symmetric and asymmetric encryption schemes which are discussed in Section 3.1.1 and Section 3.1.2 respectively.

### 3.1.1 Symmetric Encryption

In symmetric encryption schemes, denoted by  $\Sigma = (\text{Gen}, \text{Enc}, \text{Dec})$ , the identical key  $K$  is used for decryption and encryption of a message  $m \in \{0, 1\}^*$ . Formally, the encryption algorithm  $\text{Enc}$  as well as the decryption algorithm  $\text{Dec}$  with message  $m$  and ciphertext  $c$  can be defined as follows:

$$\begin{aligned} c &= \text{Enc}_K(m) \\ m &= \text{Dec}_K(c) \end{aligned} \tag{3.1}$$

Thereby, one requires that for all potential messages  $m$  we have  $m = \text{Dec}_K(\text{Enc}_K(m))$  for all  $K \in \text{Gen}(\kappa)$ . For the rest of this thesis, we assume  $\text{Gen}$  to be an algorithm, outputting the required keying material on input of a security parameter  $\kappa$ . Furthermore, we assume an oracle to be a black box, allowing to execute a certain algorithm, whereas no further information about this algorithm can be derived from the oracle.

**Notion of security:** In order to quantify the security of a given symmetric encryption scheme  $\Sigma = (\text{Gen}, \text{Enc}, \text{Dec})$ , any state-of-the-art literature, e.g., [52], defines various security measures, which are explained in the following paragraphs. The notion, as well as the definitions for our further explanations were taken from [52].

Firstly, we define the *indistinguishable encryption* property of a given symmetric cipher when an *eavesdropper* is present. This property is defined in Definition 3.1, which, in turn, is based on Game 3.1.

1. A key  $K$  is generated by running  $\text{Gen}(\kappa)$ .
2. Adversary  $\mathcal{A}$  gets the security parameter  $\kappa$  and outputs a pair of messages  $m_0, m_1$  of the same length. (These messages must be in the plaintext space associated with  $K$ )
3. A random bit  $b \in \{0, 1\}$  is chosen, and then a ciphertext  $c = \text{Enc}_K(m_b)$  is computed and given to  $\mathcal{A}$ . We call  $c$  the **challenge ciphertext**.
4.  $\mathcal{A}$  outputs a bit  $b'$ .
5. The output of the game is defined to be 1 if  $b' = b$ , and 0 otherwise.

Game 3.1: The eavesdropping indistinguishability game  $\text{PrivK}_{\mathcal{A}, \Pi}^{\text{eav}}(\kappa)$  [52]

**Definition 3.1** (EAV-security [52]). A symmetric encryption scheme  $\Sigma = (\text{Gen}, \text{Enc}, \text{Dec})$  has *indistinguishable encryption in the presence of an eavesdropper* if for all probabilistic polynomial-time adversaries  $\mathcal{A}$  there exists a negligible function  $\epsilon$  such that

$$\Pr [\text{PrivK}_{\mathcal{A}, \Pi}^{\text{eav}}(\kappa) = 1] \leq \frac{1}{2} + \epsilon(\kappa).$$

Secondly, the *indistinguishable encryption* property of a symmetric cipher with respect to the *chosen-plaintext attack* can be defined. This is done in Game 3.2 and Definition 3.2, respectively.

1. A key  $K$  is generated by running  $\text{Gen}(\kappa)$ .
2. The adversary  $\mathcal{A}$  gets the security parameter  $\kappa$  and oracle access to  $\text{Enc}_K(\cdot)$ , and outputs a pair of messages  $m_0, m_1$  of the same length.
3. A random  $b \in \{0, 1\}$  is chosen, and then a ciphertext  $c = \text{Enc}_K(m_b)$  is computed and given to  $\mathcal{A}$ . We call  $c$  the **challenge ciphertext**.
4. The adversary  $\mathcal{A}$  continues to have oracle access to  $\text{Enc}_K(\cdot)$ , and outputs a bit  $b'$ .
5. The output of the game is defined to be 1 if  $b' = b$ , and 0 otherwise. (In case  $\text{PrivK}_{\mathcal{A}, \Pi}^{\text{cpa}}(n) = 1$ , we say that  $\mathcal{A}$  succeeded)

Game 3.2: The CPA indistinguishability game  $\text{PrivK}_{\mathcal{A}, \Pi}^{\text{cpa}}(\kappa)$  [52]

**Definition 3.2** (CPA-security [52]). A symmetric encryption scheme  $\Sigma = (\text{Gen}, \text{Enc}, \text{Dec})$  has *indistinguishable encryptions under a chosen-plaintext attack* if for all probabilistic polynomial-time adversaries  $\mathcal{A}$  there exists a negligible function  $\epsilon$  such that

$$\Pr [\text{PrivK}_{\mathcal{A}, \Pi}^{\text{cpa}}(\kappa) = 1] \leq \frac{1}{2} + \epsilon(\kappa).$$

Finally, we introduce the *indistinguishable encryption* property of a symmetric cipher for *chosen-ciphertext attacks* (see Game 3.3, Definition 3.3).

1. A key  $K$  is generated by running  $\text{Gen}(\kappa)$ .
2. The adversary  $\mathcal{A}$  gets the security parameter  $\kappa$  and oracle access to  $\text{Enc}_K(\cdot)$  and  $\text{Dec}_K(\cdot)$ . It outputs a pair of messages  $m_0, m_1$  of the same length.
3. A random  $b \in \{0, 1\}$  is chosen, and then a ciphertext  $c = \text{Enc}_K(m_b)$  is computed and given to  $\mathcal{A}$ . We call  $c$  the *challenge ciphertext*.
4. The adversary  $\mathcal{A}$  continues to have oracle access to  $\text{Enc}_K(\cdot)$  and  $\text{Dec}_K(\cdot)$ , but is not allowed to query the latter on the challenge ciphertext itself. Eventually,  $\mathcal{A}$  outputs a bit  $b'$ .
5. The output of the game is defined to be 1 if  $b' = b$ , and 0 otherwise.

Game 3.3: The CCA indistinguishability game  $\text{PrivK}_{\mathcal{A}, \Pi}^{\text{cca}}(\kappa)$  [52]

**Definition 3.3** (CCA-security [52]). A symmetric encryption scheme  $\Sigma = (\text{Gen}, \text{Enc}, \text{Dec})$  has indistinguishable encryptions under a chosen-ciphertext attack if for all probabilistic polynomial-time adversaries  $\mathcal{A}$  there exists a negligible function  $\epsilon$  such that

$$\Pr [\text{PrivK}_{\mathcal{A}, \Pi}^{\text{cca}}(\kappa) = 1] \leq \frac{1}{2} + \epsilon(\kappa).$$

One state-of-the-art symmetric cipher, which is also marginally used in this thesis, is the Advanced Encryption Standard (AES) [23] cipher developed by Daemen and Rijmen. The AES cipher has been standardized by NIST in 2001 in order to replace the Data Encryption Standard (DES) [1] cipher, which had become insecure.

What remains unsolved at this point is the problem of exchanging a symmetric key amongst the communication parties in a confidential, integrity-protecting and authentic way. Two possibilities for overcoming this problem are provided later in this section (hybrid encryption) and in Section 3.1.4 (key agreement), respectively.

### 3.1.2 Public Key Encryption

In contrast to symmetric encryption, a public key encryption scheme, denoted by  $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ , distinguishes between two keys, namely, a secret key  $\text{sk}$  and a public key  $\text{pk}$ . Whilst the secret key is only known to the owner of a so-called keypair  $(\text{sk}, \text{pk})$ , the public key can be made publicly available. Note that this requires that it is hard to compute the secret key from the public key. This is usually realized by using (trapdoor) one-way functions. Formally the encryption function  $\text{Enc}$  and the decryption function  $\text{Dec}$  with message  $m$  and ciphertext  $c$  can be defined as follows:

$$\begin{aligned} c &= \text{Enc}_{\text{pk}}(m) \\ m &= \text{Dec}_{\text{sk}}(c) \end{aligned} \tag{3.2}$$

As in the symmetric case, it is required that for all potential messages  $m$  it holds that  $m = \text{Dec}_{\text{sk}}(\text{Enc}_{\text{pk}}(m))$  for all  $(\text{sk}, \text{pk}) \in \text{Gen}(\kappa)$ . This way, anyone in possession of the public key can encrypt messages for a given party  $A$  by using  $A$ 's public key  $\text{pk}$ . Once the message is encrypted,  $A$  can decrypt the message using its secret key  $\text{sk}$ . Cryptosystems, incorporating a public and a private key, are further referred to as public key cryptosystems.

**Notions of Security:** As in the symmetric case, we are going to introduce security measures for public key cryptosystems. Again, the notion as well as the definitions have been taken from [52].

Similar to the symmetric case, the definition for security against chosen-plaintext attacks looks as follows: Game 3.4 states the chosen-plaintext indistinguishability game, whereas Definition 3.4 defines the security measure against this attack.

1.  $\text{Gen}(\kappa)$  is run to obtain keys  $(\text{sk}, \text{pk})$ .
2. Adversary  $\mathcal{A}$  is given  $\text{pk}$  as well as oracle access to  $\text{Enc}_{\text{pk}}(\cdot)$ . The adversary outputs a pair of messages  $m_0, m_1$  of the same length. (These messages must be in the plaintext space associated with  $\text{pk}$ .)
3. A random bit  $b \in \{0, 1\}$  is chosen, and then a ciphertext  $c = \text{Enc}_{\text{pk}}(m_b)$  is computed and given to  $\mathcal{A}$ . We call  $c$  the **challenge ciphertext**.
4.  $\mathcal{A}$  continues to have access to  $\text{Enc}_{\text{pk}}(\cdot)$  and outputs a bit  $b'$ .
5. The output of the game is defined to be 1 if  $b' = b$ , and 0 otherwise.

Game 3.4: The CPA indistinguishability game  $\text{PubK}_{\mathcal{A}, \Pi}^{\text{cpa}}(\kappa)$  [52]

**Definition 3.4** (CPA-security [52]). A public key encryption scheme  $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$  has *indistinguishable encryptions under a chosen-plaintext attack* if for all probabilistic polynomial-time adversaries  $\mathcal{A}$  there exists a negligible function  $\epsilon$  such that

$$\Pr \left[ \text{PubK}_{\mathcal{A}, \Pi}^{\text{cpa}}(\kappa) = 1 \right] \leq \frac{1}{2} + \epsilon(\kappa).$$

Note that in the asymmetric case CPA-security and EAV-security are the same, since any adversary can simply perform an encryption using the public key (without oracle access).

At last, we state the chosen-ciphertext attack indistinguishability game, as well as the definition regarding the security against these types of attacks here (see Game 3.5, Definition 3.5).

1.  $\text{Gen}(\kappa)$  is run to obtain keys  $(\text{sk}, \text{pk})$ .
2. The adversary  $\mathcal{A}$  is given  $\text{pk}$  and access to a decryption oracle  $\text{Dec}_{\text{sk}}(\cdot)$ . It outputs a pair of messages  $m_0, m_1$  of the same length. (These messages must be in the plaintext space associated with  $\text{pk}$ .)
3. A random bit  $b \in \{0, 1\}$  is chosen, and then a ciphertext  $c = \text{Enc}_{\text{pk}}(m_b)$  is computed and given to  $\mathcal{A}$ .
4.  $\mathcal{A}$  continues to interact with the decryption oracle, but may not request a decryption of  $c$  itself. Finally,  $\mathcal{A}$  outputs a bit  $b'$ .
5. The output of the game is defined to be 1 if  $b' = b$ , and 0 otherwise.

Game 3.5: The CCA indistinguishability game  $\text{PubK}_{\mathcal{A}, \Pi}^{\text{cca}}(\kappa)$  [52]

**Definition 3.5** (CCA-security [52]). A public key encryption scheme  $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$  has *indistinguishable encryptions under a chosen-ciphertext attack* if for all probabilistic polynomial-time adversaries  $\mathcal{A}$  there exists a negligible function  $\epsilon$  such that

$$\Pr \left[ \text{PubK}_{\mathcal{A}, \Pi}^{\text{cca}}(\kappa) = 1 \right] \leq \frac{1}{2} + \epsilon(\kappa).$$

### 3.1.3 Hybrid Encryption

Whilst asymmetric encryption schemes are typically too slow for larger messages, they do not require a secure key exchange, since the public key  $\mathbf{pk}$  is public by definition. In contrast, symmetric ciphers are fast when encrypting large messages, whereas they incorporate the problem of securely exchanging the secret key  $K$ .

Hybrid encryption schemes combine the advantages of both, symmetric and asymmetric encryption schemes. Thereby, the message  $m$ , intended to be encrypted, is symmetrically encrypted under a random key  $K$ . The small key  $K$  is then asymmetrically encrypted using the recipients public key  $\mathbf{pk}$ . Suppose that a party  $A$  wants to encrypt a message  $m$  for a party  $B$ . Then,  $A$  symmetrically encrypts  $m$  under the random symmetric key  $K$ , which is in turn encrypted using the public key  $\mathbf{pk}_B$  of party  $B$ :

$$\begin{array}{ccc}
 & A & B \\
 c = \text{Enc}_K(m) & & \\
 c_{K,B} = \text{Enc}_{\mathbf{pk}_B}(K) & \xrightarrow{c, c_{K,B}} & \\
 & & K = \text{Dec}_{\mathbf{sk}_B}(c_{K,B}) \\
 & & m = \text{Dec}_K(c)
 \end{array} \tag{3.3}$$

Note that hybrid encryption can also be used to encrypt data intended to be decrypted by multiple parties. In this case, the data only needs to be encrypted once, whereas the symmetric key  $K$  is separately encrypted for each party. The encrypted data is then represented by the tuple  $(c, (c_{K,1}, \dots, c_{K,n}))$  for the recipients  $1, \dots, n$ . PKCS#7 [49], for instance, standardizes this paradigm under the name *EnvelopedData*.

### Elliptic Curve Integrated Encryption Scheme

The elliptic curve integrated encryption scheme (ECIES) is used for encryption purposes in our implementations. It is, e.g., standardized in [2] and provides means for public key encryption using elliptic curve groups. Essentially, it is a hybrid encryption scheme incorporating message integrity, i.e., providing CCA-security.

In order to present the basic algorithm we assume the following setup: We have a suitable elliptic curve group  $E(\mathbb{F}_q)$  with cofactor  $h$  and a generator  $P$  of a large prime order subgroup. The recipient of the encrypted message  $B$  has the keypair  $(\mathbf{sk}_B, \mathbf{pk}_B) = (d_B, Q_B = d_B \cdot P)$ .

Furthermore, the following list shows the cryptographic primitives being required for the ECIES to work:

- *KDF*: A key derivation function provides means for deriving a key from a piece of data (an elliptic curve point in our case). For further details see, e.g., [20].
- *MAC*: A message authentication code, also known as keyed hash function, provides means for authenticating a message w.r.t. a given key. See [3] for further details.
- An appropriate symmetric cipher, e.g., [2] proposes to use a simple exclusive or.

For a real world implementation of ECIES, additional details such as the validation of the public key would have to be considered. Since the implementation is not part of this thesis, these details are omitted for the sake of simplicity and only the encryption and decryption algorithms are presented.

**Algorithm 3.1** Elliptic Curve Integrated Encryption Scheme -  $\text{Enc}_{Q_B}$  [35]

---

```

1: Input:  $E(\mathbb{F}_q), P, Q_B, m, h, KDF, MAC$ 
2: while true do
3:   Choose:  $k \in_r \mathbb{Z}_{\#E(\mathbb{F}_q)}$ 
4:   Compute  $R = k \cdot P$  and  $Z = h \cdot k \cdot Q_B$ 
5:   if  $Z = \infty$  then
6:     continue
7:   end if
8:    $(k_1, k_2) = KDF(x_Z, R)$ , with  $x_Z$  being the  $x$ -coordinate of  $Z$ 
9:   Compute  $C = \text{Enc}_{k_1}(m)$  and  $t = \text{MAC}_{k_2}(C)$ 
10:  return  $(R, C, t)$ 
11: end while

```

---

**Algorithm 3.2** Elliptic Curve Integrated Encryption Scheme -  $\text{Dec}_{d_B}$  [35]

---

```

1: Input:  $E(\mathbb{F}_q), P, d_B, h, KDF, MAC, (R, C, t)$ 
2: Compute:  $Z = h \cdot d_B \cdot R$ 
3: if  $Z = \infty$  then
4:   return error
5: end if
6:  $(k_1, k_2) = KDF(x_Z, R)$ , with  $x_Z$  being the  $x$ -coordinate of  $Z$ 
7: Compute  $t' = \text{MAC}_{k_2}(C)$ 
8: if  $t' = t$  then
9:   return  $m = \text{Dec}_{k_1}(c)$ 
10: end if
11: return error

```

---

Note that ECIES is correct, since it holds that:

$$h \cdot k \cdot Q_B = h \cdot k \cdot d_B \cdot P = h \cdot d_B \cdot R.$$

### 3.1.4 Key Agreement

A key agreement constitutes a protocol for confidentially computing a common key over an insecure channel. The probably most popular key agreement scheme is the Diffie-Hellman key agreement [24], which also constituted the beginning of public key cryptography. This protocol makes use of the hardness of the CDHP and works as follows. With  $P$  being a generator of the used finite cyclic group  $\mathbb{G}$  of order  $p$ , the parties  $A$  and  $B$  agree on a key  $K_{AB}$  in the following way.

$$\begin{array}{ccc}
 A & & B \\
 \text{randomly choose } x \in \mathbb{Z}_p^* & & \text{randomly choose } y \in \mathbb{Z}_p^* \\
 X = x \cdot P & \xrightarrow{X} & \\
 & \xleftarrow{Y} & Y = y \cdot P \\
 K_{AB} = x \cdot Y = xy \cdot P & & K_{AB} = y \cdot X = yx \cdot P
 \end{array} \tag{3.4}$$

This scheme could, for instance, be instantiated by using an elliptic curve group, i.e.,  $P \in_g E(\mathbb{F}_q)$ . When such an elliptic curve group is used, the scheme is referred to as elliptic curve Diffie-Hellman (ECDH) key agreement.

When using a key agreement, one has to consider person-in-the-middle attacks as shown in Figure 3.2. Thereby, an adversary Eve intercepts both message  $X$  and message  $Y$  and establishes a shared key  $K_{AE}$  with Alice, and a shared key  $K_{BE}$  with Bob. Thus, Eve is able to wiretap the whole communication, whilst Alice and Bob are not aware of it.

Person-in-the-middle attacks can, for instance, be prevented by authentically providing certain algorithm parameters, e.g., the key agreement scheme shown in the next section makes use of the authentic public keys of the communication parties. Instead of directly using asymmetric encryption in the aforementioned case, one establishes session keys using the key agreement. This way the leakage of a session key does not compromise the whole system, since one can simply agree on a new key (*forward secrecy*).



Figure 3.2: Person in the middle attack

### The Elliptic Curve MQV Key Agreement

The following explanations are intended to illustrate the principle of the ECMQV key agreement. For practical implementations of ECMQV, e.g., as standardized in [2], further details need to be considered (see, e.g., [35]). However, they are omitted here since the implementation of ECMQV is not part of this thesis.

Our illustration of the ECMQV is based on [35]. We assume that parties  $A$  and  $B$  want to agree on a key. Given an appropriate elliptic curve group  $E(\mathbb{F}_q)$  of order  $p$ , and  $P \in {}_g E(\mathbb{F}_q)$ ,  $A$  has the keypair  $(\text{sk}_A, \text{pk}_A) = (d_a, Q_a = d_a \cdot P)$ , whereas  $B$  has the keypair  $(\text{sk}_B, \text{pk}_B) = (d_b, Q_b = d_b \cdot P)$ . Furthermore, with  $R$  being an elliptic curve point,  $\bar{r}$  is the integer representation of the  $x$  coordinate in  $\mathbb{Z}_p^*$  of  $R$ .

Based on the definitions above, we can introduce the ECMQV key agreement:

$$\begin{array}{ccc}
 A & & B \\
 k_A \in_r \mathbb{Z}_p, R_A = k_A \cdot P & \xrightarrow{R_A} & \\
 & \xleftarrow{R_B} & k_B \in_r \mathbb{Z}_p, R_B = k_B \cdot P \\
 s_A = (k_A + \bar{r}_A \cdot d_A) & & s_B = (k_B + \bar{r}_B \cdot d_B) \\
 K_{AB} = h \cdot s_A \cdot (R_B + \bar{r}_B \cdot Q_B) & & K_{AB} = h \cdot s_B \cdot (R_A + \bar{r}_A \cdot Q_A)
 \end{array} \quad (3.5)$$

Once a common point  $K_{AB}$  has been computed, a key can be derived using an arbitrary key derivation function  $KDF$ , see, e.g., [20]. If the public keys  $Q_A, Q_B$  of parties  $A$  and  $B$  are provided in an authentic manner, there is no longer a possibility for man-in-the-middle attacks.

Finally, it is easy to see that both parties obtain the same key, as it holds that:

$$\begin{aligned}
 h \cdot s_A \cdot (R_B + \bar{r}_B \cdot Q_B) &= h \cdot s_A \cdot (k_B + \bar{r}_B \cdot d_B) \cdot P = \\
 &= h \cdot s_A \cdot s_B \cdot P = \\
 &= h \cdot (k_A + \bar{r}_A \cdot d_A) \cdot s_B \cdot P = h \cdot s_B \cdot (R_A + \bar{r}_A \cdot Q_A).
 \end{aligned}$$

#### 3.1.5 Digital Signatures

Another application of public key cryptosystems are digital signatures schemes, denoted by  $\text{DSS} = (\text{DKeyGen}, \text{DSign}, \text{DVerify})$ . They allow to provide authenticity of messages, i.e., identification of the sender and checking the integrity of a received message. Given a key pair  $(\text{sk}_A, \text{pk}_A)$  of a party  $A$ , the secret key  $\text{sk}_A$  is used to issue a signature  $\sigma$  on a message  $m$ . Once the signature is issued, anyone in possession of the public key  $\text{pk}_A$  can verify the

signature. The signing algorithm  $\text{DSign}$  as well as the verification algorithm  $\text{DVerify}$  for a message  $m$  can be formally defined as follows:

$$\begin{aligned}\sigma &= \text{DSign}_{\text{sk}_A}(m) \\ \text{DVerify}_{\text{pk}_A}(m, \sigma) &\in \{1, 0\}\end{aligned}\tag{3.6}$$

For a digital signature scheme one requires, that for every potential message  $m$   $\text{DVerify}_{\text{pk}_A}(m, \text{DSign}_{\text{sk}_A}(m)) = 1$  for all  $(\text{sk}, \text{pk}) \in \text{DKeyGen}(\kappa)$ .

### Notion of Security

Similar to the security definitions of encryption schemes, we can also define the security of a signature scheme. In order to give a definition, we firstly need to introduce the chosen-message attack unforgeability game (Game 3.6) for digital signature schemes  $\text{DSS} = (\text{DKeyGen}, \text{DSign}, \text{DVerify})$ . Based on this game, Definition 3.6 introduces the security measure. The notion and the definition is, again, taken from [52].

1.  $\text{DKeyGen}(\kappa)$  is run to obtain keys  $(\text{sk}, \text{pk})$ .
2. Adversary  $\mathcal{A}$  is given  $\text{pk}$  and oracle access to  $\text{DSign}_{\text{sk}}(\cdot)$ . (This oracle returns a signature  $\sigma = \text{DSign}_{\text{sk}}(m)$  for any message  $m$  of the adversary's choice.) The adversary then outputs  $(m, \sigma)$ . Let  $\mathcal{Q}$  denote the set of messages whose signatures were requested by  $\mathcal{A}$  during its execution.
3. The output of the experiment is defined to be 1 if and only if:
  - $\text{DVerify}_{\text{pk}}(m, \sigma) = 1$ , and
  - $m \notin \mathcal{Q}$ .

Game 3.6: The CMA unforgeability game  $\text{Sign}_{\mathcal{A}, \text{DSS}}^{\text{uf-cma}}(\kappa)$  [52]

**Definition 3.6.** A signature scheme  $\text{DSS} = (\text{DKeyGen}, \text{DSign}, \text{DVerify})$  is existentially unforgeable under an adaptive chosen-message attack if for all probabilistic polynomial-time adversaries  $\mathcal{A}$ , there exists a negligible function  $\epsilon$  such that

$$\Pr \left[ \text{Sign}_{\mathcal{A}, \text{DSS}}^{\text{uf-cma}}(\kappa) = 1 \right] \leq \epsilon(\kappa).$$

### Elliptic Curve Digital Signature Algorithm

The elliptic curve digital signature algorithm (ECDSA), see, e.g., [31], constitutes a standardized signature algorithm making use of ECC. Thereby,  $E(\mathbb{F}_q)$  is an appropriate elliptic curve group of prime order  $p$  and  $P \in_g E(\mathbb{F}_q)$ . The signer  $A$  randomly chooses an integer  $d_A \in_r \mathbb{Z}_p^*$  as secret key and computes the public key  $Q_A = d_A \cdot P$ .

In order to sign a message  $m$ ,  $A$  firstly needs to uniquely encode the message  $m$  in an appropriate way, e.g., by using a cryptographic hash function  $H$  such as discussed in Section 3.1.7. The encoding of  $m$  is further referred to as  $H(m)$ .

Algorithm 3.3 illustrates the  $\text{DSign}$  algorithm and Algorithm 3.4 the  $\text{DVerify}$  algorithm. It can easily be shown why ECDSA is correct, since it holds that:

$$\begin{aligned}k &\equiv s^{-1} \cdot (H(m) + r \cdot \text{sk}_A) \equiv s^{-1} \cdot H(m) + s^{-1} \cdot r \cdot \text{sk}_A \equiv \\ &\equiv v \cdot H(m) + v \cdot r \cdot \text{sk}_A \equiv w_1 + w_2 \cdot \text{sk}_A \equiv k \pmod{p}, \text{ and} \\ w_1 \cdot P + w_2 \cdot \text{pk}_A &= \psi = w_1 \cdot P + w_2 \cdot \text{sk}_A \cdot P = (w_1 + w_2 \cdot \text{sk}_A) \cdot P = k \cdot P = (x, y).\end{aligned}$$



**Algorithm 3.3** Elliptic Curve Digital Signature Algorithm -  $\text{DSign}_{d_A}$  [35]

---

```

1: Input:  $d_A, H(m), E(\mathbb{F}_q), p, P$ 
2: while true do
3:   Choose:  $k \in_r \mathbb{Z}_p$ 
4:   With  $(x, y) = k \cdot P$  compute  $r = x \bmod p$ 
5:   if  $r = 0$  then
6:     continue
7:   end if
8:   Compute  $s = k^{-1} \cdot (H(m) + r \cdot d_A) \bmod p$ 
9:   if  $s = 0$  then
10:    continue
11:  end if
12: end while
13: return  $\sigma = (r, s)$ 

```

---

**Algorithm 3.4** Elliptic Curve Digital Signature Algorithm -  $\text{DVerify}_{Q_A}$  [35]

---

```

1: Input:  $\sigma = (r, s), Q_A, H(m), E(\mathbb{F}_q), p, P$ 
2: if  $r, s \notin [1, p[$  then
3:   return 0
4: end if
5: Compute  $v = s^{-1} \bmod p$ 
6: Compute  $w_1 = H(m) \cdot v \bmod p \wedge w_2 = r \cdot v \bmod p$ 
7: Compute  $\psi = w_1 \cdot P + w_2 \cdot Q_A$ 
8: if  $\psi = \infty$  then
9:   return 0
10: end if
11:  $(x, y) = \psi$ 
12: if  $x \neq r \bmod p$  then
13:   return 0
14: end if
15: return 1

```

---

### 3.1.6 Public Key Infrastructures

When using digital signatures and public key encryption in practice, some kind of binding between a real world person and a public key  $\text{pk}$  is required in order to be able to ensure the authenticity of a given public key. One possibility to achieve such a binding is to use certificates, constituting a signed tuple  $(\sigma, (\text{pk}_A, \text{id}_A))$ , with  $\text{id}_A$  denoting the identity of  $A$  and  $\sigma$  being a signature for the message  $(\text{pk}_A, \text{id}_A)$ . The signature on such a certificate is, thereby, issued by a certification authority (CA), which needs to be trusted by all participating parties. An infrastructure, providing such services, is called public key infrastructure (PKI).

The probably most famous PKI implementation is specified in RFC5280 [22], commonly referred to as PKIX. The basis of PKIX are so-called X.509 certificates. In addition to the signed tuple  $(\sigma, (\text{pk}_A, \text{id}_A))$ , introduced along with the basic idea of certificates, X.509 certificates incorporate several additional fields such as, e.g., possibilities for restricting the usage of the key or specifying a validity period for the certificate. Furthermore, PKIX [22] defines means for revoking certificates and checking their revocation status, which can for instance be useful in case of a key compromise. The revocation check can be performed in two ways:

- *Certificate Revocation Lists (CRLs)* constitute signed lists of revoked certificates together with a revocation date and a reason for their revocation. For further details

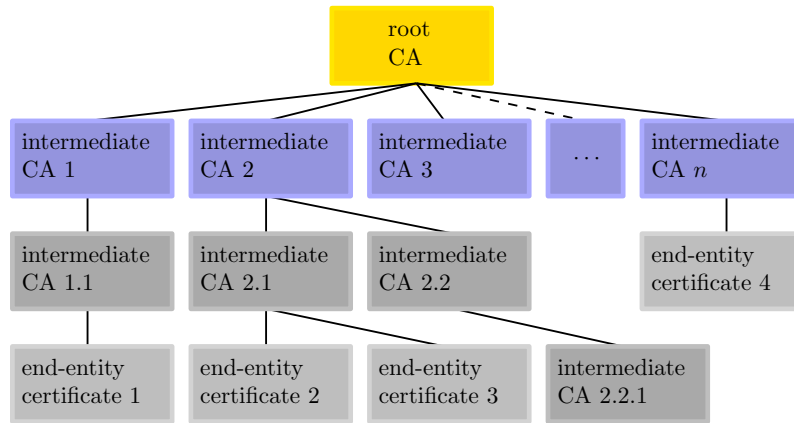


Figure 3.3: Example X.509 certificate hierarchy

we refer the reader to [22].

- *Online Certificate Status Protocol (OCSP)* is defined in [68] and defines means for requesting the status of a certificate at a webservice. It constitutes an alternative to CRLs.

Note that considering too old CRLs might be misleading, since some certificates might have been revoked in the timespan between issuance of the CRL and the time of the revocation check. The same problem might arise when dealing with cached OCSP responses.

Typically, a PKI infrastructure is organized in tree form, i.e., a certificate is always signed by the authority in the next higher level. Thus, once a CA certificate is trusted, all of its child-certificates are trusted as well. Whereas the certification authorities are always inner nodes in the tree, the end-entity certificates always constitute leaves in the tree (see Figure 3.3).

### 3.1.7 Cryptographic Hash Functions

In a nutshell, hash functions are functions mapping from bitstrings of arbitrary length to bitstrings of fixed length  $n$ :

$$H : \{0, 1\}^* \rightarrow \{0, 1\}^n.$$

**Definition 3.7.** *The following properties hold true for secure cryptographic hash functions:*

- (i)  *$H$  is efficiently computable.*
- (ii) **Preimage resistance:** *Given  $H(m)$ , it is intractable to find  $m$ .*
- (iii) **Second preimage resistance:** *Given  $H(m)$  and  $m$ , it is intractable to find  $m'$ , such that  $H(m) = H(m')$ .*
- (iv) **Collision resistance:** *It is intractable to find two messages  $m, m'$  with  $m \neq m'$ , such that  $H(m) = H(m')$ .*

The size of the hash value space, i.e.,  $2^n$ , gives a rough upper bound for finding collisions, preimages and second preimages, since hashing  $2^n + 1$  messages leads to at least one duplicate hash value. This bound can be tightened for finding collisions by using the

birthday paradox. For the following theorem, we assume an urn containing  $m$  balls numbered from 1 to  $m$  to be in place. Then,  $n$  balls are drawn from the urn, with replacement, i.e., after each draw, a ball with the same number is put back into the urn.

**Theorem 3.1** (Birthday Paradox). *For large values of  $n$  the first ball selected for the second time is expected to be drawn after  $\sqrt{\frac{\pi n}{2}}$  draws.*

When applying the birthday paradox to the collision resistance of hash functions, the number of expected trials can be reduced to  $2^{\frac{n}{2}}$ .

If there exists an attack for finding preimages or second preimages which can be completed in asymptotically less than  $2^n$  steps, or there exists an attack for finding collisions which can be completed in asymptotically less than  $2^{\frac{n}{2}}$  steps, respectively, a hash function is considered to be broken.

Hash functions have many applications in cryptography, and are, thus, widely used, e.g., for encoding the input of digital signature algorithms (hash-then-sign paradigm) in order to enable efficient signatures for larger messages and to prohibit existential forgery attacks. From a practical point of view, the most popular hashing algorithms are the NIST SHA-1 and SHA-2 algorithms. Quite recently, NIST selected Keccak [10] as their SHA-3 hashing algorithm after expecting weaknesses in the SHA-2 algorithm.

## 3.2 Proxy Signatures

Since proxy signatures constitute the basis for the two schemes implemented in this thesis, we want to briefly introduce the idea behind proxy signatures here. As already mentioned in Chapter 1, proxy signatures were firstly introduced in [60].

In contrast to conventional digital signature schemes, involving a signer and a verifier, proxy signature schemes involve three parties: an *originator*, a *proxy* and a *verifier*. Thereby, the *originator* delegates the signing rights for some particular well defined set of messages  $\mathcal{M} = \{M_1, M_2, \dots, M_n\}$  to a *proxy*. The designated *proxy* is then able to issue a signature on a message out of the set  $\mathcal{M}$  on behalf of the *originator*. Once such a signature has been issued, any *verifier* is able to check whether the *proxy* has produced the signature on behalf of the *originator* (authenticity), the integrity of the message and whether the given message is one of the "allowed" messages, i.e., contained in  $\mathcal{M}$ .

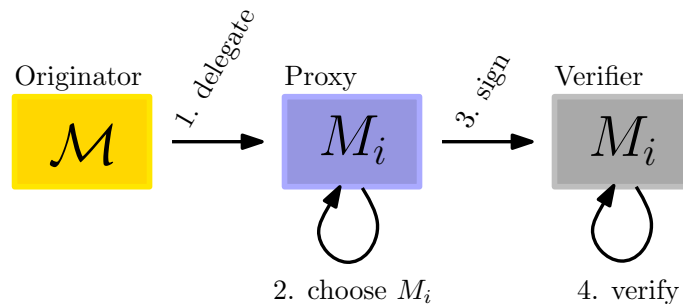


Figure 3.4: Schematic view of a proxy signature scheme

### 3.2.1 Delegation-by-Certificate

Basically, most constructions make use of conventional digital signatures for building proxy signature schemes. Thereby, the originator issues a so-called certificate by digitally signing

a description of the delegation together with the identity of the designated proxy. The proxy then issues a signature on a message conforming to the description of the originator. This signature, together with the certificate signed by the originator constitutes the proxy signature, which can be verified by simply checking both signatures and if the messages conforms to the description. Whereas, the naive approach would suggest to issue multiple certificates, each containing one possible message, state-of-the-art schemes often use a signed warrant, describing the delegation.

### 3.3 Commitments

In cryptography, commitments are used to commit to a certain value without the need for immediately revealing the value itself. From an abstract point of view, a commitment can be compared with a locked box with a particular content, being handed over to a second party without providing a key to the lock to open the box. Later on the key is provided and the second party can make sure if the content is the claimed content. They are used in many cryptographic applications, such as, e.g., the protocols implemented in this thesis. A commitment is formally defined by two protocol steps:

- $\text{Commit}(m)$  : In this algorithm a commitment  $\mathcal{C}$  to a message  $m$  is computed. This algorithm returns the commitment  $\mathcal{C}$  and the open information  $\mathcal{O}$ .
- $\text{Open}(\mathcal{C}, \mathcal{O})$  : This algorithm reveals message  $m$  in the commitment  $\mathcal{C}$  w.r.t. the open information  $\mathcal{O}$ . It returns  $m$  on success and  $\perp$  otherwise.

For a commitment scheme we require that for all potential messages  $m$  and corresponding tuples  $(\mathcal{C}, \mathcal{O}) = \text{Commit}(m)$  the algorithm  $\text{Open}(\mathcal{C}, \mathcal{O})$  returns  $m$ . Before we are able to take a closer look at the commitment schemes being required in the aforementioned protocols, we firstly need to introduce the *hiding* and the *binding* property of commitments.

**Definition 3.8** (Hiding property). *The commitment  $\mathcal{C}$  does not reveal the message  $m$ .*

**Definition 3.9** (Binding property). *It should be infeasible to find an open information  $\mathcal{O}'$  such that given a commitment  $\mathcal{C}$  with open information  $\mathcal{O}$ , the *Open* algorithm accepts  $\mathcal{O}'$ .*

Hiding and binding can be either computationally or information theoretically (unconditionally). But both properties can not be unconditional at the same time, since unconditional hiding implies that there are multiple messages yielding the same commitment, which breaks the unconditional binding property. Conversely, if there is only one possible message  $m$  for each commitment (unconditional binding) it is impossible to achieve unconditional hiding since one could simply try all possible messages in order to break the hiding.

The maximum length of the messages depends on the concrete scheme. Typically, one uses the hash-then-commit paradigm analogously to digital signatures to support larger messages.

#### 3.3.1 Discrete Logarithm and Pedersen Commitments

Two quite simple ways of computing a commitment are provided by the discrete logarithm and the Pedersen commitment. Both, discrete logarithm and Pedersen [73] commitment schemes are based on the assumption of the hardness of the DLP. Whilst discrete logarithm commitments are computationally hiding and unconditionally binding, Pedersen commitments are unconditionally hiding and computationally binding.

### Discrete Logarithm Commitments

Let  $\mathbb{G}$  be a finite cyclic group of order  $p$  and  $P \in_g \mathbb{G}$ , a discrete logarithm commitment w.r.t. the public parameter  $P$  (see, e.g., [21]) to a message  $m \in \mathbb{Z}_p$  is computed as follows:

$$\mathcal{C} = \text{Commit}(m) = m \cdot P. \quad (3.7)$$

The open information  $\mathcal{O}$  is  $m$  in this case. Discrete logarithm commitments are *computationally hiding*, since breaking the hiding property requires solving the discrete logarithm problem. Furthermore, they are *unconditionally binding*, since there exists only one  $m \in \mathbb{Z}_p$  such that (3.7) holds true.

### Pedersen Commitments

Pedersen commitments [73] in turn, are *unconditionally hiding* and *computationally binding*. These properties are achieved by computing a commitment as follows. Let  $\mathbb{G}$  be a finite cyclic group of order  $p$  and  $P, Q$  two distinct generators of  $\mathbb{G}$ . A commitment  $\mathcal{C}$  w.r.t. the public parameters  $(P, Q)$  to a message  $m \in \mathbb{Z}_p$  is computed by randomly choosing  $r \in_r \mathbb{Z}_p^*$  and computing

$$\mathcal{C} = \text{Commit}(m, r) = m \cdot P + r \cdot Q.$$

Here, the open information  $\mathcal{O} = (m, r)$ . The *unconditional hiding* property, however, requires the discrete logarithm between  $P$  and  $Q$  to be unknown, and thus the public parameters are required to be created by a trusted third party (TTP). The *unconditional hiding* property stems from the fact that, due to the random choice of  $r$ , each element in  $\mathbb{G}$  has the same likelihood to be the commitment to a message  $m$ . *Computational binding* is achieved due to the fact that breaking the binding property requires solving the discrete logarithm problem.

*Proof.* We prove the computational binding property of Pedersen commitments by polynomially reducing the discrete logarithm problem to it. Since  $Q = \alpha \cdot P$  for some unknown  $\alpha \in \mathbb{Z}_p^*$  the Pedersen commitment can be written as  $(m + \alpha r) \cdot P$ . In order to break the binding, one needs to find  $m'$  and  $r'$  such that  $m + \alpha r = m' + \alpha r' \pmod{p}$ . However, rewriting this equation yields  $\alpha = (m - m') \cdot (r - r')^{-1}$ . This is equivalent to solving the discrete logarithm for  $Q$  which gives the desired contradiction.  $\square$

Furthermore, trapdoor commitments allow for breaking the binding property when some trapdoor information is known. For the Pedersen commitments introduced above, this trapdoor information would be the discrete logarithm  $\alpha$  between  $P$  and  $Q$ . In the following section, we introduce a trapdoor commitment which is also known as chameleon hash function.

### 3.3.2 Chameleon Hash Functions

Similar to hash functions, a chameleon hashing scheme  $\text{CH} = (\text{CSetup}, \text{CKeyGen}, \text{CHash}, \text{CForge})$  basically maps a string of arbitrary length to a hash value of fixed length. In addition to that, a chameleon hash allows to find collisions for a given hash value with the knowledge of a secret key. Furthermore, the properties in Definition 3.10 apply to a secure chameleon hashing scheme.

**Definition 3.10** (Secure CH [21]). *For a secure chameleon hashing scheme the following properties hold true.*

- (i) **Collision resistance:** Without the knowledge of the secret key  $\text{csk}$ , it is intractable to find two messages  $m, m'$  such that  $\text{CHash}(\text{cpk}, m, r) = \text{CHash}(\text{cpk}, m', r)$ .
- (ii) **Semantic security:** No information about a message  $m$  can be derived from a given hash value  $\text{CHash}(\text{cpk}, m, r)$ .

Furthermore, the authors of [40] define the *message-binding* property of a chameleon hash as follows:

**Definition 3.11.** A chameleon hashing scheme  $\text{CH}$  is message-binding if for all potential messages the map  $m \mapsto \text{CHash}(\text{cpk}, m, r)$  is injective.

The chameleon hash used in this thesis was proposed by Chen et al. [21]. According to their definition, a chameleon hashing scheme is a tuple  $\text{CH} = (\text{CSetup}, \text{CKeyGen}, \text{CHash}, \text{CForge})$ . Thereby, the system parameters are generated in  $\text{CSetup}$ , whereas  $\text{CKeyGen}$  is the key generation algorithm outputting a keypair  $(\text{csk}, \text{cpk})$ . Moreover,  $\text{CHash}$  refers to the computation of the initial hash value  $h = \text{CHash}(\text{cpk}, m, r)$ , whereas  $\text{CForge}$  refers to the collision computation for a message  $m'$  such that  $\text{CHash}(\text{cpk}, m, r) = \text{CHash}(\text{cpk}, m', r)$ .

Based on this definition, Chen et al. [21] define two schemes, namely a basic and a full chameleon hashing scheme. The former allows to recover the secret key upon the verification of a hash value, whereas the latter does not reveal any information about the key upon the verification, i.e., it is *key exposure free*. Since we do not require key exposure freeness, only the basic chameleon hashing scheme is presented here (see Scheme 3.1). This scheme is message-binding [40], i.e., the map  $m \mapsto \text{CHash}(\text{cpk}, m, r)$  is injective for all potential messages  $m$  with fixed  $r$  and all keys  $(\text{csk}, \text{cpk}) \in \text{CKeyGen}(\kappa)$ .

<p><b>CSetup:</b> Let <math>\mathbb{G}</math> be a GDH group of prime order <math>p</math> and <math>P \in_g \mathbb{G}</math>. The public parameters are <math>\text{pp} = \{\mathbb{G}, p, P\}</math>.</p> <p><b>CKeyGen:</b> Each user chooses <math>\text{csk} \in_r \mathbb{Z}_p^*</math> as secret key and publishes the public key <math>\text{cpk} = \text{csk} \cdot P</math></p> <p><b>CHash:</b> Given the public key of a user, <math>a</math> is chosen randomly out of <math>\mathbb{Z}_p^*</math> and the tuple <math>r = (a \cdot P, a \cdot \text{cpk})</math> is computed. Furthermore, the hash value <math>H</math> of a message <math>m</math> is computed as</p> $\text{CHash}(\text{cpk}, m, r) = H = m \cdot P + a \cdot \text{cpk}.$ <p>The algorithm outputs the hash value <math>H</math> together with the randomizer <math>r</math>.</p> <p><b>CForge:</b> For any valid hash value <math>H</math> of a message <math>m</math> and the corresponding secret key <math>\text{csk}</math> the forgery for a message <math>m'</math> with <math>m \neq m'</math> is computed as</p> $\text{CForge}(\text{csk}, h, m, r, m') = r' = (a' \cdot P, a' \cdot \text{cpk})$ <p>with <math>a' \cdot P = a \cdot P + (x^{-1}(m - m')) \cdot P</math> and <math>a' \cdot \text{cpk} = a \cdot \text{cpk} + (m - m') \cdot P</math>. The algorithm outputs the randomizer <math>r'</math>, leading to the desired collision.</p>
--

Scheme 3.1: The basic chameleon hashing scheme [21]

It can easily be verified that the forgery of a given hash value works:

$$\begin{aligned} \text{CHash}(\text{cpk}, m', r') &= m' \cdot P + a' \cdot \text{cpk} = m' \cdot P + a \cdot \text{cpk} + (m - m') \cdot P = \\ &= m \cdot P + a \cdot \text{cpk} = \text{CHash}(\text{cpk}, m, r). \end{aligned}$$

When verifying the hash value for a given message, one can simply recompute the hash value and compare it to the given hash value. Furthermore, one has to check whether  $(P, \text{cpk}, a \cdot P, a \cdot \text{cpk})$  is a valid Diffie-Hellman tuple, e.g., for an implementation using pairing friendly curves as gap Diffie-Hellman group, check if  $e(P, a \cdot \text{cpk}) = e(a \cdot P, \text{cpk})$  holds true.

### 3.3.3 Polynomial Commitments

Based on the commitments introduced in Section 3.3.1, Kate et al. [51] proposed two schemes for constant-size commitments to polynomials. Whilst the first scheme is based on discrete logarithm commitments ( $\text{PolyCommit}_{\text{DL}}$ ), the second one makes use of Pedersen commitments ( $\text{PolyCommit}_{\text{Ped}}$ ).

Similar to the conventional discrete logarithm and Pedersen commitment case, polynomial discrete logarithm commitments are computationally hiding, whereas polynomial Pedersen commitments are unconditionally hiding. Unconditional binding cannot be achieved for both commitment types, since there are many polynomials leading to the same evaluation at certain points. Proofs for these properties are provided in [50]. These proofs are, amongst others, based on different variants of the Diffie-Hellman problem. The following list defines the basic protocol steps of a polynomial commitment scheme [51] (slightly simplified for our purposes).

**Setup( $\kappa, t$ ):** In the setup phase an appropriate group  $\mathbb{G}$  is chosen and a keypair  $(\text{sk}, \text{pk})$  for committing to a polynomial of degree  $\leq t$  is generated.

**Commit( $\text{pk}, \phi(x)$ ):** A commitment  $\mathcal{C}$  w.r.t. the polynomial  $\phi(x)$  is computed.

**Open( $\text{pk}, \mathcal{C}, \phi(x)$ ):** Outputs the polynomial  $\phi(x)$  corresponding to the commitment  $\mathcal{C}$ .

**VerifyPoly( $\text{pk}, \mathcal{C}, \phi(x)$ ):** Outputs 1 if  $\mathcal{C}$  is a commitment to  $\phi(x)$ , 0 otherwise.

**CreateWitness( $\text{pk}, \phi(x), i$ ):** Outputs the tuple  $(i, \phi(i), w_i)$ , with  $w_i$  being a witness for the evaluation  $\phi(i)$  of  $\phi(x)$  at  $i$ .

**VerifyEval( $\text{pk}, \mathcal{C}, i, \phi(i), w_i$ ):** Outputs 1 if  $\phi(i)$  is the evaluation of the polynomial committed to in  $\mathcal{C}$  and 0 otherwise.

When the secret key  $\text{sk}$  is known, we have a trapdoor commitment. Thus, the **Setup** step is required to be executed by a trusted third party in order to keep the trapdoor information secret. Furthermore, [51] defines the following properties for a secure and correct polynomial commitment scheme:

- **Correctness:** For all possible keys  $(\text{sk}, \text{pk}) = \text{Setup}(\kappa, t)$  and for all  $\phi(x)$  it holds that for all  $\mathcal{C} = \text{Commit}(\text{pk}, \phi(x))$  the output of  $\text{Open}(\text{pk}, \phi(x), i)$  is successfully verified by  $\text{VerifyPoly}(\text{pk}, \mathcal{C}, \phi(x))$ , and for all tuples  $(i, \phi(i), w_i) = \text{CreateWitness}(\text{pk}, \phi(x), i)$  it holds that  $\text{VerifyEval}(\text{pk}, \mathcal{C}, i, \phi(i), w_i) = 1$ .
- **Polynomial Binding:** For a commitment  $\mathcal{C}$  to a polynomial  $\phi(x)$  it is intractable to find a polynomial  $\phi'(x) \neq \phi(x)$  such that  $\text{VerifyPoly}(\text{pk}, \mathcal{C}, \phi(x)) = \text{VerifyPoly}(\text{pk}, \mathcal{C}, \phi'(x)) = 1$ .
- **Evaluation Binding:** For any commitment  $\mathcal{C}$  and corresponding witness  $(i, \phi(i), w_i)$  it is intractable to find a witness  $(i, \phi(i)', w_i')$  with  $\phi(i) \neq \phi(i)'$  such that  $\text{VerifyEval}(\text{pk}, \mathcal{C}, i, \phi(i), w_i) = \text{VerifyEval}(\text{pk}, \mathcal{C}, i, \phi(i)', w_i') = 1$ .
- **Hiding:** The commitment  $\mathcal{C}$  does not reveal any information about the polynomial  $\phi(x)$ .

<p><b>Setup</b>(<math>\kappa, t</math>): Two groups <math>\mathbb{G}, \mathbb{G}_T</math>, both of prime order <math>p</math>, providing <math>\kappa</math>-bit security are chosen. <math>\mathbb{G}</math> and <math>\mathbb{G}_T</math> are chosen in a way, that a bilinear pairing <math>e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T</math> exists. With <math>P, Q \in_g \mathbb{G}</math> and an <math>\text{sk} = \alpha \in_r \mathbb{Z}_p^*</math> (chosen by a TTP), the tuple <math>\mathcal{P} = (P, \alpha P, \dots, \alpha^t P)</math> is computed and the public key <math>\text{pk}</math> is as follows:</p> $\text{pk} = (e, \mathbb{G}, \mathbb{G}_T, \mathcal{P}).$ <p><b>Commit</b>(<math>\text{pk}, \phi(x)</math>): A commitment <math>\mathcal{C} = \phi(\alpha)P</math> is computed. Since <math>\alpha</math> is unknown, <math>\mathcal{C}</math> is computed as</p> $\mathcal{C} = \sum_{i=0}^{\deg(\phi)} \phi_i(\alpha^i P)$ <p>with <math>\phi_i</math> being the <math>i</math>'th coefficient of <math>\phi</math>.</p> <p><b>Open</b>(<math>\text{pk}, \mathcal{C}, \phi(x)</math>): Outputs <math>\phi(x)</math> for a commitment <math>\mathcal{C}</math>.</p> <p><b>VerifyPoly</b>(<math>\text{pk}, \mathcal{C}, \phi(x)</math>): Outputs 1 if <math>\mathcal{C} = \phi(\alpha)P</math> and 0 otherwise. The check is performed in the following way.</p> $\mathcal{C} \stackrel{?}{=} \sum_{i=0}^{\deg(\phi)} \phi_i(\alpha^i P)$ <p>Again, with <math>\phi_i</math> being the <math>i</math>'th coefficient of <math>\phi</math>.</p> <p><b>CreateWitness</b>(<math>\text{pk}, \phi(x), i</math>): Computes <math>\psi_i(x)</math> and <math>w_i</math> and outputs <math>(i, \phi(i), w_i)</math>, i.e.,</p> $\text{with } \psi_i(x) = \frac{\phi(x) - \phi(i)}{x - i} \text{ compute } w_i = \psi_i(\alpha)P.$ <p><b>VerifyEval</b>(<math>\text{pk}, \mathcal{C}, \phi(i), \psi(i), w_i</math>): Outputs 1 if <math>\phi(i)</math> is the evaluation of the polynomial committed to by <math>\mathcal{C}</math> at <math>i</math>, 0 otherwise. The check is performed as follows.</p> $e(\mathcal{C}, P) \stackrel{?}{=} e(w_i, \alpha P - iP) \cdot e(\phi(i)P, P)$
--

Scheme 3.2: PolyCommit<sub>DL</sub> [51]

Based on this, Scheme 3.2 and Scheme 3.3 illustrate the polynomial commitment schemes as proposed in [51]. For the sake of simplicity, we assume Type-1 pairings to be in place. However, the scheme can also be used with Type-2 or Type-3 pairings. For instance, Section 4.5.1 implicitly shows how to port this scheme to Type-3 pairings.

The equality of the pairings computed in **VerifyEval** for the discrete logarithm commitments is easy to verify:

$$\begin{aligned} e(w_i, \alpha P - iP) \cdot e(\phi(i)P, P) &= e(P, P)^{\psi_i(\alpha)(\alpha-i)} \cdot e(P, P)^{\phi(i)} = \\ &= e(P, P)^{\phi(\alpha) - \phi(i)} \cdot e(P, P)^{\phi(i)} = e(\phi(\alpha)P, P) = e(\mathcal{C}, P). \end{aligned}$$

Furthermore, it is easy to show that **VerifyEval** is correct in the Pedersen commitment case (we assume that  $Q = rP$ ):

$$\begin{aligned} e(w_i, \alpha P - iP) \cdot e(\phi(i)P + \hat{\phi}(i)Q, P) &= e(P, P)^{(\psi_i(\alpha) + r\hat{\psi}_i(\alpha))(\alpha-i)} \cdot e(P, P)^{\phi(i) + r\hat{\phi}(i)} = \\ &= e(P, P)^{(\phi(\alpha) - \phi(i) + r(\hat{\phi}(\alpha) - \hat{\phi}(i)))} \cdot e(P, P)^{\phi(i) + r\hat{\phi}(i)} = e(P, P)^{\phi(\alpha) + r\hat{\phi}(\alpha)} = \\ &= e(\phi(\alpha)P + \hat{\phi}(\alpha)Q, P) = e(\mathcal{C}, P). \end{aligned}$$

### 3.3.4 Merkle Hash Trees

Merkle hash trees, as proposed in [64], were originally intended for one-time signatures and are now heavily used for authenticating large pieces of data. Thereby, the data  $M$  is divided into distinct blocks  $M_i$  and considered as a sequence

$$\mathcal{M} = (M_1, M_2, \dots, M_n).$$



**Setup**( $\kappa, t$ ): Two groups  $\mathbb{G}, \mathbb{G}_T$ , both of prime order  $p$ , providing  $\kappa$ -bit security are chosen.  $\mathbb{G}$  and  $\mathbb{G}_T$  are chosen in a way, that a bilinear pairing  $e: \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$  exists. With  $P, Q \in_g \mathbb{G}$  and an  $\text{sk} = \alpha \in_r \mathbb{Z}_p^*$  (chosen by a TTP), the tuples  $\mathcal{P} = (P, \alpha P, \dots, \alpha^t P)$  and  $\mathcal{Q} = (Q, \alpha Q, \dots, \alpha^t Q)$  are computed and the public key  $\text{pk}$  is as follows:

$$\text{pk} = (e, \mathbb{G}, \mathbb{G}_T, \mathcal{P}, \mathcal{Q}).$$

**Commit**( $\text{pk}, \phi(x)$ ): A random polynomial  $\hat{\phi}(x) \in \mathbb{Z}_p[x]$  of degree  $t$  is chosen and a commitment  $\mathcal{C} = \phi(\alpha)P + \hat{\phi}(\alpha)Q$  is computed. Since  $\alpha$  is unknown,  $\mathcal{C}$  is computed as

$$\mathcal{C} = \sum_{i=0}^{\deg(\phi)} \phi_i(\alpha^i P) + \sum_{i=0}^{\deg(\hat{\phi})} \hat{\phi}_i(\alpha^i Q)$$

with  $\phi_i$  and  $\hat{\phi}_i$  being the coefficients of  $\phi$  and  $\hat{\phi}$  respectively.

**Open**( $\text{pk}, \mathcal{C}, \phi(x), \hat{\phi}(x)$ ): Outputs  $\phi(x)$  and  $\hat{\phi}(x)$ .

**VerifyPoly**( $\text{pk}, \mathcal{C}, \phi(x), \hat{\phi}(x)$ ): Outputs 1 if  $\mathcal{C} = \phi(\alpha)P + \hat{\phi}(\alpha)Q$  and 0 otherwise. The check is performed in the following way.

$$\mathcal{C} \stackrel{?}{=} \sum_{i=0}^{\deg(\phi)} \phi_i(\alpha^i P) + \sum_{i=0}^{\deg(\hat{\phi})} \hat{\phi}_i(\alpha^i Q)$$

Again, with  $\phi_i$  and  $\hat{\phi}_i$  being the coefficients of  $\phi$  and  $\hat{\phi}$  respectively.

**CreateWitness**( $\text{pk}, \phi(x), \hat{\phi}(x), i$ ): Computes  $\psi_i(x), \hat{\psi}_i(x)$  and  $w_i$  and outputs  $(i, \phi(i), \hat{\phi}(i), w_i)$ , i.e.,

$$\text{with } \psi_i(x) = \frac{\phi(x) - \phi(i)}{x - i} \wedge \hat{\psi}_i(x) = \frac{\hat{\phi}(x) - \hat{\phi}(i)}{x - i} \text{ compute } w_i = \psi_i(\alpha)P + \hat{\psi}_i(\alpha)Q.$$

**VerifyEval**( $\text{pk}, \mathcal{C}, \phi(i), \psi(i), w_i$ ): Outputs 1 if  $\phi(i)$  is the evaluation of the polynomial committed to by  $\mathcal{C}$  at  $i$ , 0 otherwise. The check is performed as follows.

$$e(\mathcal{C}, P) \stackrel{?}{=} e(w_i, \alpha P - iP) \cdot e(\phi(i)P + \hat{\phi}(i)Q, P)$$

Scheme 3.3: PolyCommit<sub>Ped</sub> [51]

For each block  $M_i$  in the sequence, a hash value  $H(m_i)$  is computed. The sequence of hash values

$$\mathcal{H} = (H(M_1), \dots, H(M_n)).$$

then constitutes the leafs of a binary tree, the so-called Merkle hash tree. The leafs of the binary tree are attached from left to right. Once the sequence of hash values is computed, the values for each inner tree node can be computed by hashing the concatenated values of the left child-node  $l$  and the right child-node  $r$ . The computation of the root hash of such a tree can be recursively defined as:

$$h_{\text{parent}} = \begin{cases} H(l||r) & \text{if parent has two child nodes, and} \\ H(l) & \text{otherwise.} \end{cases}$$

This computation rule is recursively applied till the root of the tree is reached. Once the root hash value is computed, one can simply authenticate a message  $M_i$  out of the sequence by revealing the root hash value together with  $M_i$  and the values of the nodes in the *authentication path*. Starting from the message  $M_i$  to be authenticated, the *authentication path* is the sequence of all siblings on the unique path to the root of the tree. The upper bound for the length of the authentication path is  $\log_2 n$ . Figure 3.5 shows an example hash tree for 7 messages.

Using the tree in Figure 3.5, one can, for instance, authenticate  $M_2$  by simply computing the root hash value as follows:

$$H(h_{1-4}||h_{5-7}) = H(H(H(M_1, M_2)||h_{34})||h_{5-7}).$$

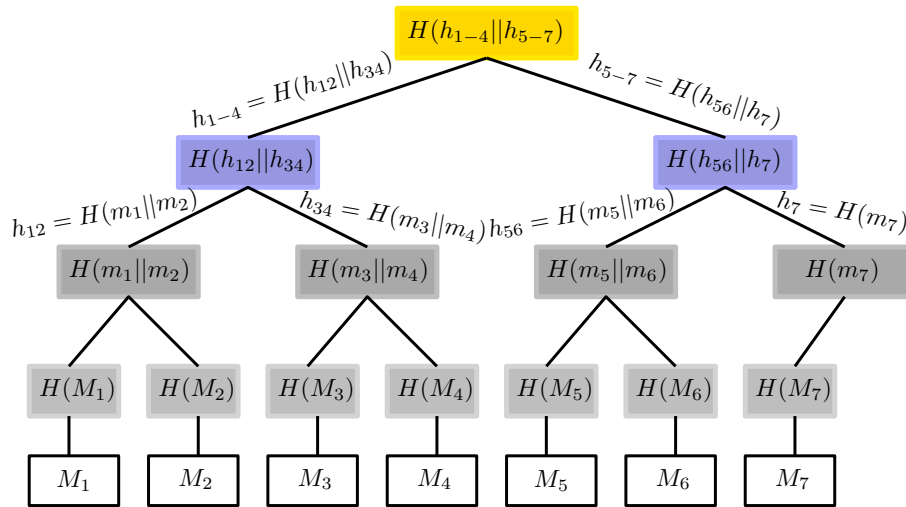


Figure 3.5: Merkle Hash Tree

Assuming that the root hash is provided in an authentic manner, one can authenticate a piece of data  $M_i$  this way. Since only the hash values in the authentication path are needed, the authentication of a value can take place without the knowledge of all remaining data pieces but only  $\log_2 n$  additional hash values. This especially saves a lot of time and bandwidth for large amounts of data.

Furthermore, Merkle hash trees can also be used as commitment scheme, i.e., the root hash is the commitment. However, when opening a commitment, which constitutes revealing the respective message  $M_i$  and corresponding authentication path, the direct sibling  $H(M_{i+1})$  or  $H(M_{i-1})$  becomes public. This, in turn, allows an adversary for testing arbitrary messages against the revealed hash values. In order to solve this problem one can make use of so-called randomized Merkle trees.

### Randomized Merkle Hash Trees

As already outlined above, the original Merkle tree construction allows for testing messages against hash values which were revealed in a preceding opening phase. Thus, randomized Merkle hash trees extend the original approach by using commitments with unconditional hiding property for the leafs of the tree (leaf commitments). When now, one wants to open a commitment, the elements in the authentication path do not reveal anything about the other messages contained in the commitment.

Note that the simplest way of creating a hiding commitment is to hash the value of each leaf together with a random number as for instance used within redactable signatures [45].

$$h_i = H(m_i || r) \text{ with } r \in_r \{0, 1\}^n \text{ for large enough } n, \text{ e.g., } n = 128.$$

This way, the previously outlined brute-force attack against elements in the authentication path is no longer possible.

For the rest of this thesis we assume the leaf commitments to be Pedersen commitments based on an elliptic curve group.

For our further explanations we also need the concept of vector commitments [17], which can, for instance, be built from randomized Merkle hash trees.

### 3.3.5 Vector Commitments

In this section we restate the notion of the vector commitment scheme shown in [17, 37]. Basically, vector commitments allow for committing to a vector and selectively opening the commitment for certain elements out of this vector. The following list gives an abstract overview over the algorithms presented in [37]. In contrast to the vector commitment defined in [17], the used vector commitment requires no update functionality. However, as discussed in [37], an update functionality could also be realized for the scheme in place.

**VKeyGen:** Generates the required parameters for the used commitment on input of a security parameter  $\kappa$ .

**VCommit:** A commitment to a sequence of messages  $\mathcal{M} = (M_1, \dots, M_n)$  w.r.t. a sequence of randomizers  $\mathcal{R} = (r_1, \dots, r_n)$  is computed and the commitment  $\mathcal{C}$  is returned as output.

**VOpen:** On input of a sequence of messages  $\mathcal{M}$ , the corresponding randomizers  $\mathcal{R}$  and an index  $i$  referring to an element  $M_i$  in the sequence  $\mathcal{M}$  this algorithm outputs the open information  $W_{M_i}$  for the element at position  $i$ .

**VVerify:** Given a commitment  $\mathcal{C}$  and an open information  $W_{M_i}$  together with an index  $i$  and the corresponding  $M_i$  and  $r_i$ , this algorithm recomputes the commitment and outputs 1 if  $\mathcal{C}$  equals the recomputed commitment and 0 otherwise.

As already outlined above, the aforementioned algorithms can be straightforwardly implemented using randomized Merkle hash trees with  $\mathcal{C}$  being the root hash of the tree (see [37]) and the open information  $W_{M_i}$  being the authentication path for the element at position  $i$ .

## Part II

# Proxy-Type Digital Signature Schemes

## Chapter 4

# Blank Digital Signatures

In this chapter we provide a comprehensive overview over the Blank Digital Signature scheme (BDSS) [38,39] as well as the extended Blank Digital Signature scheme (EBDSS) [40] which was proposed as an extension to the BDSS. We discuss the basic building blocks, as well as the principles of signature generation and validation and present the schemes in detail. Both schemes build upon a combination of standard digital signature schemes (see Section 3.1.5) and polynomial Pedersen commitments (see Section 3.3.3) as proposed by Kate et al. [51].

This chapter is organized as follows. Firstly, we introduce the basic idea of the schemes and we give a basic overview over the required setup. Then, the required encoding of the inputs for both schemes is introduced and the schemes are stated. Since the BDSS and the EBDSS only differ in the way of encoding templates and messages, most of the sections in this chapter apply to both schemes. Finally we introduce our optimizations regarding the reduction of the computation times of both schemes.

### 4.1 Basic Notion

The BDSS as well as the EBDSS, allow an *originator* to delegate the signing rights for a certain template to a *proxy*. Based on such a delegation, the *proxy* is able to issue a signature on a filled in version of the template (further referred to as instance of the template) on behalf of the *originator*.

In the BDSS, a template  $\mathcal{T}$  is a sequence of non-empty sets of bitstrings  $T_i$ , where such sets are either called *fixed* or *exchangeable*, depending on the cardinality of the respective set, i.e., exchangeable elements contain more than one bitstring, whereas fixed elements contain exactly one bitstring. Such a template is formally defined as follows:

$$T_i = \{M_{i_1}, M_{i_2}, \dots, M_{i_k}\}, \mathcal{T} = (T_1, T_2, \dots, T_n).$$

Given that, the template length is defined as the sequence length  $n$  of the template, while the template size  $|\mathcal{T}|$  is defined as  $|\mathcal{T}| = \sum_{i=1}^n |T_i|$ . Once the template is defined, the *originator* issues a signature for this template, which also specifies the proxy who is allowed to produce signatures for instantiations of this template. Based on this so called template signature, the designated *proxy* can choose concrete values for each exchangeable element and compute a so called instance signature on an instance  $\mathcal{M}$  of the template, which is formally defined as  $\mathcal{M} = (M_i)_{i=1}^n$ .

With the instance signature at hand, anyone is able to verify the validity of the instance signature, i.e., if it is a valid instantiation of the template and the delegation, whilst the

original template, i.e., the unused values of the exchangeable elements of the template, do not get revealed. The authors of [39] refer to this property as *privacy property*. Figure 4.1 shows an exemplary BDSS protocol execution. As shown in this figure, it is also possible to encode yes-/no-choices within a template by simply encoding yes and no in an exchangeable element.

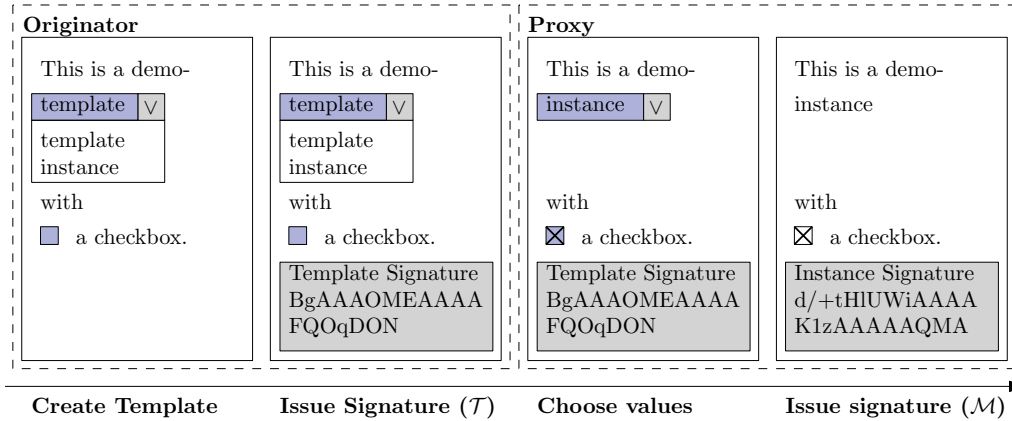


Figure 4.1: Schematic view of the BDSS

The extended Blank Digital Signature scheme (EBDSS) further extends the BDSS with the possibility for using so-called *blank* elements. These elements are intended to be replaced by arbitrary strings of predefined maximal length  $l$  and are realized by using the basic chameleon hashing scheme discussed in Section 3.3.2. Figure 4.2 illustrates a sample template running through an EBDSS protocol execution.

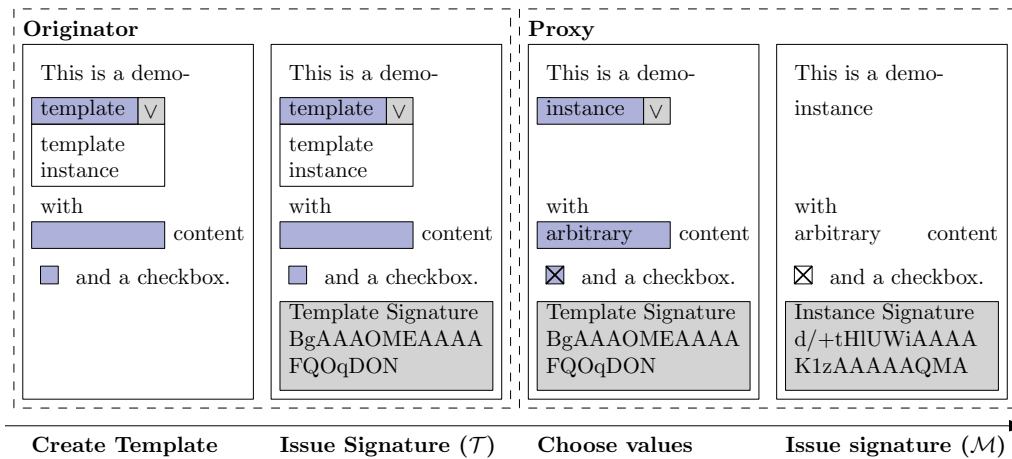


Figure 4.2: Schematic view of the EBDSS

## 4.2 Setup

Both schemes are designed for pairing friendly elliptic curve groups  $\mathbb{G}$  with subgroups of large prime order  $p$  and generator  $P$ . Subsequently, the following setup is assumed to be in place

- A bilinear pairing  $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ ,

- a full-domain cryptographic hash function  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ ,
- a digital signature scheme  $\text{DSS} = (\text{DKeyGen}, \text{DSign}, \text{DVerify})$ , and
- a public key infrastructure for retrieving the public signature keys.

In addition to that, the EBDSS requires

- a chameleon hashing scheme  $\text{CH} = (\text{CKeyGen}, \text{CHash}, \text{CForge})$ , based on the GDH group  $\mathbb{G}$ , with  $\text{CKeyGen}$  being a probabilistic algorithm, outputting the key pair  $(\text{csk}, \text{cpk})$  on input of the security parameter  $\kappa$ .

Building up on this setup, on input of a security parameter  $\kappa$  and an upper bound  $t \in \mathbb{N}$  for the size of a template, a TTP generates the public system parameters  $\text{pp}$  by choosing  $\alpha \in_r \mathbb{Z}_p^*$  and outputting

$$(H, \mathbb{G}, e, p, P, \alpha P, \dots, \alpha^t P)$$

in the BDSS case and

$$(H, \text{CH}, \mathbb{G}, e, p, P, \alpha P, \dots, \alpha^t P)$$

in the EBDSS case. Note that keeping  $\alpha$  secret is crucial for the security of both schemes, since this is the trapdoor for the polynomial commitment scheme.

### 4.3 Template and Message Encoding

Since the BDSS constructions are based on polynomial commitments, it requires a unique polynomial encoding of the inputs for the signing algorithms, i.e., templates and messages. The BDSS uses the following (unique) encoding for a template  $\mathcal{T}$  with unique identifier  $id_{\mathcal{T}}$  out of the set of all templates  $\mathbb{T}$ . The encoding function  $t : \mathbb{T} \rightarrow \mathbb{Z}_p[X]$  results in the so-called template encoding polynomial, i.e.,  $t(\mathcal{T}) = t_{\mathcal{T}}(X) \in \mathbb{Z}_p[X]$  with

$$\mathcal{T} \mapsto \prod_{i=1}^n \prod_{M \in T_i} (X - H(M || id_{\mathcal{T}} || i)).$$

Further, an instantiation (message)  $\mathcal{M}$  out of the set of all possible instantiations  $\mathbb{M}_{\mathcal{T}}$  for template  $\mathcal{T}$  is encoded in the following way. Again, the encoding function  $m_{\mathcal{T}} : \mathbb{M}_{\mathcal{T}} \rightarrow \mathbb{Z}_p[X]$  results in the so-called message encoding polynomial, i.e.,  $m_{\mathcal{T}}(\mathcal{M}) = m_{\mathcal{M}}(X) \in \mathbb{Z}_p[X]$  with

$$\mathcal{M} \mapsto \prod_{i=1}^n (X - H(M_i || id_{\mathcal{T}} || i)).$$

Finally, the complementary message encoding polynomial  $\bar{m}_{\mathcal{T}} \in \mathbb{Z}_p[X]$  is defined as

$$\bar{m}_{\mathcal{T}} = \frac{t_{\mathcal{T}}(X)}{m_{\mathcal{M}}(X)},$$

i.e., it contains the unused choices in the exchangeable element fields. The basic idea behind this encoding is that all allowed elements in the template are roots of the template encoding polynomial. As already mentioned before, the EBDSS additionally supports blank elements, and, thus, the extended encoding is presented in the following section.

### 4.3.1 Extending the Encoding for Blank Elements

In order to integrate the new element type, i.e., *blank* elements, into the existent encoding of templates and messages, blank elements are treated similar as fixed elements. The cardinality of the sets  $T_i$  corresponding to blank elements is 1. Further, blank elements are initially set to  $\perp$ , an encoding for the empty string, whereas  $\perp$  is replaced by an arbitrary string of maximum length  $l$  in an instantiation of the template. As mentioned before, this replacement is made possible by using a chameleon hash function.

The EBDSS uses the following (unique) encoding for a template  $\mathcal{T}$  with unique identifier  $id_{\mathcal{T}}$ . Assuming the same setup as above, the encoding results in the template encoding polynomial  $t_{\mathcal{T}}(X) \in \mathbb{Z}_p[X]$  through the following map

$$\mathcal{T} \mapsto \prod_{i=1}^n \begin{cases} X - H(\text{CHash}(\text{cpk}, H(\perp), r) || id_{\mathcal{T}} || i || l_i) & \text{for blank elements with} \\ & \text{maximal length } l_i, \text{ and} \\ \prod_{M \in T_i} (X - H(M || id_{\mathcal{T}} || i)) & \text{otherwise.} \end{cases}$$

Similar to the BDSS encoding, the instantiation (message)  $\mathcal{M}$  results in the message encoding polynomial  $m_{\mathcal{M}}(X) \in \mathbb{Z}_p[X]$  with

$$\mathcal{M} \mapsto \prod_{i=1}^n \begin{cases} X - H(\text{CHash}(\text{cpk}, H(M_i), r') || id_{\mathcal{T}} || i || l_i) & \text{for blank elements with} \\ & \text{maximal length } l_i, \text{ and} \\ X - H(M_i || id_{\mathcal{T}} || i) & \text{otherwise.} \end{cases}$$

The complementary message encoding polynomial is computed in the same way as previously described. Since CHash outputs an elliptic curve point, a unique encoding of this point again needs to be hashed using  $H$  [40]. Further the input of the chameleon hash is hashed (using  $H$ ) in order to enable larger message spaces. Thus, the authors of [40] define the following Lemma.

**Lemma 4.1.** *A composition of a secure chameleon hash function and a secure (full-domain) cryptographic hash function is a secure chameleon hash function.*

For convenience of the reader, we included the proof for the lemma, as shown in [40], in Appendix C.2.

When comparing the BDSS and the EBDSS encoding, one can see that the output of each encoding function is a polynomial  $f(X) \in \mathbb{Z}_p[X]$ . As already outlined before, this means that it is possible to use the same scheme for both encodings.

## 4.4 Scheme

Formally, the authors of [39, 40] define a Blank Digital Signature scheme as a tuple  $\text{BDSS} = (\text{KeyGen}, \text{Sign}, \text{Verify}_{\mathcal{T}}, \text{Inst}, \text{Verify}_{\mathcal{M}})$  of polynomial-time algorithms. The following list gives an abstract description of these algorithms.

### 4.4.1 Construction

**KeyGen:** On input of a security parameter  $\kappa$  and an upper bound for the template size, the public parameters  $\text{pp}$  are generated.



**Sign:** Given the template  $\mathcal{T}$ , the public parameters  $\text{pp}$ , the private signing key of the originator ( $\text{dsk}_O$ ) and the public signing key of the proxy ( $\text{dpk}_P$ ), this algorithm outputs a template signature  $\sigma_{\mathcal{T}}$  and a private template signing key for the proxy  $\text{sk}_P^{\mathcal{T}}$ .

**Verify $_{\mathcal{T}}$ :** Given the template  $\mathcal{T}$ , the template signature  $\sigma_{\mathcal{T}}$ , the public parameters  $\text{pp}$ , the public signing keys of originator and proxy ( $\text{dpk}_O, \text{dpk}_P$ ) and the template signing key of the proxy  $\text{sk}_P^{\mathcal{T}}$ , it checks whether  $\sigma_{\mathcal{T}}$  is a valid signature for  $\mathcal{T}$  and outputs 1 whether this check succeeds and 0 otherwise.

**Inst:** On input of a template  $\mathcal{T}$ , a corresponding instance  $\mathcal{M}$ , a signature on the template  $\sigma_{\mathcal{T}}$ , as well as the public parameters  $\text{pp}$ , the private template signing key  $\text{sk}_P^{\mathcal{T}}$  and the private signing key of the proxy  $\text{dsk}_P$ . It outputs a signature on the message  $\sigma_{\mathcal{M}}$ .

**Verify $_{\mathcal{M}}$ :** When given an instance  $\mathcal{M}$  of a template  $\mathcal{T}$ , a signature on this instance  $\sigma_{\mathcal{M}}$ , the public system parameters  $\text{pp}$  and the public signing keys of proxy and originator ( $\text{dpk}_O, \text{dpk}_P$ ), this algorithm verifies whether  $\sigma_{\mathcal{M}}$  is a valid signature on  $\mathcal{M}$  and if  $\mathcal{M}$  is a correct instantiation of  $\mathcal{T}$ .

Scheme 4.1 restates the protocol proposed in [40].

#### 4.4.2 Security

Informally, the security of a BDSS is defined as follows.

**Definition 4.1** (Security of a BDSS [38]). *A secure BDSS fulfills the following properties:*

**Correctness:** *For all keys  $(\text{dsk}_O, \text{dpk}_O) \in \text{DKeyGen}(\kappa)$ ,  $(\text{dsk}_P, \text{dpk}_P) \in \text{DKeyGen}(\kappa)$ ,  $\text{pp} \in \text{KeyGen}(\kappa, t)$  it is required that for any template  $\mathcal{T}$  and honestly computed template signature  $\sigma_{\mathcal{T}} = \text{Sign}(\mathcal{T}, \text{pp}, \text{dsk}_O, \text{dsk}_P)$  and corresponding  $\text{sk}_P^{\mathcal{T}}$*

- *Verify $_{\mathcal{T}}(\mathcal{T}, \sigma_{\mathcal{T}}, \text{pp}, \text{dpk}_O, \text{sk}_P^{\mathcal{T}}, \text{dpk}_P) = 1$  holds,*
- *for any honestly computed message signature  $\sigma_{\mathcal{M}} = \text{Inst}(\mathcal{T}, \mathcal{M}, \sigma_{\mathcal{T}}, \text{pp}, \text{sk}_P^{\mathcal{T}}, \text{dsk}_P)$  the verification  $\text{Verify}_{\mathcal{M}}(\mathcal{M}, \sigma_{\mathcal{M}}, \text{pp}, \text{dpk}_P, \text{dpk}_O) = 1$  holds, and*
- *it is intractable to find a  $(\text{sk}_P^{\mathcal{T}^*}, \mathcal{T}^*) \neq (\text{sk}_P^{\mathcal{T}}, \mathcal{T})$  such that  $\sigma_{\mathcal{T}}$  verifies for  $(\text{sk}_P^{\mathcal{T}^*}, \mathcal{T}^*)$ .*

**Unforgeability:** *Without the knowledge of  $\text{dsk}_O, \text{dsk}_P$  and  $\text{sk}_P^{\mathcal{T}}$  it is intractable to forge or existentially forge template or message signatures.*

**Immutability:** *When given  $\text{sk}_P^{\mathcal{T}}, \text{dsk}_P, \mathcal{T}$  and  $\sigma_{\mathcal{T}}$  it is intractable to forge or existentially forge template signatures or to forge signatures for messages which are not described in the template.*

**Privacy:** *It is intractable to determine template elements (except the ones revealed by instantiations) without the knowledge of  $\text{dsk}_O, \text{dsk}_P$  and  $\text{sk}_P^{\mathcal{T}}$ .*

**Theorem 4.1** (Security of Scheme 4.1 [40]). *Assuming that the  $t$ -SDH assumption in  $\mathbb{G}$  holds, the existence of secure hash functions and secure digital signature schemes, Scheme 4.1 is secure and correct w.r.t. Definition 4.1.*

For a proof of the theorem above, we refer the reader to [40].

<p><b>KeyGen:</b> On input <math>(\kappa, t)</math>, choose an elliptic curve with a subgroup <math>\mathbb{G}</math> of large prime order <math>p</math> generated by <math>P</math>, such that the bitlength of <math>p</math> is <math>\kappa</math>. Choose a pairing <math>e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T</math> and a full-domain cryptographic hash function <math>H : \{0, 1\}^* \rightarrow \mathbb{Z}_p</math> for use with the encoding functions. Pick <math>\alpha \in_R \mathbb{Z}_p^*</math>, compute <math>(\alpha P, \dots, \alpha^t P)</math> and output <math>\mathbf{pp} = (H, \mathbb{G}, e, p, P, \alpha P, \dots, \alpha^t P)</math>.</p> <p><b>Sign:</b> Given <math>\mathcal{T}, \mathbf{pp}, \mathbf{dsk}_O</math> and <math>\mathbf{dpk}_P</math>, where <math>\mathcal{T}</math> is a template of size <math> \mathcal{T}  = \ell</math> and length <math>n</math> with <math>\ell &gt; n</math>, this algorithm picks a unique <math>id_{\mathcal{T}} \in_R \{0, 1\}^{\kappa}</math>, computes <math>t_{\mathcal{T}} = t(\mathcal{T}) \in \mathbb{Z}_p[X]</math>, picks a secret <math>\rho \in_R \mathbb{Z}_p^*</math> and computes</p> $\mathcal{C} = e(\rho \cdot t_{\mathcal{T}}(\alpha)P, P) \quad \text{and} \quad \tau = \text{DSign}(id_{\mathcal{T}} \parallel \mathcal{C} \parallel n \parallel \mathbf{dpk}_P, \mathbf{dsk}_O)$ <p>and returns the template signature <math>\sigma_{\mathcal{T}} = (id_{\mathcal{T}}, \mathcal{C}, n, \tau)</math> as well as <math>\mathbf{sk}_P^{\mathcal{T}} = \rho</math>.</p> <p><b>Verify<math>_{\mathcal{T}}</math>:</b> Given <math>\mathcal{T}, \sigma_{\mathcal{T}}, \mathbf{pp}, \mathbf{dpk}_O, \mathbf{sk}_P^{\mathcal{T}}</math> and <math>\mathbf{dpk}_P</math>, where <math>\mathcal{T}</math> is a template of size <math> \mathcal{T}  = \ell</math> and length <math>n</math> with <math>\ell &gt; n</math>, this algorithm checks whether <math> \mathcal{T}  \leq t</math>. If not, it returns 0. Otherwise, it computes <math>t_{\mathcal{T}} = t(\mathcal{T})</math> and checks whether</p> $\text{DVerify}(\tau, id_{\mathcal{T}} \parallel \mathcal{C} \parallel n \parallel \mathbf{dpk}_P, \mathbf{dpk}_O) = 1 \quad \wedge \quad e(\rho \cdot t_{\mathcal{T}}(\alpha)P, P) = \mathcal{C}.$ <p>If so, return 1 and 0 otherwise.</p> <p><b>Inst:</b> Given <math>\mathcal{T}, \mathcal{M}, \sigma_{\mathcal{T}}, \mathbf{pp}, \mathbf{sk}_P^{\mathcal{T}}</math> and <math>\mathbf{dsk}_P</math>, where <math>\mathcal{T}</math> is a template of size <math> \mathcal{T}  = \ell</math> and length <math>n</math> with <math>\ell &gt; n</math>, this algorithm computes <math>m_{\overline{\mathcal{M}}} = \overline{m}_{\mathcal{T}}(\mathcal{M}) \in \mathbb{Z}_p[X]</math>. Then, it computes</p> $\mathcal{C}_{\overline{\mathcal{M}}} = \rho \cdot m_{\overline{\mathcal{M}}}(\alpha)P \quad \text{and} \quad \mu = \text{DSign}(\tau \parallel \mathcal{C}_{\overline{\mathcal{M}}} \parallel \mathcal{I}, \mathbf{dsk}_P).$ <p>It returns <math>\sigma_{\mathcal{M}} = (\mu, \mathcal{C}_{\overline{\mathcal{M}}}, \mathcal{I}, \sigma_{\mathcal{T}})</math>.</p> <p><b>Verify<math>_{\mathcal{M}}</math>:</b> Given <math>\mathcal{M}, \sigma_{\mathcal{M}} = (\mu, \mathcal{C}_{\overline{\mathcal{M}}}, \mathcal{I} = ( M_i )_{i=1}^n, \sigma_{\mathcal{T}}), \mathbf{pp}, \mathbf{dpk}_P</math> and <math>\mathbf{dpk}_O</math> this algorithm verifies whether</p> $\text{DVerify}(\tau, id_{\mathcal{T}} \parallel \mathcal{C} \parallel n \parallel \mathbf{dpk}_P, \mathbf{dpk}_O) = 1 \quad \wedge \quad \text{DVerify}(\mu, \tau \parallel \mathcal{C}_{\overline{\mathcal{M}}} \parallel \mathcal{I}, \mathbf{dpk}_P) = 1 \quad \wedge$ $ \mathcal{I}  = n \quad \wedge \quad \sum_{i=1}^n  M_i  =  \mathcal{M} $ <p>On failure return 0, otherwise evaluate <math>m_{\mathcal{M}} = m_{\mathcal{T}}(\mathcal{M})</math> and check whether</p> $e(m_{\mathcal{M}}(\alpha)P, \mathcal{C}_{\overline{\mathcal{M}}}) = \mathcal{C}$ <p>On success return 1 and 0 otherwise.</p>
---

Scheme 4.1: (E)BDSS [40]

## 4.5 Tweaks and Optimizations

This section aims to provide an overview over the optimizations we made in order to improve the efficiency of the schemes. We show how to use Type-3 pairings instead of the inefficient Type-1 pairings as originally proposed. Since the degree of the encoding polynomials seems to be the crucial bottleneck when executing the protocol, we further propose a method for aggregating elements and thus reducing the degree of the encoding polynomials.

### 4.5.1 Using Type-3 Pairings

Since the EBDSS is designed for Type-1 pairings, some modifications are necessary in order to make it suitable for much more efficient Type-3 pairings  $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$  whereas  $P$  is a generator of  $\mathbb{G}_1$  and  $P'$  is a generator of  $\mathbb{G}_2$ . As already outlined by the authors of [40], the scheme can be used with Type-3 pairings by simply duplicating some of the points in the system-wide parameters, i.e., some points in  $\mathbb{G}_1$  also have to be mapped to points in  $\mathbb{G}_2$ . In the following sections we discuss the required modifications. For all these modifications it is crucial, that the counterpart  $Q'$  in  $\mathbb{G}_2$ , of a point  $Q$  in  $\mathbb{G}_1$ , contains the

same discrete logarithm as the point  $Q$ :

$$e(Q, P') = e(P, Q') \Leftrightarrow Q = aP \text{ and } Q' = aP' \text{ for } a \in \mathbb{Z}_p.$$

### Modified (Extended) Blank Digital Signature Scheme

Currently, the system-wide parameters  $\text{pp}$  of the EBDSS contain a set of multiples of a point

$$\mathcal{P} = (P, \alpha P, \dots, \alpha^t P).$$

For Type-3 pairings, the set has to be extended with the same multiples of a point  $P' \in \mathbb{G}_2$

$$\mathcal{P}' = (P, \alpha P, \dots, \alpha^t P, P', \alpha P', \dots, \alpha^t P').$$

Given that, one has to choose the appropriate representative of the required point in all the computation steps, i.e., the representative in  $\mathbb{G}_1$  or  $\mathbb{G}_2$ .

Since the changes in the instantiation phase (Inst) are a bit more extensive, we take a closer look at them in this paragraph. The computation of the pairing

$$e(m_{\mathcal{M}}(\alpha)P, \mathcal{C}_{\overline{\mathcal{M}}})$$

requires the commitment to the complementary message encoding polynomial  $\mathcal{C}_{\overline{\mathcal{M}}}$  to be in  $\mathbb{G}_2$ . Thus, the computation of  $\mathcal{C}_{\overline{\mathcal{M}}}$  in the instantiation step changes as follows

$$\mathcal{C}_{\overline{\mathcal{M}}} = \rho \cdot m_{\mathcal{M}}(\alpha)P',$$

i.e., one needs to use the coefficients  $\alpha^i P'$  out of the public system parameters  $\text{pp}$ . The value  $\rho \in \mathbb{Z}_p^*$  is a blinding factor which is required for the modified Pedersen polynomial commitment.

Using the proposed modifications, only requires the computation of  $\mathcal{C}_{\overline{\mathcal{M}}}$  to be in  $\mathbb{G}_2$ , which is carried out by the proxy. All other computations can be performed in  $\mathbb{G}_1$  which is ideal w.r.t. efficiency, i.e., the verification only requires cheaper computations in  $\mathbb{G}_1$ .

### Modified Chameleon Hashing Scheme

The EBDSS utilizes the basic chameleon hashing scheme proposed in [21] to facilitate blank elements. This chameleon hashing scheme, among others, requires to check if a given tuple is a valid DH tuple, i.e., given  $(P, aP, bP, cP)$  to check whether  $c \equiv ab \pmod{p}$  holds. Since the authors of [40] use a pairing friendly elliptic curve group, which is a GDH group, this check is performed in the following way:

$$e(aP, bP) \stackrel{?}{=} e(cP, P).$$

Using Type-3 pairings necessitates to modify the verification of the chameleon hashing scheme, i.e., to additionally store  $aP'$ , which in turn changes the DH tuple check as follows:

$$e(aP, P') \stackrel{?}{=} e(P, aP') \quad \wedge \quad e(aP, bP') \stackrel{?}{=} e(cP, P').$$

### 4.5.2 Aggregating Elements

Arising from the fact that the degree of the template-/message-encoding polynomial has an essential influence on the performance of the scheme, we propose two crucial modifications for reducing the degree. Basically, the reduction can be achieved by using an alternative template-/message-encoding, i.e., by aggregating the fixed elements and the blank elements. The aggregation of the exchangeable elements cannot be done, since their separate encoding is inevitable for the EBDSS to work. Note that this modified encoding is still unique and preserves the security of the scheme as discussed in Appendix C.1.

#### Aggregating Fixed Elements.

The fixed elements can simply be aggregated by concatenating the identifier of the template, the messages and the positions of the messages in the template as follows:

$$\begin{aligned} m_i &= M_i||i, \quad M = m_1||m_2||\dots||m_u \\ m_{\text{fixed}}(X) &= X - H(id_{\mathcal{T}}||M) \end{aligned} \quad (4.1)$$

This encoding binds the messages and the positions of the messages in a similar way as in the original protocol [40], and, therefore, further explanations are omitted.

#### Aggregating Blank Elements.

In order to aggregate the blank elements, the homomorphic property of the used chameleon hashing scheme is exploited:

$$\begin{aligned} \text{CHash}(\text{cpk}, m, r) + \text{CHash}(\text{cpk}, m', r') &= (m + m')P + (a + a')\text{cpk} = \\ \text{CHash}(\text{cpk}, m + m', r^*) &\quad \text{with } r^* = ((a + a')P, (a + a')\text{cpk}). \end{aligned}$$

However, the aggregated evaluation of the chameleon hashes requires slightly more changes than the aggregation of the fixed elements. Firstly, the template-encoding function for blank elements has to be redefined (see (4.2)). Since the indices of the blank elements are implicitly fixed by the indices encoded within the coefficients belonging to the fixed elements and the exchangeable elements, an explicit encoding of the indices for the blank elements can be omitted in this case. This originates from the fact that the blank elements are empty in the template-signing phase, i.e., they can be arbitrarily exchanged amongst each other without changing anything. Assuming that we have  $v > 2$  blank elements in the template, we obtain:

$$t_{\text{blank}}(X) = X - H(l_1||l_2||\dots||l_v||id_{\mathcal{T}}||\mathcal{H}(\text{cpk}, v \cdot H(\epsilon), r)). \quad (4.2)$$

Secondly, a new message-encoding function has to be defined (see (4.3)). Now, since we have a concrete instantiation, the bitstrings contained in the blank elements are fixed and, therefore, an inclusion of the indices is essential.

$$\begin{aligned} m_i &= H(M_i||i) \\ m_{\text{blank}}(X) &= X - H(l_1||l_2||\dots||l_v||id_{\mathcal{T}}||\mathcal{H}(\text{cpk}, m_1 + m_2 + \dots + m_v, r')). \end{aligned} \quad (4.3)$$

Both, the aggregation of the fixed elements and the aggregation of the blank elements results in a monomial each, leading to a huge speedup regarding computation times.

## Chapter 5

# Warrant-Hiding Proxy Signatures

Usually proxy signature schemes use an originator-signed warrant for describing the messages the proxy is allowed to sign on behalf of the originator. However, in a naive approach, where the warrant contains all possible messages the proxy is allowed to sign, all these messages are revealed to a verifier upon verification. In contrast, the Warrant-Hiding Proxy Signature scheme (WHPSS) [37] allows to hide the message space from the verifier by using schemes which do not reveal any information about the warrant, whilst it is still possible to check if a signed message is in the warrant. This is realized by using unconditionally hiding commitments to the message space, which allow to check set membership, i.e., polynomial or vector commitments.

In this chapter we discuss the WHPSS as well as our optimizations regarding certain computation steps. Similar to the previous chapter, we start with the introduction of the basic notion and the setup. Later on, we are going to introduce the encoding as well as the scheme itself. Finally, our performance optimizations are presented.

### 5.1 Basic Notion

In a nutshell, the WHPSS allows an *originator* to delegate the signing rights for a message out of a well defined message space  $\mathcal{M}$  to a proxy. The message space  $\mathcal{M}$  is, thereby, a non-empty set of bitstrings (messages)  $M_i$ , i.e.,  $\mathcal{M} = \{M_1, \dots, M_n\}$ . Once the delegation is computed, the *proxy* is able to choose one bitstring  $M_i$  out of the message space  $\mathcal{M}$  and issue a so-called proxy signature on behalf of the originator for it. Similar to the EBDSS, the WHPSS builds up on conventional digital signatures for certifying the delegation. For committing to the message space and the messages the WHPSS defines two methods. The first method (WHPSS<sub>PolyCommit</sub>) makes use of polynomial commitments (see Section 3.3.3), whereas the second method (WHPSS<sub>VectorCommit</sub>) is built upon randomized Merkle hash trees (see Section 3.3.4). Both commitments ensure that the unused messages in the message space are not revealed when verifying the instance signature (*privacy property*).

Figure 5.1 gives an abstract overview over a WHPSS protocol execution.

### 5.2 Setup

Since the WHPSS defines two somewhat different approaches for committing to the message space and the chosen messages respectively, also the setup phases for these two approaches

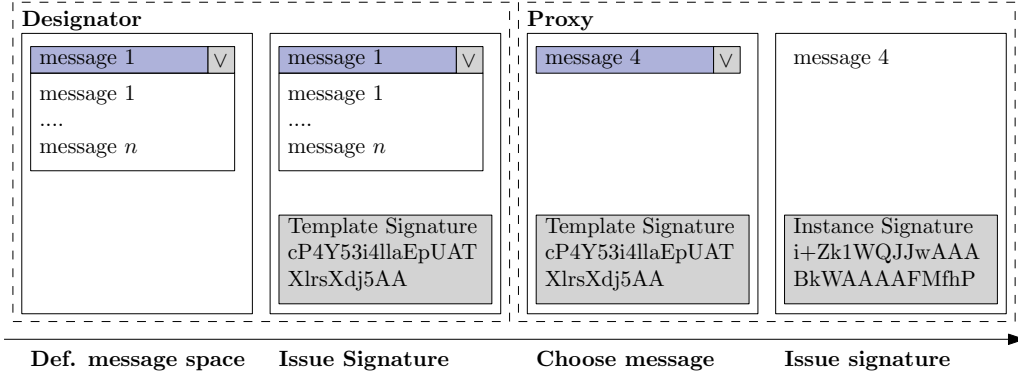


Figure 5.1: Schematic view of the WHPSS

differ. The following description gives an overview over the requirements for both variations of the WHPSS.

**PolyCommit:** The polynomial commitment version requires a pairing friendly elliptic curve group  $\mathbb{G}$  with subgroup of large prime order  $p$  and generator  $P$  to be used. Furthermore, the following setting is assumed to be in place

- A bilinear pairing  $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ ,
- a digital signature scheme  $DSS = (DKeyGen, DSign, DVerify)$ , and
- a full-domain cryptographic hash function  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ .

**VectorCommit:** Similar to the PolyCommit setup one requires a digital signature scheme  $DSS = (DKeyGen, DSign, DVerify)$ . The vector commitments used in the WHPSS are constructed from randomized Merkle hash trees (see Section 3.3.4), and, thus, a full-domain cryptographic hash function  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$  is required. Furthermore, one needs to complete the setup phase of the used (unconditionally hiding) commitment for the leafs of the tree.

Based on this setup, the inputs for the signing algorithms, i.e., the message space and the messages need to be appropriately encoded. The following section gives a brief overview over how the encoding works.

### 5.3 Message and Message Space Encoding

In contrast to the EBDSS, the encoding is quite simple in case of WHPSS. The message space  $\mathcal{M}$  as well as the messages  $M_i$  can be directly used in the vector commitment case and no additional encoding is needed. Conversely, using polynomial commitments requires a polynomial encoding of the message space  $\mathcal{M} = \{M_1, \dots, M_n\}$ . Let  $\mathbb{M}$  be the set of all possible messages and  $\phi : \mathbb{M} \rightarrow \mathbb{Z}_p[X]$ . The polynomial encoding of the message space  $m(X)$  is computed by evaluating the encoding function  $\phi$  at  $\mathcal{M}$ , which is defined as follows:

$$\mathcal{M} \mapsto \prod_{i=0}^n (X - H(M_i)).$$

Based on this encoding, the message space size  $|\mathcal{M}|$  is the size of the set  $\mathcal{M}$ . Since this scheme requires the proxy to choose a single element  $M_i$  out of the set  $\mathcal{M}$ , no separate encoding for  $M_i$  is needed and  $M_i$  can be used as-is in the protocol.

## 5.4 Schemes

In this section, we introduce  $\text{WHPSS}_{\text{PolyCommit}}$  and  $\text{WHPSS}_{\text{VectorCommit}}$  respectively. Basically the authors of [37] define four algorithms  $((D, P), PS, PV, ID)$ , based on [12] and shown in the following section.

### 5.4.1 Constructions

**Setup:** On input of a security parameter  $\kappa$  and an upper bound for the size of the message space  $t$ , this algorithm generates the public system parameters  $\text{ppk}$ . Note that  $\text{ppk}$  is assumed to be an input to all subsequent algorithms.

**(D, P):** In this phase, the originator and the proxy jointly compute a delegation as well as a proxy signing key  $\text{skp}$ . The originator, thereby, computes the delegation and outputs the delegation certificate issued using its DSS secret key  $\text{sk}_i$ , whereas the proxy verifies the delegation and composes the proxy signing key  $\text{skp}$  by joining the proxy's DSS secret key  $\text{sk}_j$  with the originators delegation.

**PS:** Here, the proxy chooses a message, and computes a so-called instance (proxy) signature  $\sigma_P$  on this message w.r.t. the proxy signing key  $\text{skp}$ .

**PV:** In this step the instance signature  $\sigma_P$  is verified using the public keys of proxy  $\text{pk}_j$  and originator  $\text{pk}_i$  as well as the signed message  $M$ . On success, this algorithm outputs 1, and 0 otherwise.

**ID:** This algorithm outputs the identity of the proxy, when given the a proxy signature  $\sigma_P$ .

Building up on this abstract description, the authors of [37] define two constructions, namely a construction using polynomial commitments and a construction using vector commitments. Scheme 5.1 presents the scheme making use of polynomial commitments, i.e.,  $\text{WHPSS}_{\text{PolyCommit}}$ . Note that the authors of [37] use a slightly different notation for the used schemes. Thus, we assume the following renaming scheme of the previously presented polynomial Pedersen commitment algorithms (see Section 3.3.3).

Commit  $\rightarrow$  PCommit,    CreateWitness  $\rightarrow$  PCreateWit,    VerifyEval  $\rightarrow$  PVerifyWit

Furthermore, the scheme building up on vector commitments ( $\text{WHPSS}_{\text{VectorCommit}}$ ) is presented in Scheme 5.2.

### 5.4.2 Security

Informally, the security of a WHPSS is defined as follows.

**Definition 5.1** (Security of a WHPSS [37]). *A secure WHPSS fulfills the following properties:*

**Correctness:** *For all  $\text{ppk} \in \text{Setup}(\kappa, t)$  and for all outputs of  $\text{skp} = (D, P)$  using the previously generated key  $\text{ppk}$ , it holds that for all warrants and proxy signatures for a message  $m$  the algorithm PV accepts a message if  $m$  is in the warrant and rejects it otherwise. Furthermore, ID is required to return the correct proxy.*

**Unforgeability:** *Without the knowledge of  $\text{sk}_i$  and  $\text{skp}$  it is intractable to forge delegations and proxy signatures which are either inside or outside the warrant.*

<p>(D, P): D and P are given local inputs <math>\mathbf{pk}_i, \mathbf{sk}_i, j, \mathbf{pk}_j, \omega</math> and <math>\mathbf{pk}_j, \mathbf{sk}_j, \mathbf{pk}_i</math>, where <math>\omega = (M_i)_{i=1}^c</math> with <math>c \leq t</math>.  D picks a seed <math>s \in_R \mathbb{Z}_p</math>, computes <math>\omega_H = \{H(M_i) : i = 1, \dots, c\}</math> and <math>m(X) = \phi(\omega_H)</math>. Then, compute <math>r(X) \in \mathbb{Z}_p[X]</math> with <math>\deg(r) = \deg(m) = c</math> with coefficients obtained evaluating <math>f(s)</math> as well as</p> $\mathcal{C} = \text{PCommit}(\text{ppk}, m(X), r(X)) \quad \text{and} \quad \text{cert} = \text{DSign}(\mathcal{C} \  j \  \mathbf{pk}_j, \mathbf{sk}_i).$ <p>It sets the proxy signing key of user <math>j</math> as <math>\text{skp}' = (\mathbf{pk}_i, s, \mathcal{C}, j, \mathbf{pk}_j, \omega, \text{cert})</math> and sends it to P. Now, P computes <math>\omega_H</math> and <math>m(X) = \phi(\omega_H)</math> as well as <math>r(X)</math> from seed <math>s</math>. It checks whether</p> $\text{DVerify}(\text{cert}, \text{PCommit}(\text{ppk}, m(X), r(X)) \  j \  \mathbf{pk}_j, \mathbf{pk}_i) = 1$ <p>If not, return 0 and terminate. Otherwise, set <math>\text{skp} = (\mathbf{sk}_j, \text{skp}')</math>, output <math>\text{skp}</math> and terminate. If P returns 0, D aborts. Otherwise, also D terminates correctly.</p> <p>PS: Given <math>\text{skp}, M</math> so that there is an index <math>l</math> with <math>1 \leq l \leq c</math> and <math>M_l = M</math>, this algorithm computes <math>\omega_H</math> and <math>m(X) = \phi(\omega_H)</math> as well as <math>r(X)</math> from seed <math>s</math>. Then, it computes <math>h_M = H(M)</math>, <math>r_M = r(h_M)</math> as well as</p> $W_M = \text{PCreateWit}(\text{ppk}, m(X), r(X), h_M) \quad \text{and} \quad \sigma = \text{DSign}(W_M \  r_M \  \mathbf{pk}_i, \mathbf{sk}_j),$ <p>and returns <math>\sigma_p = (j, \mathcal{C}, W_M, r_M, \mathbf{pk}_j, \text{cert}, \sigma)</math>.</p> <p>PV: Given <math>\mathbf{pk}_i, M, \sigma_p = (j, \mathcal{C}, W_M, r_M, \mathbf{pk}_j, \text{cert}, \sigma)</math>, this algorithm verifies whether</p> $\text{DVerify}(\text{cert}, \mathcal{C} \  j \  \mathbf{pk}_j, \mathbf{pk}_i) \wedge \text{DVerify}(\sigma, W_M \  r_M \  \mathbf{pk}_i, \mathbf{pk}_j) \wedge \text{PVerifyWit}(\text{ppk}, \mathcal{C}, H(M), 0, r_M, W_M)$ <p>yields 1. On success return 1 and 0 otherwise.</p> <p>ID: Given <math>\sigma_p = (j, \mathcal{C}, W_M, r_M, \mathbf{pk}_j, \text{cert}, \sigma)</math> output <math>j</math>.</p>
--

Scheme 5.1:  $\text{WHPSS}_{\text{PolyCommit}}$  [37]

**Privacy:** *One can not efficiently decide whether a given message (except the ones being revealed by proxy signatures) lies within or outside the message space of the warrant without the knowledge of  $\text{skp}$ .*

**Theorem 5.1** (Security of Scheme 5.1 [37]). *Assuming that the  $t$ -SDH assumption in  $\mathbb{G}$  holds, the existence of secure hash functions and secure digital signature schemes, Scheme 5.1 is secure and correct w.r.t. Definition 5.1.*

**Theorem 5.2** (Security of Scheme 5.2 [37]). *Assuming the security of the used vector commitment scheme, the existence of secure hash functions and secure digital signature schemes, Scheme 5.2 is secure and correct w.r.t. Definition 5.1.*

For proofs of the theorems above, we refer the reader to [37].

## 5.5 Tweaks and Optimizations

This section is dedicated to give an overview over our optimizations regarding the WHPSS. In contrast to the  $\text{WHPSS}_{\text{PolyCommit}}$ , the  $\text{WHPSS}_{\text{VectorCommit}}$  does not leave any optimization potential, except for an optimal implementation of the vector commitment scheme and the digital signature scheme. Thus, this section mainly concentrates on  $\text{WHPSS}_{\text{PolyCommit}}$ . Firstly, we illustrate how to make use of much more efficient Type-3 pairings instead of the originally proposed Type-1 pairings. Secondly, we introduce some optimizations regarding the polynomial commitment scheme.



<p>(D, P): D and P are given local inputs <math>\text{pk}_i, \text{sk}_i, j, \text{pk}_j, \omega</math> and <math>\text{pk}_j, \text{sk}_j, \text{pk}_i</math>, where <math>\omega = (M_i)_{i=1}^c</math>. D picks a seed <math>s \in_R \{0, 1\}^\kappa</math>, chooses a secure PRG <math>f : \{0, 1\}^\kappa \rightarrow (\{0, 1\}^\kappa)^c</math> and computes <math>\mathbb{R} = f(s)</math> and</p> $\mathcal{C} = \text{VCommit}(\omega, \mathbb{R}) \quad \text{and} \quad \text{cert} = \text{DSign}(\mathcal{C} \  j \  \text{pk}_j, \text{sk}_i).$ <p>It sets the proxy signing key of user <math>j</math> as <math>\text{skp}' = (\text{pk}_i, s, \mathcal{C}, j, \text{pk}_j, \omega, \text{cert})</math> and sends it to P. Now, P computes <math>\mathbb{R}</math> from <math>s</math> and checks whether</p> $\text{DVerify}(\text{cert}, \text{VCommit}(\omega, \mathbb{R}) \  j \  \text{pk}_j, \text{pk}_i) = 1$ <p>If not, return <math>\perp</math> and terminate. Otherwise, set <math>\text{skp} = (\text{sk}_j, \text{skp}')</math>, output <math>\text{skp}</math> and terminate. If P returns <math>\perp</math>, D aborts. Otherwise, also D terminates correctly.</p> <p>PS: Given <math>\text{skp}, M</math> so that there is an index <math>l</math> with <math>1 \leq l \leq c</math> and <math>M_l = M</math>, this algorithm computes <math>\mathbb{R}</math> from seed <math>s</math>, sets <math>r_M = (r_l, l)</math> and computes</p> $W_M = \text{VOpen}(l, \omega, \mathbb{R}) \quad \text{and} \quad \sigma = \text{DSign}(W_M \  r_M \  \text{pk}_i, \text{sk}_j),$ <p>and returns <math>\sigma_p = (j, \mathcal{C}, W_M, r_M, \text{pk}_j, \text{cert}, \sigma)</math>.</p> <p>PV: Given <math>\text{pk}_i, M, \sigma_p = (j, \mathcal{C}, W_M, r_M, \text{pk}_j, \text{cert}, \sigma)</math> with <math>r_M = (r_l, l)</math>, this algorithm verifies whether</p> $\text{DVerify}(\text{cert}, \mathcal{C} \  j \  \text{pk}_j, \text{pk}_i) \wedge \text{DVerify}(\sigma, W_M \  r_M \  \text{pk}_i, \text{pk}_j) \wedge \text{VVerify}(\mathcal{C}, l, M, r_l, W_M)$ <p>yields 1. On success return 1 and 0 otherwise.</p> <p>ID: Given <math>\sigma_p = (j, \mathcal{C}, W_M, r_M, \text{pk}_j, \text{cert}, \sigma)</math> output <math>j</math>.</p>
---

Scheme 5.2:  $\text{WHPSS}_{\text{VectorCommit}}$  [37]

### 5.5.1 Using Type-3 Pairings

Similar to the EBDSS, the  $\text{WHPSS}_{\text{PolyCommit}}$  can be ported to use Type-3 pairings by duplicating some points in the public parameters, i.e., adding a  $\mathbb{G}_2$  representative of certain  $\mathbb{G}_1$  points in the public system parameters. Currently, the public system parameters contain the tuples  $\mathcal{P}$  and  $\mathcal{Q}$

$$\mathcal{P} = (P, \alpha P, \dots, \alpha^t P)$$

$$\mathcal{Q} = (Q, \alpha Q, \dots, \alpha^t Q)$$

with P and Q being generators of  $\mathbb{G}_1$ . Using Type-3 pairings requires to extend the system parameters with a tuple

$$\mathcal{P}' = (P', \alpha P')$$

with  $P'$  being a generator of  $\mathbb{G}_2$ , which enables to compute the Type-3 pairings in the  $\text{PVerifyWit}$  step as follows:

$$e(\mathcal{C}, P') \stackrel{?}{=} e(w_i, \alpha P' - iP') \cdot e(\phi(i)P + \hat{\phi}(i)Q, P').$$

### 5.5.2 Hashing the Commitment

Moreover, it is possible to further speed up the proxy verification step PV by applying a slight modification of the delegation step (D, P) and the proxy verification step PV. In addition to the setting mentioned in Section 5.2, a cryptographic hash function  $H' : \mathbb{F}_{q^k}^* [p] \rightarrow \mathbb{Z}_p$  is required for this modification. Currently the commitment is verified using the following comparison:

$$e(\mathcal{C}, P) \stackrel{?}{=} e(w_i, \alpha P - iP) \cdot e(\phi(i)P + \hat{\phi}(i)Q, P).$$

Since the computation of a pairing is quite expensive, we propose to move one of the three pairing computations to the delegation step (D, P) which is less time critical since it is only performed once per delegation. This is achieved over the following modifications. Instead of outputting  $\mathcal{C}$  in the delegation step, we output

$$\mathcal{C}' = H'(e(\mathcal{C}, P))$$

which, in turn, changes the check in the proxy verification step to

$$\mathcal{C}' \stackrel{?}{=} H' \left( e(w_i, \alpha P - iP) \cdot e(\phi(i)P + \hat{\phi}(i)Q, P) \right).$$

## Chapter 6

# Delegation of Signing Rights to Multiple Proxies

As already outlined by the authors of [40], it is possible to delegate the signing rights to multiple proxies in the delegation-by-certificate approach. In order to achieve this, an arbitrary description of a group of authorized proxies  $\Omega$  is included in the delegation certificate instead of the public key of one proxy. This way, any verifier can check the identity of the proxy which has issued a given instance signature against the  $\Omega$  parameter contained in the delegation certificate. Whilst both, the BDSS and the WHPSS can easily make use of this approach our explanations are based on the WHPSS for the sake of simplicity. However, including such a description can have some impact on the privacy of the proxies and on the privacy property of the used scheme, respectively.

This chapter firstly introduces a concept for the confidential distribution of the proxy signing keys in order to sustain the privacy property of the schemes and then discusses some ways of hiding the proxies using an appropriate commitment scheme in order to prevent the group of authorized proxies from being revealed to a verifier. For completeness reasons, we conclude this chapter with the concept of including an abstract description of the delegation in the  $\Omega$  parameter as shown in [40].

### 6.1 Hybrid Encryption of the Proxy Signing Keys

The schemes discussed in Chapter 4 and Chapter 5 respectively, both keep the message space private when verifying a signature of a proxy (*privacy property*). When this property is required together with the delegation of the signing rights to multiple proxies, certain use cases may require a hybrid encryption of the proxy signing keys and the message space, the delegation is based on. Let us consider a use case where a delegation is computed and then distributed over a public webserver. Clearly, the  $\Omega$  parameter prevents unauthorized proxies from computing a valid instance signature in this case, but, nevertheless, the privacy property gets lost due to the public availability of the message space. This, however, can be circumvented using a hybrid encryption scheme as shown in Figure 6.1. For instance one could make use of the PKCS#7 enveloped data [49] to prevent the leakage of the whole message space.

Though, the identity of the proxies is revealed once an instance signature is issued, since the proxies are contained in the  $\Omega$  parameter of the signature. Consequently, the following section shows two methods for circumventing this privacy issue.

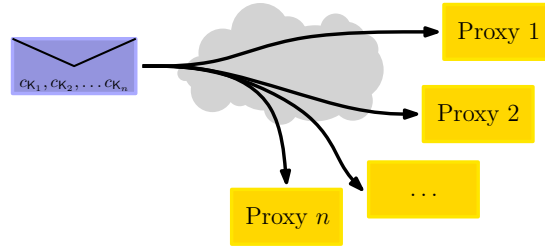


Figure 6.1: Hybrid encryption of the delegation parameters

## 6.2 Proxy Hiding

Basically, a selective hiding of the proxies contained in the  $\Omega$  parameter can be achieved by using any (unconditional hiding) commitment scheme which allows for selectively opening the commitment for subsets of the data. For our explanations, we concentrate on randomized Merkle hash trees (Section 6.2.1) and polynomial Pedersen commitments (Section 6.2.2), respectively. In the following sections, we discuss the schemes used for committing to the set of proxies. Since the schemes discussed in the following sections can be seen as plug-ins of the BDSS and the WHPSS, the computation times presented in Section 8.2.2 can simply be added to the BDSS/WHPSS computation times presented in Section 8.2.1 in order to get an insight into the overall execution times with a number of hidden proxies greater than one.

When using such a proxy hiding scheme as plug-in of a proxy-type signature scheme, one needs to perform the **Commit** step of the proxy hiding scheme in the delegation phase executed by the originator, whereas the **Open** step of the proxy hiding scheme needs to be jointly computed by the originator and the proxy before the instantiation. The output of the **Open** algorithm together with the additional verification information is then required to be included in the instance (proxy) signature. Finally, the verification of the proxy hiding commitment needs to be performed in the verification step of the proxy-type signature scheme.

### 6.2.1 Randomized Merkle Hash Trees

As already outlined in [37], a vector commitment scheme can be built from randomized Merkle trees. Since the scheme was not explicitly stated before, we restate a version adopted for proxy hiding purposes, in Scheme 6.1.

<p><b>VKeyGen:</b> Generates the required parameters for the used leaf commitment on input of a security parameter <math>\kappa</math>.</p> <p><b>VCommit:</b> On input of a sequence <math>\mathcal{W} = (w_1, \dots, w_n)</math> of appropriately encoded proxy identifiers <math>w_i</math> and a sequence of randomizers <math>\mathcal{R} = (r_1, \dots, r_n)</math> this algorithm computes and returns the root hash of the randomized Merkle tree as <math>\Omega</math> parameter for the respective scheme.</p> <p><b>VOpen:</b> Given the sequence <math>\mathcal{W}</math> as well as the sequence <math>\mathcal{R}</math> and an index <math>i</math>, the authentication path <math>W_{w_i}</math> is computed.</p> <p><b>VVerify:</b> Given a commitment <math>\Omega</math> and an authentication path <math>W_{w_i}</math> together with an index <math>i</math> and the corresponding <math>w_i</math> and <math>r_i</math>, this algorithm recomputes the commitment and outputs 1 if <math>\Omega</math> equals the recomputed commitment and 0 otherwise.</p>
---

Scheme 6.1: Proxy hiding using randomized Merkle trees [37]

Note that the required parameters for the Merkle hash tree also need to be included in the system parameters when using this proxy hiding setup. Furthermore, the commitment needs to be included in the delegation certificate in order to have an authentic version of the commitment at hand at verification time.

### 6.2.2 Polynomial Commitments

Another possibility for proxy hiding is to utilize polynomial commitments in a similar way as they are utilized in the  $\text{WHPSS}_{\text{PolyCommit}}$ . Thereby, the message space  $\mathcal{W}$  is composed out of the appropriately encoded proxy identifiers  $w_i$ :

$$\mathcal{W} = (w_1, \dots, w_n).$$

The encoding polynomial  $m(X)$  for the message space  $\mathcal{W}$  is then computed using the encoding function  $\phi : \mathbb{W} \rightarrow \mathbb{Z}_p[X]$ , with  $\mathbb{W}$  being the set of all possible proxy combinations, and the map

$$\mathcal{W} \mapsto \prod_{i=0}^n (X - H(w_i)).$$

Once the encoding polynomial is computed, the polynomial Pedersen commitment scheme can be executed as shown in Scheme 6.2.

<p><b>PKeyGen</b>(<math>\kappa, t</math>): Two groups <math>\mathbb{G}, \mathbb{G}_T</math>, both of prime order <math>p</math>, providing <math>\kappa</math>-bit security are chosen. <math>\mathbb{G}</math> and <math>\mathbb{G}_T</math> are chosen in a way, that a bilinear pairing <math>e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T</math> exists. Furthermore <math>\mathbb{G}</math> is required to be a gap Diffie-Hellman group. With <math>P, Q \in_g \mathbb{G}</math> and an <math>\alpha \in_r \mathbb{Z}_p^*</math> (chosen by a TTP), the tuples <math>\mathcal{P} = (P, \alpha P, \dots, \alpha^t P)</math> and <math>\mathcal{Q} = (Q, \alpha Q, \dots, \alpha^t Q)</math> are computed and the public key <math>\text{pk}</math> is as follows:</p> $\text{pk} = (e, \mathbb{G}, \mathbb{G}_T, \mathcal{P}, \mathcal{Q}).$ <p><b>PCommit</b>(<math>\text{pk}, \phi(x)</math>): A random polynomial <math>\hat{\phi}(x) \in \mathbb{Z}_p[x]</math> of degree <math>t</math> is chosen and a commitment <math>\Omega = \phi(\alpha)P + \hat{\phi}(\alpha)Q</math> is computed. Since <math>\alpha</math> is unknown, <math>\omega</math> is computed as</p> $\Omega = \sum_{i=0}^{\deg(\phi)} \phi_i(\alpha^i P) + \sum_{i=0}^{\deg(\hat{\phi})} \hat{\phi}_i(\alpha^i Q)$ <p>with <math>\phi_i</math> and <math>\hat{\phi}_i</math> being the coefficients of <math>\phi</math> and <math>\hat{\phi}</math> respectively.</p> <p><b>POpen</b>(<math>\text{pk}, \phi(x), \hat{\phi}(x), i</math>): Computes <math>\psi_i(x), \hat{\psi}_i(x)</math> and <math>w_i</math> and outputs <math>(i, \phi(i), \hat{\phi}(i), w_i)</math>, i.e.,</p> $\text{with } \psi_i(x) = \frac{\phi(x) - \phi(i)}{x - i} \wedge \hat{\psi}_i(x) = \frac{\hat{\phi}(x) - \hat{\phi}(i)}{x - i} \text{ compute } w_i = \psi_i(\alpha)P + \hat{\psi}_i(\alpha)Q.$ <p><b>PVerifyEval</b>(<math>\text{pk}, \omega, \phi(i), \psi(i), w_i</math>): Outputs 1 if <math>\phi(i)</math> is the evaluation of the polynomial committed to by <math>\Omega</math> at <math>i</math>, 0 otherwise. The check is performed as follows.</p> $e(\Omega, P) \stackrel{?}{=} e(w_i, \alpha P - iP) \cdot e(\phi(i)P + \hat{\phi}(i)Q, P)$
---

Scheme 6.2:  $\text{PolyCommit}_{\text{Ped}}$  slightly modified for proxy hiding [51]

Again, the parameters, being required by the polynomial Pedersen commitments are required to be included in the public system parameters. The output  $\Omega$  of the commitment is again intended to be included in the delegation certificate, such that an authentic version of the commitment is available at verification time for comparison purposes.

The main difference between the two previously introduced proxy hiding schemes is that the size of the commitment grows logarithmically with the number of proxies when using randomized Merkle hash trees and is constant when using polynomial commitments.

Conversely, the computations performed when using polynomial commitments are more time consuming than the ones performed when using randomized Merkle hash trees. Thus, choosing one scheme is always a tradeoff between computation time and commitment size (see Section 8.2.2).

### 6.3 Abstract Description of the Delegation

For the sake of completeness another approach for a multi-proxy setup is discussed here. This approach, is the approach which was initially proposed in [40] and is intended for use cases which require a large number of proxies, e.g., when a governmental organization delegates the signing rights for a template to all citizens of a country. In such a use case the  $\Omega$  parameter contains an appropriate encoding of the abstract group description, i.e., an appropriate encoding of the string "All citizens of Country X". Using such an approach, however, requires special treatment when verifying the signatures, e.g., when considering the aforementioned example, one would have to check if the signer is indeed a citizen of the respective country.

**Part III**

**Implementation Aspects**

## Chapter 7

# Implementation in Java and Integration into JCA

In this chapter we provide an in-depth description of the implementation related aspects of the optimized BDSS as well as the optimized WHPSS. We show how the schemes are integrated into the Java Cryptography Architecture [71], the key material is integrated into X.509 and propose three signature formats, namely, an XML signature format for the BDSS and the WHPSS (Section 7.4.1, Section 7.4.2) and a PDF signature format for the BDSS (Section 7.4.1).

### 7.1 Background

This section is intended to give a basic overview over the infrastructure our implementations are integrated in. Moreover, we briefly introduce the BNpairings library, which is used for the pairing computations.

#### 7.1.1 Java Cryptography Architecture

The Java Cryptography Architecture (JCA) [71] constitutes an API, providing standardized access to cryptographic algorithms. Each library, implementing this API, needs to implement a so-called cryptographic provider, registering the provided algorithm implementations at the JCA. The desired provider is then set by the user of the library, and instances of the algorithm implementations can be obtained using the JCA-provided factories, e.g., `Signature` or `MessageDigest`. For a comprehensive list of the provided factories see [71]. This way, entire implementations can be exchanged by simply setting another provider. Furthermore, it is possible to register multiple providers, providing different algorithm implementations. Figure 7.1 illustrates the different ways of retrieving a `MessageDigest` implementation with multiple registered providers and gives a basic architectural overview over the JCA. The figure can be interpreted in the following way. When no provider for retrieving an algorithm is specified, the first provider implementing this algorithm is chosen. Otherwise, the JCA chooses the implementation from the specified provider.

Since the way of using the implementations, once they are retrieved from the JCA, is illustrated with concrete examples later in this chapter, further explanations are omitted here. For a more detailed overview over the JCA, we refer the reader to [71].



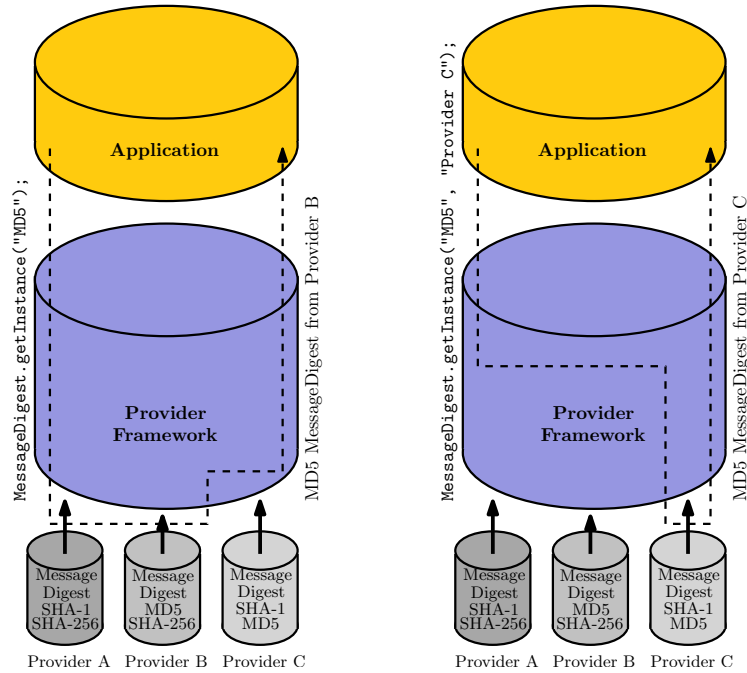


Figure 7.1: Basic JCA architecture [71]

### 7.1.2 BNpairings Library

The BNpairings [32] library by Geovandro and Barreto is a Java library, providing means for efficiently computing pairings on BN curves [8]. The library provides multiple pairing implementations such as the Tate [56, 79, 80] pairing, the Eta [7] pairing, the Ate [41] pairing as well as the optimal Ate [83] pairing.

Thus, this section is dedicated to justify our choice regarding the used pairing. Since our implementation aims at reaching minimal computation times, the obvious criteria for choosing a pairing is the computation time needed for a single pairing evaluation using a group size of 256bit. Table 7.1 gives an overview over the achieved computation times on a single core of a *Lenovo ThinkPad T420s* with an *Intel Core i5 2540M* with 2.6/3.3 GHz and 8 GB of RAM. On the software side, Java 1.7.0\_21 was used on top of Ubuntu 12.10/amd64. Note that each value is an average over 100 executions in order to eliminate unwanted influences such as the garbage collector.

Tate	Eta	Ate	Optimal Ate
33.56ms	25.30ms	17.34ms	11.13ms

Table 7.1: Pairing computation times using the BNpairings library

Based on the timings in Table 7.1 and Figure 7.2 respectively, our choice fell on the optimal Ate pairing.

## 7.2 Overview

Figure 7.3 and Figure 7.4 provide a fine-grained overview over the computations and the communication during a BDSS and a WHPPS protocol execution respectively. The

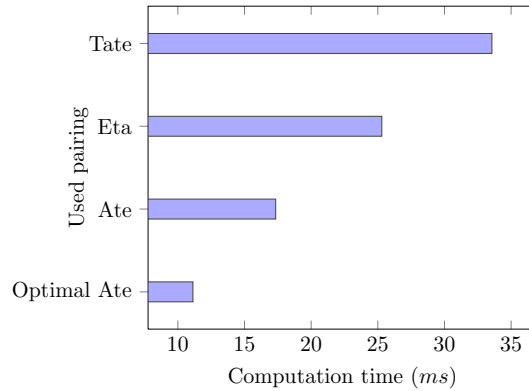


Figure 7.2: Computation times in relation to the used pairing

gray boxes in the figures logically group consecutive computation steps to units with defined input and output. For the sake of simplicity, we omitted the computation of the system parameters and the visualization of the distribution of the public keys by the TTP. However, we assume that the TTP provides means for retrieving the public keys in an authentic manner, e.g., encapsulated within a X.509 certificate (see Section 7.3). To provide maximum flexibility, the TTP implementation is left open to the user and it is only required to register an arbitrary implementation of the ITTP interface at the TTPAdapter contained in our library.

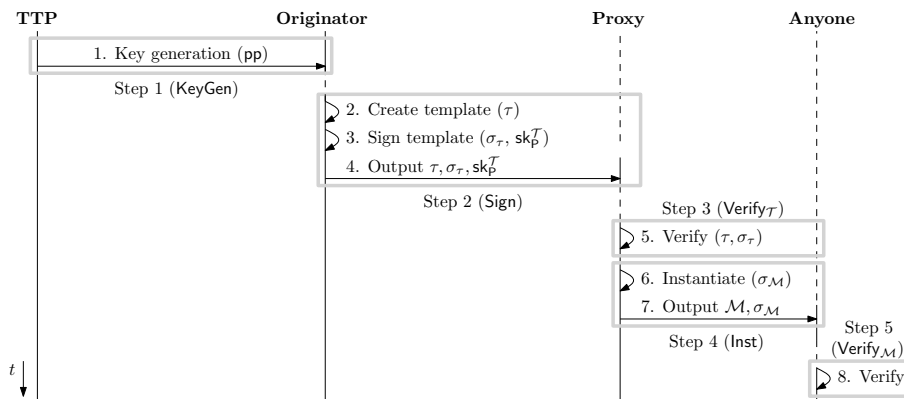


Figure 7.3: Computations and communication in the (E)BDSS

From a JCA point of view, the system parameter generation phase (KeyGen) is wrapped in a `KeyPairGeneratorSpi` implementation, being usable by any custom ITTP implementation to generate the BDSS parameters `pp` and the WHPSS parameters `ppk`.

The subsequent steps, i.e., Step 2-5 in Figure 7.3 and Figure 7.4, are packed into two `SignatureSpi` implementations each. The BDSS steps are included in the `BDSSTemplateSignature` (Step 2 and 3) and the `BDSSTemplateSignature` (Step 4 and 5), whereas the WHPSS steps are included in the `WHPSSInstanceSignature` (Step 2 and 3) and the `WHPSSInstanceSignature` (Step 4 and 5).

Moreover, both protocol implementations distinguish between a single- and a multi-proxy setup by encoding the authorized proxies within a parameter  $\Omega$ . In the single-proxy setup, the fingerprint of the provided X.509 proxy certificate is encoded within  $\Omega$ , whereas, in the multi-proxy setup this encoding is open to the user, e.g.,  $\Omega$  could contain a suitable

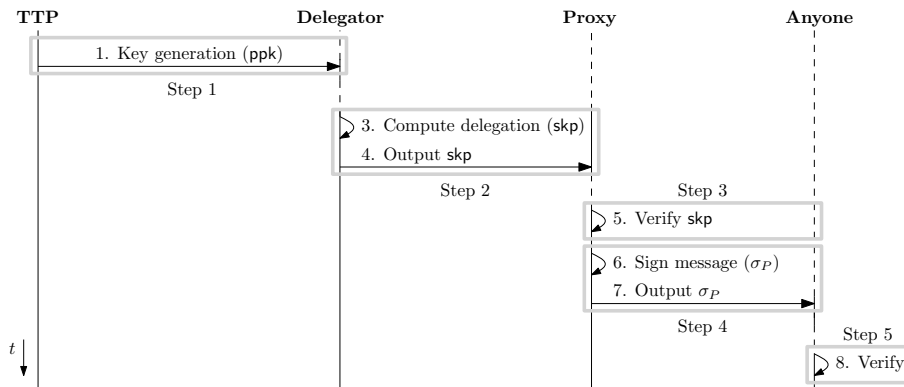


Figure 7.4: Computations and communication in the WHPSS

representation of the string "All citizens of Country X". Consequently, the multi-proxy setup requires the user of the library to verify whether a proxy is authorized to issue an instance signature, whilst this is done by the library in the single-proxy setup. The distinction between the single- and multi-proxy setup is done by providing different parameters to the JCA `Signature` instance (see Listing 7.1, Line 2-6 and 11-15).

Note that, if the privacy property of the schemes is required, a secure transmission of the outputs of Step 2 in Figure 7.3 and Figure 7.4 is inevitable. Consequently, our library provides means for ECIES (see Section 3.1.3) encryption and decryption.

In order to make the JCA `Signature` suitable for the aforementioned protocols, the `engineSetParameter` method is overridden. This way, it is possible to supply `AlgorithmParameterSpec` implementations containing the additional parameters, being necessary for a protocol execution. Listing 7.1 provides an example for obtaining an BDSS signature on a template. All other `SignatureSpi` implementations can be used in the same way.

```

1 Signature signature = Signature.getInstance("BDSSTemplateSignature");
2 SingleProxyOSPS osps = new SingleProxyOSPS(pkOrig, skOrig,
3     pkProxyDss, sysParams, pkProxyDss.getFingerprintSHA());
4 //MultipleProxyOSPS osps = new MultipleProxyOSPS(pkOrig,
5 //     skOrig, sysParams, omega);
6 signature.setParameter(osps);
7 signature.initSign(skOrigDss);
8 signature.update(template.toByteArray());
9 byte[] templateSignature = signature.sign();
10
11 SingleProxyPVPS pvps = new SingleProxyPVPS(pkOrig, pkOrigDss,
12     pkProxyDss, sysParams);
13 //MultipleProxyPVPS pvps = new MultipleProxyPVPS(pkOrig,
14 //     pkOrigDss, sysParams);
15 signature.setParameter(pvps);
16 signature.initVerify(pkOrigDss);
17 signature.update(template.toByteArray());
18 boolean success = signature.verify(templateSignature);

```

Listing 7.1: Java code for obtaining an BDSS template signature

### 7.3 Encoding and Key Representation in X.509

The need for storing and transferring the intermediate results, i.e., template-, instance- and proxy-signatures, necessitates a serialized representation of the aforementioned objects. Consequently, a compact encoding with minimal overhead is desired in order to keep the transmission times low and to enable the de-/encoding on constrained devices. Due to the overhead of ASN.1 [42], we propose a lightweight encoding which is similar to the BER/DER [43] encoding of ASN.1 and works in the following way. Each block starts with a magic number byte in order to uniquely identify the block contents. After the magic number an arbitrary number of data blocks, each prefixed with 4 length bytes, can follow. This format also allows for the encapsulation of blocks within a data block of another block, suiting our needs for signature representation. Furthermore, the encoding is unique, which is crucial for cryptographic applications.

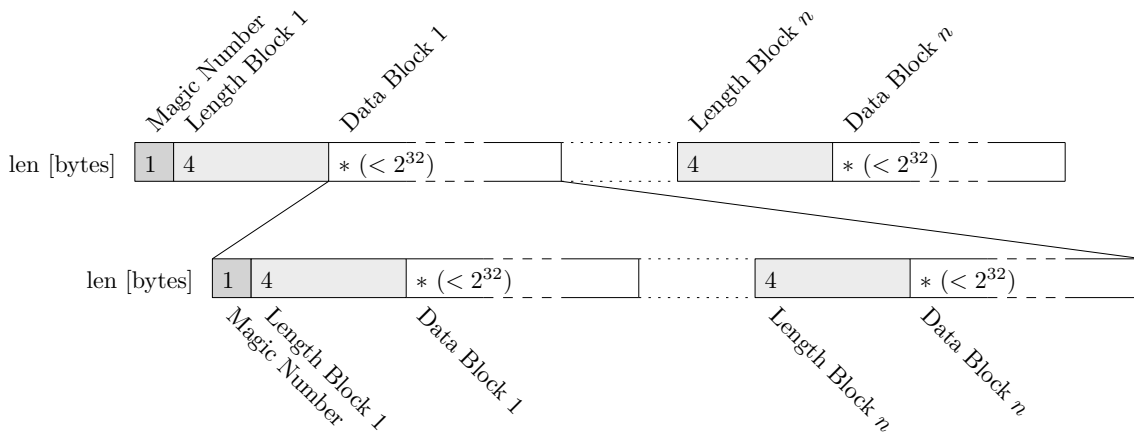


Figure 7.5: Signature encoding format

Based on this encoding format, means for integrating the keying material as public key info into X.509 certificates [22] are provided in our implementation. This way, it is possible to use X.509 provided methods for ensuring key authenticity and integrity. Furthermore, the revocation mechanisms of PKIX [22] can be put to use. For the sake of completeness, we included the object identifiers for the keys to be integrated into X.509 certificates in Appendix B.2.

In order to (re-)extract the serialized keys from the public key info, our Java cryptographic provider provides the appropriate `KeyFactorySpi` implementations, being capable of extracting the keys from a `X509EncodedKeySpec`. Since the design of our library aims at maximum flexibility, it is open to the user to encapsulate the keys within a X.509 certificate or to employ other mechanisms. Consequently, we do not limit the user to one signature format, enabling a broad spectrum of applications.

### 7.4 Proposed Signature Formats

In the following subsections, we introduce two example signature formats for the BDSS as well as one signature format for the WHPSS, which can be seen as basic formats, covering all protocol related issues. However, these formats are easily extendable for certain use cases requiring additional parameters.

### 7.4.1 BDSS Signature Formats

In the BDSS case, we define a XML and a PDF signature format. Whilst the former is suited for many different applications, the latter is intended to illustrate the applicability of the BDSS to a widespread application such as PDF forms.

#### Defining an XML Signature Format

In order to facilitate the usage of XML, we added Java Annotation for XML binding (JAXB) annotations, as defined in [28], to the classes serving as input-/output-containers. Using these annotations together with the appropriate XML schema allows for an easy processing using the JAXB marshalling and unmarshalling routines provided by the Java platform. In a nutshell, the BDSS needs an appropriate representation for templates and instances of those templates. Furthermore, appending a signature to a template and instantiations should be possible. Listing 7.2 and Listing 7.3 show the proposed signature format, with "?", "+" and "\*" denoting the multiplicity of the tags, i.e., "?" means at most once, "+" means at least once and "\*" means any number of times. Whilst the template schema allows for multiple `message` tags within one `templateentry` tag, the instance schema requires to choose one `message` for each `templateentry` and to encode it in a series of `message` tags. To allow for blank elements, the message tag contains a `type` attribute and a `length` attribute. Since the blank elements are left empty in the template, the `text` tag in the `message` tag can be omitted. However, the `text` tag is required in the instance. The signature, being attachable to both, the template and the instance, basically contains the encoded `signaturevalue` of the BDSS template-/instance-signature, the `keyid` identifying the originator-specific public key and the X.509 certificates of TTP, originator and proxy. Whilst the `signaturevalue` tag is required once a `signature` tag is included, the other fields are optional.

```

1 <template id="...">
2   (<templateentry>
3     (<message type="blank|exch|fix" length="[Integer]">
4       (<text>[String]</text>)?
5     </message>)*
6   </templateentry>)+
7   (<signature>
8     <signaturevalue>[Base64 encoded string]</signaturevalue>
9     (<keyId>[String]</keyId>)?
10    (<ttpcert>[Base64 encoded string]</ttpcert>)?
11    (<originatorcert>[Base64 encoded string]</originatorcert>)?
12    (<proxycert>[Base64 encoded string]</proxycert>)?
13  </signature>)?
14 </template>

```

Listing 7.2: BDSS template format

Since we propose XML as standard signature format, our library directly includes means for marshalling and unmarshalling objects.

```

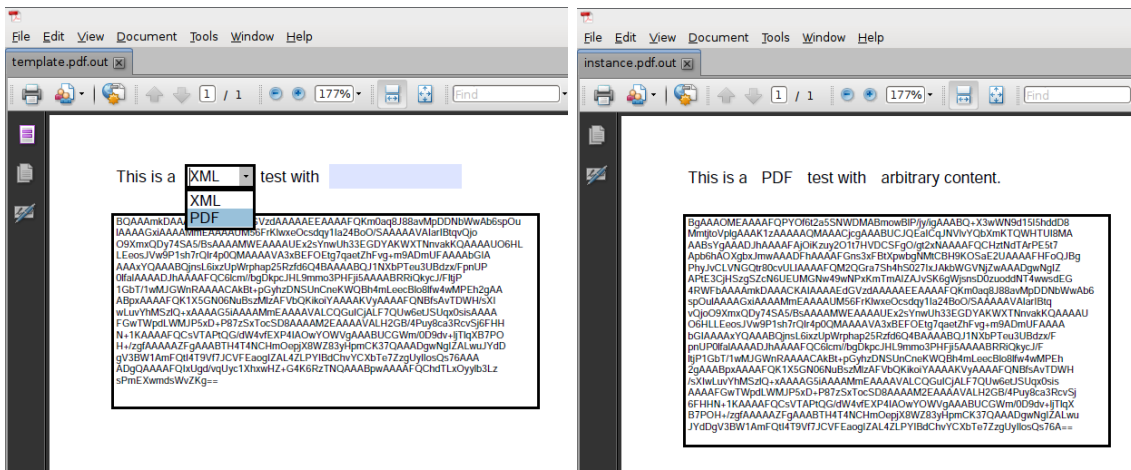
1 <instance id="...">
2   (<message type="blank|exch|fix" length="[Integer]">
3     <text>[String]</text>
4   </message>)+
5   (<signature>
6     <signaturevalue>[Base64 encoded string]</signaturevalue>
7     (<keyId>[String]</keyId>)?
8     (<ttpcert>[Base64 encoded string]</ttpcert>)?
9     (<originatorcert>[Base64 encoded string]</originatorcert>)?
10    (<proxycert>[Base64 encoded string]</proxycert>)?
11  </signature>)?
12 </instance>

```

Listing 7.3: BDSS instance format

### Moving to PDF as an Alternative Signature Format

Arising from the fact that signable PDF Forms seem to be an essential application of the BDSS, a proof-of-concept implementation utilizing PDF as signature format is introduced in this subsection. The PDF file manipulation is, thereby, done using the `iText`<sup>®</sup> Java library [44]. In order to model the element types being required by the BDSS, the `TextField`, providing the possibility for encoding text fields (fixed and variable) and multiple choice fields, is put to use. The variable fields can be exploited to encode the blank elements, since the `TextField` also allows for defining a maximum length of the variable text. Fixed fields and multiple choice fields can be used without any additional configuration. Figure 7.6 shows a sample template and a corresponding instance, both containing a signature.



(a) Signed template (b) Signed instance

Figure 7.6: BDSS PDF signature format

In addition to Adobe Reader provided means for signing PDF forms, using the proposed signature format facilitates to prove that an instance is a correct instantiation of a template, whilst the unused choices of the exchangeable elements still stay private (*privacy property*). In contrast, when using Adobe signed forms, one can either choose to ensure the privacy property or to show that an instance is a correct instantiation of a template. This owes

from the fact that ensuring the privacy property would require to keep the original form secret, whereas keeping the original form secret prevents from proving that an instance is a correct instantiation of a template.

As the purpose of this implementation is to serve as demonstration for the possibility to easily switch over to PDF as signature format, only the Base64 encoded BDSS signature value is attached to the signed document. Nevertheless, the inclusion of arbitrary attributes, such as the ones shown in the `signature` tag in Listing 7.2, can be easily performed.

#### 7.4.2 WHPSS XML Signature Format

Similar as in the proposed BDSS XML signature format, we also made use of JAXB [28] annotations for the WHPSS signature format, which allows for an easy processing in Java. In the WHPSS case, appropriate representations of the message space and the instances of this message space are needed. Again, a signature is required to be attachable. For our following explanations we use the notation already outlined in the previous section, that is, "?", "+" and "\*" denote the multiplicity of the tags, i.e., "?" means at most once, "+" means at least once and "\*" means any number of times.

Listing 7.4 shows the proposed message space format, which contains at least one `messageentry`. Furthermore, the same signature as in the BDSS format is attachable.

```

1 <message id="...">
2   (<messageentry>[String] </messageentry>)+
3   (<signature>
4     <signaturevalue>[Base64 encoded string] </signaturevalue>
5     (<keyId>[String] </keyId>)?
6     (<ttpcert>[Base64 encoded string] </ttpcert>)?
7     (<originatorcert>[Base64 encoded string] </originatorcert>)?
8     (<proxycert>[Base64 encoded string] </proxycert>)?
9   </signature>)?
10 </message>

```

Listing 7.4: WHPSS message space format

Computing an instance of such a message space requires an encoding as shown in Listing 7.5. This encoding requires to choose one `messageentry` out of the set of `messageentries` in the message space format and to encode it into a single `messageentry` tag. Moreover, means for encoding an optional signature of the aforementioned format are provided.

```

1 <messageInstance id="...">
2   <messageentry>[String] </messageentry>
3   (<signature>
4     <signaturevalue>[Base64 encoded string] </signaturevalue>
5     (<keyId>[String] </keyId>)?
6     (<ttpcert>[Base64 encoded string] </ttpcert>)?
7     (<originatorcert>[Base64 encoded string] </originatorcert>)?
8     (<proxycert>[Base64 encoded string] </proxycert>)?
9   </signature>)?
10 </messageInstance>

```

Listing 7.5: WHPSS message instance format

# Chapter 8

## Performance Evaluation

Based on the previously introduced efficient implementations of the optimized protocols, we give a comprehensive overview over the performance of the (E)BDSS and the WHPSS in this chapter. Furthermore, we discuss our performance evaluation results of the two proxy hiding approaches introduced in Chapter 6.

This chapter is organized as follows. Firstly, we introduce our test setup and the strategy which we used for our timings. Then, the results of the performance evaluation of both, the (E)BDSS and the WHPSS are presented and compared. Subsequently, the performance of the proxy hiding schemes, presented earlier in this thesis, are discussed.

### 8.1 Test Setup

For the timings, we utilize the BNPairings library [32] for computing the optimal Ate pairing [83] on BN curves [8] with 256 bit group size and an embedding degree of 12. In order to obtain the required digital signatures, we utilize ECDSA (see Section 3.1.5) with the NIST P-224 curve [31]. The timings were performed on a single core of a *Lenovo ThinkPad T420s* with an *Intel Core i5 2540M* with 2.6/3.3 GHz and 8 GB of RAM. On the software side, Java 1.7.0\_21 was used on top of Ubuntu 12.10/amd64.

**Timing Strategy:** Basically, our timing strategy is quite different for the (E)BDSS and the WHPSS. Whilst the timing strategy in the (E)BDSS comes down to measuring the execution time of the four protocol steps (Step 2-5 in Figure 7.3) for different template sizes and template compositions, the timing strategy in the WHPSS measures the execution time of both, the  $\text{WHPSS}_{\text{PolyCommit}}$  and the  $\text{WHPSS}_{\text{VectorCommit}}$ , for different message space sizes. Similarly, in the proxy hiding case we illustrate the influence of the number of proxies on the computation times. In order to isolate unwanted timing related influences, e.g., the garbage collector, each timing presented in this chapter represents the mean of 100 consecutive runs.

### 8.2 Results

In this section we present the performance evaluation results of the two proxy-type signature scheme implementations and the two proxy hiding approaches respectively.



### 8.2.1 Proxy-Type Signature Schemes

Table 8.1 shows the (E)BDSS computation times for template sizes ranging from 3 to 1000. For the purpose of illustrating the influence of the distribution of the element types on the timing, we provide timings for three different element type distributions, namely a distribution without blank elements, a uniform distribution of blank, exchangeable and fixed elements and a distribution without exchangeable elements. The used distribution is indicated by the percentage values in the top row of Table 8.1 ([exchangeable]%-[blank]%-[fixed]%). Figure 8.1 gives an overview over the computation time with increasing template size.

$ \mathcal{T} $	50%-0%-50%				33%-33%-34%				0%-50%-50%			
	Template		Instance		Template		Instance		Template		Instance	
	Sign	Verify	Inst	Verify	Sign	Verify	Inst	Verify	Sign	Verify	Inst	Verify
3	18	17	16	14	32	68	73	65	28	63	60	62
5	20	19	16	16	30	65	70	63	28	63	60	62
10	23	23	27	17	34	69	81	63	28	63	61	63
15	27	27	37	18	35	70	81	64	27	62	60	62
30	35	34	57	21	44	79	102	67	27	63	60	62
50	51	50	100	26	52	86	124	71	28	62	60	62
70	63	62	132	30	60	95	145	73	27	62	60	62
100	83	82	185	37	72	107	178	77	27	62	60	62
150	115	115	270	47	96	131	243	86	28	62	61	62
300	220	219	536	81	163	199	416	107	28	63	60	62
500	371	370	903	128	258	293	651	139	28	63	61	63
1000	770	765	1811	244	509	544	1254	214	28	64	61	64

Table 8.1: (E)BDSS timings for various template sizes in milliseconds

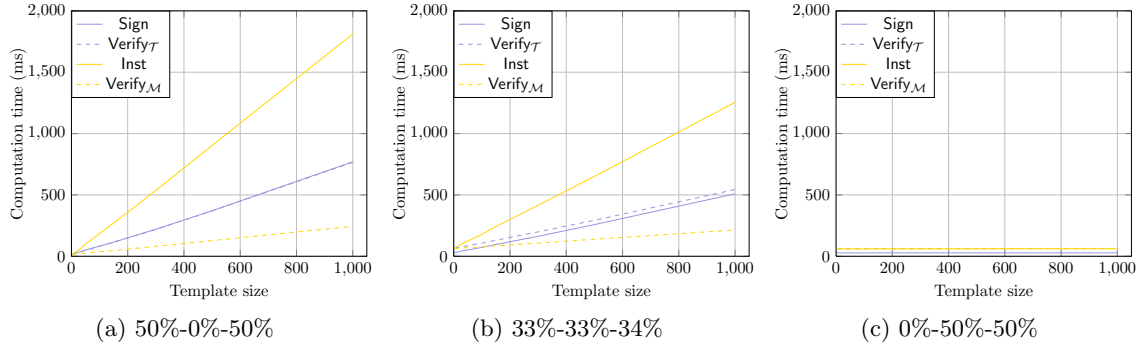


Figure 8.1: (E)BDSS computation times in relation to the template size, with the percentage values conforming to the number of the particular element types ([exchangeable]%-[blank]%-[fixed]%).

As expected, the computation times depend heavily on the degree of the template encoding polynomial. Consequently, using a template without blank elements leads to the largest computation times (Figure 8.2a). Increasing the amount of blank elements, whilst reducing the fixed elements (Figure 8.2b) reduces these computation times, as the degree of the polynomial reduces due to the aggregated evaluation of the chameleon hash for the blank elements and the hash for the fixed elements (see Section 4.5.2). Moving on to the usage of a template without exchangeable elements further reduces the polynomial degree to a constant degree of 2, as the blank elements and the fixed elements are aggregated within one monomial each. Consequently, this case leads to constant computation times of

below 65ms, being independent of the template-size, for each step (Figure 8.1c).

From a practical point of view, most forms are mainly composed out of blank and fixed elements, extended by a quite small amount of exchangeable elements. Therefore, we expect the practical execution time to be somewhere between the case shown in Figure 8.2b and the case with constant computation times ( $< 65\text{ms}$ ) for each step, which is perfectly acceptable for practical use. Furthermore, we assume practical template sizes to be below 100 elements, and, thus, every step can be performed within below 190ms for each algorithm.

Moreover, Table 8.2 shows the computation times for both WHPSS variants with message space sizes from 3 to 1000. Since the separate encoding of the elements, is crucial for the WHPSS to work, the computation times grow drastically with increasing message spaces. This arises from the high polynomial degree in the  $\text{WHPSS}_{\text{PolyCommit}}$  case and from the large number of leaf commitments in the  $\text{WHPSS}_{\text{VectorCommit}}$  case. Nevertheless, the

$ \mathcal{T} $	Polynomial Commitments				Vector Commitments			
	Message Space		Instance		Message Space		Instance	
	D	P	PS	PV	D	P	PS	PV
3	21	21	7	31	8	8	5	4
5	26	27	12	30	13	13	10	4
10	39	40	25	31	27	27	23	4
15	52	53	39	31	40	40	37	4
30	91	91	78	31	79	80	77	4
50	145	146	132	31	132	134	130	4
70	198	199	184	31	186	186	183	4
100	279	281	265	31	265	266	262	4
150	414	419	403	31	400	400	396	4
300	839	840	824	31	797	797	795	4
500	1423	1431	1411	31	1333	1331	1330	4
1000	3019	3035	3011	31	2663	2661	2660	4

Table 8.2: WHPSS timings for various template sizes in milliseconds

verification step of the instance signature PV runs in constant time in the polynomial commitment case and increases logarithmically with the message space size in the vector commitment case. Since, PV is the step which is performed most often, the scheme is still useable for practical applications which do not require a fast computation of the delegation and the instantiation. Furthermore, again a message space size of 100 and below is considered to be realistic for real world applications, which means that every protocol step can be performed within  $< 285\text{ms}$ , which seems to be acceptable for most practical use cases.

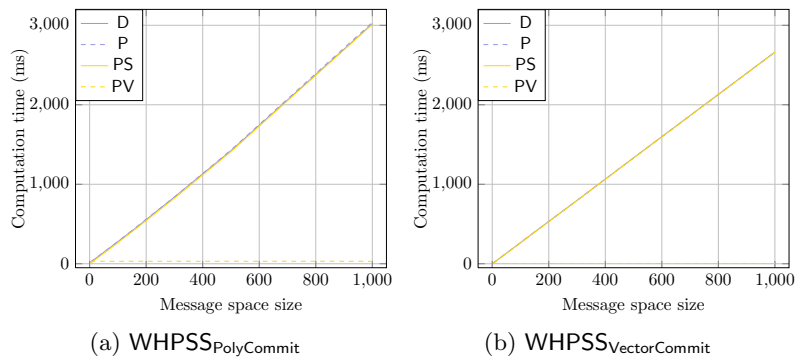


Figure 8.2: WHPSS computation times in relation to the message space size

When comparing this to the (E)BDSS, which could be used to achieve a similar behavior

by using a single exchangeable element field containing all messages out of the message space, the (E)BDSS performs better in most steps and, thus, it might be beneficial to use the (E)BDSS instead of the WHPSS. However, when a fast verification of the instance signature is required, whereas the runtime of the other algorithms does not matter at all, we recommend using the WHPSS.

### 8.2.2 Comparison of the two Proxy Hiding Approaches

This section aims at giving an overview over the performance of the proxy hiding feature described in Chapter 6. First of all, choosing one of the proxy hiding approaches is always a tradeoff between the size of the witness and the required computation times. Whilst the witness size in the vector commitment scheme grows logarithmically with the number of proxies, the size of the commitment in the polynomial commitment scheme is constant. Conversely, the computations performed in the vector commitment scheme are less time consuming than the ones performed in the polynomial commitment scheme.

$ \mathcal{W} $	Polynomial Commitments			Vector Commitments		
	PCommit	PCreateWit	PVerifyWit	PCommit	PCreateWit	PVerifyWit
3	20	6	29	7	5	2
5	25	11	29	13	10	2
10	38	25	29	26	23	2
15	52	39	29	39	36	2
30	92	80	29	78	76	2
50	148	132	29	131	128	2
70	201	187	29	183	182	2
100	281	267	29	260	260	2
150	419	405	29	399	398	2
300	841	823	29	798	802	2
500	1426	1413	29	1335	1334	2
1000	3023	3016	29	2685	2691	2

Table 8.3: Proxy hiding computation times for different numbers of proxies in milliseconds

Table 8.3 as well as Figure 8.3 illustrate the computation times for different numbers of proxies. As expected, the vector commitment version outperforms the polynomial commitment version for a larger number of proxies, i.e., the difference in the timings grows with increasing number of proxies. Furthermore, proxy hiding will most likely make no

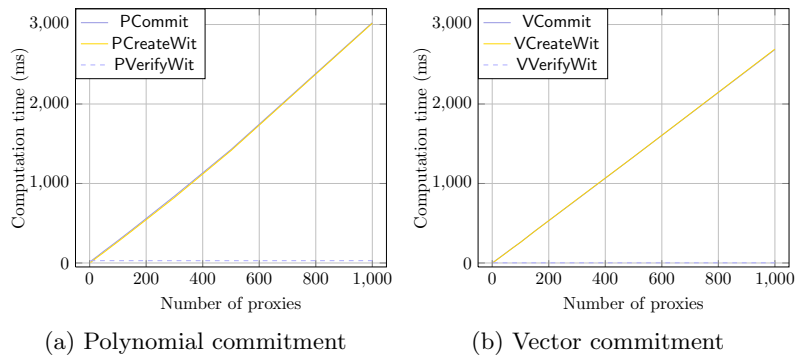


Figure 8.3: Computation times in relation to the number of proxies

sense for use cases exceeding 50 proxies, since an abstract description of the delegation, e.g., an appropriate encoding of the string "All people in Group A", would be more practical

in such a case. For use cases up to 50 proxies every step can be performed in  $< 150\text{ms}$ , independent of the used commitment, which is totally acceptable.

## Chapter 9

# Conclusion

In this thesis we have optimized the (E)BDSS from [38–40] as well as the WHPSS from [37] regarding performance and introduced a JCA-based interoperable framework for the aforementioned schemes, providing an easy to use API. In order to illustrate the capabilities, concerning the integration into other applications, two signature formats for the (E)BDSS and one signature format for the WHPSS are proposed. We expect the PDF signature format for the (E)BDSS to be very important for practical applications, since the templates and messages can be viewed using standard tools, i.e., an appropriate PDF viewer. Furthermore, we do not expect any problems when integrating the XML Signatures proposed in Section 7.4.1 and Section 7.4.2 into other XML signature formats such as XMLDSig [25] or the various types of XML Advanced Electronic Signatures [26]. The latter format would, in turn, enable long term signature validation [27], which could be of particular interest for (E)BDSS signed contracts.

From a practical point of view, the (E)BDSS execution times presented in Chapter 8 seem totally acceptable, since practical templates will most likely have a template-size of less than 100, leading to computation times of less than 190ms for any template constellation. Since the WHPSS does not perform that well and the same behavior can be modeled using the (E)BDSS, the usage of the (E)BDSS in practical applications might be beneficial.

To conclude, this thesis shows that the (E)BDSS and the WHPSS are fully feasible for practical use. Thus, a next interesting step would be to increase the practical usability over integrating (E)BDSS within a plug-in of a PDF reader. Furthermore, an investigation of the relationship between (E)BDSS and redactable/sanitizable signatures would be of interest for the future.

Finally, from a performance point of view, an investigation regarding the optimization potential and, if necessary, an optimization of the used library for computing the pairings would be of interest.

# Appendix A

## Definitions

### A.1 Abbreviations

<b>BDHP</b>	Bilinear Diffie-Hellman Problem
<b>BDS</b>	Blank Digital Signatures
<b>BDSS</b>	Blank Digital Signature Scheme
<b>CA</b>	Certification Authority
<b>CDHP</b>	Computational Diffie-Hellman Problem
<b>CRL</b>	Certificate Revocation List
<b>DBDHP</b>	Decisional Bilinear Diffie-Hellman Problem
<b>DDHP</b>	Decisional Diffie-Hellman Problem
<b>DH</b>	Diffie-Hellman
<b>DHP</b>	Diffie-Hellman Problem
<b>DLP</b>	Discrete Logarithm Problem
<b>DSS</b>	Digital Signature Scheme
<b>EBDS</b>	Extended Blank Digital Signatures
<b>EBDSS</b>	Extended Blank Digital Signature Scheme
<b>ECC</b>	Elliptic Curve Cryptography
<b>ECDH</b>	Elliptic Curve Diffie-Hellman
<b>ECDL</b>	Elliptic Curve Discrete Logarithm
<b>ECDLP</b>	Elliptic Curve Discrete Logarithm Problem
<b>ECDSA</b>	Elliptic Curve Digital Signature Algorithm
<b>FFC</b>	Finite-Field Cryptography
<b>gcd</b>	Greatest Common Divisor
<b>IFC</b>	Integer-Factorization Cryptography
<b>JCA</b>	Java Cryptography Architecture
<b>KDF</b>	Key Derivation Function
<b>MAC</b>	Message Authentication Code
<b>NIST</b>	National Institute of Standards and Technology
<b>PKCS</b>	Public Key Cryptography Standards
<b>PKI</b>	Public Key Infrastructure
<b>SHA</b>	Secure Hashing Algorithm
<b>TTP</b>	Trusted Third Party
<b>WHPS</b>	Warrant-Hiding Proxy Signatures
<b>WHPSS</b>	Warrant-Hiding Proxy Signature Scheme

## A.2 Used Symbols

$a \in_g G$	The element $a$ is a generator of the group $G$
$a \in_r G$	The element $a$ is randomly selected out of the set $G$
$P_1 \leq_P P_2$	The computational problem $P_1$ is polynomial reducible to the computational problem $P_2$
$\epsilon$	A negligible function
$Pr[\cdot]$	The probability of the argument stated within the braces
$a^{-1}$	The multiplicative inverse of an element
$\mathbb{P}$	The set of all primes
$\mathbb{R}$	The set of real numbers
$\mathbb{N}$	The set of natural numbers
$\mathbb{N}_0$	The set of natural numbers with 0
$\mathbb{Z}$	The set of integers
$\mathbb{Z}_n$	The set of integers modulo $n$
$\mathbb{Z}_n^*$	The set of invertible integers modulo $n$
$\mathbb{Z}_p[X]$	The ring of polynomials with coefficients in $\mathbb{Z}_p$
$\deg(f)$	The degree of a polynomial $f$
$\forall$	Forall quantifier
$\exists$	Existential quantifier
$\mathbb{F}$	A finite field
$\mathbb{F}_q$	A finite field with $q = p^m$
$\mathbb{F}_{2^m}$	A binary extension field
$\mathbb{F}_{p^m}$	A prime extension field
$E/F$	An elliptic curve $E$ defined over field $F$
$E(F)$	The set of all points on an elliptic curve $E$ over the field $F$ together with $\infty$
$\#E(F)$	The order of an elliptic curve group $E$ defined over the field $F$
$E(F)[p]$	The order $p$ subgroup of an elliptic curve group defined over the field $F$
$e$	A bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$
$P$	An element of $\mathbb{G}_1$
$P'$	An element of $\mathbb{G}_2$
$K$	A symmetric encryption key
pk	The public key of an asymmetric encryption scheme
sk	The secret key of an asymmetric encryption scheme
$\{0, 1\}^*$	The set of all bitstrings of arbitrary length
$\{0, 1\}^n$	The set of all bitstrings of length $n$
$\Sigma$	An alphabet
$\mathcal{C}$	A commitment
CH	A chameleon hash (trapdoor commitment)
$\perp$	An encoding for the empty string

# Appendix B

## Implementational Details

### B.1 Polynomial Arithmetic

The schemes, this thesis builds up on, rely on polynomial encodings for certain sequences of messages. Those polynomials are of the form shown in Equation (B.1), with  $M_i$  being an appropriate representation of a message out of a sequence  $\mathcal{M} = (M_1, \dots, M_n)$ . Each polynomial is in  $\mathbb{Z}_p[X]$  (with  $p$  being a prime) and is of the following form:

$$f(X) = \prod_{i=0}^n (X - M_i) \text{ with } f(X) \in \mathbb{Z}_p[X] \quad (\text{B.1})$$

From an implementation point of view the polynomials can be stored as arrays  $A = [A_0, \dots, A_n]$  of `BigIntegers`, with  $A_i$  representing the coefficient belonging to  $X^i$ . However, to get from Equation (B.1) to the aforementioned way of storing the polynomials, an expansion by multiplying the monomials  $(X - M_i)$  with each other has to be performed. Algorithm B.1 gives a detailed overview over the expansion process.

---

**Algorithm B.1** Polynomial Expansion of the equation shown in B.1

---

```
Input: Coefficients  $\mathcal{M} = (M_1, \dots, M_n)$  of  $(X - M_i)$ , reduction prime  $p$   
 $result[0] \leftarrow M[0], result[1] \leftarrow 1, d \leftarrow 1$   
for  $i = 1$  to  $n$  do  
  for  $j = d$  to  $0$  do  
     $result[j + 1] \leftarrow result[j]$   
  end for  
   $result[0] \leftarrow 0$   
  for  $j = 0$  to  $d$  do  
     $result[j] \leftarrow result[j] + result[j + 1] \cdot (-M[i]) \pmod p$   
  end for  
   $d = d + 1$   
end for  
return  $result$ 
```

---

Furthermore a method for dividing polynomials by polynomials of the form  $(X - M)$  is needed. Since, we only deal with divisions without remainder, a quite simple algorithm can be put to use (see Algorithm B.2).

Finally, the evaluation of polynomials is required. Arising from the fact that this is done in the obvious way, i.e., by summing up the evaluations for each coefficient which can be efficiently done using Horner's method, we do not further discuss the algorithm here.



**Algorithm B.2** Polynomial division without remainder using a divisor of the form  $(X - M)$ 


---

**Input:** Polynomial  $f(X)$ , coefficient  $M$  of  $(X - M)$ , reduction prime  $p$   
**for**  $i = \deg(f)$  **to** 0 **do**  
     $result[i] \leftarrow f[i + 1]$   
     $f[i] \leftarrow f[i + 1] \cdot (-M) + f[i] \pmod p$   
**end for**  
**if**  $f[0] \neq 0$  **then**  
    **return** *ERROR*:  $(X - M)$  does not perfectly divide  $f$   
**end if**  
**return**  $result$

---

## B.2 Used Object Identifiers

The following listing shows the object identifiers corresponding to the keys to be included within keystores or X.509 certificates.

```

1 iaik OBJECT IDENTIFIER ::= { ISO assigned(1) Organization acknowledged by ISO(3)
   US Department of Defense(6) Internet(1) Private(4) IANA registered private
   enterprises(1) 2706 }
2   — 1.3.6.1.4.1.2706
3
4 iaik-ebdss-pkorig OBJECT IDENTIFIER ::= { iaik-20 1 }
5   — 1.3.6.1.4.1.2706.20.1
6 iaik-ebdss-keygenoutput OBJECT IDENTIFIER ::= { iaik-20 2 }
7   — 1.3.6.1.4.1.2706.20.2
8 iaik-whpss-pkdeleg OBJECT IDENTIFIER ::= { iaik-20 3 }
9   — 1.3.6.1.4.1.2706.20.3
10 iaik-ebdss-skttp OBJECT IDENTIFIER ::= { iaik-20 4 }
11  — 1.3.6.1.4.1.2706.20.4
12 iaik-ebdss-skproxy OBJECT IDENTIFIER ::= { iaik-20 6 }
13  — 1.3.6.1.4.1.2706.20.6
14 iaik-whpss-skttp OBJECT IDENTIFIER ::= { iaik-20 7 }
15  — 1.3.6.1.4.1.2706.20.7
16 iaik-whpss-skproxy OBJECT IDENTIFIER ::= { iaik-20 8 }
17  — 1.3.6.1.4.1.2706.20.8

```

Listing 1: Object Identifiers for the BDSS Keying Material

# Appendix C

## Proofs

### C.1 Security Analysis of the (E)BDSS Modifications

#### C.1.1 Aggregating Fixed Elements

In the original construction [39, 40], every fixed element represents a monomial in the template encoding polynomial and in further consequence in every message encoding polynomial. The modification proposed here integrates all fixed elements into a single monomial, which reduces the degree of the respective polynomials. Now, we have to show that this has no impact on the security of the construction. Our argumentation is as follows.

*Proof.* Using one monomial for the fixed elements in the modified version can be seen as the original construction using only a single fixed element in the template. Therefore, the construction as such still remains secure. What remains to show, however, is that the modified encoding does not influence the signature soundness and the unforgeability as well as immutability, respectively.

The signatures soundness in this context essentially says that given a template signature  $\sigma_{\mathcal{T}}$  for some template  $\mathcal{T}$ , the probability that this signature will verify for any  $\mathcal{T}' \neq \mathcal{T}$  is negligible in the security parameter  $\kappa$ . In order to achieve this (for fixed elements), one would need to find

$$H(id_{\mathcal{T}} \| m_{i_1} \| i_1 \| \dots \| m_{i_u} \| i_u) = H(id_{\mathcal{T}'} \| m'_{i'_1} \| i'_1 \| \dots \| m'_{i'_u} \| i'_u)$$

which is clearly intractable if  $H$  is collision resistant. The same argumentation holds for unforgeability and immutability (cases **T1**, **M1** [40]), where here the problem is to find a second preimage for  $H(id_{\mathcal{T}} \| m_{i_1} \| i_1 \| \dots \| m_{i_u} \| i_u)$ .  $\square$

#### C.1.2 Aggregating Blank Elements

In the original construction [40], every blank element represents a monomial in the template encoding polynomial and, in further consequence, in every message encoding polynomial. The modification proposed here integrates all blank elements into a single monomial, which reduces the degree of the respective polynomials. Now, we have to show that this has no impact on the security of the construction. Our argumentation is as follows.

*Proof.* Using one monomial for the blank elements in the modified version can be seen as the original construction using only a single blank element in the template. Therefore,

the construction as such still remains secure. What remains to show, however, is that the modified encoding does not influence the unforgeability as well as immutability, respectively.

As in the message signature  $\sigma_{\mathcal{M}}$  the randomizers  $\mathbb{R} = (r_{i_j})_{j=1}^v$  are signed, this means that  $\sum_{j=1}^v a_{i_j} \text{cpk}$  is fixed. Due to the used chameleon hash being message-binding (as defined in [40]), meaning that the map  $m \mapsto \text{CHash}(\text{cpk}, m, r)$  is injective, the value  $\sum_{j=1}^v m_{i_j}$  with  $m_{i_j} = H(M_{i_j} \| i_j)$  is fixed too. Consequently, one would need to find  $\sum_{j=1}^v H(M_{i_j} \| i_j) = \sum_{j=1}^v H(M'_{i_j} \| i_j)$  which either requires finding second preimages for all  $H(M_{i_j} \| i_j)$ , or choosing all but one of the addends in the sum and computing a preimage for the remaining addend. Clearly, both cases are intractable when assuming that  $H$  is a secure hash function.  $\square$

## C.2 Composition of Secure Chameleon Hash and Secure Hash

The following proof was taken from [40] and is included here for convenience of the reader.

*Proof.* Let  $\text{CH} = (\text{CKeyGen}, \text{CHash}, \text{CForge})$  be a secure chameleon hash scheme and  $H$  be a secure cryptographic hash function, i.e.,  $H$  is collision-, preimage- and second preimage-resistant.

At first, we prove the collision resistance of the composition  $\mathcal{CH}(\text{cpk}, m, r) = (H \circ \text{CHash})(\text{cpk}, m, r)$ . Suppose that  $\mathcal{CH}$  is not collision-resistant. Then, either  $\text{CH}$  is not collision-resistant, or  $H$  is not collision-resistant, i.e., it is possible to find two chameleon hash values  $c \neq c'$  such that  $H(c) = H(c')$ . In both cases, we get a contradiction. Obviously, in the latter case it must, additionally, be possible to find values  $(m, r) \neq (m', r')$  such that  $c = \text{CHash}(\text{cpk}, m, r)$  and  $c' = \text{CHash}(\text{cpk}, m', r')$ .

Secondly, if  $\text{CH}$  is semantically secure, then it follows directly that this is the case for the  $\mathcal{CH}$  construction and if  $\text{CH}$  is also key-exposure free, then this is trivially the case for  $\mathcal{CH}$ .  $\square$

# Bibliography

- [1] Federal Information Processing Standard: Data Encryption Standard. In *FIPS Publication 46*, 1977.
- [2] *Public Key Cryptography for the Financial Services Industry - Key Agreement and Key Transport Using Elliptic Curve Cryptography: ANSI American National Standard for Financial Services, X9.63-2001*. American national standard / ANSI. Accredited Standards Committee X9, Incorporated, 2001.
- [3] ISO/IEC 9797. Information Technology: Security Techniques : Message Authentication Codes (MACs). Technical report, International Standards Organization, 2011.
- [4] G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable Signatures. In *ESORICS*, volume 3679 of *LNCS*, pages 159–177. Springer, 2005.
- [5] R. Barbulescu, P. Gaudry, A. Joux, and E. Thome. A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. *Cryptology ePrint Archive*, Report 2013/400, 2013. <http://eprint.iacr.org/>.
- [6] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for Key Management - Part 1: General (Revision 3). In *NIST Special Publication*, 2012.
- [7] P. S. Barreto, S. D. Galbraith, C. O. Héigearthaigh, and M. Scott. Efficient pairing computation on supersingular Abelian varieties. *Des. Codes Cryptography*, 42(3):239–271, Mar. 2007.
- [8] P. S. L. M. Barreto and M. Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In *Selected Areas in Cryptography*, LNCS, pages 319–331. Springer, 2005.
- [9] M. Bellare and P. Rogaway. *Introduction to Modern Cryptography*. <http://cseweb.ucsd.edu/~mihir/cse207/classnotes.html>, 2005.
- [10] G. Bertoni, J. Daemen, P. Michael, and G. Van Assche. The Keccak sponge function family. [keccak.noekeon.org](http://keccak.noekeon.org).
- [11] A. Boldyreva, A. Palacio, and B. Warinschi. Secure Proxy Signature Schemes for Delegation of Signing Rights. *Cryptology ePrint Archive*, Report 2003/096, 2003. <http://eprint.iacr.org/>.
- [12] A. Boldyreva, A. Palacio, and B. Warinschi. Secure Proxy Signature Schemes for Delegation of Signing Rights. *J. Cryptology*, 25(1):57–115, 2012.
- [13] D. Boneh and X. Boyen. Short Signatures Without Random Oracles. *IACR Cryptology ePrint Archive*, 2004:171, 2004.

- [14] D. Boneh and M. K. Franklin. Identity-Based Encryption from the Weil Pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 213–229, London, UK, UK, 2001. Springer-Verlag.
- [15] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '01, pages 514–532, London, UK, UK, 2001. Springer-Verlag.
- [16] C. Brzuska, H. Busch, Ö. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder. Redactable Signatures for Tree-Structured Data: Definitions and Constructions. In *ACNS*, volume 6123 of *LNCS*, pages 87–104. Springer, 2010.
- [17] D. Catalano and D. Fiore. Vector Commitments and Their Applications. In K. Kurosawa and G. Hanaoka, editors, *Public Key Cryptography*, volume 7778 of *LNCS*, pages 55–72. Springer, 2013.
- [18] S. Chatterjee and A. Menezes. On Cryptographic Protocols Employing Asymmetric Pairings – The Role of  $\Psi$  Revisited. Cryptology ePrint Archive, Report 2009/480, 2009. <http://eprint.iacr.org/>.
- [19] S. Chatterjee and A. Menezes. On Cryptographic Protocols Employing Asymmetric Pairings - The Role of Revisited. *Discrete Applied Mathematics*, 159(13):1311–1322, 2011.
- [20] L. Chen. Recommendation for Key Derivation using Pseudorandom Functions. *NIST Special Publication*, 800:108.
- [21] X. Chen, F. Zhang, and K. Kim. Chameleon Hashing Without Key Exposure. In K. Zhang and Y. Zheng, editors, *ISC*, volume 3225 of *LNCS*, pages 87–98. Springer, 2004.
- [22] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008.
- [23] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Verlag, Berlin, Heidelberg, New York, 2002.
- [24] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
- [25] D. Eastlake, J. Reagle, and D. Solo. *XML-Signature Syntax and Processing*. 2002. W3C Recommendation.
- [26] European Telecommunications Standards Institute. Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAAdES); ETSI TS 101 903, 2010.
- [27] European Telecommunications Standards Institute. Electronic Signatures and Infrastructures (ESI); Signature verification procedures and policies; ETSI TS 102 853, 2012.

- [28] J. Fialli and S. Vajjhala. Java Architecture for XML Binding (JAXB) 2.0. Java Specification Request (JSR) 222, October 2005.
- [29] G. Frey, M. Muller, and H.-G. Ruck. The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. *Information Theory, IEEE Transactions on*, 45(5):1717–1719, 1999.
- [30] G. Frey and H.-G. Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Math. Comput.*, 62(206):865–874, Apr. 1994.
- [31] P. Gallagher, D. D. Foreword, and C. F. Director. FIPS Publication 186-3 Federal Information Processing Standards Publication: Digital Signature Standard (DSS), 2009.
- [32] C. C. F. P. Geovandro and P. S. L. M. Barreto. bnpairings - A Java implementation of efficient bilinear pairings and elliptic curve operations. Public Google code project at: <https://code.google.com/p/bnpairings/>, 5 November 2012.
- [33] O. Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.
- [34] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.
- [35] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.
- [36] C. Hanser. New Trends in Elliptic Curve Cryptography. Master’s thesis, Graz University of Technology, 2010.
- [37] C. Hanser and D. Slamanig. Warrant-Hiding Delegation-by-Certificate Proxy Signature Schemes. In *Progress in Cryptology - INDOCRYPT 2013, 14th International Conference on Cryptology in Mumbai, India, December 7-10, 2013. Proceedings*, LNCS. Springer. to appear.
- [38] C. Hanser and D. Slamanig. Blank Digital Signatures. In *8th ACM SIGSAC Symposium on Information, Computer and Communications Security (AsiaCCS)*, pages 95–106. ACM, 2013.
- [39] C. Hanser and D. Slamanig. Blank Digital Signatures. Cryptology ePrint Archive, Report 2013/130, 2013. <http://eprint.iacr.org/>.
- [40] C. Hanser and D. Slamanig. Extending Blank Digital Signatures or How to Delegate Signing of Arbitrary Authorized Forms, 2013. manuscript.
- [41] F. Hess, N. Smart, F. Vercauteren, and T. U. Berlin. The Eta Pairing Revisited. *IEEE Transactions on Information Theory*, 52:4595–4602, 2006.
- [42] International Telecommunication Union. Information Technology — Abstract Syntax Notation One (ASN.1): Specification of Basic Notation. ITU-T Recommendation X.680, July 2002.

- [43] International Telecommunication Union. Information Technology — ASN.1 Encoding Rules — Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER). ITU-T Recommendation X.690, July 2002.
- [44] iText Software Corp. iText<sup>®</sup>, a Java-PDF library. Public sourceforge project at: <http://sourceforge.net/projects/itext/files/>, 11 April 2013.
- [45] R. Johnson, D. Molnar, D. X. Song, and D. Wagner. Homomorphic Signature Schemes. In *CT-RSA*, volume 2271 of *LNCS*, pages 244–262. Springer, 2002.
- [46] A. Joux. A One Round Protocol for Tripartite Diffie-Hellman. In *Proceedings of the 4th International Symposium on Algorithmic Number Theory, ANTS-IV*, pages 385–394, London, UK, UK, 2000. Springer-Verlag.
- [47] A. Joux. A new index calculus algorithm with complexity  $L(1/4 + o(1))$  in very small characteristic. Cryptology ePrint Archive, Report 2013/095, 2013. <http://eprint.iacr.org/>.
- [48] A. Joux and K. Nguyen. Separating Decision Diffie-Hellman from Computational Diffie-Hellman in Cryptographic Groups. *Journal of Cryptology*, 16(4):239–247, 2003.
- [49] B. Kaliski. PKCS#7: Cryptographic Message Syntax Version 1.5. RFC 2315 (Informational), March 1998.
- [50] A. Kate, G. Zaverucha, and I. Goldberg. Polynomial Commitments, 2010. Available at <http://cacr.uwaterloo.ca/techreports/2010/cacr2010-10.pdf>.
- [51] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-Size Commitments to Polynomials and Their Applications. In *ASIACRYPT*, volume 6477 of *LNCS*, pages 177–194. Springer, 2010.
- [52] J. Katz and Y. Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.
- [53] A. Kerckhoffs. La Cryptographie Militaire. *Journal des sciences militaires*, IX:5–83, Jan. 1883.
- [54] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, Jan. 1987.
- [55] B. Libert and J.-J. Quisquater. Efficient Signcryption with Key Privacy from Gap Diffie-Hellman Groups. In F. Bao, R. Deng, and J. Zhou, editors, *Public Key Cryptography - PKC 2004*, volume 2947 of *LNCS*, pages 187–200. Springer Berlin Heidelberg, 2004.
- [56] S. Lichtenbaum. Duality theorems for curves over P-adic fields. *Inventiones mathematicae*, 7(2):120–136, 1969.
- [57] P. Linz. *An Introduction to Formal Language and Automata*. Jones and Bartlett Publishers, Inc., USA, 2006.
- [58] B. Lynn. *On the Implementation of Pairing-Based Cryptosystems*. PhD thesis, Dept. of Computer Science, Stanford University, 2007.

- [59] M. Madritsch. *Mathematische Grundlagen der Kryptografie*. Lecture Notes to the Lecture: *Mathematische Grundlagen der Kryptografie* at the Graz University of Technology, 2009.
- [60] M. Mambo, K. Usuda, and E. Okamoto. Proxy Signatures for Delegating Signing Operation. In *ACM Conference on Computer and Communications Security'96*, pages 48–57, 1996.
- [61] A. Menezes. *An Introduction to Pairing-Based Cryptography*. Notes from lectures given in, 2005.
- [62] A. Menezes, S. Vanstone, and T. Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing, STOC '91*, pages 80–89, New York, NY, USA, 1991. ACM.
- [63] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.
- [64] R. C. Merkle. Method of providing digital signatures, May 1982.
- [65] V. S. Miller. Use of Elliptic Curves in Cryptography. In *LNCS; 218 on Advances in cryptology—CRYPTO 85*, pages 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc.
- [66] K. Miyazaki, M. Iwamura, T. Matsumoto, R. Sasaki, H. Yoshiura, S. Tezuka, and H. Imai. Digitally Signed Document Sanitizing Scheme with Disclosure Condition Control. *IEICE Transactions*, 88-A(1):239–246, 2005.
- [67] C. Moore and S. Mertens. *The Nature of Computation*. OUP Oxford, 2011.
- [68] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 2560 (Proposed Standard), June 1999.
- [69] B. C. Neuman. Proxy-Based Authorization and Accounting for Distributed Systems. In *Proceedings of the 13th International Conference on Distributed Computing SYSTEMS*, pages 283–291, 1993.
- [70] T. Okamoto and D. Pointcheval. The Gap-Problems: A New Class of Problems for the Security of Cryptographic Schemes. In K. Kim, editor, *Public Key Cryptography*, volume 1992 of *LNCS*, pages 104–118. Springer Berlin Heidelberg, 2001.
- [71] Oracle. Java™ Cryptography Architecture (JCA) Reference Guide. <http://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html>.
- [72] C. Paar and J. Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer-Verlag New York Inc, 2010.
- [73] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '91*, pages 129–140, London, UK, UK, 1992. Springer-Verlag.



- [74] H. Riesel. *Prime numbers and computer methods for factorization*. Birkhauser Boston Inc., Cambridge, MA, USA, 1985.
- [75] V. Rijmen. Applied Cryptography 1. Slides to the Lecture: Applied Cryptography 1 at the Graz University of Technology, 2012.
- [76] I. A. Semaev. Evaluation of discrete logarithms in a group of  $p$ -torsion points of an elliptic curve in characteristic  $p$ . *Mathematics of Computation*, 1998.
- [77] M. Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 2nd edition, 1996.
- [78] R. Steinfeld, L. Bull, and Y. Zheng. Content Extraction Signatures. In *ICISC*, volume 2288 of *LNCS*, pages 285–304. Springer, 2001.
- [79] J. Tate. *WC - groups over  $p$ -adic fields*, volume 13 of *Séminaire Bourbaki; 10e année: 1957/1958. Textes des conférences; Exposés 152 à 168; 2e éd. corrigée, Exposé 156*. Secrétariat mathématique, Paris, 1958.
- [80] J. Tate. Duality theorems in Galois cohomology over number fields. In *Proc. Internat. Congr. Mathematicians (Stockholm, 1962)*, pages 288–295. Inst. Mittag-Leffler, Djursholm, 1963.
- [81] The European Parliament and the Council of the European Union. Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures. *Official Journal of the European Communities*, L 12:12–20, Feb000-0Jan-JanSep 2000.
- [82] V. Varadharajan, P. Allen, and S. Black. An Analysis of the Proxy Problem in Distributed Systems. In *IEEE Symposium on Security and Privacy*, pages 255–277, 1991.
- [83] F. Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010.
- [84] L. C. Washington and W. Trappe. *Introduction to Cryptography: With Coding Theory*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2002.