**UNIVERSIDAD DE VIGO**

**ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN**

**Proyecto Fin de Carrera**

# Public-key Encryption with Oblivious Keyword Search. Priced Oblivious Transfer

Autor: Alfredo Rial Durán
Tutor: Juan Carlos Burguillo Rial

Titulación: Ingeniería de Telecomunicación

Curso 2007-2008

**KATHOLIEKE UNIVERSITEIT LEUVEN**
FACULTEIT INGENIEURSWETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK-ESAT
Kasteelpark Arenberg 10, 3001 Leuven-Heverlee

**Master Thesis**

# Public-key Encryption with Oblivious Keyword Search. Priced Oblivious Transfer

Author:
Alfredo Rial Durán

Promoter:
Prof. Bart Preneel

Assessors:
Claudia Diaz
Gregory Neven

Supervisors:
Claudia Diaz
Markulf Kohlweiss

Framework: Erasmus program

**Title:** Public-key Encryption with Oblivious Keyword Search. Priced Oblivious Transfer
**Author:** Alfredo Rial Durán
**Supervisor:** Juan Carlos Burguillo Rial

# Abstract

The use of electronic communication services can pose a threat to the privacy of individuals. Currently, when carrying out different kinds of transactions some personal information is disclosed. This information can be collected to make profiles that may be sold.

However, the characteristics of this communication services provide the possibility of having better privacy preserving properties than their counterparts in the physical world. This is an important feature since users may prefer the service providers that offer solutions in which they have to reveal as little information as possible about themselves.

Therefore, we provide constructions in which the party who requests a service discloses the minimum personal information, whereas the party that offers data is guaranteed that no more data than the one being requested is going to be learned. Particularly, we focus on two applications: the search by keywords on encrypted data and the buying of digital goods.

For the former, there existed two approaches that aimed at solving it in a privacy preserving manner: Public-key Encryption with Keyword Search (PEKS) and Oblivious Keyword Search (OKS). We present here a new concept, Public-key Encryption with Oblivious Keyword Search (PEOKS), that extends PEKS and that offers similar properties to OKS. In order to build a PEOKS scheme, we have designed the first anonymous Identity-Based Encryption scheme with blind key derivation. Furthermore, we have applied PEOKS to solve in a privacy-enhancing way the obligation for communication service providers to retain data, as stated by the Directive 2006/24/EC of the EU. Finally, we propose other applications of blind-anonymous-IBE.

For the buying of digital goods, we propose an improvement of the Priced Oblivious Transfer primitive that makes it possible for the vendor to apply marketing techniques to his business. Moreover, we provide an implementation of this scheme and we discuss its security design and its performance.

**Keywords:** Identity-Based Encryption, Public-key Encryption with Keyword Search, Oblivious Keyword Search, Priced Oblivious Transfer.

# Acknowledgements

First and foremost, I would like to thank Markulf Kohlweiss, who has been a great supervisor for me. I am grateful to him for all the work he has done for this thesis, for the interest he has always shown, for his patience when answering questions, for all the time he has spent with me, for being kind and helpful and for everything I have learnt from him.

My gratitude also goes to Claudia Diaz for letting me work at COSIC and for supporting and helping me whenever necessary. My sincere thanks for her useful advices and discussions.

Gregory Neven has been an inexhaustible source of inspiration. Since many concepts developed here are based on his previous work, I feel that I have been truly privileged having him as my assessor. I am grateful for his interest and for his important comments.

I am very grateful to Caroline Sheedy, who has also collaborated on this work. I would like to thank her for encourage me and for all the discussions we had. I would also like to thank Jan Camenisch for proposing the data retention scenario.

The ESAT-SCD-COSIC research group at Katholieke Universiteit Leuven has been a motivating and helpful environment. Particularly, I am very grateful to my officemate Stefan Schiffner for creating a nice working atmosphere, for helping me with every technical or administrative doubt and for his comments on how to structure and typeset a thesis.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The use of electronic communication services can cause a disclosure of data that may be useful to describe an individual. Among this data not only personal information such as the name or the address should be included, but also information that expounds the preferences of the users when browsing the web or when carrying out different transactions. For example, the terms that a user chooses when searching for information in the web can reveal her hobbies, her political membership, her religion or her sexual preferences.

On the other hand, the progress of information technologies has provided tools by means of which data can be easily collected, stored and shared. By using this tools, it is possible to automatize the process of creating personal profiles by using information that describes the behavior of an individual when using electronic communication services. Since this kind of profiles can be utilized for several purposes, such as offering advertisements specific for each consumer or even the prosecution of crime, the interest in building and selling these profiles has been increasing during the last years.

However, the right to privacy states that everybody should be respected for her private and family life, her home and her correspondence. This must include not only information useful to identify and locate an individual, but also data that could be utilized to describe her thoughts or her behavior. Therefore, there exists the necessity of providing privacy preserving electronic communication services, in which users have to reveal no more than the minimum information needed to carry out their transactions. This can be a real concern for service providers, since users may prefer services that respect their privacy.

One approach to construct privacy-enhancing services consists in providing anonymity. In this case, privacy is protected by hiding the identity of the user, and thus it is not possible to relate the collected data with a particular individual.

Nevertheless, in some contexts anonymity can be difficult to achieve or even undesirable. If it is the case, services that provide privacy by revealing

as little information as possible about the choices and the behavior of a user when utilizing electronic communication services should be applied.

## 1.1 Goal of the Thesis

We propose constructions that solve the problem of revealing private information in two scenarios: the search by keywords on encrypted data and the buying of digital goods. In both scenarios we provide a solution in which users disclose as little information as necessary for obtaining the service, whereas the party that offers the service is guaranteed that no more than the requested service is provided.

For the first application, there existed two methods that aimed at protecting privacy: Public-key Encryption with Keyword Search (PEKS) [BDOP04] and Oblivious Keyword Search (OKS) [OK04].

In PEKS the parties that encrypt messages attach to each message one or more keywords that describe its content in an encrypted manner in order to make it possible to efficiently search for some particular messages. This encrypted keywords are referred to as searchable encryptions. Then, when a user wants to search for the messages that were attached to a specific keyword, she first gets a trapdoor for this keyword and by means of the trapdoor she can find and retrieve the corresponding messages.

PEKS is usually instantiated as an application of anonymous Identity-Based Encryption (IBE) [ABC+05]. In IBE [BF01], the identity of a user acts as her public key, and there exists a master authority, usually referred to as Key Generation Center (KGC), that derives secret keys for identities. Intuitively, its application to PEKS consists in using identities as keywords and secret keys as trapdoors.

However, this instantiation can be a problem when the party that derives the trapdoors, the Trapdoor Generation Center (TGC), is not the same as the one that chooses the keywords. Namely, since in IBE the KGC learns the identities for which it derives secret keys, the TGC will learn all the keywords for which users request trapdoors. This is a serious threat to privacy since the selected keywords reveal the kind of information that is of interest for a particular user.

In order to solve this problem, we designed a blind key derivation protocol for an existing anonymous-IBE scheme, and thus we build the first blind-anonymous-IBE scheme. Using this scheme to build an application that allows searches by keywords on encrypted data, we are able to hide the keywords from the party that derives trapdoors. We call this solution Public-key Encryption with Oblivious Keyword Search (PEOKS). We define its security properties and we prove that this properties hold for a construction for PEOKS that is based on blind-anonymous-IBE.

Moreover, we apply PEOKS to propose a solution that preserves the privacy of the police in the data retention scenario that is specified by the Directive 2006/24/EC of the EU. This directive requires the communication service providers to retain the traffic and location data of a communication along with the information needed to identify the user. By means of PEOKS, we achieve a solution in which the communication service providers can store this data in an efficient and secure way. Moreover, the investigators, when allowed to by a judge, can efficiently search for the data that was attached to a specific keyword without revealing the keyword. Finally, it is guaranteed that the investigators do not get any information about data that was attached to keywords they were not allowed to search for.

Furthermore, we propose other applications of blind-anonymous-IBE.

For the second application, the buying of digital goods, there existed a privacy preserving primitive called Priced Oblivious Transfer (POT) [AIR01], which is an extension of the Oblivious Transfer (OT) primitive [Rab81].

In OT we have a scenario with a sender and a receiver. The sender offers a set of messages to the receiver, and the receiver gets several messages from the set in such a way that the sender does not learn which messages the receiver obtains, whereas the receiver learns nothing about the rest of messages.

In POT, the receiver has to pay for obtaining messages. Therefore, apart from providing the security and privacy properties of OT, it must be guaranteed that the receiver pays the right price for the messages she gets.

Here, we propose a POT scheme that provides more features than the existing ones. Namely, in previous schemes the price of a specific message was the same for all the receivers, but in our scheme the sender can charge different prices for the same message depending on the particular receiver. This allows the sender to apply marketing techniques to his business. For example, he can offer smaller prices to underage receivers.

## 1.2 Structure of the Thesis

The thesis is structured as follows:

**Chapter 2: Preliminaries** gives the mathematical background needed to understand the rest of the chapters. In addition, it introduces some concepts that are used in this thesis such as bilinear pairings, commitment schemes, proofs of knowledge, blind signature schemes and private credentials.

**Chapter 3: Basic and Related Work** explains the existing schemes on which the constructions presented in Chapter 4 and in Chapter 5 are

based. Namely, an introduction to Identity-Based Encryption, Public-key Encryption with Keyword Search, Oblivious Transfer and Oblivious Keyword Search is given. In addition, some new results, such as a generalized definition for PEKS or a security definition in the full-simulation model for OKS, are provided.

**Chapter 4: Public-key Encryption with Oblivious Keyword Search** defines the concept of PEOKS, shows how to instantiate a PEOKS scheme from any blind-anonymous-IBE scheme and proves the security of this instantiation. Moreover, it provides a blind key derivation protocol for an existing anonymous IBE scheme and proves its security. Apart from that, it also explains how to apply PEOKS to construct an efficient solution for the data retention scenario specified by the Directive 2006/24/EC of the EU. Finally, it proposes other applications of blind-anonymous-IBE.

**Chapter 5: Priced Oblivious Transfer** provides different variants of a Priced Oblivious Transfer scheme and compares them with previous solutions.

**Chapter 6: Implementation of a Priced Oblivious Transfer scheme** describes the implementation of one of the Priced Oblivious Transfer schemes depicted in Chapter 5 and discusses the implementation's characteristics.

**Chapter 7: Conclusion and Future Work** draws a conclusion and gives some hints of possible future topics of research.

## 1.3 Conventions

Throughout the thesis, the parties that offer data are of masculine gender, whereas the parties that are the final destination of data are of feminine gender. For example, in IBE, the sender will be of masculine gender and the receiver of feminine gender. Parties like the KGC are of neuter gender. Finally, if a party is referred to as the adversary it is defined to be of masculine gender.

# Chapter 2

# Preliminaries

In this chapter we present a brief explantation of the basic concepts used later on in our constructions. We start by describing the notation utilized in the rest of the thesis and we show a few concepts of probability theory, complexity theory and Turing machines. Then we give an introduction to abstract algebra and in particular to groups in Section 2.5, and we talk about number-theory problems that appear in such settings in Section 2.6. These are basic concepts utilized in many cryptographic algorithms, but we do not pretend to explain them in detail. The reader interested in having a strong background in number theory can consult many books of mathematics, but we recommend to use two cryptography-oriented books: the *Handbook of Applied Cryptography* [MvOV96] and *The Foundations of Cryptography* [Gol01].

Armed with these concepts, we introduce more specific concepts that are also used in this thesis. In Section 2.7 we talk about bilinear pairings. Although bilinear pairings usually involve the use of elliptic curve cryptography we introduce them using abstract notation. The reader interested in elliptic curve cryptography and its application to pairings can consult [Maa04] or [Lyn07]. After that, we explain commitments in Section 2.8 and proofs of knowledge in Section 2.4. A good introduction to proofs of knowledge can be found in [Cam98]. Finally, we define the concept of blind signature scheme in Section 2.10 and credentials in Section 2.11.

Readers that are already familiar with these concepts can skip this chapter without losing any relevant information about the constructions presented in the thesis.

## 2.1 Notation

A set is a collection of objects which satisfy some property. To denote a set we enumerate its elements using the set notation, e.g., $\{1, 2, \ldots, 100\}$ or the interval notation. We distinguish between closed integer intervals, e.g.,

[1..100], and open integer intervals, e.g., ]0..101[. All these examples define the same set.

To name a set, we use capital letters, e.g., $M = \{m_1, \ldots, m_n\}$, whereas the notation $\bar{M}$ denotes a set of sets $\bar{M} = \{M_1, \ldots, M_r\}$. $|M|$ stands for the number of elements of a set, and $m_1 \in M$ means that $m_1$ is a member of the set $M$. If $M$ and $R$ are sets, $M \subseteq R$ means that $M$ is a subset of $R$, whereas $M \subset R$ means that $M$ is a proper subset of $R$, i.e., $M \subseteq R$ and $M \neq R$. Finally, $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ denote the set of non-negative integers, the set of integers and the set of real numbers respectively.

Let $a$ be a real number. We denote by $\lfloor a \rfloor$ the largest integer $b$ such that $b \leq a$, by $\lceil a \rceil$ the smallest integer $b \geq a$, and by $\lceil a \rfloor$ the largest integer $b \leq a + \frac{1}{2}$.

A function or mapping $f : M \rightarrow R$, where $M$ and $R$ are sets, is a rule that assigns to each element $m \in M$ precisely one element $r \in R$.

When speaking about parties we use again a capital letter to name a party. Some characters are reserved as follows: $P$ denotes a prover and $V$ a verifier; $S$ stands for sender and $R$ for receiver; finally, $B$ is a buyer and $V$ is a vendor. We use $\tilde{P}$ to refer to a malicious prover and so on.

When referring to algorithms, sometimes a capital letter as subindex of the name of the party is used to denote an algorithm executed by this party, e.g., $S_I$ stands for the initialization algorithm of the sender. $S_I(\cdot)$ means that the algorithm has one input, whereas $S_I(\cdot, \cdot, \ldots)$ means that it has more than one input. $y \leftarrow S_I(x)$ means that algorithm $S_I$ outputs $y$ on input $x$. If $S_I$ is deterministic then it defines a function and thus $y$ is unique, but if $S_I$ is probabilistic, $y$ is a random variable. Character $A$ is reserved to denote the algorithm executed by a malicious party, often referred to as the adversary.

Finally, $\epsilon$ stands for an empty string and $\perp$ denotes that an algorithm or a protocol aborts.

## 2.2 Complexity Theory

In this section we explain several basic notions of complexity theory and we give a brief classification for algorithms according to their running-time. After that, we define the concepts of intractable problem and negligible function.

Complexity theory aims at classifying computational problems in order to measure the intrinsic difficulty of the problem, taking into consideration the resources needed. There are different types of resources: size, time, etc., but we focus on time. First of all, we recall some well-known definitions [MvOV96].

**Definition 1 (Algorithm)** *An algorithm is a well-defined computational*

*procedure that takes a variable input and halts with an output.*

The "well-defined computational procedure" mentioned in the definition of algorithm will be made precise when using Turing machines (see Section 2.3). On the other hand, one party can execute one or more algorithms.

**Definition 2 (Running time)** *The running time of an algorithm on a particular input is the number of primitive operations or "steps" executed.*

**Definition 3 (Worst-case running time)** *The worst-case running time of an algorithm is an upper bound on the running time for any input, expressed as a function of the input size.*

**Definition 4 (Average-case running time)** *The average-case running time of an algorithm is the average running time over all inputs of a fixed size, expressed as a function of the input size.*

In cryptography average-case complexity is more important than worst-case complexity because a necessary condition for an encryption scheme to be considered secure is that the corresponding cryptanalysis problem is difficult on average and not only in the worst case.

Since measuring the exact running time of an algorithm is difficult, commonly an asymptotic notation is used, by means of which it is shown that the running time of an algorithm is given by a function that grows asymptotically as fast as a reference function on the size of the input. More formally, consider functions $f : \mathbb{N} \to \mathbb{R}$ and $g : \mathbb{N} \to \mathbb{R}$ that map from the positive integers to real values that are positive from some point onwards, and let $n$ be the size of the input. Then:

**Definition 5 (Asymptotic upper bound)** $f(n) \in O(g(n))$ *expresses that there exist a constant* $c > 0$ *and an integer* $n_0 > 0$ *such that* $0 \leq f(n) \leq cg(n)$, $\forall n \geq n_0$.

This means that $f$ grows no faster asymptotically than $g$ to within a constant multiple, where usually $f$ expresses the running time of the algorithm an $g$ is the reference function.

An algorithm runs in polynomial time if $f(n) \in O(n^k)$ for some constant $k$, whereas it runs in exponential time if $f(n) \in O(c^n)$ for some $c > 1$. Informally speaking, a subexponential time algorithm is an algorithm that is asymptotically faster than any exponential time algorithm but slower than any polynomial time algorithm.

Armed with this classification, we can define intractable problems as follows (for a more formal definition see [OP01]):

**Definition 6 (Intractable problem)** *A problem is intractable (or hard) when no polynomial time algorithm on the size of the input exists to solve this problem.*

Nowadays, it is not known if there exists any intractable problem. Therefore, from now on, when talking about intractable problems we are referring to potentially intractable problems.

We divide intractable problems into two groups: one group that includes all the problems for which a subexponential time algorithm is known and another one for the problems that can be solved only by using exponential time algorithms.

Finally, we define the concept of negligible function, which we use later on mainly to show that the probability of an event is very small.

**Definition 7 (Negligible function)** *A function $f$ is negligible in $n$ if for every polynomial $p$ and sufficiently large $n$, $f(n) < \frac{1}{p(n)}$.*

In the following we use $\nu(k)$ to denote a function negligible in the security parameter $k$.

## 2.3 Turing Machines

In this section we will explain briefly the computation model for computers called Turing machines, since we assume that Turing machines are the computational devices used by every party in our constructions. After that we define the concept of oracle access, which is used for security definitions.

A Turing machine is essentially a finite-state sequential machine that has the ability to communicate with an external store of information [Boo67]. At any time the machine is in one specific state, and there are rules that specify conditions under which the machine will change its state.

The machine has an infinite one-dimensional tape divided into cells. Each cell contains one symbol that belongs to an alphabet. It also has a read-write head to scan single cells on the tape. It can move left and right along the tape to scan successive cells. Finally, it has a table of transition rules that specify the action that has to be executed.

The action of a Turing machine is completely determined by three elements: its current state, the symbol that is currently read from the tape and the table of transition rules. The rule that should be applied is determined by the current state and the symbol. Each transition rule establish the next state to which the machine has to move by taking a specific action. The actions available to a Turing machine are either to write a symbol on the current cell or to move the head one position to the left or to the right (in some models, both actions can be indicated in the same rule). When the machine reaches a situation in which there is no transition rule specified, it halts.

Until now we have described the most basic model of Turing machines, but we need to extend it. A Turing machine is called probabilistic if it has

an additional random tape, whereas it is polynomial time if it executes only a polynomial number of steps. By a non-uniform Turing machine we mean a Turing machine that executes a different program for each different size of inputs.

To give some intuition, we can say that a computer can be seen as a Turing machine that does not have infinite memory. The tape would be the memory and the table of transition rules the "program", whereas the read-write head can be seen as the bus through which data is accessed and updated by the computer.

Turing machines aim at defining what is computable by using algorithms, which given an input provide an output. However, in a computation that involves machines that communicate with each other there are tasks that cannot be reduced to algorithms. Therefore, to capture the notion of interactive computation, the concept of interactive Turing machine has been proposed. An interactive Turing machine is a multi-tape Turing machine that can share tapes with other interactive Turing machines, such that the write-only output tape of one machine constitutes the read-only input tape of the other. A pair of interactive Turing machines can run interactive protocols, in which at each round one of them runs and writes to its output tape, which becomes available to the other one on its input tape at the beginning of the next round.

Let $U$ and $V$ be Turing machines. We say that $U$ has oracle access to $V$, short $U^{\hookrightarrow V}$, if $V$ has a tape which it can access only if $U$ allows it, and that $U$ cannot access directly, in such a way that $U$ can order $V$ to store its state on this tape at any step of $V$'s computation. In addition, $U$ can order $V$ to return to any state previously stored in the tape.

## 2.4 Probability Theory

In this section we want to explain different notions of indistinguishability between ensembles of random variables and, after that, we explain the notation we use to describe probabilities. First, we remind several basic concepts of probability theory.

**Definition 8 (Experiment)** *An experiment is a procedure that yields one of a given set of outcomes. This set is called sample space $S$.*

We consider only discrete sample spaces, i.e., sample spaces with a finite number of outcomes. We denote them by $S = \{s_1, \ldots, s_n\}$.

**Definition 9 (Discrete probability distribution)** *A discrete probability distribution $P$ is a sequence of non-negative real numbers $\{p_1, \ldots, p_n\}$ such that they sum to 1. Each $p_i$ is the probability of $s_i$ being the outcome of the experiment, i.e., $Pr[s_i] = p_i$.*

**Definition 10 (Discrete random variable)** *Let $S$ be a discrete sample space with discrete probability distribution $P$. A discrete random variable $X$ is a function $X : S \to \mathbb{R}$ that maps each element of the sample space $s_i$ to a real number $X(s_i) = x_i$. If we do not care about the sample space but only about the value of the random variable we can write $X = x_i$.*

**Definition 11 (Ensemble of discrete random variables)** *An ensemble of discrete random variables is a set of indexed random variables $E = \{X_n\}_{n\in\mathbb{N}}$.*

We are ready to introduce three notions of indistinguishability between ensembles of random variables: perfect, statistical and computational indistinguishability, where perfect indistinguishability is the strongest one and computational indistinguishability is the weakest one.

**Definition 12 (Indistinguishability between two ensembles)** *Let $E_1 = \{X_n\}$ and $E_2 = \{Y_n\}$ be two ensembles of discrete random variables, such that $X_n$ and $Y_n$ have a common discrete sample space $S_n$ for all $n$. Then $E_1$ and $E_2$ are:*

***perfectly indistinguishable*** *if, for all $n$, $X_n$ and $Y_n$ have the same discrete probability distribution $P_n$.*

***statistically indistinguishable*** *if for every polynomial $p(\cdot)$ and sufficiently large $n$:*

$$\sum_{s\in S_n} |Pr[X_n(s)] - Pr[Y_n(s)]| \leq \frac{1}{p(n)}$$

***computationally indistinguishable*** *if no efficient algorithm exists that can distinguish them with non-negligible probability, i.e., for every probabilistic polynomial-time algorithm $D$, for every polynomial $p(\cdot)$ and sufficiently large $n$*

$$|Pr[D(X_n) = 1] - Pr[D(Y_n) = 1]| \leq \frac{1}{p(n)}$$

Note that by defining the discrete sample space as $S_n \subseteq \{0,1\}^{q(n)}$, where $q(n)$ is a polynomial, the distribution can be parameterized according to the input length or security parameter of an algorithm.

The notation we use to describe probabilities is based on the one proposed by Goldwasser, Micali and Rivest in [GMR88]. Let $A(input_{A_1}, \ldots, input_{A_n})$ be a probabilistic algorithm that receives $(input_{A_1}, \ldots, input_{A_n})$ as inputs. Then $(output_{A_1}, \ldots, output_{A_m}) \leftarrow A(input_{A_1}, \ldots, input_{A_n})$ denotes that algorithm $A$ outputs values $(output_{A_1}, \ldots, output_{A_m})$ according to its strategy. Finally, the following notation:

$$Pr[(output_{A_1}, \ldots, output_{A_m}) \leftarrow A(input_{A_1}, \ldots, input_{A_n});$$
$$(output_{B_1}, \ldots, output_{B_r}) \leftarrow B(input_{B_1}, \ldots, input_{B_s});$$
$$\ldots : p(\cdot, \cdot, \ldots)] = x$$

means that predicate $p(\cdot, \cdot, \ldots)$, in which some of the outputs of the algorithms form part of its inputs, is true with probability $x$ after the (ordered) execution of algorithms $(A, B, \ldots)$.

## 2.5 Abstract Algebra

Abstract algebra is a field that includes several algebraic objects such as groups, rings, fields or vector spaces. In this section we focus on groups because those are the objects we utilize in our constructions. First, we define the concept of group and of cyclic group and then we explain its properties. Second, we explain several important results related with groups and the concept of subgroup, and finally we explain more in detail characteristics of the groups that we use in our constructions.

To start with, before defining groups it is necessary to explain the concept of binary operation on a set and its properties.

**Definition 13 (Binary operation on a set and properties)** *A binary operation $*$ on a set $S$ is a mapping from $S \times S$ to $S$, i.e., it is a rule that assigns to each ordered pair of elements from $S$ an element of $S$. A binary operation can have several properties defined as follows:*

**Associative** *For all $a, b, c \in S$, $(a * b) * c = a * (b * c)$.*

**Commutative** *For all $a, b \in S$, $a * b = b * a$.*

**Identity element** *An element $e$ is the identity element if for all $a \in S$, $a * e = e * a = a$.*

**Inverse** *An inverse of an element $a \in S$ is an element $b \in S$ such that $b * a = a * b = e$.*

In this definition, the inverse element $b$ is usually referred to as $-a$ and the identity element $e$ as 0 if additive notation is used. With multiplicative notation, the inverse element is $a^{-1}$ and the identity element 1. Now we can define the concept of group and different kinds of groups:

**Definition 14 (Group)** *A group $(G, *)$ is a set $G$ with an associative binary operation $*$ on $G$ satisfying that $G$ contains an identity element for $*$ and that for every element of $G$ there exists an inverse element under $*$. If $*$ is commutative, the group is called abelian or commutative. A group is finite if $|G|$ is finite. The number of elements of a finite group is called its order.*

From now on we refer to the group $(G, *)$ as $G$ since there is no risk of ambiguity. Just to make clear what a group is, we give some examples: the set of integers $\mathbb{Z}$ with the operation of addition or the set $\mathbb{Z}_n$ with the operation of addition modulo $n$ are groups. On the other hand, the set $\mathbb{Z}_n$ with the operation of multiplication modulo n is not a group because not all the elements have inverses.

A kind of group very important for cryptography is the cyclic group. To give this definition, first we explain what the order of a specific element of a group is.

**Definition 15 (Order of an element)** *The order of an element $a \in G$ is the least positive integer $t$ such that $a^t = 1$. If $t$ does not exist the order of $a$ is infinite.*

**Definition 16 (Cyclic group)** *A group $G$ is cyclic if there exists an element $a \in G$ such that for every element $b \in G$, $b = a^i$ for some $i \in \mathbb{Z}$. The element $a$ is called a generator of $G$, and $G = \langle a \rangle$ means that $a$ generates $G$.*

*An element $a \in G$ with order $n$ is a generator of $G$ if $n = |G|$. If $a$ is generator, an element $b = a^i$ has order $n/gcd(n, i)$. Therefore, if $gcd(n, i) = 1$ the order of $b$ is $n$ and $b$ is a generator of $G$.*

Thus, if the order of a group $G$ is prime then all the elements of the group, except 1, are generators. For a given group $G$ of order $n$ we can calculate the number of generators by means of the Euler $\phi$-function (also Euler totient function).

**Definition 17 (Euler $\phi$-function)** *For $n \geq 1$, let $\phi(n)$ denote the number of positive integers $a$ smaller than $n$ such that they are relatively prime to $n$ (i.e., $gcd(n, a) = 1$). The function $\phi$ is called the Euler $\phi$-function. If $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$ is the factorization of $n$, then $\phi(n) = n \prod_{i=1}^{k} (1 - p_i^{-1})$.*

Second, we define the concept of subgroup. We also define the relation between the order of the subgroup and the order of the group that contains the subgroup, and at the end we state that the order of every element of a group divides the order of the group.

**Definition 18 (Subgroup)** *A non-empty subset $H$ of $G$ is a subgroup of $G$ if $H$ is itself a group with respect to the operation of $G$. If $G$ is cyclic then $H$ is also cyclic.*

**Definition 19 (Lagrange's Theorem)** *If $G$ is a finite group and $H$ is a subgroup of $G$, then $|H|$ divides $|G|$.*

Therefore, if $a \in G$ has order $t$, then the order of the subgroup generated by $a$, $|\langle a \rangle|$, is $t$, and thus the order of every element of a group divides the order of the group. Furthermore, for every positive divisor $d$ of $n$, G contains exactly one subgroup of order $d$ and $\phi(d)$ elements of order $d$.

Finally, we are ready to explain which groups are used in our constructions and their characteristics. These groups are the group of integers modulo a prime number and the group of integers modulo a composite n.

**Definition 20 (Group and subgroups modulo a prime p)** *The remainders modulo p form a cyclic multiplicative group $\mathbb{Z}_p^*$. The order of this group is $p-1$.*

There exists a cyclic group $G_t$ of order $t$ for every divisor $t|p-1$ of $\mathbb{Z}_p^*$. This group is particularly interesting for $t = q$ with $q$ a large prime. Algorithms for creating the modulus and generators for such groups can be found in [MvOV96].

To introduce the concept of group modulo a composite $n$ we should first explain the concept of special RSA modulus.

**Definition 21 (Special RSA modulus)** *A special RSA modulus n is the product of two primes $p = 2p' + 1$ and $q = 2q' + 1$. A prime $p'$ for which a corresponding prime $p = 2p' + 1$ exists is called Sophie Germain prime.*

**Definition 22 (Groups and subgroups modulo a composite n)** *The subgroup $QR_n$ of the group $\mathbb{Z}_n^*$, where $n = pq$ is a special RSA modulus, is a cyclic group under multiplication denoting the quadratic residues modulo n. The order of this group is $p'q'$ and therefore $p'q'$ divides $\phi(n)$.*

## 2.6 Number-Theory Problems

In this section we show the problems whose intractability ensures the security of our constructions. First of all, we define the discrete logarithm (DL) problem, which has been used to prove the security of Pedersen commitments (see Section 2.8.2) and also of proofs of knowledge about discrete logarithms (see Section 2.9.3). After that we define the computational Diffie-Hellman (CDH) and the decision Diffie-Hellman (DDH) problems in order to introduce the concept of gap Diffie-Hellman (GDH) group, which is interesting for us because bilinear pairings groups (see Section 2.7) are GDH groups. Finally, we also mention the RSA problem and some related assumptions that are used when proving the security of those constructions that utilize the group of quadratic residues modulo a special RSA modulus $n$.

Therefore, let $Setup(1^k)$ be an algorithm that on input a security parameter $k$ outputs a cyclic group $G_q$ of prime order $q$ and a generator $g$ of $G_q$.

**Definition 23 (DL)** *Let $x \in \mathbb{Z}_q$ be randomly chosen. Given $(g, g^x)$ the discrete logarithm problem is to compute $x$. Formally, the DL assumption holds if there exists a negligible function $\nu$ such that:*

$$\Pr[(q, G_q, g) \leftarrow Setup(1^k); x \leftarrow \mathbb{Z}_q; z \leftarrow A(g, g^x) : z = x] < \nu(k).$$

**Definition 24 (CDH)** *Let $a, b \in \mathbb{Z}_q$ be randomly chosen. Given $(g, g^a, g^b)$ the computational Diffie-Hellman problem is to compute $g^{ab}$. Formally, the CDH assumption holds if there exists a negligible function $\nu$ such that:*

$$\Pr[(q, G_q, g) \leftarrow Setup(1^k); a, b \leftarrow \mathbb{Z}_q; z \leftarrow A(g, g^a, g^b) : z = g^{ab}] < \nu(k).$$

**Definition 25 (DDH)** *Let $a, b, c \in \mathbb{Z}_q$ be randomly chosen. Given $(g, g^a, g^b, g^{ab})$ and $(g, g^a, g^b, g^c)$ the decision Diffie-Hellman problem is to distinguish between them. Formally, the DDH assumption holds if there exists a negligible function $\nu$ such that:*

$$\begin{aligned}\Pr[(q, G_q, g) \leftarrow Setup(1^k); a, b, c \leftarrow \mathbb{Z}_q; z_0 \leftarrow g^{ab}; z_1 \leftarrow g^c;\\ b \leftarrow \{0,1\}; b' \leftarrow A(g, g^a, g^b, z_b) : b = b'] < 1/2 + \nu(k).\end{aligned}$$

Nowadays, the best known algorithm that solves the DL problem runs in subexponential time (see Section 2.2).

The CDH problem is not harder than the DL problem because you can solve it if you can solve the DL problem. Namely, you can get $a, b$ from $(g, g^a, g^b)$ and compute $g^{ab}$. Therefore, the DL problem is hard if the CDH problem is hard. However, it is not known if the hardness of the DL problem guarantees the hardness of the CDH problem [MW99], although they are widely believed to be equivalent and for some classes of groups it has been proved that they are [MW98].

Like in the previous case, the DDH problem is not harder than the CDH problem because if you can compute $g^{ab}$ from $(g, g^a, g^b)$, immediately you can distinguish between $(g, g^a, g^b, g^{ab})$ and $(g, g^a, g^b, g^c)$. Thus the hardness of the DDH problem guarantees the hardness of the CDH problem. Conversely, for most groups it is not clear if the DDH problem is easier than the CDH problem. The groups that fulfill this property are called gap Diffie-Hellman (GDH) groups. For these groups a variant of the CDH and the DDH problems is defined:

**Definition 26 (GDH)** *Let $a, b \in \mathbb{Z}_q$. Given $(g, g^a, g^b)$, the gap Diffie-Hellman problem is to solve the CDH problem for this tuple with the help*

*of a DDH oracle. Formally, the GDH assumption holds if there exists a negligible function $\nu$ such that:*

$$\Pr[(q, G_q, g) \leftarrow Setup(1^k); a, b \leftarrow \mathbb{Z}_q; z \leftarrow A(g, g^a, g^b)^{\leftrightarrow Oracle_{DDH}(\cdot,\cdot,\cdot,\cdot)} : z = g^{ab}] < \nu(k).$$

In this definition, $Oracle_{DDH}(\cdot, \cdot, \cdot, \cdot)$ can be queried as many times as needed by the adversary. It receives an instance $(g, g^a, g^b, z)$ of the DDH problem and outputs whether $z = g^{ab}$ or not.

Finally, we define the RSA problem and the flexible RSA problem and their related assumptions, the RSA assumption and the strong RSA assumption (see [RSA78]). They are used to prove the security of commitment schemes and proofs of knowledge that use groups of hidden order (see Section 2.8.3 and Section 2.9.3).

Let $Setup(1^k)$ be an algorithm that on input a security parameter $k$ outputs an RSA modulus $n$ and its corresponding group $Z_n^*$.

**Definition 27 (RSA)** *Given integers $v$ and $x$ such that $v \in \mathbb{Z}_n^*$ and $2 < x < n$, where $x$ is coprime to $\phi(n)$, the RSA problem consists in finding $u \in Z_n^*$ such that $u^x = v \bmod n$. Formally, the RSA assumption holds if there exists a negligible function $\nu$ such that:*

$$\Pr[(n, Z_n^*, x) \leftarrow Setup(1^k) \wedge x \in \{3, \ldots, n-1\} \wedge gcd(x, \phi(n)) = 1; v \leftarrow \mathbb{Z}_n^*; z \leftarrow A(v, x) : z^x = v \bmod n] < \nu(k).$$

**Definition 28 (Flexible RSA)** *Given $v \in \mathbb{Z}_n^*$, the flexible RSA problem consists in finding $u \in Z_n^*$ and $2 < x < n$, where $x$ is coprime to $\phi(n)$, such that $u^x = v \bmod n$. Formally, the strong RSA assumption holds if there exists a negligible function $\nu$ such that:*

$$\Pr[(n, Z_n^*) \leftarrow Setup(1^k); v \leftarrow \mathbb{Z}_n^*; (z_1, z_2) \leftarrow A(v) : z_1^{z_2} = v \bmod n \wedge z_2 \in \{3, \ldots, n-1\} \wedge gcd(z_2, \phi(n)) = 1] < \nu(k).$$

Note that the strong RSA assumption is stronger than the RSA assumption in the sense that solving the flexible RSA problem may be easier than solving the RSA problem, because $x$ can be chosen in a way that depends on $v$.

## 2.7 Bilinear Pairings

The concept of pairing-based cryptography has increased in importance during the last years, mainly because of its use in Identity-Based Encryption

(see Section 3.1). It can also be used to build other applications such as key agreement protocols [Jou00] or signature schemes [BLS01].

In a nutshell, a pairing is a map between two cryptographic groups that allows us to construct new cryptographic schemes based on the reduction of one problem in one group to another (usually easier) problem in the other group. The two known constructions for pairings (the Tate and Weil pairings) involve complex mathematics, but a lot of cryptographic schemes have been designed by using an abstract notation for pairings, so we describe pairings following this notation.

First, we give a definition of the concept of bilinear map. Then, we describe the security assumption under which a lot of schemes built by using bilinear maps are secure, i.e., the bilinear Diffie-Hellman assumption, and finally we show the assumptions that we utilize in our constructions.

### 2.7.1 Definition

A bilinear map involves three groups, namely, $G_1, G_2$ and $G_T$ of prime order $q$, where $G_1$ and $G_2$ are commonly referred to as gap groups (see Section 2.6), and $G_T$ as the target group. A bilinear map is called symmetric if $G_1 = G_2$, and asymmetric otherwise. In order to give a general view, we utilize an asymmetric one in the definition below, but it can be easily particularized to symmetric pairings. Although usually the group operation is very different from the arithmetic multiplication, it is common to use multiplicative notation for both the gap group and the target group (sometimes, additive notation is used for the gap group).

Therefore, consider groups $G_1, G_2, G_T$ of prime order $q$ and two generators $g \in G_1$ and $h \in G_2$. A bilinear map $e$ is defined as a function:

$$e : G_1 \times G_2 \rightarrow G_T$$

with the following properties:

**Bilinearity** Given $a, b \in \mathbb{Z}_q$, $e(g^a, h^b) = e(g, h)^{ab}$

**Non-Degeneracy** $\forall g \in G_1, g \neq 1$, and $h \in G_2, h \neq 1, < e(g, h) >= G_T$, i.e., $e(g, h) \neq 1$

**Efficiency** $e$ should be easily computable. This property is needed if we want the map to be useful to construct cryptographic schemes.

Usually, $G_1$ and $G_2$ are elliptic-curve groups and $G_T$ is a finite field.

For practical constructions there will be an efficient algorithm *BilinearSetup*$(1^k)$ such that, on input a security parameter $k$, it outputs a bilinear instance $(q, G_1, G_2, G_T, g, h, e)$.

### 2.7.2 Bilinear Diffie-Hellman Assumption

In Section 2.6 we defined the concept of gap Diffie-Hellman (GDH) groups and the GDH problem. Now we show that GDH groups can be realized with pairings. Namely, given $e : G_1 \times G_2 \to G_T$ that fulfills the properties mentioned in the previous subsection, where $G_1$ and $G_2$ are groups in which solving the computational Diffie-Hellman (CDH) problem is supposed to be hard, it is easy to solve the decision Diffie-Hellman (DDH) problem for the tuples $(g, g^a, h^b, h^{ab})$ and $(g, g^a, h^b, h^c)$ as follows:

1. Compute $e(g^a, h^b) = e(g, h^{ab})$.

2. Check $e(g, h^{ab}) \stackrel{?}{=} e(g, h^c)$.

Therefore, since the DDH problem is easy in either $G_1$ or $G_2$, either of those groups is a GDH group.

In addition, the pairings allow us to construct new cryptographic schemes based on the possibility of moving a secret exponent from one coordinate of the pairing to the other one, thanks to the bilinearity property. The security of these schemes relies on a different problem called the bilinear Diffie-Hellman (BDH) problem. Let $(q, G_1, G_2, G_T, g, h, e)$ be a bilinear instance. Then the BDH problem is defined as follows:

**Definition 29 (BDH)** *Let $a, b, c \in \mathbb{Z}_q$. Given $(g, g^a, g^b, g^c) \in G_1$ and $(h, h^a, h^b, h^c) \in G_2$ the bilinear Diffie-Hellman problem is to compute $e(g, h)^{abc} \in G_T$. Formally, the BDH assumption holds if there exists a negligible function $\nu$ such that:*

$$\Pr[(q, G_1, G_2, G_T, g, h, e) \leftarrow \mathit{BilinearSetup}(1^k); a, b, c \leftarrow \mathbb{Z}_q; \\ z \leftarrow A(g, h, g^a, g^b, g^c, h^a, h^b, h^c) : z = e(g, h)^{abc}] < \nu(k).$$

The BDH problem is not harder than the CDH problem in $G_1$, in $G_2$ and in $G_T$. Namely, let $g_t = e(g, h)$ be a generator of $G_T$. In both $G_1$ and $G_2$, if we can solve CDH we can compute $g^{ab}$ and then $e(g^{ab}, h^c) = e(g, h)^{abc}$, and in $G_T$, from $g_t^a = e(g^a, h)$ and $g_t^{bc} = e(g^b, h^c)$ we can compute $g_t^{abc} = e(g, h)^{abc}$. Conversely, it is widely believed that the hardness of CDH guarantees the hardness of BDH, although it remains an open problem to demonstrate it. Cheon and Lee [CL] have shown that if the pairing is weak-invertible, which means that from a generator $g_t$ of $G_T$ is easy to compute an inverse image $(g, h)$ of generators of $G$ such that $e(g, h) = g_t$, then the BDH problem is equivalent to the CDH problem, but for the modified Weil and Tate pairings it is not known whether they are weak-invertible or not (see [Maa04]).

### 2.7.3 Other Assumptions

Since bilinear pairings appeared in cryptography [Jou00], they have been used in a large variety of ways under many different complexity assumptions with different strength. We distinguish between two types: static and dynamic assumptions. A complexity assumption is called static if its problem instances are stated non-interactively (i.e., independently of the number of adversarial queries) and dynamic if it is not the case.

For the construction presented in Chapter 4, we need the following two static complexity assumptions, the decision bilinear Diffie-Hellman (DBDH) assumption and the decision linear assumption (see [BW06]).

**Definition 30 (DBDH)** *Given* $(g, g^a, g^c, h, h^a, h^b, Z) \in G_1^3 \times G_2^3 \times G_T$ *for random exponents* $a, b, c \in \mathbb{Z}_q$*, decide whether* $Z = e(g,h)^{abc}$*. Formally, the DBDH assumption holds if there exists a negligible function* $\nu$ *such that:*

$$\begin{aligned}\Pr[&(q, G_1, G_2, G_T, g, h, e) \leftarrow BilinearSetup(1^k); a, b, c \leftarrow \mathbb{Z}_q;\\ &z_0 \leftarrow e(g,h)^{abc}; z_1 \leftarrow G_T; b \leftarrow \{0,1\};\\ &b' \leftarrow A(g, g^a, g^c, h, h^a, h^b, z_b) : b = b'] < 1/2 + \nu(k).\end{aligned}$$

**Definition 31 (Decision Linear)** *Given* $(g, g^a, g^b, g^{ac}, g^{bd}, h, h^a, h^b, Z) \in G_1^5 \times G_2^3 \times G_T$ *for random exponents* $a, b, c, d \in \mathbb{Z}_q$*, decide whether* $Z = g^{c+d}$*. Formally, the Decision Linear assumption holds if there exists a negligible function* $\nu$ *such that:*

$$\begin{aligned}\Pr[&(q, G_1, G_2, G_T, g, h, e) \leftarrow BilinearSetup(1^k); a, b, c, d \leftarrow \mathbb{Z}_q;\\ &z_0 \leftarrow g^{c+d}; z_1 \leftarrow G_1; b \leftarrow \{0,1\};\\ &b' \leftarrow A(g, g^a, g^b, g^{ac}, g^{bd}, h, h^a, h^b, z_b) : b = b'] < 1/2 + \nu(k).\end{aligned}$$

Finally, for the scheme presented in Chapter 5, we use two dynamic complexity assumptions, the strong Diffie-Hellman assumption [BB04b] and the power decisional Diffie-Hellman assumption [CNS07].

**Definition 32 (l-SDH)** *Given* $(g, g^x, \ldots, g^{x^l}) \in G_1^{l+1}$*, where* $x \in \mathbb{Z}_q$*, output a pair* $(c, g^{1/(x+c)})$*, where* $c \in \mathbb{Z}_q$*. Formally, the l-SDH assumption holds if there exists a negligible function* $\nu$ *such that:*

$$\begin{aligned}\Pr[&(q, G_1, G_2, G_T, g, h, e) \leftarrow BilinearSetup(1^k); x \leftarrow \mathbb{Z}_q;\\ &(z_1, z_2) \leftarrow A(g, g^x, \ldots, g^{x^l}) : (z_1, z_2) = (c, g^{1/(x+c)}) \wedge c \in \mathbb{Z}_q] < \nu(k).\end{aligned}$$

**Definition 33 (l-PDDH)** *Given* $(g, g^x, \ldots, g^{x^l}, H) \in G_1^{l+1} \times G_T$*, where* $x \in \mathbb{Z}_q$*, distinguish between the vector* $(H^x, H^{x^2}, \ldots, H^{x^l}) \in G_T^l$ *and a random vector* $T \in G_T^l$*. Formally, the l-PDDH assumption holds if there exists*

*a negligible function $\nu$ such that:*

$$\Pr[(q, G_1, G_2, G_T, g, h, e) \leftarrow BilinearSetup(1^k); x \leftarrow \mathbb{Z}_q; H \leftarrow G_T;$$
$$T_0 \leftarrow (H^x, H^{x^2}, \ldots, H^{x^l}); T_1 \leftarrow G_T^l; b \leftarrow \{0,1\};$$
$$b' \leftarrow A(g, g^x, \ldots, g^{x^l}, H, T_b) : b = b'] < 1/2 + \nu(k).$$

## 2.8 Commitment Schemes

Commitment schemes are a basic tool used in many cryptographic protocols such as zero-knowledge proofs and multi-party computation protocols. Although the concept was formalized in 1988 [BCC88], it had been used before. There are several ways to construct commitment schemes: by using one-way functions, pseudorandom generators, etc. In this section, first we explain what a commitment scheme is and we show its properties. After that, we focus on Pedersen commitments and on commitment schemes for groups of hidden order because those are the ones used later on in our constructions.

### 2.8.1 Definition

A commitment scheme is a method that allows a party to commit to a value while keeping it hidden, preserving the possibility that this party can reveal the value later. The interaction takes place in two phases. During the commit phase the sender gives a commitment to the value and the receiver stores it. After that, during the reveal phase the sender allows the receiver to get the value and check it. More formally, a generic commitment scheme can be defined as follows:

**Definition 34 (Commitment scheme)** *A commitment scheme consists of a set of three polynomial time randomized algorithms:*

**SetupCommit**$(1^k)$**.** *On input a security parameter $k$, it outputs $params$, the parameters of the scheme.*

**Commit**$(params, m, open_m)$**.** *On input a message $m$ and a random string $open_m$, it outputs a commitment $C$.*

**Open**$(params, C, m', open_{m'})$**.** *On input a message $m'$, a random string $open_{m'}$ and a commitment $C$, it outputs accept if $C = Commit(params, m', open_{m'})$ and rejects otherwise.*

Therefore, during the commit phase the sender utilizes *Commit* to compute a commitment $C$ and he sends it to the receiver, and in the reveal phase the sender sends values $m$ and $open_m$ and the receiver uses *Open* to check the validity of the commitment $C$. The main idea is that between the

two phases the receiver cannot get the value and the sender cannot modify the commitment.

This is formalized by the two properties of a commitment scheme: the hiding property and the binding property. The hiding property ensures that the receiver cannot get the value from the commitment, while the binding property states that when the receiver opens the commitment she can only get a unique value fixed during the commit phase. Formally speaking, these properties can be defined as follows:

**Definition 35 (Hiding property)** *Even when the adversary chooses the input distribution $D$ for the message $m$, he learns no more information about this value than what he can learn from the input distribution itself. In other words, there exist a simulator $S$ for every adversary $A$ and a negligible function $\nu$ such that:*

$$\begin{aligned}\Pr[&params \leftarrow SetupCommit(1^k); (u, D) \leftarrow A(k, params); open_m \leftarrow \{0,1\}^k;\\ &m \leftarrow D; C_0 \leftarrow Commit(params, m, open_m); C_1 \leftarrow S(params, D);\\ &b \leftarrow \{0,1\}; b' \leftarrow A(u, C_b) : b' = b] \leq 1/2 + \nu(k).\end{aligned}$$

**Definition 36 (Binding property)** *The adversary cannot open a commitment in two different ways, i.e., for all $A$ there exists a negligible function $\nu$ such that:*

$$\begin{aligned}\Pr[&params \leftarrow SetupCommit(1^k);\\ &(C, m_0, open_{m_0}, m_1, open_{m_1}) \leftarrow A(params);\\ &a = Open(params, C, m_0, open_{m_0}); b = Open(params, C, m_1, open_{m_1}) :\\ &a = accept \wedge b = accept] < \nu(k)\end{aligned}$$

It has been demonstrated that, if a commitment scheme is information theoretically hiding, then it can be at most computationally binding, and vice versa, if it is information theoretically binding then it is computationally hiding. Another property related with the hiding property is semantic security, which means that the receiver cannot distinguish between two commitments to different values even if she knows the values.

**Definition 37 (Semantic security)** *For all $A$ there exists a negligible function $\nu$ such that:*

$$\begin{aligned}\Pr[&params \leftarrow SetupCommit(1^k); (m_0, m_1, state) \leftarrow A(params);\\ &b \leftarrow \{0,1\}; open_m \leftarrow \{0,1\}^k; C_b = Commit(params, m_b, open_m);\\ &b' \leftarrow A(state, C_b) : b' = b] \leq 1/2 + \nu(k)\end{aligned}$$

### 2.8.2 Pedersen Commitment Scheme

In this scheme, the public parameters are a cyclic group $G_q$ of prime order $q$ and two generators $g$ and $h$ of $G_q$, such that nobody knows $log_g(h)$. Pedersen [Ped91] proposed to use a trusted center or to run a coin-flipping protocol between the parties to assure this, but for some protocols it is sufficient if nobody except the verifier knows $log_g(h)$. With these parameters, the sender commits himself to a value $m \in \mathbb{Z}_q$ by choosing at random a value $open_m \in \mathbb{Z}_q$ and computing:

$$Commit(params, m, open_m) = g^m h^{open_m}$$

As $open_m$ is chosen at random, $Commit(params, m, open_m)$ is uniformly distributed in $G_q$. Therefore, the scheme is information theoretically hiding and semantically secure. The binding property is assured under the assumption that computing discrete logarithms is hard. Pedersen proved that the sender cannot make the receiver open the commitment to another value $m' \neq m$ unless he can compute $log_g(h)$.

The scheme can be turned into one that is computationally hiding and unconditionally binding by computing $C = Commit(params, m, open_m) = (g^m h^{open_m}, g^{open_m})$, which is very similar to the ElGamal encryption scheme [Gam84].

### 2.8.3 Commitment Scheme for Groups of Hidden Order

We present here an instance of a class of commitment schemes due to Damgård and Fujisaki [DF02], which is a correction and generalization of the scheme proposed by Fujisaki and Okamoto [FO97]. It allows to commit to arbitrary size integers and is based on commutative groups that fulfill certain properties, being the most important that computing the order of the group is hard for the committer.

Damgård and Fujisaki stated the conditions that any group should fulfill in order to be used in this scheme; they also said that the cyclic group of quadratic residues modulo a special RSA modulus $n$, i.e., $QR_n$, can be used. We use this group, which implies that the scheme is secure under the strong RSA assumption. The construction works as follows:

**SetupCommit**$(1^k)$ This algorithm can be run by the verifier or by a trusted third party. It outputs the parameters $params = (n, g, h)$ of the scheme, where $n$ is a special RSA modulus and $g$ and $h$ are generators of $QR_n$. The verifier has to prove that $g \in \langle h \rangle$, which can be done by using the Schnorr identification protocol (see Section 2.9.3).

**Commit**$(params, m, open_m)$ On input the parameters of the scheme, an integer $m$ and randomness $open_m \in [0..\lfloor \frac{n}{4} \rfloor]$, it computes the commit-

ment $C = g^m h^{open_m} \bmod n$. Here $\lfloor \frac{n}{4} \rfloor$ is a good upper bound for the order of the group, as required in [DF02].

**Open**$(params, C, m', open_{m'}, b)$ On input the commitment C and values $(m', open_{m'}, b)$, it outputs accept if $C = g^m h^{open_{m'}} b$, where $b^2 = 1$, and reject otherwise. An honest prover can always use $b = 1$.

This commitment scheme is statistically hiding and semantically secure. It is computationally binding if the strong RSA assumption holds.

## 2.9 Proofs of Knowledge

Roughly speaking, a proof of knowledge is a two-party protocol between a prover and a verifier by means of which the verifier is convinced of the validity of a statement, e.g., that the prover knows some secret values. Proofs of knowledge are a fundamental tool used to provide privacy and security in our protocols.

In this section we start by giving a definition of proof of knowledge along with its properties. In Section 2.9.2 we define the zero-knowledge property, which makes it possible that the prover convince the verifier without the verifier learning anything apart from the validity of the proof. Finally, in Section 2.9.3 we focus on different kinds of proofs that are used later on in our constructions.

### 2.9.1 Definition

Traditionally, a proof of a statement $S$ is a sequence of statements such that, when interpreted, they lead to the validity of the proof. Anyone who is able to interpret the statements can verify the proof and send it to other parties in order to convince them of the validity of the statement $S$.

However, an interactive proof of a statement is a protocol between two parties, a prover and a verifier, by means of which the verifier is convinced of the validity of the statement. Interactive proofs are more powerful because there are statements that can be proven interactively but not with a conventional proof [Sha92]. Apart from that, in zero-knowledge protocols, the verifier only learns that the statement is valid and thus she cannot prove it to other parties. On the other hand, interactive proofs are probabilistic, i.e., they are valid with probability close to one.

The concept of interactive proof systems was defined in [Bab85], [BM88] and [GMR85]. [GMR85] also proposed proofs of knowledge although they did not give a formal definition. After that, definitions of proofs of knowledge were proposed, for instance, by [BCC88], [BC86], [Cha86] and [FFS88]. In these definitions, the prover is a polynomial-time machine, whereas in

the definitions given for interactive proof systems he is computationally unbounded.

Nevertheless, in 1992 Bellare and Goldreich [BG92] proposed a definition of proof of knowledge in which the prover is computationally unbounded and in which they solved other shortcomings of the definitions mentioned above. Before introducing their definition we define some concepts utilized in it.

**Definition 38 (Polynomially bounded relation)** *Let $R \subset \{0,1\}^* \times \{0,1\}^*$ be a relation. It is polynomially bounded if there exists a polynomial $p$ such that for all $(x,w) \in R$, $|w| \leq p(|x|)$. If $(x,w) \in R$, then $w$ is called the witness for $x$. $L_R$ denotes $\{x \mid \exists w : (x,w) \in R\}$ and $R(x)$ is $\{w \mid \exists x : (x,w) \in R\}$.*

$L_R$ is the language and $R(x)$ is the witness set for $x$ and language $L_R$.

**Definition 39 (Knowledge error)** *The knowledge error $\kappa(x)$ is the probability that the verifier is convinced when the prover does not know a witness $w$ for $x$.*

In applications $\kappa(x)$ is small or even 0. Now we introduce the definition given in [BG92].

**Definition 40 (Proof of knowledge)** *Let $R \subset \{0,1\}^* \times \{0,1\}^*$ be a polynomially bounded relation. An interactive proof of knowledge for relation $R$ with knowledge error $\kappa$ is a two party protocol between a prover $P$ and a verifier $V$ that fulfills two properties:*

**Completeness.** *If $x \in L_R$, then $Pr[P(x) \leftrightarrow V(x) \rightarrow accept] = 1$*

**Validity (with error $\kappa$).** *There exists a knowledge extractor $K$ such that, given oracle access to $\tilde{P}$, for every $\tilde{P}$ and every $x \in L_R$, $K(x)^{\leftrightarrow \tilde{P}(x)} \in R(x) \cup \{\perp\}$ and*

$$Pr[K(x)^{\leftrightarrow \tilde{P}(x)} \in R(x)] \geq Pr[\tilde{P}(x) \leftrightarrow V(x) \rightarrow accept] - \kappa(x).$$

*Therefore, validity requires that the probability of the knowledge extractor $K$ in extracting a witness is at least the probability that $\tilde{P}$ convince the verifier, minus error $\kappa$. Given $K$, it is possible to construct a knowledge extractor $K'$ that succeeds in outputting a witness in time proportional to $1/(Pr[\tilde{P}(x) \leftrightarrow V(x) \rightarrow accept] - \kappa(x))$*

The completeness property means that if the prover and the verifier are honest, the verifier accepts the proof. On the other hand, the validity property prevents a dishonest prover from convincing an honest verifier by showing that if the dishonest prover succeeds with some probability then we can construct a knowledge extractor that with at least the same probability,

minus error $\kappa$, outputs the secret belonging to an honest prover. This definition can be easily adapted to define proofs of knowledge where the prover is a polynomial time machine. In this case, proofs of knowledge are sometimes referred to as arguments of knowledge.

**Definition 41 (Argument of knowledge)** *Let $R \subset \{0,1\}^* \times \{0,1\}^*$ be a polynomially bounded relation. An interactive argument of knowledge for relation $R$ with knowledge error $\kappa$ is a two party protocol between a polynomial time prover $P$ and a verifier $V$ that fulfills two properties:*

**Completeness.** *If $(x,w) \in R$, then $Pr[P(x,w) \leftrightarrow V(x) \rightarrow accept] = 1$*

**Validity (with error $\kappa$).** *There exists a knowledge extractor $K$ such that, given oracle access to $\tilde{P}$, for every polynomial time non-uniform $\tilde{P}$ and every $x \in L_R$, $K(x)^{\leftrightarrow \tilde{P}(x)} \in R(x) \cup \{\perp\}$ and*

$$Pr[K(x)^{\leftrightarrow \tilde{P}(x)} \in R(x)] \geq Pr[\tilde{P}(x) \leftrightarrow V(x) \rightarrow accept] - \kappa(x).$$

*Therefore, validity requires that the probability of the knowledge extractor $K$ in extracting a witness is at least the probability that $\tilde{P}$ convince the verifier, minus error $\kappa$. Given $K$, it is possible to construct a knowledge extractor $K'$ that succeeds in outputting a witness in time proportional to $1/(Pr[\tilde{P}(x) \leftrightarrow V(x) \rightarrow accept] - \kappa(x))$*

On the other hand, we note that in Definition 40 and in Definition 41 we give no requirements for the case $x \notin L_R$. In this case, the witness $w$ for $x$ and language $L_R$ does not exist. Therefore, the soundness property described below is implied by the validity property.

**Definition 42 (Soundness)** *For every $\tilde{P}$ and for all $x \notin L_R$,*

$$Pr[\tilde{P}(x) \leftrightarrow V(x) \rightarrow accept] < 1/2.$$

### 2.9.2 Zero-knowledge Property

The zero-knowledge property, introduced by [GMR85], provides security for the prover. Namely, it ensures that the verifier cannot get any information from the protocol apart from whether the prover succeeds or not. In other words, a proof of knowledge fulfills the zero-knowledge property if anything that the verifier learns from the interaction with the prover, she could have computed herself.

Formally speaking, we define this property by using the definition of [GMR85] (see Section 2.4 for definitions of indistinguishability between probability distributions).

**Definition 43 (Zero-knowledge)** *An interactive protocol between a prover $P$ and a verifier $V$ is said to be perfect, statistical or computational zero-knowledge if for every probabilistic polynomial-time verifier $\tilde{V}$ there exists a probabilistic polynomial-time simulator $S_{\tilde{V}}$ such that the two ensembles*

1. $\{P(x) \leftrightarrow \tilde{V}(x) \rightarrow .\}_{x \in L_R}$
2. $\{S_{\tilde{V}}(x) \rightarrow .\}_{x \in L_R}$

*are perfectly, statistically or computationally indistinguishable.*

This definition means that there exists a polynomial-time algorithm (the simulator), which, on input the assertions to be proven, can produce transcripts that are indistinguishable from those obtained from the interaction with the real prover.

To prove that a protocol fulfills the zero-knowledge property one needs to construct a simulator for each possible verifier. In order to allow the construction of one simulator that works for all verifiers, another definition, black-box zero-knowledge, was proposed. In this definition, there exists a simulator that can compute whatever the verifier needs to accept the proof without the verifier noticing the difference.

**Definition 44 (Black-box zero-knowledge)** *An interactive protocol between a prover $P$ and a verifier $V$ is said to be perfect, statistical or computational black-box zero-knowledge if there exists a probabilistic polynomial-time simulator $S$ such that for every probabilistic polynomial-time verifier $\tilde{V}$ the two ensembles*

1. $\{P(x) \leftrightarrow \tilde{V}(x) \rightarrow .\}_{x \in L_R}$
2. $\{S^{\leftrightarrow \tilde{V}(x)} \rightarrow .\}_{x \in L_R}$

*are perfectly, statistically or computationally indistinguishable.*

This definition is very similar to the other one; the only difference is that the quantifiers are reversed and that the simulator $S$ is given only black-box access to $\tilde{V}$. [GO94] shown that all protocols that are black-box zero-knowledge are a subset of the protocols that are zero-knowledge according to Definition 43.

### 2.9.3 Proofs of Knowledge about Discrete Logarithms

In this section we want to explain how to construct proofs of knowledge about discrete logarithms. First of all, we give a general framework to design proofs of knowledge about discrete logarithms and then we apply it to a simple example, the Schnorr's identification protocol, to show how it works. After that, we use this framework to give proofs of knowledge of

representations and proofs that show relations (e.g., equality), between the elements of a representation. Finally, we present in detail some proofs that are used later on in our protocols. This section follows the exposition given in [Koh04].

### Framework

We have a scenario with a prover $P$ and a verifier $V$. The prover has one or more secret values, usually referred to as secret keys, whose possession he wants to prove to the verifier. Both prover and verifier have as input one or more common values, usually referred to as public keys.

For the proofs given in the following sections, we consider two different algebraic settings: a finite cyclic group of prime order and the group of quadratic residues modulo a special RSA modulus.

In the former, we have a finite cyclic group $G_q$ of prime order $q$ with random generators $g, h, g_1, \ldots, g_k \in G$, such that computing discrete logarithms of any group element (apart from the identity element) with respect to one of the generators is infeasible. In the latter, we have a group $QR_n$ of order $p'q'$, where $p'$ and $q'$ are Sophie Germain primes such that $n = (2p'+1)(2q'+1)$ (see Section 2.5). We denote this group as $G_{p'q'}$, and its generators again as $g, h, g_1, \ldots, g_k \in G_{p'q'}$ (in this group, the number of non-generators, given by $p'q' - \phi(p'q')$, is negligible). For both settings, the generators should be chosen at random in order to ensure that the discrete logarithm of any generator with respect to another generator is unknown, so that the computation of a representation with respect to multiple generators is as hard as the discrete logarithm problem in $G_q$ or as hard as solving the flexible RSA problem in $G_{p'q'}$.

To represent the proofs we use the notation proposed by Camenisch and Stadler [CS97]. For example,

$$PK\{(x_1, x_2) : y_1 = g_1^{x_1} \wedge y_2 = g_1^{x_2} g_2^{x_1}\}$$

represents a proof by means of which $P$ proves knowledge of the secret keys $x_1, x_2$ such that $y_1 = g_1^{x_1}$ and $y_2 = g_1^{x_2} g_2^{x_1}$ hold. The values $y_1, y_2$ are the public keys (in [CS97] Greek characters are used for secret keys, whereas we do not). Usually public keys go to the left of the equal sign and the secret keys are the exponents of some representation on the right, but there are more possibilities.

To show how this framework works, we apply it to the Schnorr identification protocol [Sch91] with the group $G_q$. In this protocol both prover and verifier know a public key $y$ and the prover proves knowledge of a secret key $x$ such that $y = g^x$. Therefore, using the notation of [CS97],

$$PK\{(x) : y = g^x\}$$

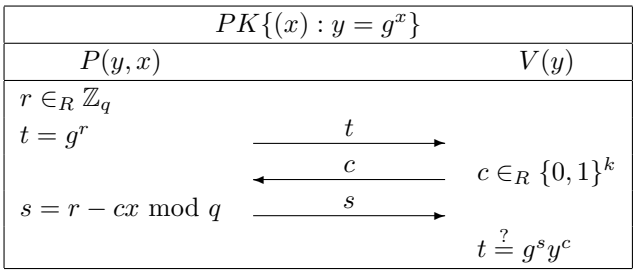


Figure 2.1: Schnorr's identification protocol

The protocol is depicted in Figure 2.1. In the first round the prover commits himself to randomness $r$ by means of the message $t = g^r$, which is called the commitment. In the second round the verifier sends a challenge $c$, and in the third round the prover computes a response $s$, which is used by the verifier to check the validity of the proof. If $t = g^s y^c$, the verifier accepts the proof. Namely, the completeness of the proof is easy to demonstrate:

$$g^s y^c = g^{r-xc} g^{xc} = g^r = t$$

The prover can compute a valid response if he knows the secret key or if he had guessed the challenge in the first round (in this case, he chooses a random response $s$ at the beginning and computes $t = g^s y^c$). The validity of the protocol has also been demonstrated. A reader interested in this demonstration and further explanations can consult [Cam98].

If we want to design proofs of knowledge for the group $G_{p'q'}$, we should note that the intervals used to take the different values of the proof change. Since the prover does not know the order of the group, he uses $\lfloor \frac{n}{4} \rfloor$ as an upper bound of the order. Therefore, the secret keys are taken from the interval $x \in [0..\lfloor \frac{n}{4} \rfloor]$. In addition, the operation $s = r - cx \ mod \ p'q'$ cannot be computed and thus the prover has to compute it in $\mathbb{Z}$. Unfortunately, now the verifier can get some knowledge about the secret key. For instance, if the challenge $c$ is large and $s$ is also large, the verifier might conclude that the secret key is small. To avoid this problem, $r$ should be taken from an interval much bigger than the domain of $x$, i.e., $r \in ]-n2^{-2+k+k'}..n2^{-2+k+k'}[$ [1], where $k'$ is used to avoid this leakage of information and $k$ is used to make the probability that a prover who does not know the witness succeed in the proof negligible. Therefore, the challenge $c$ is taken, as when using $G_q$, from the interval $\{0,1\}^k$.

All the proofs described later work like the Schnorr protocol, albeit they are more complex. All of them are three-round protocols in which the prover sends one or more commitments in the first round, the verifier submits one or more challenges in the second round and the prover replies with one or more

[1] The fact that this interval is symmetric is only a convention

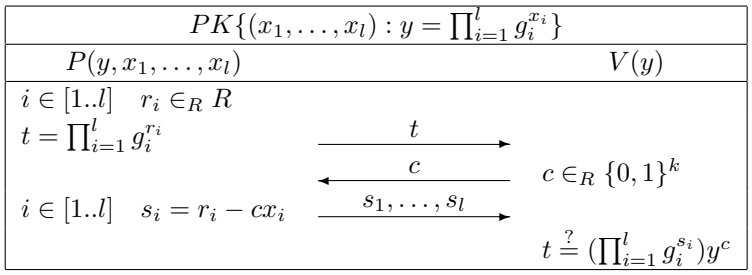


Figure 2.2: Proof of knowledge of a representation

responses in the third round. Finally, the verifier checks if the responses received were correctly computed by using the commitments obtained in the first round.

To show proofs that can be used both with $G_q$ and $G_{p'q'}$ at once, in the following we denote by $R$ the interval from which the randomness is taken, which is $\mathbb{Z}_q$ for $G_q$ and $r \in ]-n2^{-2+k+k'}..n2^{-2+k+k'}[$ for $G_{p'q'}$. We denote by $g, h, g_1, \ldots, g_k$ the generators, which can belong to $G_q$ or $G_{p'q'}$ depending on the specific case, and by $-$ the operation of substraction in $Z_q$ for $G_q$ and in $Z$ for $G_{p'q'}$.

The proofs described in the next section are not zero-knowledge, but there are techniques that make them zero-knowledge [CDM00]. In [CCT07] it is provided a theorem that avoids the necessity of stating explicit security proofs when utilizing zero-knowledge proofs for any set of discrete-log relations as a building block to construct a more complex protocol.

### Detailed Proof Examples

*Proof of knowledge of a representation.* In this setting, we have a prover with secret keys $x_1, \ldots, x_l$ and one public key $y = \prod_{i=1}^{l} g_i^{x_i}$. Therefore, we can describe the proof as follows:

$$PK\{(x_1,\ldots,x_l) : y = \prod_{i=1}^{l} g_i^{x_i}\}$$

The protocol is described in Figure 2.2. At the beginning, the prover takes one random value per secret key and computes the commitment $t$. When he receives the challenge, he computes one response per secret key and finally the verifier checks whether these responses are consistent with $t$.

As a Pedersen commitment (see Section 2.8), is a representation of values $x, open_x$ with two generators, we can use this proof of knowledge to show that one party knows the values used to compute a commitment that he has sent to another party.

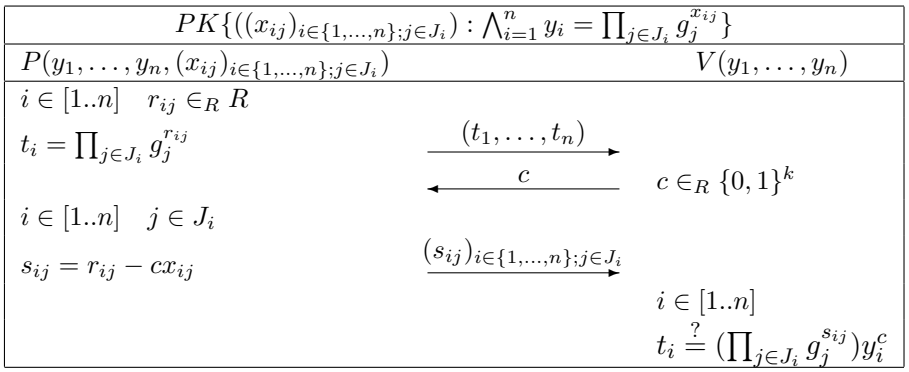


Figure 2.3: Proof of knowledge of several representations

*Proof of knowledge of several representations.* Now there are several secret keys and several public keys. One can use the previous proof for each representation, but it is possible to use another protocol in which only one challenge is needed. Having generators $g_1,\ldots,g_k$ and public keys $y_1,\ldots,y_n$, we denote by $J_i$ the set of generators used in public key $y_i$, and we denote each secret key by $x_{ij}$, where $j$, $1 \le j \le k$, indicates the generator $g_j$ used for this secret key in public key $y_i$, $1 \le i \le n$. For example, $x_{12}$ means that this secret key takes generator $g_2$ in public key $y_1$. Then we can have a public key $y_2$ with set of generators $J_2 = \{g_1, g_3, g_4\}$ and computed with secret keys $x_{21}, x_{23}, x_{24}$ as follows: $y_2 = g_1^{x_{21}} g_3^{x_{23}} g_4^{x_{24}}$.

The protocol is depicted in Figure 2.3. It works in the same way as the previous ones. First, the prover takes one random value per secret key and computes one commitment per public key. After receiving the challenge he computes one response per secret key and finally the verifier checks if the responses are consistent with the commitments.

Note that with this proof we cannot use the same secret key in several public keys. If it is needed we have to use the following proof.

*Proof of knowledge of equality and other linear relations between secret keys.* We have again a set of generators $g_1,\ldots,g_k$ and a set of public keys $y_1,\ldots,y_n$, but now we have a set of secret keys $x_1,\ldots,x_l$ and one secret key can be used in more than one representation. We denote by $J_i$ the set of generators used in public key $i$, and by $e_{ij}$, where $1 \le e_{ij} \le l$ and $j \in J_i$, that the secret key $x_{e_{ij}}$ is used in public key $y_i$ with generator $g_j$. For example, $e_{25} = e_{13} = 3$ means that the secret key $x_3$ is used in public key $y_2$ with generator $g_5$ and in public key $y_1$ with generator $g_3$.

The proof is depicted in Figure 2.4. The prover picks again one random value per secret key, but now this value is used more than once when computing the commitments if the corresponding secret key appears in more than one representation. The prover sends one commitment per public key and

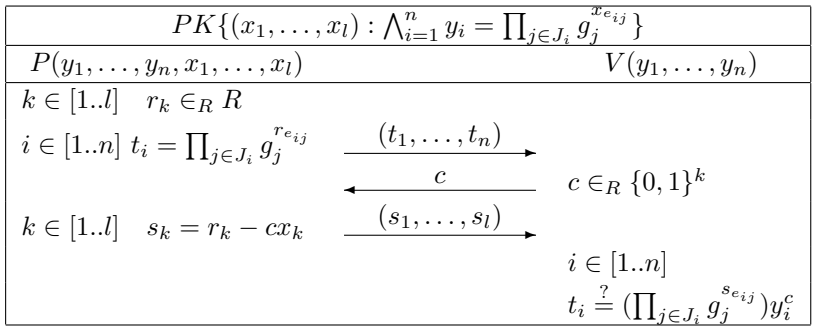


Figure 2.4: Proof of knowledge of equality of secret keys

after receiving the challenge he computes one response per secret key, using in each response the random value corresponding to a particular secret key like in previous schemes. Finally, the verifier also performs the same checks, but now she has to use one particular response in several checks if the secret key corresponding to that response appears in several representations. This proves that the prover knows this secret key and that this secret key appears in several representations.

To make it clear consider the following example:

$$PK\{(x_1,x_2,x_3) : y_1 = g_1^{x_1} g_2^{x_2} \wedge y_2 = g_1^{x_2} g_2^{x_3}\}$$

First the prover picks random values $r_1, r_2, r_3$ and computes commitments $t_1 = g_1^{r_1} g_2^{r_2}$ and $t_2 = g_1^{r_2} g_2^{r_3}$. With challenge $c$ he computes responses $s_1 = r_1 - cx_1$, $s_2 = r_2 - cx_2$ and $s_3 = r_3 - cx_3$, and finally the verifier checks that $t_1 = g_1^{s_1} g_2^{s_2} y_1^c$ and that $t_2 = g_1^{s_2} g_2^{s_3} y_2^c$. As can be seen, $s_2$ is used in both checks so the same value $x_2$ appears in public keys $y_1$ and $y_2$.

This protocol can be easily extended to prove other linear relations between secret keys, but we do not need them for our constructions.

*Proof of knowledge of multiplicative relations between secret keys.* By using the techniques shown in the previous proof one can also prove that one secret key $x_3$ is the product of two other secret keys $x_1$ and $x_2$:

$$PK\{(x_1,x_2,x_3) : y_1 = g^{x_1} \wedge y_2 = g^{x_2} \wedge 1 = y_1^{x_2}(1/g)^{x_3}\}$$

Another interesting proof, used later in our constructions, consists in proving that a committed value is the product of two committed values. Namely, having commitments $C_1 = g^{x_1} h^{open_{x_1}}$, $C_2 = g^{x_2} h^{open_{x_2}}$ and $C_3 = g^{x_3} h^{open_{x_3}}$, one can prove that $x_3 = x_1 x_2$ by running the following proof:

$$PK\{(x_1,x_2,x_3,open_{x_1},open_{x_2},open_{x_3},open_{x_1 x_2}) : C_1 = g^{x_1} h^{open_{x_1}} \wedge \\ C_2 = g^{x_2} h^{open_{x_2}} \wedge C_3 = g^{x_3} h^{open_{x_3}} \wedge 1 = C_1^{x_2}(1/g)^{x_3}(1/h)^{open_{x_1 x_2}}\}$$

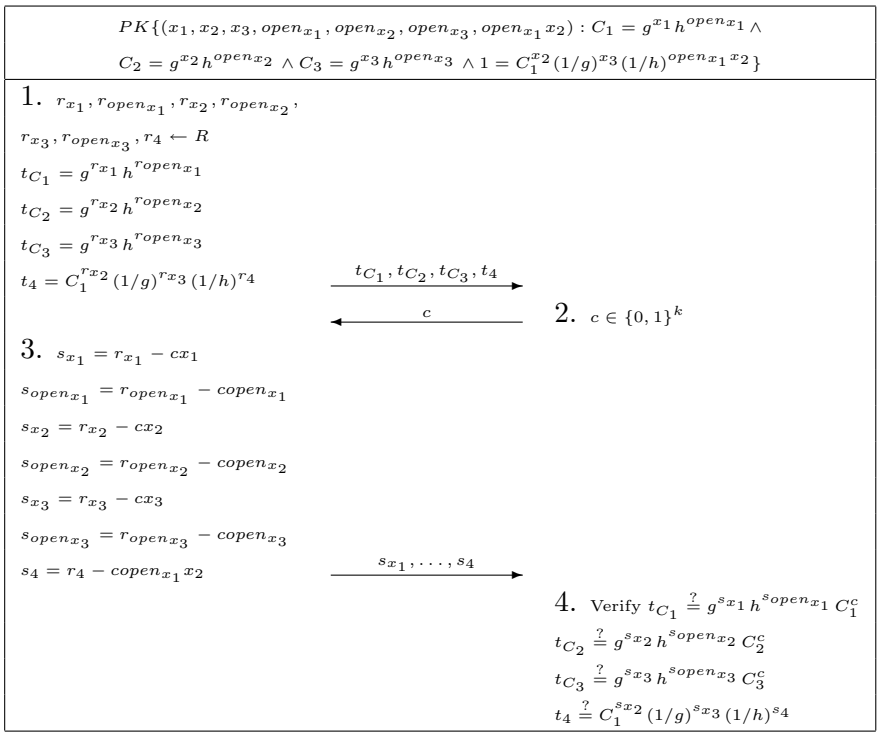


Figure 2.5: Proof of knowledge of multiplicative relation between committed values

This proof is depicted in detail in Figure 2.5. More information about proofs of multiplicative relations between secret keys can be found in [CM99].

*Proof with interval checks.* In this case, apart from proving knowledge of a secret key, the prover also shows that this secret key lies in a specific interval. A technique to implement this proof is described in [Bou00] and must be implemented using groups of hidden order. In the following we show a proof that a secret key $x \in \mathbb{Z}_q$ and a commitment $C = g^x h^{open_x}$, where $C$ is a Damgård-Fujisaki commitment (see Section 2.8.3), have the same value $x$ (by using the technique depicted in Figure 2.4), and that this value lies in an interval of the form $\Gamma = ]-2^{k+k'+l_x}..2^{k+k'+l_x}[$, where $l_x$ is such that $2^{l_x} < q2^{-k-k'-1}$. Note that $x$ must lie in a more restricted interval $\Gamma' = ]-2^{l_x}..2^{l_x}[$ to guarantee the success of the proof. Therefore, we have the following proof:

$$PK\{(x, open_x) : y = g^x \wedge C = g^x h^{open_x} \ mod\ n \wedge x \in \Gamma\}$$

Recall that $g \in G_q$, $g, h \in G_{p'q'}$ and randomness $open_x \in [0..\lfloor \frac{n}{4} \rfloor]$. The proof is depicted in Figure 2.6. First, the prover picks two random values

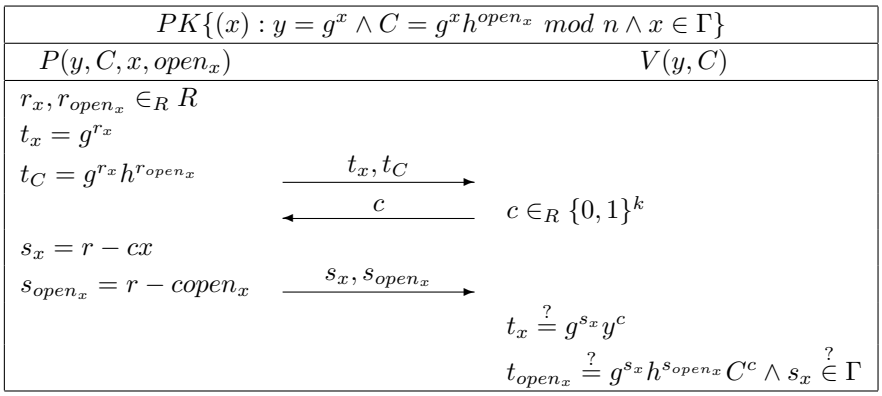


Figure 2.6: Proof with interval checks

$r_x, r_{open_x} \in R$, where $R = ]-2^{k+k'+l_x-1}..2^{k+k'+l_x-1}[$, and he sends the corresponding commitments $t_x$ and $t_C$. Upon receiving the challenge, he computes the responses $s_x, s_{open_x}$ and sends them. Finally, the verifier checks the equality of the secret key in $y$ and $C$ and that $s_x \in \Gamma$, and then she is convinced that $x \in \Gamma$. For the analysis of the proof we refer to [Bou00].

*Proof of knowledge of the inclusion of a secret key in an interval.* We want to prove that a secret key lies in some interval $[a..b]$. There are different approaches to construct this proof. We follow the one given in [Lip03]. Here, to prove that the secret key lies in an interval $[a..b]$ one has to prove that values $x - a$ and $b - x$ are non-negative. The proof that a value is non-negative can be accomplished by representing it as a sum of four squares and proving knowledge of this representation (Lagrange demonstrated that every positive number can be represented as a sum of four squares). Therefore, by applying the Rabin-Shallit algorithm [RS85] or an improved version presented in [Lip03], the prover obtains the four values $x_1, x_2, x_3, x_4$ such that $value = x_1^2 + x_2^2 + x_3^2 + x_4^2$, where $value$ is either $x - a$ or $b - x$. After that, prover and verifier run the following proof for both values:

$$PK\{(value, x_1, x_2, x_3, x_4) : y = g^{value} \wedge y_1 = g^{x_1} \wedge y_2 = g^{x_2} \wedge y_3 = g^{x_3} \wedge y_4 = g^{x_4} \wedge y = y_1^{x_1} y_2^{x_2} y_3^{x_3} y_4^{x_4}\}$$

## 2.10 Blind Signature Schemes

In this section we define signature schemes, blind signature schemes and finally unique blind signature schemes. We also explain the security definitions they should fulfill.

In order to define the concept of blind signature scheme, we first recall

the definition of a signature scheme for signing blocks of messages, which we use in Section 2.11. This definition can be easily particularized for the case in which there is only one message.

**Definition 45 (Signature scheme for blocks of messages)** *A signature scheme for blocks of messages is a set of three probabilistic polynomial time algorithms (SetupSign, Sign, VerifySign) as follows:*

**SetupSign**($1^k$)**.** *On input a security parameter $k$, it outputs a secret key $s_k$ and the parameters of the scheme params.*

**Sign**($params, s_k, M$)**.** *On input a block of messages $M = \{m_1, \ldots, m_l\}$ and the secret key $s_k$, it outputs a signature $S$ on that block of messages.*

**VerifySign**($params, S, M'$)**.** *On input the signature $S$ and the block of messages $M' = \{m'_1, \ldots, m'_l\}$, it either accepts or rejects the signature.*

The main idea behind a blind signature scheme consists in running a two-party protocol between the user and the signer in order to let the user obtain a signature on a message $m$ instead of making the signer execute the *Sign* algorithm. This protocol, *BlindSign*, is defined as follows:

**Definition 46 (BlindSign(S($params$,$s_k$),U($params$,$m$)))** *The input of the signer is the secret key $s_k$, whereas the input of the user is the message $m$. The output of the user is a signature $S$ on $m$ or $\perp$ if the protocol fails, and the output of the signer is nothing or $\perp$.*

For both definitions, correctness implies that, for all $(s_k, params)$ output by algorithm *SetupSign*, the *VerifySign* algorithm accepts on input a message $m$ and a signature $S$ on $m$ that was output either by the *Sign* algorithm or by the *BlindSign* protocol.

The security notion for a blind signature scheme involves two concepts: one-more unforgeability [PS96] and selective-failure blindness. The former means that an adversary has a negligible probability in outputting $n + 1$ valid message-signature pairs after getting the parameters of the scheme as input and after at most $n$ interactions with a signing oracle. For the latter see Definition 55.

Finally, we say that a blind signature scheme is unique if for each parameters *params* and each message $m$ there exists at most one signature $S$ such that the verification algorithm accepts [GO92].

## 2.11 Credentials

Credentials can be seen as the electronic counterparts of paper documents issued by trusted parties in order to ensure the validity of a set of statements

about an entity. However, by means of digital credentials it is possible to achieve extra features. For example, the owner of a credential can select which statements she wants to reveal when showing it, or she can prove that one or more statements fulfill certain conditions without revealing the exact values of these statements. Therefore, credentials can be used to prevent the disclosure of personal data and to protect the privacy of individuals.

In this section we define credentials and their properties, and we show how to get and show credentials. Afterwards, we describe a credential scheme based on a signature scheme proposed by Camenisch and Lysyanskaya [CL02].

### 2.11.1 Definition

The entities of a credential system are owners, issuers and verifiers of credentials. Owners obtain credentials from issuers and show them to verifiers. There are parties that can be both issuers and verifiers.

A credential is a set of certified statements about the owner of the credential. Credentials constitute a mechanism by means of which it is possible to obtain a signature on a set of statements from an issuer and prove possession of this signature to verifiers in such a way that the owner of the credential can choose the amount of information she wants to disclose about the statements. The owner uses them to certify to a verifier that she is given authorization to perform some actions or that she fulfills certain conditions. In turn he may allow her to perform these actions or to get another credential from him.

In some sense, credentials are similar to certificates. In fact, these words can be used as synonyms. However, certificate is usually utilized to refer to conventional certificates as defined by the standards X.509 or SPKI, which do not fulfill important privacy properties [CH02]. For instance, different uses of the same certificate are linkable to each other or to the transaction in which the certificate was issued. Moreover, more information apart from the data that the owner wants to disclose can be obtained from a combination of context and addressing information from one or a series of transactions.

Therefore, in order to protect the privacy of the owner we require the following properties in the process by which a credential is shown [BCL04]:

**Multi-show unlinkability.** The show of a credential can be linked neither to the issuing of the credential nor to another show of the same credential unless the information being disclosed allow for such linking.

**Selective show of data items.** The owner of a credential can select which statements she wants to disclose when showing the credential. In addition, it should be possible to prove predicates about data items without revealing the value. For example, the owner should be able to prove that she is not underage without revealing her age.

**Conditional showing of statements.** When showing a credential, certain provided information should be revealed only if one or more predicates are true. To achieve this, the owner sends encrypted data that cannot be decrypted by the verifier of the credential, but that the verifier can send to a trusted third party that, if the predicates are true, will reveal the data to the verifier. This can be used, for example, to prevent the owners from misbehaving. The verifier is allowed to get information about them only if they misbehave.

**Proving relations between statements.** It should be possible to prove that values that belong to different credentials fulfill one or more relations without disclosing these values. This is useful when the owner needs to show more than one credential at once, for example, to ensure that these credentials were issued to the same owner without revealing the owner's identity.

On the other hand, it is necessary to prevent fraud [CL01]. For this purpose, other properties are needed (the last one is applied only to anonymous credentials):

**Unforgeability of credentials** An owner should not be able to show credentials that have not been issued to her, even if she colludes with some collaborators. In addition, the owner should not be able to change the statements certified in the credential.

**Discourage of credential sharing.** Some measures should be taken to discourage the sharing of credentials between owners. These measures usually reveal a secret key belonging to an owner that shares her credentials.

**Consistency of credentials.** It should not be possible for a group of owners to team up and show some of their credentials to obtain a permission or a credential that one owner alone would not have gotten.

Apart from this, credentials may have extra features, like expiration date or the possibility of being revoked, and they can have options, like being a one-show credential or a multi-show credential [Koh04].

### 2.11.2 Implementation of a Credential Scheme based on any Signature Scheme

We want to show how to construct credentials and how to implement protocols that allow to get and show credentials. For this purpose, we provide ourselves with a signature scheme ($SetupSign$, $Sign$, $VerifySign$) that works as explained in Definition 45.

A credential $Cred$ is a signature on a block of messages $Cred = Sign(params, s_k, M)$. However, instead of utilizing algorithms $Sign$ and $VerifySign$ for issuing and showing credentials, we need to use two interactive protocols in order not to disclose all the data being signed when issuing a credential or the data that has been signed when showing the credentials. In other words, instead of making the issuer of credentials run algorithm $Sign$, we use a protocol $GetCredential$ between the issuer and the owner by means of which the owner can obtain a signature for the block of messages or statements of the credential without letting the issuer know all the messages. On the other hand, instead of making the verifier use algorithm $VerifySign$, we run a protocol $ShowCredential$ between the verifier and the owner by means of which the verifier is convinced that the owner has been issued a credential that ensures certain statements without learning all these statements. Therefore, a credential scheme can be defined as follows:

**Construction 1 (Credentials based on any signature scheme)** *Given an underlying signature scheme SignSch = (SetupSign, Sign, VerifySign) and a commitment scheme CommSch = (SetupCommit, Commit, Open), a credential scheme is a set of one probabilistic polynomial time algorithm (SetupCred) and two interactive protocols (GetCredential, ShowCredential) as follows:*

**SetupCred($1^k$).** *On input a security parameter $k$, it runs $SetupSign(1^k)$ to output the parameters of the underlying signature scheme $params_{sig}$ and the secret key $s_k$, which is given to the issuer. It also runs $SetupCommit(1^k)$ and outputs $params_{comm}$. $params$ = ($params_{sig}$, $params_{comm}$) is the joint set of parameters.*

**GetCredential(Issuer($input_{issuer}$), Owner($input_{owner}$)).** *We have that $input_{issuer} = \{params, s_k, M_k, C_s\}$ and $input_{owner} = \{params, M_k, M_s, Open_s\}$. The input of the issuer is the secret key $s_k$, the set of known messages $M_k = \{m_{l'+1}, \ldots, m_l\}$, and the set of commitments $C_s = \{c_{m_1}, \ldots, c_{m_{l'}}\}$ to the secret messages $M_s = \{m_1, \ldots, m_{l'}\}$. The input of the owner is the set of known messages $M_k$, the set of secret messages $M_s$ and the set containing the randomness for each commitment $Open_s = \{open_{m_1}, \ldots, open_{m_{l'}}\}$. The output of the owner is a valid credential $Cred$ according to $SignSch$ or rejection. The output of the issuer is nothing or rejection.*

**ShowCredential(Verifier($input_{verifier}$), Owner($input_{owner}$)).** *We have that $input_{verifier} = \{params, \bar{CredV}, policy\}$ and $input_{owner} = \{params, CredO, policy\}$. The input of the verifier is the set $\bar{CredV} = \{CredV_1, \ldots, CredV_n\}$ about all the credentials that the owner should show in order to get authorization from the verifier, where $n$ is the total number of credentials. Each element $CredV_i = \{C_{u_i}, M_{r_i}\}$, where*

$1 \leq i \leq n$, *consists of the set of commitments for the unrevealed messages* $C_u = \{c_{m_{i1}}, \ldots, c_{m_{il^*}}\}$ *and of the set of revealed messages* $M_{r_i} = \{m_{i(l*+1)}, \ldots, m_{il}\}$. *In addition, it has as input policy, which is a set of predicates that should be fulfilled by the credentials presented by the owner. The input of the owner is the set* $\bar{CredO} = \{CredO_1, \ldots, CredO_n\}$, *where each* $CredO_i = \{Cred_i, M_{r_i}, M_{u_i}, Open_{u_i}\}$ *with* $1 \leq i \leq n$. *The output of the verifier is acceptance or rejection. He accepts when the owner demonstrates that she has valid credentials whose messages fulfill the predicates specified in policy. The output of the owner is being accepted or being rejected.*

Therefore, *GetCredential* is a protocol that allows an owner to obtain a credential on a set of messages $M = M_s||M_k$, where $M_s$ stands for the set of messages known only by the owner and $M_k$ denotes the set of messages also known by the issuer and possibly chosen by him. Note that, for simplicity, we do not consider the case in which some messages can be jointly chosen by the owner and the organization. In addition, we consider the general case in which the owner is required to commit to the values that are only known by her using a commitment scheme which is not related with the signature scheme, although this is not always necessary.

On the other hand, *ShowCredential* allows an owner to obtain authorization from a verifier in order to get another credential from him or to perform some actions. We consider again the general case in which the owner uses a commitment scheme independent from the signature scheme. In addition, we consider the case in which more than one credential needs to be shown. One could think that it can be done by showing only one credential at once and running the protocol as many times as necessary, but we consider the case in which the verifier wants a subset or the whole set of credentials to fulfill a certain predicate given in policy, so they have to be shown at once. *policy* is a set of predicates specifying the properties that the credentials shown by the owner should fulfill in order to give her authorization. For example, if the verifier is a shop that sells goods to people who has a membership and that makes discounts for underage people, he not only wants the owner to show a valid credential that demonstrates her membership but also to show that the value indicating the age, contained in the credential, is lower than 18. If the credential that contains the age is different from the one ensuring the membership, the verifier may require that both credentials were issued under the same identity, and for this purpose the owner needs to show both credentials at once. *policy* may also include extra features like the expiration date.

For the case in which it is enough to show the validity of one credential, the following *ShowCredential* protocol should be applied:

**ShowCredential(Verifier(**$input_{verifier}$**), Owner(**$input_{owner}$**)).** We have that $input_{verifier}$=$\{params, C_u, M_r\}$ and $input_{owner}$=$\{params, Cred,$

$M_r$, $M_u$, $Open_u\}$. The input of the verifier is the set of commitments to the secret messages $C_u$ and the set of revealed messages $M_r$. The input of the owner is the credential $Cred$, the set of revealed and unrevealed messages and the set of randomness of those commitments $Open_u$. The output of the verifier is acceptance or rejection. It accepts when the owner demonstrates that she has a valid credential $Cred$. The output of the owner is being accepted or being rejected.

### 2.11.3 Instantiation using the Camenisch-Lysyanskaya Signature Scheme

We want to instantiate the construction given in the previous section utilizing a signature scheme proposed by Camenisch and Lysyankaya [CL02]. First of all, we show how this scheme works. For this purpose, let $l_n$ be the length of a special RSA modulus $n = pq$, $l_m$ the length of the messages, $k$ the length of the challenge, $k'$ a parameter governing the statistical indistinguishability of two distributions and $l_e = l_m + 3$. The message space is $\{(m_1, \ldots, m_l) : m_i \in \{0,1\}^{l_m}\}$.

**Construction 2 (Camenisch-Lysyanskaya signature scheme)** *The Camenisch-Lysyanskaya signature scheme is a set of three probabilistic polynomial time algorithms as follows:*

**SetupSign**$(1^k)$ *On input a security parameter $k = l_n$, it computes an $l_n$-bit special RSA modulus $n = pq$, where $p = 2p' + 1$ and $q = 2q' + 1$. It randomly picks $s \in QR_n$ and $r_1, \ldots, r_l, z \in \langle s \rangle$. It also provides a non-interactive proof that $r_1, \ldots, r_l, z \in \langle s \rangle$:*

$$NIPK\{(\rho_1, \ldots, \rho_l, \alpha) : r_1 = s^{\rho_1} \bmod n \wedge \ldots \wedge r_l = s^{\rho_l} \bmod n \wedge z = s^{\alpha} \bmod n\}$$

*It outputs the secret key $s_k = (p, q)$ and the parameters $params = (n, r_1, \ldots, r_l, s, z, l_m)$ of the scheme.*

**Sign**$(params, s_k, M)$ *On input a block of messages $M = \{m_1, \ldots, m_l\}$ and the secret key $s_k$, it picks a random prime number $e$ of length $l_e + k' + k + 1$ such that $e > 2^{l_e+k'+k}$, and a random number $v$ of length $l_v = l_n + l_m + l_r$, where $l_r$ is a security parameter. It outputs the signature $S = (e, A, v)$, where $A$ is such that $z = r_1^{m_1} \cdots r_l^{m_l} s^v A^e \bmod n$.*

**VerifySign**$(params, S, M')$**.** *On input the signature $S$ and the block of messages $M' = \{m'_1, \ldots, m'_l\}$, it checks whether $z = r_1^{m'_1} \cdots r_l^{m'_l} s^v A^e \bmod n$ and $2^{l_e+k'+k} < e < 2^{l_e+k'+k+1}$. If both conditions are fulfilled, it accepts. Otherwise it rejects.*

This signature scheme is secure under the strong RSA assumption (see Section 2.6). For the proof of security we refer to [CL02].

Now we apply this signature scheme and Pedersen commitments (see Section 2.8.2) to construct a credential scheme. For simplicity, we show a protocol *ShowCredential* in which the owner only proves the validity of one credential. Both protocols *GetCredential* and *ShowCredential* are taken from [BCL04].

**Construction 3 (Credentials based on CL signature scheme)** *Given the underlying Camenisch-Lysyanskaya signature scheme CLSignSch = (SetupSign, Sign, VerifySign) and the Pedersen commitment scheme PedCommSch = (SetupCommit, Commit, Open), the credential scheme is a set of one probabilistic polynomial time algorithm (SetupCred) and two interactive protocols (GetCredential, ShowCredential) as follows:*

**SetupCred($1^k$).** *On input a security parameter $k$, it runs $SetupSign(1^k)$ to output the parameters of the signature scheme $params_{sig} = (n, r_1, \ldots, r_l, s, z, l_m)$ and the secret key $s_k$, which is given to the issuer. It also runs $SetupCommit(1^k)$ and outputs $params_{comm} = (G_q, g, h)$. $params = (params_{sig}, params_{comm})$ is the joint set of parameters.*

**GetCredential(Issuer($input_{issuer}$), Owner($input_{owner}$)).** *We have that $input_{issuer} = \{params, s_k, M_k, C_s\}$ and $input_{owner} = \{params, M_k, M_s, C_s, Open_s\}$. The input of the issuer is the secret key $s_k = (p, q)$, the set of known messages $M_k = \{m_{l'+1}, \ldots, m_l\}$, and the set of commitments $C_s = \{c_{m_1}, \ldots, c_{m_{l'}}\}$ to the secret messages $M_s = \{m_1, \ldots, m_{l'}\}$, where each $c_{m_i} = g^{m_i} h^{open_{m_i}}$ and $1 \leq i \leq l'$. The input of the owner is the set of known messages $M_k$, the set of secret messages $M_s$ and the set containing the randomness for each commitment $Open_s = \{open_{m_1}, \ldots, open_{m_{l'}}\}$. The protocol works as follows:*

1. *The owner picks randomly $v' \in \{0,1\}^{l_n + k'}$, computes $C = r_1^{m_1} .. r_{l'}^{m_{l'}} s^{v'} \bmod n$ and sends $C$ to the issuer.*
2. *Owner and issuer engage in the following proof of knowledge:*

$$PK\{(m_1, \ldots, m_{l'}, open_{m_1}, \ldots, open_{m_{l'}}, v') : c_{m_1} = g^{m_1} h^{open_{m_1}} \wedge \ldots \wedge c_{m_{l'}} = g^{m_{l'}} h^{open_{m_{l'}}} \wedge C = r_1^{m_1} .. r_{l'}^{m_{l'}} s^{v'} \bmod n \wedge m_1, \ldots, m_{l'} \in \{0,1\}^{l_m + k + k'}\}$$

3. *The issuer picks a random $l_e$-bit integer $e'$ such that $e = 2^{l_e + k + k' + 1} + e'$ is a prime number. He also picks randomly $v'' \in \mathbb{Z}_e$, computes*

$$A = \left(\frac{z}{C r_{l'+1}^{m_{l'+1}} .. r_l^{m_l} s^{v''}}\right)^{\frac{1}{e}} \bmod n$$

*and sends (e,A,v") to the owner.*

**4.** *To convince the owner that $A \in \langle s \rangle$, they run the following proof of knowledge:*

$$PK\{(\frac{1}{e}) : A = (\frac{z}{Cr_{l'+1}^{m_{l'+1}}..r_l^{m_l} s^{v''}})^{\frac{1}{e}} \ mod \ n\}$$

**5.** *The owner verifies that $e > 2^{l_e+k+k'+1}$ is prime and stores $(e, A, v = v' + v'')$ as the credential.*

*Therefore, the output of the owner is a valid credential $Cred = (e, A, v)$ according to SignSch on the block of messages $\{m_1, \ldots, m_l\}$ or rejection. The output of the issuer is nothing or rejection.*

**ShowCredential(Verifier($input_{verifier}$), Owner($input_{owner}$)).** *We have that $input_{verifier}$=$\{params, C_u, M_r\}$ and $input_{owner}$=$\{params, Cred, M_r, M_u, Open_u\}$. The input of the verifier is the set of commitments $C_u = \{c_{m_1}, \ldots, c_{m_{l*}}\}$ to the unrevealed messages $M_u = \{m_1, \ldots, m_{l*}\}$ and the set of revealed messages $M_r = \{m_{l*+1}, \ldots, m_l\}$. The input of the owner is the credential $Cred = (e, A, v)$, the set of revealed and unrevealed messages and the set of randomness of these commitments $Open_u$. The protocol works as follows:*

**1.** *The owner picks a random $r_A \in \{0,1\}^{l_n+k}$, computes $\tilde{A} = As^{r_A}$, and sends $\tilde{A}$ to the verifier.*

**2.** *The owner and the verifier run the following proof:*

$$\begin{aligned} PK\{&(\lambda, m_1, \ldots, m_{l*}, open_{m_1}, \ldots, open_{m_{l*}}, v) : \\ &c_{m_1} = g^{m_1} h^{open_{m_1}} \wedge \ldots \wedge c_{m_{l*}} = g^{m_{l*}} h^{open_{m_{l*}}} \wedge \\ &\frac{z}{\tilde{A}^{2^{l_e+k+k'+1}} r_{l*+1}^{m_{l*+1}}..r_l^{m_l}} = \tilde{A}^{\lambda} r_1^{m_1}..r_{l*}^{m_{l*}} s^v \ mod \ n \wedge \\ &\lambda \in \{0,1\}^{l_e+k+k'} \wedge m_1, \ldots, m_{l*} \in \{0,1\}^{l_m+k+k'}\} \end{aligned}$$

*The output of the verifier is acceptance or rejection. It accepts when the owner demonstrates that she has a valid credential. The output of the owner is being accepted or being rejected.*

# Chapter 3

# Basic and Related Work

We explain the basic concepts used later in our constructions. First, we introduce the concept of Identity-Based Encryption (IBE) along with the conditions that an IBE scheme should fulfill in order to be anonymous, and the concept of blind-IBE. Second, in Section 3.2 we define Public-key Encryption with Keyword Search (PEKS) and the properties that a PEKS scheme should fulfill in order to be secure and consistent. For this purpose, we generalize the definitions that can be found in [BDOP04] or [ABC$^+$05] to make a definition useful for all the application scenarios in which PEKS can be applied. After that, we compare these applications and we explain how to build a PEKS scheme that follows our definition of PEKS by using any anonymous-IBE scheme.

These two concepts are used to build Public-key Encryption with Oblivious Keyword Search in Chapter 4, whereas the following concept is used to construct a Priced Oblivious Transfers scheme in Chapter 5. In Section 3.3 we introduce the concept of Oblivious Transfer (OT), and we explain its security properties. In addition, we show how to build an OT scheme using any blind-IBE scheme. Finally, in Section 3.4 we define Oblivious Keyword Search along with its security properties and we compare OKS with OT.

## 3.1 Identity-Based Encryption

Identity-Based Encryption (IBE) is a kind of asymmetric key encryption in which the identity of the user is her public key. Its main advantage is that there is no need for certification authorities (CA) to provide certificates in order to let users know the validity of public keys because the digital identity of a user is publicly available information that uniquely and undeniably identifies her.

In this section, we first define IBE schemes and we show the conditions they should hold in order to provide consistency and security. Then we explain the concept of anonymous-IBE and blind-IBE along with the prop-

erties that an IBE scheme should fulfill in order to provide anonymity and blind key derivation. Finally, we give an overview on the previous work on this topic.

### 3.1.1 Definition

In order to define IBE schemes we need to explain the concept of message space [ABC+05]:

**Definition 47 (Message space)** *A message space $Msg(k)$ is a map assigning to every $k \in \mathbb{N}$ a set of strings such that $\{0,1\}^k \subseteq Msg(k) \subseteq \{0,1\}^*$ and the following two conditions hold: first, there is a probabilistic polynomial time algorithm that on input $1^k$ outputs 1 if a message $M$ belongs to the message space ($M \in Msg(k)$) and 0 otherwise; second, $\{0,1\}^{|M|} \subseteq Msg(k)$ for all $k \in \mathbb{N}$ and $M \in Msg(k)$.*

The following definition follows the one given in [BF01]:

**Definition 48 (IBE)** *An IBE scheme is a set of four probabilistic polynomial time algorithms (SetupIBE, Extract, Encrypt, Decrypt) as follows:*

**SetupIBE($1^k$).** *On input a security parameter $k$ it outputs a master secret key $s_k$ and the parameters of the scheme params, which include a description of the message space $Msg(k)$, of the identity space $I(k)$ and of the ciphertext space $C(k)$.*

**Extract($params$,$s_k$,$id$).** *On input an identity id and the master secret key $s_k$, it outputs the secret key $sk_{id}$ corresponding to that identity.*

**Encrypt($params$,$id$,$M$).** *On input an identity id, it outputs a ciphertext $C$ of a message $M$ under identity id.*

**Decrypt($sk_{id}$,$C$).** *On input the secret key $sk_{id}$ related to identity id and a ciphertext $C = Encrypt(params, id', M)$, it outputs the message $M \in Msg(k)$ if $id = id'$. Otherwise it outputs a random message.*

The scheme works as depicted in Figure 3.1. First, algorithm $SetupIBE$ is run, the Key Generation Center (KGC) is given the master secret key and the parameters of the scheme are published. Then, when a sender wants to encrypt a message intended for a receiver with identity $id$, he executes algorithm *Encrypt* using $id$ as input to obtain the corresponding ciphertext. In parallel, a receiver who wants to decrypt messages intended for her sends her identity to the KGC in order to get a secret key for her identity. Upon receiving the identity, the KGC runs *Extract* and sends this secret key to the receiver. Finally, a receiver who has a secret key for her identity and a ciphertext uses algorithm *Decrypt* to obtain the plaintext.

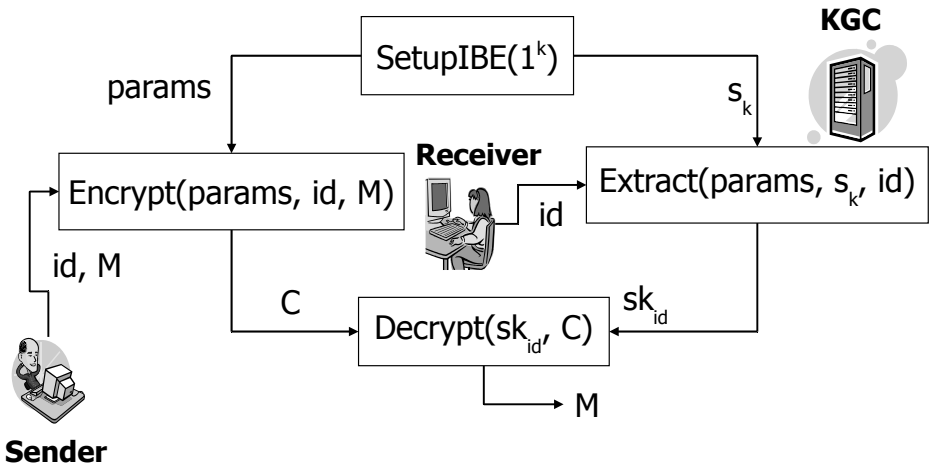


Figure 3.1: Identity-Based Encryption scheme.

Therefore, the way by which the secret key is obtained in IBE is different from the way used in traditional public-key encryption (PKE). In PKE, one can choose a secret key at random and compute by means of a possibly randomized one-way function the public key, and finally submit this public key to a CA in order to get the corresponding certificate. Since in IBE the public key is fixed beforehand, if users were able to compute the corresponding secret key themselves everyone could have the secret key related to every identity. To avoid that, each user has to send her identity to the KGC, which computes the corresponding secret key by using the identity and its master secret key. For a detailed comparison between PKE and IBE we refer to [Maa04].

Every cryptographic primitive should fulfill two properties: security and consistency. Consistency means that the primitive should work properly and in IBE this means that decryption reverses encryption. We can formalize this notion as follows:

**Definition 49 (Consistency for IBE)** *An IBE scheme is consistent if it fulfills the following condition:*

$$\Pr[(s_k, params) \leftarrow SetupIBE(1^k); sk_{id} \leftarrow Extract(params, s_k, id); \\ C \leftarrow Encrypt(params, id, M); M' \leftarrow Decrypt(sk_{id}, C) : M' = M] = 1$$

In IBE, security means privacy and it can be formalized in several ways, e.g., indistinguishability under chosen plaintext attack (IBE-IND-CPA) (also semantic security) or indistinguishability under chosen ciphertext attack (IBE-IND-CCA). The former means that an adversary cannot

distinguish between the encryption of two challenge messages of his choice even when he is allowed to obtain secret keys for any identity (except the one used to compute the ciphertexts). According to [BF01], it can be defined as follows:

**Definition 50 (IBE-IND-CPA)** *An IBE scheme is said to be IBE-IND-CPA secure if there exists a negligible function $\nu(k)$ such that:*

$$\begin{aligned}&\Pr[IDSet \leftarrow \emptyset; (s_k, params) \leftarrow SetupIBE(1^k);\\ &\quad (id, M_0, M_1, state) \leftarrow A^{\leftrightarrow Oracle_{Extract}(\cdot)}(params) \wedge M_0, M_1 \in Msg(k) \wedge\\ &\quad |M_0| = |M_1|; b \leftarrow \{0,1\}; C \leftarrow Encrypt(params, id, M_b);\\ &\quad b' \leftarrow A^{\leftrightarrow Oracle_{Extract}(\cdot)}(params, C, state);\\ &\quad b' \leftarrow 0 \wedge IDSet \cap \{id\} \neq \emptyset : b = b'] < 1/2 + \nu(k)\end{aligned}$$

*In the expression above, $Oracle_{Extract}(id)$ works as follows: it adds id to the set, $IDSet \leftarrow IDSet \cup \{id\}$, and it returns $sk_{id} \leftarrow Extract(params, s_k, id)$. $|M_0|$ denotes the length of the message.*

More intuitively, we can describe IBE-IND-CPA by means of an IBE-IND-CPA game:

**1.** Run algorithm $SetupIBE(1^k)$ and give the parameters $params$ to the adversary.

**2.** The adversary makes adaptive queries to $Oracle_{Extract}(id)$ and gets back the secret keys for these identities.

**3.** The adversary outputs two equal length plaintexts $M_0, M_1 \in Msg(k)$ and an identity $id$.

**4.** Pick a random bit and run $Encrypt(params, id, M_b)$ to obtain a ciphertext $C$ for message $M_b$ under identity $id$. Give $C$ to the adversary.

**5.** The adversary continues making adaptive queries to $Oracle_{Extract}(id)$.

**6.** The adversary outputs a bit $b'$ in order to guess which message was used to build the ciphertext. If the identity $id$ was used to query $Oracle_{Extract}$ in Step 2 or in Step 5 the game returns $b' = 0$. Otherwise it returns $b'$.

The advantage of the adversary is $Adv(A) = |Pr[b = b'] - 1/2|$. As in Definition 50, we say that the IBE scheme is IBE-IND-CPA secure when the advantage of the adversary is negligible.

In IBE-IND-CCA the adversary can also make queries to an oracle $Oracle_{Decrypt}(id, C)$ by means of which he obtains a decryption using the

secret key corresponding to $id$ for a ciphertext $C$ in Step 2 and in Step 5. The game returns $b' = 0$ if the adversary makes a query to $Oracle_{Decrypt}$ where $id$ is the one output by the adversary in Step 3 and $C$ is the ciphertext computed in Step 4. The IBE scheme is said to be IBE-IND-CCA secure when the advantage of the adversary in the game is negligible.

Apart from that, another (slightly weaker) notion of security is called selective-ID [CHK03], and then we get both IBE-IND-sID-CPA and IBE-IND-sID-CCA. In the above-mentioned game, the difference is that the adversary has to output the identity under which he wants to be challenged at the beginning, so he cannot take that decision depending on the information obtained during Step 2.

### 3.1.2 Anonymous Identity-Based Encryption

A public-key encryption scheme is said to be anonymous when the ciphertext does not leak the public key [BBDP01]. In particular, an IBE scheme is anonymous when the ciphertext does not leak the identity of the receiver. Anonymous-IBE schemes are interesting because they allow us to build Public-key Encryption with Keyword Search (PEKS) schemes [ABC$^+$05] [BDOP04] (see Section 3.2.3). Therefore, in this section we show the condition that an IBE scheme should fulfill in order to be anonymous.

As for privacy, anonymity can be formalized as anonymity under chosen plaintext attack (IBE-ANO-CPA) and under chosen ciphertext attack (IBE-ANO-CCA). [ABC$^+$05] adapted the definition of anonymity for public-key encryption proposed by [BBDP01] to IBE.

**Definition 51 (IBE-ANO-CPA)** *An IBE scheme is said to fulfill IBE-ANO-CPA if there exists a negligible function $\nu(k)$ such that:*

$$\begin{aligned}
&\Pr[IDSet \leftarrow \emptyset; (s_k, params) \leftarrow SetupIBE(1^k);\\
&\quad (id_0, id_1, M, state) \leftarrow A^{\leftrightarrow Oracle_{Extract}(\cdot)}(params) \wedge M \in Msg(k);\\
&\quad b \leftarrow \{0,1\}; C \leftarrow Encrypt(params, id_b, M);\\
&\quad b' \leftarrow A^{\leftrightarrow Oracle_{Extract}(\cdot)}(params, C, state);\\
&\quad b' \leftarrow 0 \wedge IDSet \cap \{id_0, id_1\} \neq \emptyset : b = b'] < 1/2 + \nu(k)
\end{aligned}$$

*In the expression above, $Oracle_{Extract}(id)$ works as follows: it adds $id$ to the set, $IDSet \leftarrow IDSet \cup \{id\}$, and it returns $sk_{id} \leftarrow Extract(params, s_k, id)$.*

More intuitively, we can describe IBE-ANO-CPA by means of an IBE-IND-CPA game:

**1.** Run algorithm $SetupIBE(1^k)$ and give the parameters $params$ to the adversary.

**2.** The adversary makes adaptive queries to $Oracle_{Extract}(id)$ and gets back the secret keys for these identities.

**3.** The adversary outputs a message $M \in Msg(k)$ and two identities $id_0, id_1$.

**4.** Pick a random bit and run $Encrypt(params, id_b, M)$ to obtain a ciphertext $C$ for message $M$ under identity $id_b$. Give $C$ to the adversary.

**5.** The adversary continues making adaptive queries to $Oracle_{Extract}(id)$.

**6.** The adversary outputs a bit $b'$ in order to guess which identity was used to build the ciphertext. If the identities $id_0$ or $id_1$ were used to query $Oracle_{Extract}$ in Step 2 or in Step 5 the game returns $b' = 0$. Otherwise it returns $b'$.

The advantage of the adversary is $Adv(A) = |Pr[b = b'] - 1/2|$. As in Definition 51, we say that the IBE scheme is IBE-ANO-CPA anonymous when the advantage of the adversary is negligible.

In IBE-ANO-CCA the adversary is provided with two additional oracles: $Oracle_{Decrypt_0}$, which given a ciphertext $C$ outputs the decryption of $C$ by using identity $id_0$, and $Oracle_{Decrypt_1}$, which behaves equivalently using identity $id_1$. The game returns $b' = 0$ if the adversary makes a query where $C$ is the ciphertext computed in Step 4. The IBE scheme is said to be IBE-ANO-CCA anonymous when the advantage of the adversary in the game is negligible.

Finally, we show how to combine IBE-IND-CPA and IBE-ANO-CPA in one property that is called IBE-IND-ANO-CPA security, which was done in [Gen06].

**Definition 52 (IBE-IND-ANO-CPA)** *An IBE scheme is said to be IBE-IND-ANO-CPA secure if there exists a negligible function $\nu(k)$ such that:*

$$\begin{aligned}
&\Pr[IDSet \leftarrow \emptyset; (s_k, params) \leftarrow SetupIBE(1^k);\\
&\quad (id_0, id_1, M_0, M_1, state) \leftarrow A^{\leftrightarrow Oracle_{Extract}(\cdot)}(params) \wedge\\
&\quad M_0, M_1 \in Msg(k) \wedge |M_0| = |M_1|;\\
&\quad b \leftarrow \{0,1\}; c \leftarrow \{0,1\}; C \leftarrow Encrypt(params, id_b, M_c);\\
&\quad (b', c') \leftarrow A^{\leftrightarrow Oracle_{Extract}(\cdot)}(params, C, state);\\
&\quad b' \leftarrow 0 \wedge c' \leftarrow 0 \wedge IDSet \cap \{id_0, id_1\} \neq \emptyset : b = b' \wedge c = c'] < 1/4 + \nu(k)
\end{aligned}$$

*In the expression above, $Oracle_{Extract}(id)$ works as follows: it adds id to the set, $IDSet \leftarrow IDSet \cup \{id\}$, and it returns $sk_{id} \leftarrow Extract(params, s_k, id)$. $|M_0|$ denotes the length of the message.*

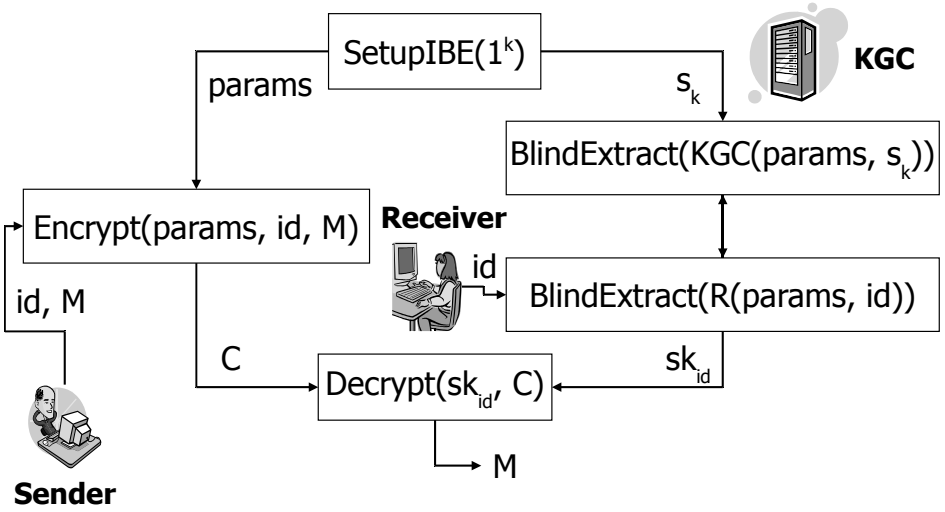


Figure 3.2: Blind-IBE scheme.

### 3.1.3 Blind Identity-Based Encryption

In IBE a user who wants to obtain a secret key for her identity has to send her identity to the KGC in order to get the secret key. Therefore, the KGC learns both the identity of the user and her secret key, so the user has to trust it. However, by means of blind-IBE the KGC learns neither the identity nor the secret key. The problem that arises now is that everyone can get from the KGC secret keys for identities of other users. Despite of this, blind-IBE is useful to build other primitives such as Oblivious Transfer schemes (see Section 3.3.2) and is also used in the construction of the Public-key Encryption with Oblivious Keyword Search scheme presented in Chapter 4.

The main idea behind blind-IBE consists in running a two-party protocol between the user and the KGC in order to compute the secret key for the user instead of making the KGC execute the *Extract* algorithm. This protocol, *BlindExtract*, is defined as follows [GH07]:

**Definition 53 (BlindExtract(KGC($params$,$s_k$), U($params$,$id$)))** *The input of the KGC is the master secret key $s_k$, whereas the input of the user is her identity $id$. The output of the user is the secret key $sk_{id}$ corresponding to identity $id$ or $\perp$, and the output of the KGC is nothing or $\perp$.*

The resultant IBE scheme is depicted in Figure 3.2. In the picture the word *BlindExtract* is written in two boxes indicating that it refers to a protocol between the receiver and the KGC.

A blind-IBE scheme is secure when the underlying IBE scheme is secure and the *BlindExtract* protocol fulfills two properties: leak freeness and selective-failure blindness. More formally [GH07]:

**Theorem 1** *A blind-IBE scheme (SetupIBE, BlindExtract, Encrypt, Decrypt) is called IBE-IND-CPA secure if and only if the underlying IBE scheme (SetupIBE, Extract, Encrypt, Decrypt) is IBE-IND-CPA secure and protocol BlindExtract is leak free and selective-failure blind.*

Informally speaking, leak freeness ensures that a malicious user cannot learn anything by running the *BlindExtract* protocol with an honest KGC that she could not have learned by sending her identity to the KGC and receiving the secret key corresponding to that identity from the KGC, which executes algorithm *Extract*. More formally, this property is defined as follows [GH07]:

**Definition 54 (Leak freeness)** *A protocol BlindExtract associated with an IBE scheme (SetupIBE, Extract, Encrypt, Decrypt) is leak free if, for all efficient adversaries A, there exists an efficient simulator S such that for every security parameter k no efficient distinguisher D can distinguish whether it is faced with A playing Game Real or with S playing Game Ideal with non-negligible advantage:*

**Game Real.** *Run* $SetupIBE(1^k)$. *As many times as D wants, A chooses an identity id and executes the BlindExtract protocol with the KGC.*

**Game Ideal.** *Run* $SetupIBE(1^k)$. *As many times as D wants, S chooses an identity id and queries a trusted party to obtain the output of Extract(params,*$s_k$*,id), if* $id \in I(k)$ *and* $\perp$ *otherwise.*

*Here D and A (or S) may communicate at any time.*

On the other hand, selective-failure blindness ensures that a malicious KGC cannot learn anything about the user's choice during the *BlindExtract* protocol and thus it cannot cause the protocol to fail depending on the user's choice. More formally, this property is defined as follows [CNS07]:

**Definition 55 (Selective-failure blindness)** *A protocol BlindExtract* $(A(\cdot,\cdot),\ U(\cdot,\cdot))$ *is said to be selective-failure blind if there exists a negligible function* $\nu(k)$ *such that:*

$$\Pr[(params, id_0, id_1, state) \leftarrow A(1^k); b \leftarrow \{0,1\};$$
$$b' \leftarrow A(state)^{Oracle_U(params,id_b);Oracle_U(params,id_{1-b});Oracle_{Aux}()}$$
$$: b = b'] < 1/2 + \nu(k)$$

$Oracle_U(params, id_b)$ *interacts with A by executing the user's part of protocol BlindExtract. It returns, not until the execution of* $Oracle_{Aux}()$, $sk_{id_b}$ *or* $\perp$ *if the protocol aborts. On the other hand,* $Oracle_{Aux}()$ *returns* $\perp$ *if it is executed before the other two oracles have ended. Otherwise, it returns* $(sk_{id_b}, sk_{id_{1-b}})$ *if* $sk_{id_b} \neq \perp$ *and* $sk_{id_{1-b}} \neq \perp$, $(\perp, \epsilon)$ *if* $sk_{id_b} = \perp$ *and* $sk_{id_{1-b}} \neq \perp$, $(\epsilon, \perp)$ *if* $sk_{id_b} \neq \perp$ *and* $sk_{id_{1-b}} = \perp$ *and* $(\perp, \perp)$ *if* $sk_{id_b} = \perp$ *and* $sk_{id_{1-b}} = \perp$.

More intuitively, we can describe the above-mentioned game as follows:

**1.** The adversary outputs the parameters of the scheme and two identities $id_0$ and $id_1$.

**2.** Pick a random bit $b$, and interact with the adversary by executing the user's part of the *BlindExtract* protocol, using $id_b$ the first time and $id_{1-b}$ the second time. Output $sk_b$ and $sk_{1-b}$ respectively.

**3.** Give to the adversary ($sk_{id_b}$,$sk_{id_{1-b}}$) if $sk_{id_b} \neq \perp$ and $sk_{id_{1-b}} \neq \perp$, $(\perp, \epsilon)$ if $sk_{id_b} = \perp$ and $sk_{id_{1-b}} \neq \perp$, $(\epsilon, \perp)$ if $sk_{id_b} \neq \perp$ and $sk_{id_{1-b}} = \perp$ and $(\perp, \perp)$ if $sk_{id_b} = \perp$ and $sk_{id_{1-b}} = \perp$. The adversary outputs $b'$ in order to guess which identity was used in each execution of the protocol.

The advantage of the adversary is $Adv(A) = |Pr[b = b'] - 1/2|$. As in Definition 55, we say that the *BlindExtract* protocol is selective-failure blind when the advantage of the adversary is negligible.

### 3.1.4 Historical Overview

Although IBE was proposed by Shamir [Sha84] in 1984, it was not until 1996 when Maurer and Yacobi proposed the first construction for IBE [MY96]. In 2001 Boneh and Franklin [BF01] designed the first practical scheme, which was proven secure under adaptive-ID security in the random oracle model. In 2003, Canetti, Halevi and Katz [CHK03] proposed a weaker notion of security, selective-ID security, and constructed an inefficient IBE scheme secure in the standard model. Afterwards, efficient selective-ID secure schemes in the standard model were designed by Boneh and Boyen [BB04a]. Although they showed an extension to construct an IBE scheme under adaptive-ID security, their solution was impractical. Later, Waters [Wat05] designed a simpler extension to improve its efficiency. This simplified extension was generalized by Naccache [Nac05] to allow for shorter public parameters.

Until that moment, there were only two IBE schemes which provide anonymity, i.e., which ensure that the ciphertext does not leak the identity of the recipient. They were the one due to Boneh and Franklin [BF01] and another one due to Sakai and Kasahara [CCMLS06] (also in the random oracle model). The IBE schemes designed in order to avoid random oracles introduce randomness in the generation of the secret key. This made them non-anonymous, because the information included in the ciphertext to cancel that randomness leaks the identity. However, recently two anonymous IBE schemes were constructed by Boyen and Waters [BW06] and by Gentry [Gen06]. The former achieves anonymity by splitting the identity into two randomized complementary components and was used to construct the first anonymous hierarchical IBE scheme. The latter achieves this by moving an essential element of the ciphertext into the target group of the bilinear

map instead of the gap group. It is secure under a stronger assumption and it cannot be used to design an anonymous HIBE scheme, but has several advantages like computational efficiency or shorter public parameters.

All the above-mentioned IBE schemes are not blind. In 2007 Green and Hohenberger [GH07] derived the first blind IBE schemes from the Boneh and Boyen scheme and from the Waters-Naccache scheme and applied them to construct Oblivious Transfer schemes secure in the full-simulation model while requiring only static complexity assumptions (see Section 3.3.2).

## 3.2 Public-key Encryption with Keyword Search

Public-key Encryption with Keyword Search (PEKS) is a mechanism that allows to search by keywords on encrypted data. It aims at preserving the privacy of the receiver of the data while providing a way that allows her to search efficiently without decrypting.

In this section, we start by explaining the concept of PEKS, its motivation and its properties. Then we show two application scenarios to which PEKS was applied and we explain how to build a PEKS scheme from an anonymous-IBE scheme. Finally, we give an overview on the existing work on this topic.

### 3.2.1 Definition

Consider a scenario in which one ore more senders encrypt messages and store them in a database or server. Consider also that the receiver of the messages is interested in retrieving only the ones that fulfill some criteria. For example, messages that contain certain type of information, urgent messages or messages sent by a specific sender. An inefficient solution that respects receiver's privacy would be to give her all the encrypted messages and make her decrypt all the messages in order to search for the ones she needs to obtain. However, it would be desirable that the server or database could do this work instead of the receiver. For this purpose, the senders attach to each message one or several keywords by means of which the receiver can select the messages she wants to get. If these keywords were not encrypted, then the privacy of the receiver would not be respected because the server would learn the keywords she is looking for when she sends them in order to make the server search for the messages that were attached to these keywords. Nevertheless, if Public-key Encryption with Keyword Search is implemented, then the senders can attach encrypted keywords to the messages and the receiver can compute trapdoors that allow the server to test if a message was attached to a specific keyword without decrypting the messages (as with non-encrypted keywords) and without learning the keyword (although this is not fully achieved, see Section 3.2.2). PEKS was also applied to scenarios in which the keywords are not encrypted using

the public key of the receiver and thus she cannot compute trapdoors herself. Namely, she needs to query a third party that might decide whether the receiver should be allowed to search for messages attached to a specific keyword and gives her the trapdoor in that case (see Section 3.2.2).

From now on we focus on the formal definition of PEKS and its security and consistency properties, whereas the application of PEKS will be detailed in Section 3.2.2. Therefore, we define PEKS as follows:

**Definition 56 (PEKS)** *A non-interactive public-key encryption with keyword search scheme consists of a set of four probabilistic polynomial time algorithms (SetupPEKS, PEKS, Trapdoor, Test):*

**SetupPEKS**($1^k$)**.** *On input a security parameter $k$, it outputs a secret key $s_k$ and params, the parameters of the scheme.*

**PEKS**($params$,$W$,$M$)**.** *On input a message $M$ and a keyword $W$, it computes a searchable encryption $S_{W,M}$ for $W$.*

**Trapdoor**($params$,$s_k$,$W$)**.** *On input a secret key $s_k$ and a keyword $W$, it computes a trapdoor $T_W$ for $W$.*

**Test**($params$,$S_{W,M}$,$T_{W'}$)**.** *On input a searchable encryption $S_{W,M}$ and a trapdoor $T_{W'}$, it outputs $M$ if $W = W'$ and $\perp$ otherwise.*

Generically speaking, the scheme works as follows: first, algorithm *SetupPEKS* is run and the party that computes the trapdoors is given the secret key. When a sender wants to send an encrypted message to a receiver, he attaches to this message several searchable encryptions computed by means of the $PEKS$ algorithm. Then algorithm $Trapdoor$ is run in order to obtain trapdoors for specific keywords. Finally, algorithm $Test$ allows to check if a message was attached to a specific keyword by using each searchable encryption with the trapdoor corresponding to that keyword.

Note that the definition given here is more general than the original definition of PEKS given in [BDOP04]. The difference consists in introducing a message as input of the $PEKS$ algorithm, which is obtained afterwards when computing the $Test$ algorithm with a searchable encryption and a trapdoor for the same keyword as inputs. In the original definition there was no message and the output of the *Test* algorithm was "yes" or "no". By means of this message we allow the sender to include some useful information in the searchable encryption. For example, it can be used by the sender to include the symmetric key that was used to encrypt the message to which the searchable encryption was attached (see Section 3.2.2). There are PEKS schemes, like the first one given in [BDOP04], in which introducing this message is not possible and thus the original definition suits better, but

their definition can be seen as a special case where $M = \epsilon$. In addition, Definition 56 is interesting for us because we can instantiate it when deriving a PEKS scheme from an anonymous-IBE scheme (see Section 3.2.3).

In order to define privacy and consistency for PEKS, we have adapted the definitions of [ABC+05] in order to apply them to the general case in which the sender can introduce a message in the searchable encryptions.

Like in IBE, security means privacy. Usually, the notion used to ensure privacy in PEKS is indistinguishability under chosen plaintext attack (PEKS-IND-CPA). This means that an adversary cannot distinguish between two searchable encryptions for keywords of his choice, even when he is allowed to obtain trapdoors for any non-challenge keywords. In addition, we also require that the searchable encryptions do not leak any information about the message introduced in them. Therefore, privacy in PEKS is defined as follows:

**Definition 57 (PEKS-IND-CPA)** *A PEKS scheme is said to be PEKS-IND-CPA secure if there exists a negligible function $\nu(k)$ such that:*

$$\begin{aligned}\Pr[\, & WSet \leftarrow \emptyset; (s_k, params) \leftarrow SetupPEKS(1^k);\\ & (W_0, W_1, M_0, M_1, state) \leftarrow A^{\leftrightarrow Oracle_{Trapdoor}(\cdot)}(params);\\ & b \leftarrow \{0,1\}; c \leftarrow \{0,1\}; S_{W_b,M_c} \leftarrow PEKS(params, W_b, M_c);\\ & (b', c') \leftarrow A^{\leftrightarrow Oracle_{Trapdoor}(\cdot)}(params, S_{W_b,M_c}, state);\\ & b' \leftarrow 0 \wedge c' \leftarrow 0 \wedge WSet \cap \{W_0, W_1\} \neq \emptyset : b = b' \wedge c = c'] < 1/4 + \nu(k)\end{aligned}$$

*In the expression above, $Oracle_{Trapdoor}(W)$ works as follows: it adds $W$ to the set, $WSet \leftarrow WSet \cup \{W\}$, and it returns $T_W \leftarrow Trapdoor(params, s_k, W)$.*

More intuitively, we can describe PEKS-IND-CPA by means of a PEKS-IND-CPA game:

1. Run algorithm $SetupPEKS(1^k)$ and give the parameters *params* to the adversary.

2. The adversary makes adaptive queries to $Oracle_{Trapdoor}(W)$ and gets back the trapdoors for these keywords.

3. The adversary outputs two keywords $W_0, W_1$ and two messages $M_0, M_1$.

4. Pick two random bits $b$ and $c$ and run $PEKS(params, W_b, M_c)$ to obtain a searchable encryption $S_{W_b,M_c}$ for keyword $W_b$, using also message $M_c$. Give $S_{W_b,M_c}$ to the adversary.

5. The adversary continues making adaptive queries to $Oracle_{Trapdoor}(W)$.

**6.** The adversary outputs two bits $b'$ and $c'$ in order to guess which keyword and which message were used to compute the searchable encryption. If the keywords $W_0$ or $W_1$ were used to query $Oracle_{Trapdoor}$ in Step 2 or in Step 5 the game returns $b' = 0$ and $c' = 0$. Otherwise it returns $b'$ and $c'$.

The advantage of the adversary is $Adv(A) = |Pr[b = b' \wedge c = c'] - 1/4|$. As in Definition 57, we say that the PEKS scheme is PEKS-IND-CPA secure when the advantage of the adversary is negligible.

On the other hand, according to [ABC+05], consistency in PEKS can be divided into two parts. The first one requires that, given a searchable encryption and a trapdoor computed using the same keyword, algorithm $Test$ never outputs $\perp$. More formally, given $W \in \{0,1\}^*$ and $M \in \{0,1\}^*$:

$$\begin{aligned}\Pr[&(s_k, params) \leftarrow SetupPEKS(1^k); S_{W,M} \leftarrow PEKS(params, W, M);\\ &T_W \leftarrow Trapdoor(params, s_k, W);\\ &M' \leftarrow Test(params, S_{W,M}, T_W) : M' = M] = 1\end{aligned}$$

This condition will be always required for correctness, and thus we reserve the term "consistency for PEKS" for the second condition. This second condition states that when the searchable encryption and the trapdoor were computed by using different keywords, then algorithm $Test$ should output $\perp$. Despite of what one could expect, there is not any known PEKS scheme that fulfills this property, which is thus called perfect consistency. In order to label the consistency of a concrete PEKS scheme, [ABC+05] proposed to run a game with an adversary that aims at representing the worst-case real life behavior. By means of this game they considered two relaxations for consistency, statistical and computational:

**Definition 58 (Consistency for PEKS)** *Consider a PEKS scheme (SetupPEKS, PEKS, Trapdoor, Test) and the following probability P:*

$$\begin{aligned}\Pr[&(s_k, params) \leftarrow SetupPEKS(1^k); (W, W', M) \leftarrow A(params);\\ &S_{W,M} \leftarrow PEKS(params, W, M); T_{W'} \leftarrow Trapdoor(params, s_k, W');\\ &M' \leftarrow Test(params, S_{W,M}, T_{W'}) : M' \neq \perp]\end{aligned}$$

*Then this PEKS scheme is said to be:*

***perfectly consistent*** *if, for all computationally unbounded adversaries, $P = 0$.*

***statistically consistent*** *if, for all computationally unbounded adversaries, there exists a negligible function $\nu(k)$ such that $P \leq \nu(k)$.*

***computationally consistent*** *if, for all probabilistic polynomial time adversaries, there exists a negligible function $\nu(k)$ such that $P \leq \nu(k)$.*

The main difference between this definition and the original consists in allowing the adversary to output a message along with the keywords on which he wishes to be challenged.

### 3.2.2 Applications

We can distinguish two kinds of application scenarios for PEKS: one in which the searchable encryptions are computed using the public key of the receiver of the messages, who therefore is also the one who compute the trapdoors, and another one in which they are computed using the public key of a third party, either because there will be more than one receiver or because the receiver is not known at this moment, and thus each receiver needs to query this third party to get the trapdoors.

In this subsection we discuss both scenarios by showing two concrete applications: a mail server for the first scenario and an audit log for the second one, and we point out their shortcomings.

#### Mail Server

This application was proposed in [BDOP04] and it is depicted in Figure 3.3. We have a setting with three parties: senders, receivers and a mail server. At the beginning, the receiver runs $SetupPEKS(1^k)$ in order to obtain a secret key $s_k$ and to publish the parameters $params$ of the scheme. Then a sender who wants to send an email with a set of keywords $W = \{W_1, \ldots, W_n\}$ encrypts the content $msg$ of the email under the public key of the receiver (given in $params$) and runs $S_{W_j,\epsilon} = PEKS(params, W_j, \epsilon)$, for all $j \in 1, \ldots, n$, in order to obtain the searchable encryptions $S = \{S_{W_1,\epsilon}, \ldots, S_{W_n,\epsilon}\}$. Finally, he sends $[Enc_{p_k}(msg)||S]$.

Now suppose a receiver who wants to get emails containing a specific keyword $\hat{W}$. She runs $Trapdoor(params, s_k, \hat{W})$ to get a trapdoor $T_{\hat{W}}$ and she sends it to the mail server. After that, the mail server runs $Test(params, S_{W_j,\epsilon}, T_{\hat{W}})$ for the elements of the set $S$ of searchable encryptions of each email in order to know which emails are described by keyword $\hat{W}$. Finally, it sends these emails to the receiver.

The main purpose of this application is that the receiver can delegate the computation of the $Test$ algorithm to a third party (the mail server) in order to retrieve only the messages that are attached to keywords of her choice. Note that in the original proposal there was no message $M$ introduced in the PEKS element and thus the output of the $Test$ algorithm was always "yes" or "no". In Figure 3.3 we can instantiate this by sending an useless message $M = \epsilon$ (do not confuse $M$ with the email). [BDOP04] suggest having a small number of keywords. For example, the words in the subject line may act as keywords.

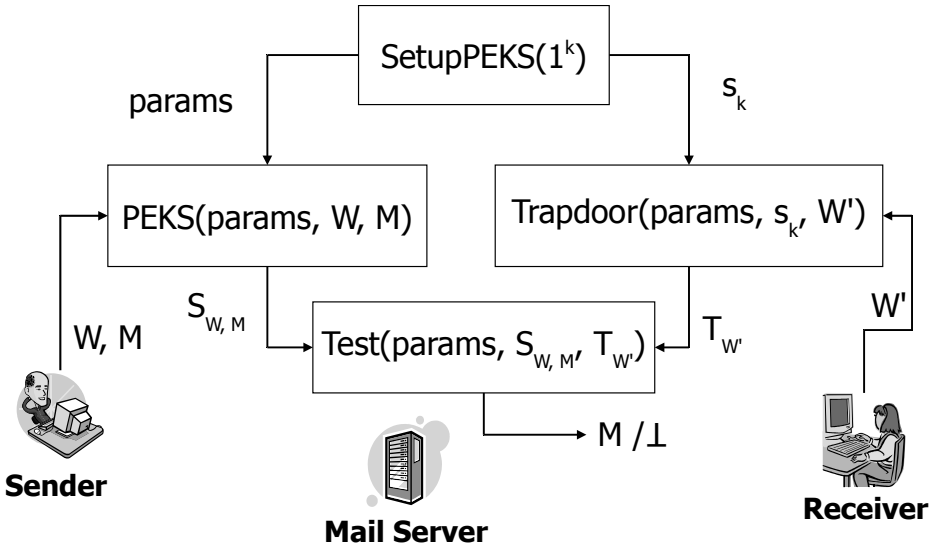


Figure 3.3: PEKS in the mail server scenario.

**Discussion:** This construction does not ensure full privacy for the keywords selected by the receiver. Namely, the server can store trapdoors sent by the receiver and it is able to compute searchable encryptions for any keyword. In addition, although the set of keywords is, in theory, big, in real life applications there will be a subset of keywords frequently attached by senders and requested by receivers. Therefore, the mail server can compute searchable encryptions for this subset and check by means of the *Test* algorithm if the receiver has sent a trapdoor corresponding to a specific keyword.

One may argue that a sender and a receiver can agree on a particular *nonce* and concatenate it to each keyword in order to make the computation of searchable encryptions by the server impracticable. However, as stated in [BSNS], this is not a right solution since PEKS aims at providing keyword search without interaction between senders and receivers. [BSNS] suggest concatenating some time period information to keywords when computing searchable encryptions and trapdoors in order to limit the time in which the server can look for searchable encryptions that work with these trapdoors. This concept was improved and formalized by [ABC+05] as Public-key Encryption with Temporary Keyword Search (PETKS). However, PETKS is not useful to limit the time period as proposed by [BSNS] because, even if the time period has expired, the server can still compute searchable encryptions with past time periods in order to guess the keyword corresponding to a trapdoor (although the trapdoor is not useful anymore). Therefore, PETKS can only help by incrementing the size of the subset of frequently used keywords and by making it impossible for the server to check if a trap-

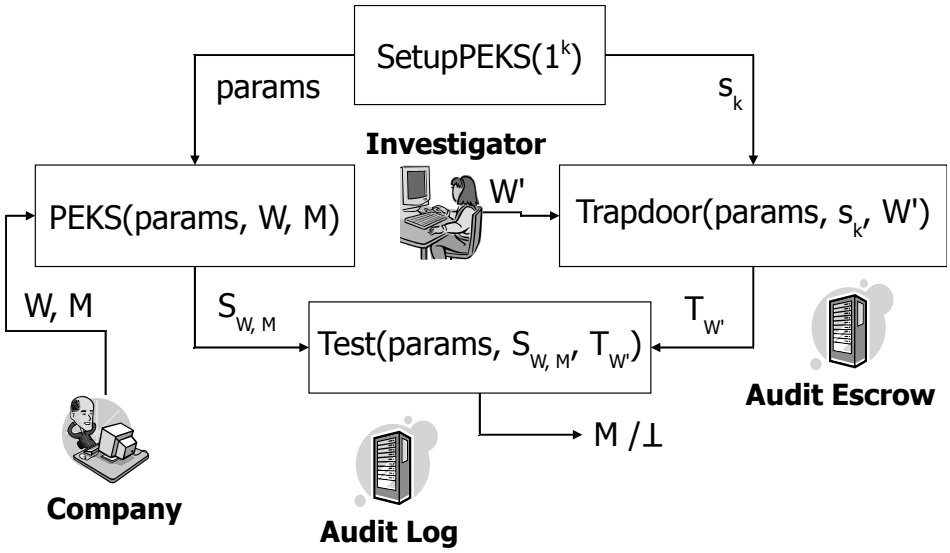


Figure 3.4: PEKS in the audit log scenario.

door works with searchable encryptions submitted by senders in a different time period.

### Audit Log

This application was proposed in [WBDS04] and it is depicted in Figure 3.4. We have a setting with three parties: a company or similar entity that stores information about its activity in an audit log, an investigator who wants to retrieve some information from the audit log and an audit escrow, who gives authorization to the investigator depending on what she wants to search for. At the beginning, the company runs $SetupPEKS(1^k)$ in order to obtain a secret key $s_k$ and to publish the parameters of the scheme $params$. Then, in order to add a record with keywords $\{W_1, \ldots, W_n\}$ to the audit log, the company picks at random a symmetric encryption key, $K$, and encrypts the record using this key. Then, for each $W_j$, where $1 \leq j \leq n$, the company computes a searchable encryption $S_{W_j,K} = PEKS(params, W_j, K)$, and finally the encrypted record is stored with its searchable encryptions. After that, when an investigator wants to get information related with a specific keyword $W'$, she sends it to the audit escrow. If the audit escrow authorizes her, it runs $T_{W'} = Trapdoor(params, s_k, W')$ and sends her $T_{W'}$. Finally, the investigator that has access to the encrypted audit log uses this trapdoor to search for the records attached to keyword $W'$. When the $Test$ algorithm does not reject, she gets the encryption key $K$ used to encrypt the record and finally she decrypts it and gets the information.

**Discussion:** The main difference between this scenario and the mail

server scenario consists in computing the searchable encryptions using the public key of a third party (the audit escrow) different from the receiver or investigator. Thus the investigator has to get the trapdoors from this party, who may or may not authorize her. Apart from that, in this kind of scenario we avoid the problem of leaking information about the keyword choice of the investigator, either because we can think that the audit escrow also belongs to the company, so the company inevitably knows the keywords, or because we can think that the investigator is able to access the audit log in a private manner, e.g., taking a private copy of the audit log. In this case, the searchable encryptions ensure that the investigator learns nothing about records described by keywords for which she has not got a trapdoor.

**Remark**

We notice that in the audit log scenario the audit escrow learns the keywords. In this scenario it makes sense because the audit escrow has to give authorization depending on the keyword requested by the investigator. However, there exist applications in which the party that should give authorization is not the one able to compute the trapdoors. In that case, it is undesirable that the party that computes the trapdoors learns the keywords. Therefore, in Chapter 4 we provide a solution by means of which receivers can get trapdoors from this party without revealing the keywords.

### 3.2.3 Instantiation using Anonymous-IBE

As demonstrated in [ABC+05], there exists a generic construction by means of which it is possible to build a PEKS scheme from any anonymous-IBE scheme. In the following, we modify this transformation to adapt it to our definition of PEKS and we also prove the privacy and the consistency of the resultant PEKS scheme when the IBE scheme is private and anonymous.

**Transformation 1 (IBE-to-PEKS)** *Given an anonymous-IBE scheme with algorithms (SetupIBE, Extract, Encrypt, Decrypt) the PEKS scheme works as follows:*

**SetupPEKS**$(1^k)$**.** *On input a security parameter $k$, it runs $SetupIBE(1^k)$ to obtain the secret key $s_k$ and params, the parameters of the scheme.*

**PEKS**$(params, W, M)$**.** *On input a keyword $W$ and a message $M$, it picks a random value $C_2 \in \{0,1\}^k$ and computes $C_1 = Encrypt(params, W, C_2||M)$. It outputs the tuple $S_{W,M} = (C_1, C_2)$.*

**Trapdoor**$(params, s_k, W)$**.** *The trapdoor $T_W$ associated with the keyword $W$ is the secret key $sk_W$ associated with this keyword (acting as an identity), so it can be obtained by running $T_W = Extract(params, s_k, W)$.*

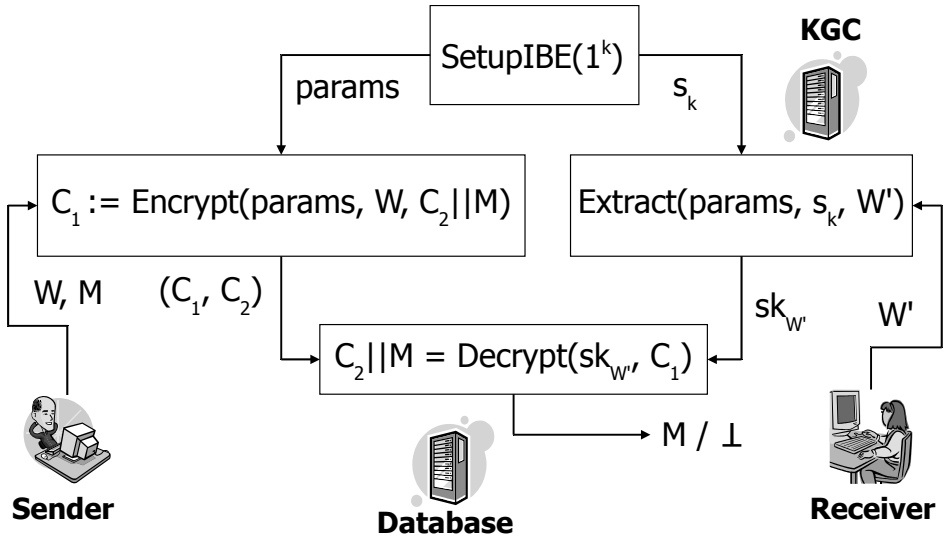


Figure 3.5: Transformation from an IBE scheme to a PEKS scheme.

**Test**$(params, S_{W,M}, T_{W'})$**.** *On input the searchable encryption* $S_{W,M}$ *and the trapdoor* $T_{W'}$*, it outputs* $M$ *if* $C_2||M = Decrypt(T_{W'}, C_1)$ *and* $\perp$ *otherwise.*

The construction is depicted in Figure 3.5. Intuitively, the main idea behind it is to utilize the keyword $W$ as an identity used in the *Encrypt* algorithm when computing the $PEKS$ algorithm, and thus the trapdoor that has to be used when running the $Test$ algorithm is the secret key corresponding to that identity. As explained in Section 3.2.2, the parties computing each algorithm can vary depending on the application. This construction is very similar to the second scheme by Boneh et al. [BDOP04] (see Section 3.2.4), but instead of always using $0^k$ as the message that is encrypted, here a random message $C_2$ is chosen and is attached in the clear to the ciphertext.

Note that we have adapted this transformation to our general definition of PEKS in which a message can be introduced when computing a searchable encryption, so the proof given in [ABC+05] needs to be adequately modified.

**Theorem 2** *Let IBEsch be an IBE scheme and let PEKSsch be the PEKS scheme derived from IBE via IBE-to-PEKS. If IBEsch is IBE-IND-CPA secure, then PEKSsch is computationally consistent. Moreover, if IBEsch is IBE-ANO-CPA secure and again IBE-IND-CPA secure, then the PEKS scheme is PEKS-IND-CPA secure.*

*Proof.* First, we prove that an IBE-IND-CPA secure IBE scheme leads to a computationally consistent PEKS scheme. Let $E$ be any probabilistic

polynomial time adversary attacking the consistency of the PEKS scheme. We build a probabilistic polynomial time algorithm $A$ that attacks the IBE-IND-CPA security of the IBE scheme. First, the adversary $A$ runs $E$ in order to obtain the keywords $(W, W')$ and the message $M$. Then $A$ outputs the keyword $W$ and the messages $R_0||M, R_1||M$, where $R_0, R_1 \in \{0,1\}^k$, on which he wishes to be challenged. Given challenge ciphertext $C = Encrypt(params, W, R_b||M)$, $A$ queries $Oracle_{Extract}(W')$ to obtain the trapdoor $T_{W'}$ for keyword $W'$. At this point, recall that the advantage of $E$ attacking the consistency of the PEKS scheme is the probability of finding two keywords $W, W'$ and a message $M$ such that the $Test$ algorithm outputs $M$ on inputs a searchable encryption $S_{W,M}$ and a trapdoor $T_{W'}$. When using IBE-to-PEKS this equals the probability that $C_2||M = Decrypt(T_{W'}, C_1)$, where $C_1 = Encrypt(params, W, C_2||M)$. Therefore, it can be seen that after getting the trapdoor from $Oracle_{Extract}(W')$, $A$ can check whether $Decrypt(T_{W'}, C) = R_1||M$. Assume now that $A$ outputs bit $b' = 1$ when this happens. It can be easily seen that the probability of $A$ guessing bit $b$ when $b = 1$, i.e., $Pr[b = b'|b = 1]$, is at least the advantage of $E$ in breaking the consistency of the PEKS scheme, i.e., $Adv(E) \leq Pr[b = b'|b = 1]$. However, it does not mean that $A$ has broken the security of the IBE scheme, because it is possible that $b = 0$ and $Decrypt(T_{W'}, C) = R_1||M$. Fortunately, we can bound this probability as follows: $Pr[b \neq b'|b = 0] \leq 2^{-k}$, because if $b = 0$ the ciphertext $C = Encrypt(params, W, R_0||M)$ is independent of the random string $R_1$, so the probability that it decrypts to $R_1$ is $2^{-k}$. Therefore, defining the advantage of $A$ as his probability of guessing when $b = 1$ minus his probability of failing when $b = 0$, i.e., $Adv(A) = Pr[b = b'|b = 1] - Pr[b \neq b'|b = 0]$, we have that $Adv(E) \leq Pr[b = b'|b = 1]$ and thus:

$$Adv(E) \leq Adv(A) + 2^{-k}$$

Therefore, if the advantage of $A$ in breaking the IBE-IND-CPA security is negligible then the advantage of $E$ in breaking the consistency of the PEKS scheme is also negligible, so the first claim of Theorem 2 is proved.

Second, we prove that an IBE-IND-ANO-CPA secure IBE scheme leads to a PEKS-IND-CPA secure PEKS scheme. Let $E$ be any probabilistic polynomial time adversary attacking the PEKS-IND-CPA security of the PEKS scheme, and let $A$ be a probabilistic polynomial time adversary attacking the IBE-IND-ANO-CPA security of the IBE scheme. First, $A$ runs $E$ and gets the challenge keywords $W_0, W_1$ and the challenge messages $M_0, M_1$. $A$ answers the trapdoor oracle of $E$ by means of his extraction oracle in every stage. Then, $A$ outputs two keywords $W_0, W_1$ and two challenge messages $R_0||M_0$ and $R_1||M_1$, where $R_0, R_1 \in \{0,1\}^k$. Given the challenge ciphertext $C = Encrypt(params, W_b, R_c||M_c)$, it runs $E$ on input $C$ to get bits $(b', c')$,

which $A$ returns. It is easy to see that, for $b, c \in \{0, 1\}$, the probability of guessing $(b, c)$ in the IBE-IND-ANO-CPA game equals the probability of guessing $(b, c)$ in the PEKS-IND-CPA game.

In conclusion, the advantage $Adv(E)$ equals the advantage of $A$ in breaking both the anonymity and the privacy of the IBE scheme at the same time, and thus if the advantage of $A$ in breaking IBE-IND-ANO-CPA is negligible, then the PEKS scheme is PEKS-IND-CPA secure. □

Before the results of [ABC$^+$05], [BDOP04] tried to demonstrate that it is possible to build an IBE scheme from any PEKS scheme. In fact, they proposed another generic transformation from PEKS to IBE. However, this transformation requires a perfectly consistent PEKS scheme and until now none is known. Therefore, if we apply a statistically or computationally consistent PEKS scheme as input of the transformation we obtain, respectively, a statistically or computationally consistent IBE scheme and this does not match the standard definitions for IBE schemes (see Section 3.1.1).

### 3.2.4 Historical Overview

Previously to the existence of PEKS, the applications for searching on encrypted data can be divided into two groups: private databases and public databases.

In private databases we have a scenario in which a user sends encrypted data to a database and afterwards she wants to retrieve records containing keywords of her choice. In the early 1990's two solutions were proposed by Ostrovsky [Ost] and by Ostrovsky and Goldreich [GO96]. These solutions require poly-logarithmic rounds in the size of the database and poly-logarithmic work per query in the database. In addition, they can hide the access pattern of the user from the database. More recently, Song et al. [SWP00] published a new solution that requires little communication between the user and the database (proportional to the security parameter) and only one round of interaction.

In public databases we consider a scenario in which the data is publicly accessible and the user wants to hide the records she wants to get or search for. Private Information Retrieval (PIR) permits this with little communication. PIR was shown to be possible in a scenario where there are several copies of the database that cannot communicate with each other by Chor et al. [CGKS95]. Afterwards, Kushilevitz and Ostrovsky [KO97] proposed a solution that requires only one instance of the database in which the communication complexity is $O(n^\alpha)$, where $n$ is the size of the database and $\alpha > 0$. This was reduced to poly-logarithmic overhead by Cachin et al. [CMS99]. In addition, [KO97] shown that PIR can be extended to 1-out-of-N Oblivious Transfer.

However, the mail server scenario does not correspond with none of the two above-mentioned problems. Unlike in the private database problem,

in PEKS the data is sent by third parties and thus it cannot be stored conveniently, and unlike in PIR, here the information is private.

In 2004 the notion of PEKS was defined formally by Boneh et al. [BDOP04]. They also gave a definition of semantic security and explained how to achieve CCA security. In addition, they proposed a theoretical transformation from a perfectly consistent PEKS scheme to an IBE scheme (see Section 3.2.3), and they proposed two constructions for PEKS. The first one is based on the Boneh and Franklin IBE scheme [BF01]. It uses bilinear pairings and is secure under the BDH assumption in the random oracle model. The second one assumes that the total number of keywords is bounded by a polynomial function in the security parameter. Then, claiming that by means of any trapdoor permutation family they can construct a semantically secure anonymous encryption scheme, they constructed a simple semantically secure scheme using trapdoor permutations.

Waters et al. [WBDS04] shown that by using another PEKS scheme based on [BF01], it is possible to construct encrypted and searchable audit logs. They compared two solutions: the first uses symmetric keys and the second, based on [BF01], public keys. They concluded that the security of the second scheme is much better than the security of the first one, although its efficiency is worse.

In 2005, Baek et al. [BSNS] addressed three issues that in [BDOP04] were not considered. The first one, "refreshing keywords", is motivated by the fact that the server could store the trapdoors sent by the receiver and, since it is probable that the keywords are reused frequently, then for new data it could know if a searchable encryption is related with a trapdoor without having the receiver sent a new trapdoor. They proposed to concatenate the keyword with the date on which the searchable encryption is computed to make the trapdoors expire. The second one, "removing secure channel", is related with the fact that trapdoors must be sent through an encrypted and authenticated channel to the server in the schemes by [BDOP04]. They proposed a new efficient scheme in which a secure channel is not needed because public and private keys of the server and the receiver are generated by using a common parameter. Finally, they show that in [BDOP04], when the sender wants to attach more than one encrypted keyword to a message, he just generates them separately and finally concatenates them. Then they proposed a more efficient scheme to attach several searchable encryptions to a message.

Abdalla et al. [ABC+05] explained the notion of privacy and consistency for PEKS. They defined three types of consistency: perfect (stronger), statistical and computational (weaker), and they demonstrated that the first scheme by [BDOP04] is only computationally consistent. Therefore, they proposed a new statistically consistent scheme. In addition, they stated that the second scheme by [BDOP04], from which, in principle, one could think on designing a PEKS scheme by using an anonymous IBE scheme as

the semantically secure source indistinguishable encryption scheme, does not always yield to a computationally consistent PEKS scheme, so they proposed a similar transformation that provides computational consistency (and also semantic security). Therefore, they demonstrated that anonymous IBE implies PEKS. The authors also gave a sufficient condition for anonymity in IBE and in hierarchical IBE. They also provided an anonymous-at-level-1 HIBE scheme (a fully anonymous one was proposed in 2006 by Boyen and Waters [BW06]). Apart from that, they considered the problem of "refreshing keywords", which they formalized as Public-key Encryption with Temporary Keyword Search (PETKS). In their definition, the sender uses the time period in which the keyword is encrypted to generate the searchable encryption and the receiver uses a time interval to compute the trapdoor such that the *Test* algorithm outputs "yes" on inputs this trapdoor and any searchable encryption whose time period belongs to the time interval of the trapdoor. They provided two constructions, one of which uses the anonymous-at-level-1 HIBE scheme. Finally, they defined the concept of Identity-Based Encryption with Keyword Search (IBEKS), i.e., a PEKS scheme in which the identity of the receiver is used to encrypt keywords and trapdoors are generated by using the secret key corresponding to this identity. They provided a generic transformation from a level-2 anonymous HIBE scheme to an IBEKS scheme.

## 3.3 Oblivious Transfer

Oblivious Transfer (OT) is a cryptographic primitive that allows a receiver to obtain one message from a set of messages offered by the sender without letting the sender know which one, whereas the sender is guaranteed that the receiver does not learn anything about the rest of the messages. It has potential uses for many applications as a privacy protection mechanism, e.g., for patent searches, location-based services or medical databases [NP99b]. On the other hand, it has been demonstrated that any secure multiparty computation protocol can be realized if OT is possible [GMW87].

In this section we give a definition of OT, we talk about its security properties and we give an historical overview about the different notions of security considered for OT. After that, we explain how to build an OT scheme by using a blind-IBE scheme (see Section 3.1.3), and finally we review the existing work on OT.

### 3.3.1 Definition

In OT we have a scenario that involves a sender and a receiver. The sender has a set of messages $M = \{M_1, \ldots, M_N\}$ and the receiver a selection value $\sigma \in \{1, \ldots, N\}$, which means that she wants to obtain the message $M_\sigma$. By means of OT, the receiver gets the message $M_\sigma$ in such a way that the sender

does not learn anything about $\sigma$ and the receiver does not learn anything about the other messages.

The concept of OT has been evolving over time, from 1-out-of-2 OT ($OT_1^2$) to 1-out-of-N OT ($OT_1^N$), and finally to k-out-of-N OT ($OT_k^N$). The first one means that the sender has two messages and the receiver gets one of them. $OT_1^2$ and $OT_1^N$ are equivalent in the information theoretic sense [BCR86b], which means that it is possible to implement $OT_1^N$ using $OT_1^2$ without losing security.

In $OT_k^N$ we have a commit phase and a transfer phase. In the commit phase the sender commits to $N$ messages and sends the commitments to the receiver. During the transfer phase the sender and the receiver interact in such a way that the receiver can obtain $k$ messages of her choice by using the commitments. We distinguish two settings: non-adaptive $OT_k^N$, where the receiver states at the beginning of the transfer phase which messages she wants to obtain, and adaptive $OT_k^N$ (also $OT_{k\times 1}^N$), where the receiver is able to decide which message she wants to get in the transfer subphase $i \in \{1, \ldots, k\}$ after $i-1$ subphases, in such a way that her decision can depend on the messages obtained before.

More formally, according to [CNS07], $OT_{k\times 1}^N$ is defined as follows:

**Definition 59** (**$OT_{k\times 1}^N$**) *An $OT_{k\times 1}^N$ scheme is a tuple of algorithms $(S_I, R_I, S_T, R_T)$ used in matched pairs. During the initialization phase the sender runs $S_I(M_1, \ldots, M_N)$ and obtains state value $S_0$, and the receiver runs $R_I()$ and obtains $R_0$. During the transfer phase sender and receiver execute $k$ times $(S_T, R_T)$. During the $i$th $(1 \leq i \leq k)$ transfer the sender runs $S_T(S_{i-1})$ to obtain $S_i$, and the receiver runs $R_T(R_{i-1}, \sigma_i)$ to obtain state value $R_i$ and the message $M'_{\sigma_i}$ (or $\perp$ indicating failure).*

In this definition, correctness requires that $M'_{\sigma_i} = M_{\sigma_i}$ for all messages $(M_{\sigma_1}, \ldots, M_{\sigma_k})$, for all selections $(\sigma_1, \ldots, \sigma_k) \in \{1, \ldots, N\}$ and for all coin tosses of the algorithms.

The notion of security in OT has also been evolving as time goes by, from an honest-but-curious model to a half-simulation model, and recently to a full-simulation model.

In the honest-but-curious model all the parties behave honestly. Security guarantees that after running the protocol any curious party cannot learn anything else by analyzing the transcript.

In the half-simulation model [NP05] security holds separately for senders and receivers. Security for receivers implies that the sender's view of the protocol when the receiver chooses $\sigma$ is indistinguishable from his view when she chooses $\sigma'$. However, security for senders involves a stronger notion following the ideal-world/real-world paradigm, by means of which it is possible to describe an ideal-world counterpart for every real-world malicious

receiver such that the adversary in the real world gains no more information as counterpart in an ideal world in which OT is implemented by a trusted third party.

The half-simulation model is vulnerable to selective-failure attacks. Namely, a sender is always able to make a transfer fail by sending bogus messages during the initialization phase. However, it is possible to prevent him from making the transfers fail depending on some property of the receiver's selection or to prevent him from making a different message fail in different transfers within the same transfer phase. Although in such an attack the sender gains no information about the receiver's selection, the receiver cannot complain because she would lose her privacy. Therefore, a full-simulation model [CNS07], in which both sender and receiver security follow the ideal-world/real-world paradigm was defined to avoid this kind of attacks. In the real experiment both parties run the protocol, but in the ideal experiment the functionality is implemented by a trusted third party. In this model it is also required that the combined outputs of the sender and the receiver be indistinguishable, not only the output of the malicious party.

According to [CNS07], security for $OT^N_{k\times 1}$ can be defined as follows ($\kappa$ is the security parameter):

**Definition 60 (Security for $OT^N_{k\times 1}$)**

*Real experiment. In* $Real_{\tilde{S},\tilde{R}}(N, k, M_1, \ldots, M_N, \sigma_1, \ldots, \sigma_k)$ *the possibly cheating sender* $\tilde{S}$ *has messages* $(M_1, \ldots, M_N)$ *as input and interacts with possibly cheating receiver* $\tilde{R}$*. In their first run, both* $\tilde{S}$ *and* $\tilde{R}$ *output initial states* $(S_0, R_0)$*. Then, for* $1 \leq i \leq k$ *the sender computes* $S_i \leftarrow \tilde{S}(S_{i-1})$*, and the receiver computes* $(R_i, M'_{\sigma_i}) \leftarrow \tilde{R}(R_{i-1}, \sigma_i)$*, where* $\sigma_i \in \{1, \ldots, N\}$ *is a message index and* $M'_{\sigma_i}$ *equals* $M_{\sigma_i}$ *if neither participant cheats. At the end of the* $k$*th transfer the output of the experiment is* $(S_k, R_k)$*. We define the honest sender* $S$ *as the one that runs* $S_I(M_1, \ldots, M_N)$ *in the initialization phase, runs* $S_T(S_{i-1})$ *in the* $i$*th transfer phase and outputs* $S_k = S_0$*, and the honest receiver* $R$ *as the one that runs* $R_I()$ *during the initialization phase, runs* $R_T(R_{i-1}, \sigma_i)$ *in the* $i$*th transfer and returns* $R_k = R_0$ *and the list of received messages* $\{M'_{\sigma_1}, \ldots, M'_{\sigma_k}\}$*.*

*Ideal experiment. In* $Ideal_{\tilde{S}',\tilde{R}'}(N, k, M_1, \ldots, M_N, \sigma_1, \ldots, \sigma_k)$ *the possibly cheating sender algorithm* $\tilde{S}'(M_1, \ldots, M_N)$ *generates messages* $(M'_1, \ldots, M'_N)$ *and transmits them to a trusted party* $T$*. In the* $i$*th round* $\tilde{S}'$ *sends a bit* $b_i$ *to* $T$ *and the possibly cheating receiver* $\tilde{R}'(\sigma_i)$ *transmits* $\sigma'_i$ *to* $T$*. If* $b_i = 1$ *and* $\sigma'_i \in \{1, \ldots, N\}$ *then* $T$ *hands* $M'_{\sigma'_i}$ *to* $\tilde{R}'$*. Otherwise it hands* $\perp$ *to* $\tilde{R}'$*. At the end of the* $k$*th transfer the output of the experiment is* $(S_k, R_k)$*. We define the honest sender* $S'(M_1, \ldots, M_N)$ *as the one that sends messages* $\{M_1, \ldots, M_N\}$ *in the initialization phase, sends* $b_i = 1$ *in the* $i$*th transfer phase and outputs* $S_k = S_0$*, and the honest receiver* $R$ *as the one that sends*

*$\sigma_i$ in the ith transfer and returns $R_k = R_0$ and the list of received messages $\{M'_{\sigma_1}, \ldots, M'_{\sigma_k}\}$.*

*Sender Security. $OT^N_{k\times 1}$ provides sender security if for every real-world probabilistic polynomial time receiver $\tilde{R}$ there exists an ideal-world receiver $\tilde{R}'$ such that for any polynomial $N_m(\kappa)$, any $N \in \{1, \ldots, N_m(\kappa)\}$, any $k \in \{1, \ldots, N\}$, any $(M_1, \ldots, M_N)$ and any indices $\sigma_1, \ldots, \sigma_k \in \{1, \ldots, N\}$, the advantage of every probabilistic polynomial time distinguisher in distinguishing $Real_{S,\tilde{R}}(N, k, M_1, \ldots, M_N, \sigma_1, \ldots, \sigma_k)$ and $Ideal_{S',\tilde{R}'}(N, k, M_1, \ldots, M_N, \sigma_1, \ldots, \sigma_k)$ is negligible in $\kappa$.*

*Receiver Security. $OT^N_{k\times 1}$ provides receiver security if for every real-world probabilistic polynomial time sender $\tilde{S}$ there exists an ideal-world sender $\tilde{S}'$ such that for any polynomial $N_m(\kappa)$, any $N \in \{1, \ldots, N_m(\kappa)\}$, any $k \in \{1, \ldots, N\}$, any $(M_1, \ldots, M_N)$ and any indices $\sigma_1, \ldots, \sigma_k \in \{1, \ldots, N\}$, the advantage of every probabilistic polynomial time distinguisher in distinguishing $Real_{\tilde{S},R}(N, k, M_1, \ldots, M_N, \sigma_1, \ldots, \sigma_k)$ and $Ideal_{\tilde{S}',R'}(N, k, M_1, \ldots, M_N, \sigma_1, \ldots, \sigma_k)$ is negligible in $\kappa$.*

### 3.3.2 Instantiation using Blind-IBE

Green and Hohenberger [GH07] shown how to build an $OT^N_{k\times 1}$ scheme by using any blind-IBE scheme. A simple protocol to build an $OT^N_{k\times 1}$ scheme from any blind-IBE scheme ($SetupIBE$, $BlindExtract$, $Encrypt$, $Decrypt$) would consist in the following steps:

**1.** The sender runs the $SetupIBE(1^k)$ algorithm, gets the master secret key $s_k$ and publishes $params$, the parameters of the scheme.

**2.** For every message $M_i$, where $i \in \{1, \ldots, N\}$, the sender computes $C_i = Encrypt(params, i, M_i)$, where $i$ acts as an identity, and sends the ciphertexts to the receiver.

**3.** The receiver engage with the sender $k$ times by running $BlindExtract$ $(S(params, s_k),\ R(params, \sigma_i))$ in order to retrieve the secret key $sk_{\sigma_i}$ for each identity $\sigma_i \in \{\sigma_1, \ldots, \sigma_k\}$.

**4.** The receiver executes $Decrypt(sk_{\sigma_i}, C_i)$ to retrieve each message $M_i$.

However, according to [GH07], it is difficult to show that this protocol is fully simulatable, and thus they did several modifications. They proposed two protocols: one non-adaptive in the standard model and one adaptive in the random oracle model. The latter works as follows:

**Transformation 2 (IBE-to-OT) Initialization phase.** *The sender runs $S_I(M_1, \ldots, M_N)$ and the receiver runs $R_I()$:*

1. *The sender runs* $SetupIBE(1^k)$*, gets the secret key* $s_k$ *and the parameters of the scheme* $params$*. He also chooses a function* $H : M \to \{0,1\}^n$*.*
2. *The sender selects random messages* $\{W_1, \ldots, W_N\} \in Msg(k)$ *and, for* $j = 1, \ldots, N$*, computes* $A_j \leftarrow Encrypt(params, j, W_j)$*,* $B_j \leftarrow H(W_j) \oplus M_j$ *and* $C_j = (A_j, B_j)$*.*
3. *The sender conducts the proof of knowledge* $PK\{(s_k) : (params, s_k) \in SetupIBE(1^k)\}$*.*
4. *The sender transmits* $(params, C_1, \ldots, C_N)$ *to the receiver.*
5. *The receiver aborts if the proof fails.*
6. *The sender outputs state value* $S_0 = (params, s_k)$ *and the receiver outputs state value* $R_0 = (params, C_1, \ldots, C_N)$*.*

**Transfer phase.** *The sender runs* $S_T(S_{i-1})$ *and the receiver runs* $R_T(R_{i-1}, \sigma_i)$ *during the* $i$*th phase* $(i \in \{1, \ldots, k\})$*:*

1. *The receiver engages with the sender in the protocol* $BlindExtract(S(params, s_k), R(params, \sigma_i))$ *in order to obtain the secret key* $sk_{\sigma_i}$ *for identity* $\sigma_i$*.*
2. *The receiver computes* $M'_{\sigma_i} \leftarrow B_{\sigma_i} \oplus H(Decrypt(sk_{\sigma_i}, A_{\sigma_i}))$ *or outputs* $\perp$ *if* $BlindExtract$ *failed.*
3. *The sender outputs state value* $S_i = S_{i-1}$ *and the receiver outputs* $R_i = R_{i-1}$ *and the message* $M'_{\sigma_i}$*.*

In order to construct a similar protocol in the standard model they proposed two options. The first one consists in executing a $OT_1^N$ protocol $k$ times (it uses their construction for $OT_k^N$ with $k = 1$), but it is inefficient for large values of $k$ or $N$. The second option consists in combining their protocol with the protocol of Camenisch et al. [CNS07], which is secure under the $l$-PDDH assumption and the $l$-SDH assumption, in order to keep $l$ small and thus reduce the length of the security parameters.

### 3.3.3 Historical Overview

Introduced by Rabin [Rab81] in 1981, the concept of Oblivious Transfer was extended by Even, Goldreich and Lempel [EGL85], who defined $OT_1^2$, and afterwards by Brassard, Crépeau and Robert ([BCR86b], [BCR86a]), who proposed $OT_1^N$ and gave a construction using $N$ applications of a $OT_1^2$ protocol. In 1990 Bellare and Micali [BM89] shown practical implementations of $OT_1^2$ under the honest-but-curious model.

Under half simulation, in 1999, Naor and Pinkas [NP99a] gave a more efficient construction for $OT_1^N$. They also proposed in [NP99b] the first adaptive $OT_k^N$. In 2004, Ogata and Kurosawa [OK04] and, in 2005, Chu and

Tzeng [CT05] proposed two efficient adaptive $OT_k^N$ schemes in the random oracle model.

In 2007, Camenisch, Neven and shelat [CNS07] solved the selective-failure problem by proving both sender and receiver security in the real-world/ideal-world paradigm. Therefore, they defined the full-simulation model. They also provide two adaptive $OT_k^N$ schemes whose efficiency is comparable to schemes in the half-simulation model. The first one, whose security is proved in the random oracle model, uses a unique blind signature scheme. The schemes by [OK04] and by [CT05] are particular cases of this scheme. The second one, in the standard model, also uses a blind signature scheme, but it is not a blackbox special purpose construction.

Afterwards, Green and Hohenberger [GH07] realized that the schemes given by [CNS07] involve stronger security assumptions than those used to build OT in the half-simulation model. In particular, the first scheme, if instantiated with the unique blind signature schemes due to Chaum [Cha82] or Boldyreva [Bol03], requires interactive complexity assumptions (one-more-inversion RSA and chosen-target CDH, respectively), and the second scheme requires dynamic assumptions like $l$-PDDH or $l$-SDH (see Section 2.7.3). Therefore, they proposed two new $OT_k^N$ schemes, one non-adaptive (without random oracles) and the other one adaptive (in the random oracle model), that only require static complexity assumptions such as DBDH (see Section 2.7.3). These schemes are constructed by using a blind IBE scheme (see Section 3.1). They also discussed how to remove the oracles at an additional cost, but they left as an open problem to achieve the same improvement in terms of complexity assumptions for the second scheme by [CNS07].

## 3.4 Oblivious Keyword Search

The concept of Oblivious Keyword Search (OKS) generalizes Oblivious Transfer. We have a scenario in which, like in the previous case, the receiver wants to get data from a sender without letting the sender know which data was obtained, and the sender wants to reveal no information about the data that was not requested by the receiver. However, in OKS the receiver wants to get all messages related with a specific keyword, whereas in OT she retrieves the message at the position specified by a selection value. Therefore, it can be seen that OT is a particular case in which every record of data is specified by a unique keyword, which is the selection value, and the receiver knows the selection values that specify the records she wants to retrieve.

In this section we give a formal definition of OKS and we talk about its security properties. Then we look more in depth at the relation between OKS and OT and finally we give an overview on the previous work on this topic.

### 3.4.1 Definition

In OKS we have a scenario that involves a sender and a receiver. The sender has a set of message-keyword pairs $M = \{(M_{W_1}, W_1), \ldots, (M_{W_N}, W_N)\}$, where the keywords belong to a keyword space $W_{sp}$, and the receiver has a keyword $\hat{W}_i$, which means that she wants to obtain the message $M_{\hat{W}_i}$. We assume without loss of generality that each keyword is related with at most one message. Namely, as the receiver wants to get all the messages related with keyword $\hat{W}_i$, we can build a message $M_{\hat{W}_i}$ that includes all the information related with this keyword. By means of OKS, the receiver gets the message $M_{\hat{W}_i}$ in such a way that the sender does not learn anything about $\hat{W}_i$ and the receiver does not learn anything about the other messages.

Unlike the concept of OT, which has evolved as time goes by, OKS is a recent concept and the first OKS scheme that was defined is already an adaptive k-out-of-N OKS scheme. Therefore, although it would be possible to start by giving definitions for more simple constructions, such as 1-out-of-N OKS ($OKS_1^N$), we give a definition of the former. Adaptive k-out-of-N OKS can be defined as follows:

**Definition 61 ($OKS_{k\times 1}^N$)** *An $OKS_{k\times 1}^N$ scheme is a tuple of algorithms $(S_I, R_I, S_T, R_T)$ used in matched pairs. During the initialization phase the sender runs $S_I((M_{W_1}, W_1), \ldots, (M_{W_N}, W_N))$, where each $W_j \in W_{sp}$, $1 \leq j \leq N$, is a keyword, and obtains state value $S_0$, and the receiver runs $R_I()$ and obtains $R_0$. During the transfer phase sender and receiver execute k times $(S_T, R_T)$. During the ith transfer, $1 \leq i \leq k$, the sender runs $S_T(S_{i-1})$ to obtain $S_i$, and the receiver runs $R_T(R_{i-1}, \hat{W}_i)$, $\hat{W}_i \in W_{sp}$, to obtain state value $R_i$ and the message $M'_{\hat{W}_i}$ (or $\perp$ indicating failure).*

Notice that in both definitions every message is associated with only one keyword. It would be nice to have a more general definition in which they were associated with several keywords and the receiver could search for several keywords in a round.

So far, the definitions of security for OKS follow the half-simulation model [OK04]. However, it is better to give a definition in the full-simulation model in order not to let the sender launch selective-failure attacks. Therefore, we write the following definition by modifying the definition of security for OT ($\kappa$ is the security parameter):

**Definition 62 (Security for $OKS_{k\times 1}^N$)**

*Real experiment. In $Real_{\tilde{S},\tilde{R}}(N, k, (M_{W_1}, W_1), \ldots, (M_{W_N}, W_N), \hat{W}_1, \ldots, \hat{W}_k)$ the possibly cheating sender $\tilde{S}$ has message-keyword pairs $\{(M_{W_1}, W_1), \ldots, (M_{W_N}, W_N)\}$ as input and interacts with possibly cheating receiver $\tilde{R}$. In their first run, both $\tilde{S}$ and $\tilde{R}$ output initial states $(S_0, R_0)$. Then, for $1 \leq i \leq k$ the sender computes $S_i \leftarrow \tilde{S}(S_{i-1})$, and the receiver computes*

$(R_i, M'_{\hat{W}_i}) \leftarrow \tilde{R}(R_{i-1}, \hat{W}_i)$, *where* $\hat{W}_i \in W_{sp}$ *is a keyword and* $M'_{\hat{W}_i}$ *equals* $M_{\hat{W}_i}$ *if neither participant cheats. Both algorithms update their states to* $S_i$ *and* $R_i$ *respectively. At the end of the* $k$*th transfer the output of the experiment is* $(S_k, R_k)$*. We define the honest sender* $S$ *as the one that runs* $S_I((M_{W_1}, W_1), \ldots, (M_{W_N}, W_N))$ *in the initialization phase, runs* $S_T(S_{i-1})$ *in the* $i$*th transfer phase and outputs* $S_k = S_0$*, and the honest receiver* $R$ *as the one that runs* $R_I()$ *during the initialization phase, runs* $R_T(R_{i-1}, \hat{W}_i)$ *in the* $i$*th transfer and returns* $R_k = R_0$ *and the list of received messages* $\{M'_{\hat{W}_1}, \ldots, M'_{\hat{W}_k}\}$*.*

*Ideal experiment. In* $Ideal_{\tilde{S}', \tilde{R}'}(N, k, (M_{W_1}, W_1), \ldots, (M_{W_N}, W_N), \hat{W}_1, \ldots, \hat{W}_k)$ *the possibly cheating sender algorithm* $\tilde{S}'((M_{W_1}, W_1), \ldots, (M_{W_N}, W_N))$ *generates message-keyword pairs* $\{(M'_{W_1}, W'_1), \ldots, (M'_{W_N}, W'_N)\}$ *and transmits them to a trusted party* $T$*. In the* $i$*th round* $\tilde{S}'$ *sends a bit* $b_i$ *to* $T$ *and the possibly cheating receiver* $\tilde{R}'(\hat{W}_i)$ *transmits* $\hat{W}'_i$ *to* $T$*. If* $b_i = 1$ *and there exists a message attached to a keyword* $\hat{W}'_i$ *then* $T$ *hands* $M'_{\hat{W}'_i}$ *to* $\tilde{R}'$*. Otherwise it hands* $\perp$ *to* $\tilde{R}'$*. At the end of the* $k$*th transfer the output of the experiment is* $(S_k, R_k)$*. We define the honest sender* $S'((M_{W_1}, W_1), \ldots, (M_{W_N}, W_N))$ *as the one that sends message-keyword pairs* $\{(M_{W_1}, W_1), \ldots, (M_{W_N}, W_N)\}$ *in the initialization phase, sends* $b_i = 1$ *in the* $i$*th transfer phase and outputs* $S_k = S_0$*, and the honest receiver* $R$ *as the one that sends* $\hat{W}_i$ *in the* $i$*th transfer and returns* $R_k = R_0$ *and the list of received messages* $\{M'_{\hat{W}_1}, \ldots, M'_{\hat{W}_k}\}$*.*

*Sender Security.* $OKS^N_{k\times 1}$ *provides sender security if for every real-world probabilistic polynomial time receiver* $\tilde{R}$ *there exists an ideal-world receiver* $\tilde{R}'$ *such that for any polynomial* $N_m(\kappa)$*, any* $N \in \{1, \ldots, N_m(\kappa)\}$*, any* $k \in \{1, \ldots, N\}$*, any* $\{(M_{W_1}, W_1) \ldots, (M_{W_N}, W_N)\}$ *and any keywords* $\hat{W}_1, \ldots, \hat{W}_k$*, the advantage of every probabilistic polynomial time distinguisher in distinguishing* $Real_{S,\tilde{R}}(N, k, (M_{W_1}, W_1), \ldots, (M_{W_N}, W_N), \hat{W}_1, \ldots, \hat{W}_k)$ *and* $Ideal_{S', \tilde{R}'}(N, k, (M_{W_1}, W_1), \ldots, (M_{W_N}, W_N), \hat{W}_1, \ldots, \hat{W}_k)$ *is negligible in* $\kappa$*.*

*Receiver Security.* $OKS^N_{k\times 1}$ *provides receiver security if for every real-world probabilistic polynomial time sender* $\tilde{S}$ *there exists an ideal-world sender* $\tilde{S}'$ *such that for any polynomial* $N_m(\kappa)$*, any* $N \in \{1, \ldots, N_m(\kappa)\}$*, any* $k \in \{1, \ldots, N\}$*, any* $\{(M_{W_1}, W_1) \ldots, (M_{W_N}, W_N)\}$ *and any keywords* $\hat{W}_1, \ldots, \hat{W}_k$*, the advantage of every probabilistic polynomial time distinguisher in distinguishing* $Real_{\tilde{S},R}(N, k, (M_{W_1}, W_1), \ldots, (M_{W_N}, W_N), \hat{W}_1, \ldots, \hat{W}_k)$ *and* $Ideal_{\tilde{S}', R'}(N, k, (M_{W_1}, W_1), \ldots, (M_{W_N}, W_N), \hat{W}_1, \ldots, \hat{W}_k)$ *is negligible in* $\kappa$*.*

### 3.4.2 Relation with Oblivious Transfer

As explained at the beginning of this section, OKS is a generalization of the concept of OT in which the receiver can select a keyword in order to search in the database instead of choosing a selection value that is associated with

a specific record of the database. In fact, when Ogata and Kurosawa [OK04] proposed OKS, they also proved that it is always possible to build an $OT^N_{k\times 1}$ scheme from any $OKS^N_{k\times 1}$ scheme by executing the $OKS^N_{k\times 1}$ scheme only once, and they built an $OT^N_{k\times 1}$ scheme as an application of their first scheme (see the following subsection). Intuitively, they just change the keywords for selection values.

Conversely, they also show how to build an $OKS^N_{k\times 1}$ scheme from any $OT^N_{k\times 1}$ scheme, although the result is inefficient. This construction holds the case in which a keyword can be related with more than one message. Namely, consider a $|W_{sp}| \times n$ matrix, where $W_{sp}$ is the set of all possible keywords and $n$ is the amount of records, in which the $i$th row includes all the indices of the records that include $W_i$ as a keyword. Then the size of the commitments is $O(n|W_{sp}| + nB)$, where $B$ is the size of a record, and in the other $OKS^N_{k\times 1}$ schemes their size is $O(nB)$, independently of $W_{sp}$.

Despite of this, we note that some $OT^N_{k\times 1}$ schemes can be easily modified to build an $OKS^N_{k\times 1}$ scheme. For example, [CNS07] show how to modify their first construction (see Section 3.3.3) in order to build an $OKS^N_{k\times 1}$ scheme. If we use the RSA blind signature scheme in this transformation the scheme proposed by [OK04] is obtained.

### 3.4.3 Historical Overview

In 2004, Ogata and Kurosawa [OK04] defined the concept of OKS. They also show how to build an OKS scheme from an OT scheme, but they explained that such a construction is not efficient. Therefore, they proposed two OKS schemes whose security is proven in the half-simulation model with random oracles. The first one uses the RSA blind signature scheme [Cha82] and thus is secure under the assumption that the one-more RSA inversion problem is hard ([BNPS03], [BNPS01]). The second one uses an oblivious polynomial evaluation protocol and thus is secure under the polynomial reconstruction problem ([NP99a], [BN00]). This second protocol is less efficient than the first one but it is based on a more widely accepted assumption. The size of the commitments in both protocols is $O(nB)$ regardless the size of the set of possible keywords, where $B$ is the size of a record and $n$ is the amount of records.

# Chapter 4

# Public-key Encryption with Oblivious Keyword Search

Both Public-key Encryption with Keyword Search (PEKS) and Oblivious Keyword Search (OKS) aim at providing a mechanism to efficiently search on encrypted data while preserving the secrecy of both the search terms and the content that is not queried. We show that PEKS cannot protect the privacy of the search terms in some application scenarios and thus we propose a new concept, Public-key Encryption with Oblivious Keyword Search (PEOKS), that extends PEKS and that offers similar properties to OKS.

Analogously to the use of anonymous-IBE to build PEKS, we show how to use any blind-anonymous-IBE scheme to construct a PEOKS scheme. In addition, since until now no blind-anonymous-IBE scheme was designed, we provide a blind key derivation protocol for the anonymous-IBE scheme due to Boyen and Waters [BW06]. With this protocol and the transformation mentioned above a PEOKS scheme can be designed in a straightforward manner.

First, we define the concept of PEOKS and its security properties. In Section 4.2 we show how to build a PEOKS scheme from any blind-anonymous-IBE scheme, and we provide a blind-anonymous-IBE scheme in Section 4.3. After that, we show how PEOKS is successfully applied to an application scenario for which PEKS is not enough. Finally, in Section 4.5 we propose other applications of blind-anonymous-IBE.

## 4.1 Definition

As explained in Section 3.2.2, there exist application scenarios in which the current notion of PEKS cannot be successfully used. These scenarios are similar to the audit log scenario described there. However, now the party that authorizes the receivers to obtain trapdoors depending on the keyword is not the one that derives the trapdoors. Therefore, the party that de-

rives trapdoors, which we from now on call the Trapdoor Generation Center (TGC), does not need to learn the keywords. Nevertheless, if we use PEKS, the TGC has to learn the keywords to be able to compute the trapdoors. In order to solve this problem, we build a mechanism by means of which the receivers can get trapdoors for keywords of their choice without revealing the keywords to the TGC. We call this solution Public-key Encryption with Oblivious Keyword Search (PEOKS). In this section we give a formal definition of PEOKS and we explain its security properties, whereas in Section 4.4 we detail a concrete application scenario in which PEOKS should be used instead of PEKS.

We give a definition of PEOKS based on the definition of PEKS given in Section 3.2.1.

**Definition 63 (PEOKS)** *A Public-key Encryption with Oblivious Keyword Search scheme consists of a set of three probabilistic polynomial time algorithms (SetupPEOKS, PEOKS, Test), an interactive protocol, BlindTrapdoor, and a commitment scheme ComSch as follows:*

**SetupPEOKS**($1^k$)**.** *On input a security parameter $k$, it outputs a secret key $s_k$ and params, the parameters of the scheme.*

**PEOKS**($params$,$W$,$M$)**.** *On input a message $M$ and a keyword $W$, it computes a searchable encryption $S_{W,M}$ for $W$.*

**BlindTrapdoor(TGC**($params$,$s_k$,$C$)**, U**($params$,$W$,$open_W$)**).** *The input of the TGC is the secret key $s_k$ and a commitment $C = Commit(params_{comm}, W', open_{W'})$ to a keyword, whereas the input of the user is the keyword $W$ and a random value $open_W$. The output of the user is the trapdoor $T_W$ or $\perp$ if the protocol fails, whereas the output of the TGC is nothing or $\perp$. The protocol fails if $W \neq W'$.*

**Test**($params$,$S_{W,M}$,$T_{W'}$)**.** *On input a searchable encryption $S_{W,M}$ and a trapdoor $T_{W'}$, it outputs $M$ if $W = W'$ and $\perp$ otherwise.*

The scheme is depicted in Figure 4.1. At the beginning, algorithm $SetupPEOKS$ is run in order to give the secret key to the TGC and to publish the parameters of the scheme. After that, the sender, who wants to store encrypted information in a database and who, in addition, wants to allow the receivers to search by keywords, uses algorithm $PEOKS$ to attach searchable encryptions to each record of data. Then, a receiver runs with the TGC protocol $BlindTrapdoor$ to obtain trapdoors for the keywords she has been authorized to search for by a third party. Finally, the receiver uses algorithm $Test$ to look for the records of data that were attached to a specific keyword.

As can be seen, the main difference between PEOKS and PEKS is that in PEOKS we use a protocol $BlindTrapdoor$ instead of algorithm $Trapdoor$.

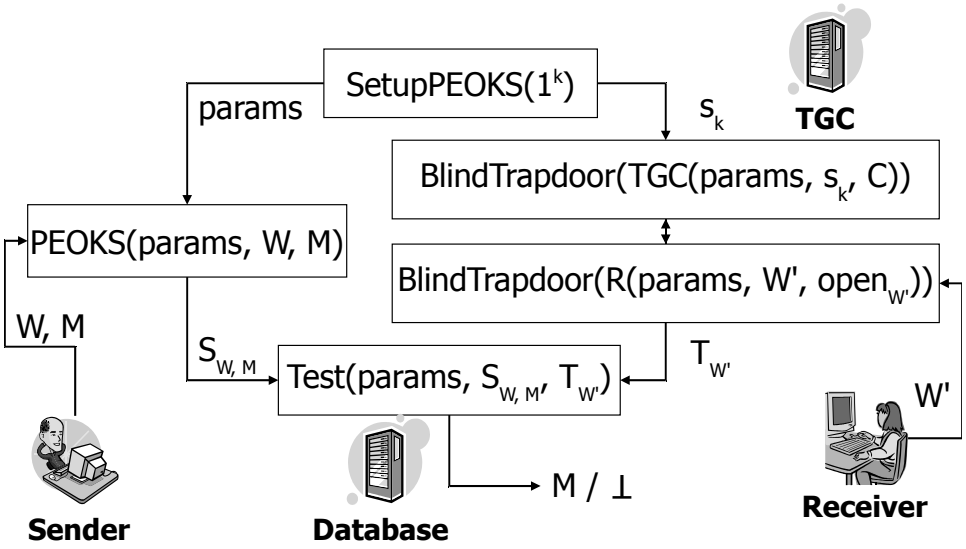


Figure 4.1: Public-key Encryption with Oblivious Keyword Search scheme.

By means of this protocol the receiver obtains trapdoors from the TGC without revealing the keywords. The commitment $C$ is used by the receiver to show to the TGC that she has authorization to obtain a trapdoor for the keyword used to compute the commitment. The way by which this is achieved will be detailed when instantiating PEOKS with blind-anonymous-IBE (see Section 4.3). On the other hand, the party that gives authorization and the way by which this authorization is given will depend on the concrete application. In Section 4.4 we detail this step for our application scenario.

The message $M$ introduced as input of the $PEOKS$ algorithm will be used for the same purposes for which it can be used in a PEKS scheme. In our application this message will be a pointer to a linked list that contains all the necessary information to retrieve all the records of data related with that keyword along with the symmetric key that was used to encrypt the first node of the list. Then the receiver can get the symmetric key and the pointer as output of the $Test$ algorithm.

Now we want to extend the concepts of privacy and consistency for PEKS that are defined in Section 3.2.1 to apply them to PEOKS. First of all, we define the concept of PEOKS-IND-CPA security by means of a PEOKS-IND-CPA game:

**1.** Run algorithm $SetupPEOKS(1^k)$ and give the parameters $params$ to the adversary.

**2.** The adversary makes adaptive queries to $Oracle_{BlindTrapdoor}(C_W)$. First, the adversary computes a commitment $C_W$ and gives it to the oracle. After that, the adversary interacts with the oracle by running the

user's side of the *BlindTrapdoor* protocol in order to obtain a trapdoor $T_W$. The oracle runs the TGC's side of the protocol.

**3.** The adversary outputs two keywords $W_0, W_1$ and two messages $M_0, M_1$.

**4.** Pick two random bits $b$ and $c$ and run $PEOKS(params, W_b, M_c)$ to obtain a searchable encryption $S_{W_b,M_c}$ for keyword $W_b$, using also message $M_c$. Give $S_{W_b,M_c}$ to the adversary.

**5.** The adversary continues making adaptive queries to $Oracle_{BlindTrapdoor}(C_W)$.

**6.** The adversary opens the commitments that he has sent in Step 2 and in Step 5.

**7.** The adversary outputs two bits $b'$ and $c'$ in order to guess which keyword and which message were used to compute the searchable encryption. If at least one challenge keyword was used to compute one of those commitments, or the adversary does not open all the commitments, the game returns $b' = 0$ and $c' = 0$. Otherwise it returns $b'$ and $c'$.

The advantage of the adversary is $Adv(A) = |Pr[b = b' \wedge c = c'] - 1/4|$. We say that the PEOKS scheme is PEOKS-IND-CPA secure when this advantage is negligible.

We note that, although we call the privacy property PEOKS-IND-CPA when applying it to a PEOKS scheme, the meaning is similar to the PEKS-IND-CPA property for PEKS, i.e., that a searchable encryption leaks neither information about the keyword nor about the message that were used to compute the searchable encryption. In addition, we require for the PEOKS-IND-CPA property a binding commitment scheme.

As the main difference between PEKS and PEOKS consists in exchanging algorithm *Trapdoor* for protocol *BlindTrapdoor*, we claim that the properties of the underlying PEKS scheme remain fulfilled if *BlindTrapdoor* is leak free (with commitment) (see Section 4.2). We recall that a protocol *BlindTrapdoor* that is leak free (with commitment) needs to be extractable, i.e., the keyword for which a trapdoor is being requested can be extracted.

**Theorem 3** *Given a PEKS scheme (SetupPEKS, PEKS, Trapdoor, Test) and a protocol BlindTrapdoor, if the PEKS scheme is PEKS-IND-CPA secure, the BlindTrapdoor protocol is leak free (with commitment) and ComSch is a binding commitment scheme, then the PEOKS scheme (SetupPEOKS, PEOKS, BlindTrapdoor, Test, ComSch) is PEOKS-IND-CPA secure.*

*Proof.* First, we show that if an adversary has non-negligible advantage in winning the PEOKS-IND-CPA game when the *BlindTrapdoor* protocol is

leak free (with commitment) and the commitment scheme is binding, then we can build an algorithm $A$ that has non-negligible advantage in winning the PEKS-IND-CPA game.

Let $E$ be a probabilistic polynomial time adversary that has non-negligible advantage in winning the PEOKS-IND-CPA game with the protocol *BlindTrapdoor* being leak free (with commitment) and the commitment scheme being binding. Then we can build a polynomial time algorithm $A$ that has non-negligible advantage in winning the PEKS-IND-CPA game as follows. First, $A$ hands the parameters of the scheme to $E$. At every stage, $A$ receives a commitment $C_W$ and answers the oracle queries of $E$ by running the TGC's side of the *BlindTrapdoor* protocol. This can be done because the leak freeness property ensures that the *BlindTrapdoor* protocol can be simulated. $A$ uses rewinding capabilities to extract the keyword that is being queried in order to give it to its $Oracle_{Trapdoor}$ and get the corresponding trapdoor, which allows $A$ to simulate the protocol.

When $E$ outputs the challenge keywords $W_0, W_1$ and the challenge messages $M_0, M_1$, $A$ uses them as its own challenges. Given the searchable encryption $S_{W_b,M_c}$, $A$ hands it to $E$. Finally, $E$ opens the commitments and outputs bits $b'$ and $c'$, and $A$ outputs these bits. If all the keywords queried by $A$ to $Oracle_{Trapdoor}$ were different from the challenge keywords $W_0, W_1$, the PEKS-IND-CPA game outputs $b'$ and $c'$. Since $E$ has non-negligible advantage in winning the PEOKS-IND-CPA game even when the commitment scheme is binding and the protocol $BlindTrapdoor$ is leak free (with commitment), and it is clear that in this case $E$ does not get any knowledge from the protocol, then $A$ wins the PEKS-IND-CPA game with non-negligible advantage by outputting $b'$ and $c'$.

Second, we show that if an adversary has non-negligible advantage in winning the PEOKS-IND-CPA game when the underlying PEKS scheme is PEKS-IND-CPA secure and $ComSch$ is a binding commitment scheme, then we can build a distinguisher $D$ that can distinguish between Game Real and Game Ideal with non-negligible advantage.

Let $E$ be a probabilistic polynomial time adversary that has non-negligible advantage in winning the PEOKS-IND-CPA game with the underlying PEKS scheme being PEKS-IND-CPA secure and $ComSch$ being binding. Consider that Game Real consists in $E$ playing a PEOKS-IND-CPA game in which $Oracle_{BlindTrapdoor}$ has the secret $s_k$ and runs the TGC's side of the $BlindTrapdoor$ protocol, and consider that Game Ideal consists in $E$ playing a PEOKS-IND-CPA game in which $Oracle_{BlindTrapdoor}$ is implemented by a simulator $S$ that extracts the keyword that is being queried by $E$ using its rewinding capabilities and that uses the keyword to query a trusted third party in order to obtain the trapdoor that corresponds to the keyword, which allows $S$ to simulate the protocol. Note that in Game Ideal $E$ cannot get more knowledge from the interaction with $S$ than the knowledge he

can get from the interaction with a trusted party that computes algorithm $Trapdoor$.

In Game Real, first the game hands the parameters of the scheme to $E$ and then answers the oracle queries of $E$ as mentioned above. When $E$ sends the challenge keywords and the challenge messages the game answers as usual. Finally $E$ opens the commitments and the bits $b'$ and $c'$. If all keywords used to compute the commitments are different from the challenge keywords, then the game outputs $b'$ and $c'$. Since we have a binding commitment scheme and an underlying PEKS scheme that is PEKS-IND-CPA secure, we have that, since $E$ is still able to win the game with non-negligible probability, it must be the case that $E$ gets some knowledge from the interaction with $Oracle_{BlindTrapdoor}$, and so the protocol is not leak free (with commitment).

In Game Ideal, first the game hands the parameters of the scheme to $E$. When $E$ makes an oracle query, $S$ is not able to simulate the TGC's side of the $BlindTrapdoor$ protocol because it is not leak free (with commitment), and so $D$ is able to distinguish whether it is faced with $E$ playing Game Real or with $S$ playing Game Ideal with non-negligible probability.

Third, we show that if an adversary has non-negligible advantage in winning the PEOKS-IND-CPA game when the underlying PEKS scheme is PEKS-IND-CPA secure and the $BlindTrapdoor$ protocol is leak free (with commitment), then we can build an adversary that has non-negligible advantage in breaking the binding property of the commitment scheme.

Let $E$ be a probabilistic polynomial time adversary that has non-negligible advantage in winning the PEOKS-IND-CPA game with the underlying PEKS scheme being PEKS-IND-CPA secure and the $BlindTrapdoor$ protocol being leak free (with commitment). Then we can build a polynomial time algorithm $A$ that has non-negligible advantage in breaking the binding property of the commitment scheme as follows. First $A$ runs $SetupPEOKS$, keeps the secret key $s_k$ and hands the parameters of the scheme to $E$. $A$ runs the TGC's side of the $BlindTrapdoor$ protocol to answer the oracle queries of $E$. $A$ uses rewinding capabilities to extract the keyword $W$ and the random value $open_W$ that $E$ uses to compute $C_W$ and stores them along with the commitment $C_W$ that $E$ sent. When $E$ sends the challenge keywords and the challenge messages, $A$ picks random bits $b$ and $c$, computes $S_{W_b,M_c}$ and hands it to $E$. Finally, $E$ opens the commitments and outputs the bits $b'$ and $c'$. Since the underlying PEKS scheme is PEKS-IND-CPA secure and the $BlindTrapdoor$ protocol is leak free (with commitment), it must be the case that $E$ was able to query for a challenge keyword and after that open the corresponding commitment to another keyword (otherwise $E$ does not win the game with non-negligible advantage). Therefore, since $A$ knows the keywords that were used to compute each commitment, $A$ knows when $E$ opens a commitment by utilizing a keyword different from the one

used to compute the commitment, and thus $A$ can use this keyword $W'$ and the random value $open_{W'}$ to output a tuple $(C_W, W, open_W, W', open_{W'})$ that breaks the binding property of the commitment scheme. □

The other property of the *BlindTrapdoor* protocol, selective-failure blindness (with commitment) (see Section 4.2), ensures that the TGC cannot induce protocol failures that depend on the keyword for which a trapdoor is requested. Note that this property needs a semantically secure commitment scheme in order to be fulfilled. On the other hand, we note that the consistency definition for PEKS is valid for PEOKS. Therefore, we arrive to the following definition.

**Definition 64 (Consistency and security for PEOKS)** *A PEOKS scheme (SetupPEOKS, PEOKS, BlindTrapdoor, Test, ComSch) is secure if an only if it is PEOKS-IND-CPA secure and the BlindTrapdoor protocol is selective-failure blind. It is consistent if the underlying PEKS scheme (SetupPEKS, PEKS, Trapdoor, Test) is consistent.*

## 4.2 Instantiation using Blind-Anonymous-IBE

In Section 3.2.3 we showed how to build a PEKS scheme by using any anonymous-IBE scheme. Now we explain how to construct a PEOKS scheme by using any blind-anonymous-IBE scheme. First of all, we need to adapt slightly the definition of protocol *BlindExtract* given in Section 3.1.3 in order to introduce a commitment that the user utilizes to show that she has been authorized to get a secret key for her identity. We call this modified protocol *ComBlindExtract*.

**Definition 65 (ComBlindExtract)** *An interactive protocol ComBlindExtract(KGC(params,$s_k$,C), U(params,id,$open_{id}$)) between a user and the KGC. The input of the KGC is the master secret key $s_k$ and a commitment $C = Commit(params_{comm}, id', open_{id'})$, whereas the input of the user is her identity id and a random value $open_{id}$. The output of the user is the secret key $sk_{id}$ corresponding to identity id or $\perp$ if the protocol fails, and the output of the KGC is nothing or $\perp$. The protocol fails if $id \neq id'$.*

Intuitively, when running the protocol the user has to show that the value used to compute the commitment and her identity are the same values. Otherwise the KGC will abort the execution.

A blind-IBE constructed using a protocol *ComBlindExtract* is, like when using a protocol *BlindExtract*, secure if the underlying IBE scheme is secure and the protocol is leak free and selective-failure blind. In addition, we require a semantically secure and binding commitment scheme. In the following we formalize this notion and we adapt the definitions of leak freeness and selective-failure blindness given in Section 3.1.3 to use them with a protocol *ComBlindExtract*.

**Definition 66 (Leak freeness (with commitment))** *A protocol ComBlindExtract associated with an IBE scheme (SetupIBE, Extract, Encrypt, Decrypt) is leak free (with commitment) if, for all efficient adversaries A, there exists an efficient simulator S such that for every security parameter k no efficient distinguisher D can distinguish whether it is faced with A playing Game Real or with S playing Game Ideal with non-negligible advantage:*

**Game Real.** *Run* $SetupIBE(1^k)$. *As many times as D wants, A chooses an identity id, a random value* $open_{id}$ *and a commitment C and executes protocol ComBlindExtract(KGC(*$params$,$s_k$,$C$*), A(*$params$,$id$, $open_{id}$*)) with the KGC.*

**Game Ideal.** *Run* $SetupIBE(1^k)$. *As many times as D wants, S chooses an identity id, a random value* $open_{id}$ *and a commitment C and queries a trusted party to obtain the output of Extract(*$params$,$s_k$,$id$*), if* $id \in I(k)$ *and* $C = Commit(params_{comm}, id, open_{id})$, *or* $\perp$ *otherwise.*

*Here D and A (or S) may communicate at any time.*

**Definition 67 (Selective-failure blindness (with commitment))** *A protocol* $ComBlindExtract(KGC(\cdot,\cdot,\cdot),\ U(\cdot,\cdot,\cdot))$ *is said to be selective-failure blind (with commitment) if there exists a negligible function* $\nu(k)$ *such that:*

$$\begin{aligned}&\Pr[(params, id_0, id_1, state) \leftarrow A(1^k); b \leftarrow \{0,1\};\\ &\quad b' \leftarrow A(state)^{Oracle_U(params,id_b);Oracle_U(params,id_{1-b});Oracle_{Aux}()}\\ &\quad : b = b'] < 1/2 + \nu(k)\end{aligned}$$

$Oracle_U(params, id_b)$ *picks a random* $open_{id_b}$, *creates a commitment* $C = Commit(id_b, open_{id_b})$ *and sends it to A. Then it interacts with A by executing the user's part of the ComBlindExtract protocol. It returns, not until the execution of* $Oracle_{Aux}()$, $sk_{id_b}$ *or* $\perp$ *if the protocol aborts. On the other hand,* $Oracle_{Aux}()$ *returns* $\perp$ *if it is executed before the other two oracles have ended. Otherwise, it returns* ($sk_{id_b}$,$sk_{id_{1-b}}$) *if* $sk_{id_b} \neq \perp$ *and* $sk_{id_{1-b}} \neq \perp$, $(\perp, \epsilon)$ *if* $sk_{id_b} = \perp$ *and* $sk_{id_{1-b}} \neq \perp$, $(\epsilon, \perp)$ *if* $sk_{id_b} \neq \perp$ *and* $sk_{id_{1-b}} = \perp$ *and* $(\perp, \perp)$ *if* $sk_{id_b} = \perp$ *and* $sk_{id_{1-b}} = \perp$.

The properties of privacy and anonymity for blind-anonymous-IBE can be defined via the following blind-IBE-IND-ANO-CPA game:

**1.** Run algorithm $SetupIBE(1^k)$ and give the parameters $params$ to the adversary.

**2.** The adversary makes adaptive queries to $Oracle_{ComBlindExtract}(C_{id})$. First, the adversary computes a commitment $C_{id}$ and gives it to the oracle. After that, the adversary interacts with the oracle by running the user's side of the $ComBlindExtract$ protocol in order to obtain a secret key $sk_{id}$. The oracle runs the KGC's side of the protocol.

**3.** The adversary outputs two identities $id_0, id_1$ and two messages $M_0, M_1$.

**4.** Pick two random bits $b$ and $c$ and run $Encrypt(params, id_b, M_c)$ to obtain a ciphertext $C$ for identity $id_b$ and message $M_c$. Give $C$ to the adversary.

**5.** The adversary makes more queries to $Oracle_{ComBlindExtract}(C_{id})$.

**6.** The adversary opens the commitments that he has sent in Step 2 and in Step 5.

**7.** The adversary outputs two bits $b'$ and $c'$ in order to guess which identity and which message were used to compute the ciphertext. If at least one challenge identity was used to compute one of those commitments, or the adversary does not open all the commitments, the game returns $b' = 0$ and $c' = 0$. Otherwise it returns $b'$ and $c'$.

The advantage of the adversary is $Adv(A) = |Pr[b = b' \wedge c = c'] - 1/4|$. We say that the IBE scheme is blind-IBE-IND-ANO-CPA secure when this advantage is negligible.

This game combines the games defined for privacy and anonymity in IBE that are described in Section 3.1. Apart from that, the main difference is that here the adversary interacts with the oracle by running the user's side of the *ComBlindExtract* protocol instead of giving the identity to the oracle. Therefore, if the protocol is leak free (with commitment) and the commitment scheme *ComSch* is binding, the blind-anonymous-IBE scheme is blind-IBE-IND-ANO-CPA secure.

**Theorem 4** *A blind-anonymous-IBE scheme (SetupIBE, ComBlindExtract, Encrypt, Decrypt, ComSch) is called blind-IBE-IND-ANO-CPA secure if the underlying IBE scheme (SetupIBE, Extract, Encrypt, Decrypt) is IBE-IND-ANO-CPA secure, the protocol ComBlindExtract is leak free (with commitment) and ComSch is a binding commitment scheme.*

*Proof.* This proof is very similar to the proof given for Theorem 3. For the first part, we show that if an adversary $E$ is able to win with non-negligible advantage the blind-IBE-IND-ANO-CPA game when the protocol *ComBlindExtract* is leak-free (with commitment) and the commitment scheme is binding, then we can build an algorithm $A$ that breaks the IBE-IND-ANO-CPA security of the underlying anonymous-IBE scheme with non-negligible advantage. $A$ is constructed in the same way as in the above mentioned proof. When $E$ queries $Oracle_{ComBlindExtract}$, it extracts the identity queried by $E$ from the protocol and sends it to $Oracle_{Extract}$ in order to be able to simulate the KGC's part of the protocol. $A$ uses the challenge sent by $E$ and when $E$ sends the bits $b'$ and $c'$, $A$ uses them as its own guess to win the IBE-IND-ANO-CPA game with non-negligible advantage.

For the second part, we show that if an adversary $E$ is able to win with non-negligible advantage the blind-IBE-IND-ANO-CPA game when the underlying anonymous-IBE scheme is IBE-IND-ANO-CPA secure and the commitment scheme is binding, then we can build a distinguisher D that can distinguish whether it is faced with Game Real, in which $Oracle_{ComBlindExtract}$ has the secret key $s_k$, and Game Ideal, in which this oracle is implemented by a simulator that extracts the identity being queried from the protocol and uses it to obtain the corresponding secret key $sk_{id}$ from a trusted third party. The strategy followed here is the same as in the proof of Theorem 3.

For the third part, we show that if an adversary $E$ is able to win with non-negligible advantage the blind-IBE-IND-ANO-CPA game when the underlying anonymous-IBE scheme is IBE-IND-ANO-CPA secure and the *ComBlindExtract* protocol is leak free (with commitment), then we can build an algorithm $A$ that has non-negligible advantage in breaking the binding property of the commitment scheme. The strategy followed here is again the same as in the proof of Theorem 3. □

The selective-failure blindness (with commitment) property of the *ComBlindExtract* protocol ensures that the KGC cannot induce protocol failures that depend on the identity being extracted. Note that this property needs a semantically secure commitment scheme in order to be fulfilled. With these properties we arrive to the following definition.

**Definition 68 (Security for blind-anonymous-IBE)** *A blind-anonymous-IBE scheme (SetupIBE, ComBlindExtract, Encrypt, Decrypt, ComSch) is secure if an only if it is blind-IBE-IND-ANO-CPA secure and the protocol ComBlindExtract is selective-failure blind (with commitment).*

Armed with these definitions we are ready to explain how to build a PEOKS scheme using any blind-anonymous-IBE scheme.

**Transformation 3 (IBE-to-PEOKS)** *Given a blind-anonymous-IBE scheme (SetupIBE, ComBlindExtract, Encrypt, Decrypt, ComSch) the PEOKS scheme can be built as follows:*

**SetupPEOKS($1^k$).** *On input a security parameter $k$, $SetupIBE(1^k)$ is run in order to obtain the secret key $s_k$ and params, the parameters of the scheme.*

**PEOKS($params, W, M$).** *On input a keyword $W$ and a message $M$, it picks a random value $C_2 \in \{0,1\}^k$ and computes $C_1 = Encrypt(params, W, C_2||M)$. It outputs the tuple $S_{W,M} = (C_1, C_2)$.*

**BlindTrapdoor(TGC($params$,$s_k$,$C$), U($params$,$W$,$open_W$)).** *The trapdoor $T_W$ associated with the keyword $W$ is the secret key $sk_W$ associated with this keyword (acting as an identity), so it can be obtained by*

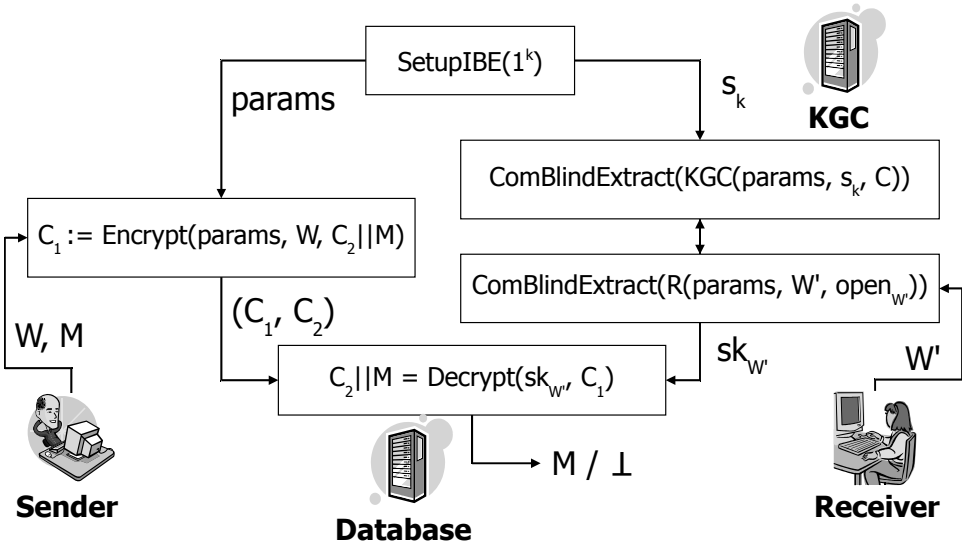


Figure 4.2: Transformation from an IBE scheme to a PEOKS scheme.

*running ComBlindExtract(KGC(params,$s_k$,C), U(params,W, $open_W$)) between the user and the TGC.*

**Test**$(params, S_{W,M}, T_{W'})$**.** *On input the searchable encryption $S_{W,M}$ and the trapdoor $T_{W'}$, it outputs $M$ if $C_2||M = Decrypt(T_{W'}, C_1)$ and $\perp$ otherwise.*

The construction is depicted in Figure 4.2. The main difference between this transformation and the Transformation IBE-to-PEKS given in Section 3.2.3 is that now we are using a blind-anonymous-IBE scheme and therefore we use protocol $ComBlindExtract$ instead of algorithm $Extract$. By means of this protocol, we can implement a protocol $BlindTrapdoor$ simply by using keywords as identities.

The following theorem expresses that a PEOKS scheme constructed by using Transformation IBE-to-PEOKS is consistent and PEOKS-IND-CPA secure.

**Theorem 5** *A PEOKS scheme constructed by following Transformation IBE-to-PEOKS is consistent and secure according to Definition 64 if and only if the blind-anonymous-IBE scheme is secure in the sense expressed by Definition 68.*

*Proof.* For consistency, we note that the consistency proof given for Theorem 2, that proves the security of Transformation IBE-to-PEKS, remains valid. The key observation is that, in the consistency game, the adversary outputs the keywords before the $Trapdoor$ algorithm (in PEKS) or the

*BlindTrapdoor* protocol (in PEOKS) are run, so he cannot use the (possible) knowledge that he would obtain from running protocol *BlindTrapdoor* to output the challenge keywords.

For selective-failure blindness (with commitment), we note that the definition of selective-failure blindness (with commitment) is the same for protocol *ComBlindExtract* and for protocol *BlindTrapdoor*. Therefore, when instantiating PEOKS with blind-anonymous-IBE, since the *BlindTrapdoor* protocol is instantiated by the *ComBlindExtract* protocol, if this protocol is selective-failure blind (with commitment) then *BlindTrapdoor* is selective-failure blind (with commitment).

For PEOKS-IND-CPA security, a similar proof to the one given to demonstrate PEKS-IND-CPA security in Transformation IBE-to-PEKS can be constructed. Let $E$ be any probabilistic polynomial time adversary attacking the PEOKS-IND-CPA security of the PEOKS scheme, and let $A$ be a probabilistic polynomial time adversary attacking the blind-IBE-IND-ANO-CPA security of the blind-anonymous-IBE scheme. First, $A$ runs $E$ and gets the challenge keywords $W_0, W_1$ and the challenge messages $M_0, M_1$. $A$ answers the trapdoor oracle of $E$ by means of his extraction oracle in every stage. For this purpose, $A$ forwards the messages corresponding to the protocol *ComBlindExtract* between $E$ and the oracle. Then, $A$ outputs two keywords $W_0, W_1$ and two challenge messages $R_0||M_0$ and $R_1||M_1$, where $R_0, R_1 \in \{0,1\}^k$. Given the challenge ciphertext $C = Encrypt(params, W_b, R_c||M_c)$, it runs $E$ on input $C$. Finally, $E$ opens the commitments and $A$ uses this information to open the commitments in the blind-IBE-IND-ANO-CPA game. $E$ sends bits $(b', c')$, which $A$ returns. It is easy to see that, for $b, c \in \{0,1\}$, the probability of guessing $(b, c)$ in the blind-IBE-IND-ANO-CPA game equals the probability of guessing $(b, c)$ in the PEOKS-IND-CPA game. □

## 4.3 A Blind-Anonymous-IBE Scheme

The design of a PEOKS scheme presented in the previous section requires the use of a blind-anonymous-IBE scheme. However, so far none is known. Until now four anonymous-IBE schemes have been proposed: two in the random oracle model, one due to Boneh and Franklin [BF01] and another one due to Sakai and Kasahara [CCMLS06], and two in the standard model, one due to Gentry [Gen06] and the other one due to Boyen and Waters [BW06]. Despite Gentry's scheme being more efficient, the latter relies on weaker complexity assumptions and generalizes to anonymous-hierarchical-IBE. In addition, it seems hard to find a blind key derivation protocol for Gentry's scheme. Therefore, we have used Boyen and Waters scheme to build a blind-anonymous-IBE scheme.

In this section we present a blind key derivation protocol for the anony-

mous-IBE scheme due to Boyen and Waters. First, we review their scheme. In Section 4.3.2 we present the protocol and we prove that it is leak free and selective-failure blind according to the definitions given in the previous section, and later we detail some parts of the protocol. Finally, as the Boyen and Waters scheme is secure under selective-id security (see Section 3.1.1) we show how to modify their scheme and our protocol in order to achieve adaptive-id security in Section 4.3.4. In conclusion, with the transformation presented in the previous section and the blind-anonymous-IBE scheme presented here, a PEOKS scheme can be designed in a straightforward manner.

### 4.3.1 The Boyen and Waters Anonymous-IBE Scheme

The Boyen and Waters IBE scheme [BW06] is the only scheme known to us that provides anonymity and hierarchical key delegation at the same time. However, as we do not need hierarchical key delegation to build PEOKS, we use their one-level scheme. The security of the scheme relies on two static assumptions: DBDH and Decision Linear (see Section 2.7.3). For generality, we present here a scheme that uses asymmetric pairings, but as noted by [BW06] their scheme is also anonymous if it is implemented with symmetric pairings. On the other hand, for simplicity we present now an IBE scheme that is selective-id secure (IBE-IND-sID-CPA) and in the next subsections we construct a blind key derivation protocol for this scheme. Another reason is that some other applications of this protocol, like its use in the OT scheme due to [GH07] (see Section 3.3.2) do not require stronger notions of security. We show the modifications needed to provide adaptive-id security (IBE-IND-CPA) in the scheme and the consequent changes in the protocol in Section 4.3.4.

Therefore, the Boyen and Waters IBE scheme works as follows:

**SetupIBE($1^k$).** Run $BilinearSetup(1^k)$ to obtain a bilinear map setup $paramsBM = (p, G_1, G_2, G_t, e, g, h)$. Pick $\omega, z_0, z_1, t_1, t_2, t_3, t_4 \in \mathbb{Z}_p^*$ and keep $s_k = (\omega, t_1, t_2, t_3, t_4)$ as the master secret key. Compute the system parameters as

$$params = (\Omega = e(g,h)^{t_1 t_2 \omega}, g, h, g_0 = g^{z_0}, g_1 = g^{z_1}, h_0 = h^{z_0}, h_1 = h^{z_1}, \\ v_1 = g^{t_1}, v_2 = g^{t_2}, v_3 = g^{t_3}, v_4 = g^{t_4}).$$

**Extract($params$,$s_k$,$id$).** Pick two random values $r_1, r_2 \in \mathbb{Z}_p^*$ and compute the private key as

$$[sk_{id} = (h^{r_1 t_1 t_2 + r_2 t_3 t_4}, h^{-\omega t_2}(h_0 h_1^{id})^{-r_1 t_2}, h^{-\omega t_1}(h_0 h_1^{id})^{-r_1 t_1}, \\ (h_0 h_1^{id})^{-r_2 t_4}, (h_0 h_1^{id})^{-r_2 t_3})]$$

**Encrypt(*params*,*id*,*M*).** To encrypt a message $M \in G_t$ first pick $s, s_1, s_2 \in \mathbb{Z}_p$ and then create the ciphertext

$$C = (\Omega^s \cdot M, (g_0 g_1^{id})^s, v_1^{s-s_1}, v_2^{s_1}, v_3^{s-s_2}, v_4^{s_2}).$$

**Decrypt(*sk*$_{id}$,*C*).** Parse $sk_{id}$ as $(d_0, d_1, d_2, d_3, d_4)$ and $C$ as $(c', c_0, c_1, c_2, c_3, c_4)$ and output

$$M = c' \cdot e(c_0, d_0) \cdot e(c_1, d_1) \cdot e(c_2, d_2) \cdot e(c_3, d_3) \cdot e(c_4, d_4)$$

### 4.3.2 Blind Key Derivation Protocol

To fulfill the security requirements, the blind key derivation protocol *ComBlindExtract*$(KGC(params, s_k, C),\ U(params, id, open_{id}))$ needs to be a cryptographic two-party protocol that outputs a randomly distributed secret key to the user. The randomness needs to be chosen jointly by the user and the KGC in a way that neither of the two parties learns the other party's randomness. Namely, a user that learns the KGC's randomness can potentially decrypt messages of other users, whereas a KGC that learns the user's randomness could break the blindness of the keys.

The KGC picks random $\hat{r}_1, \hat{r}_2 \in \mathbb{Z}_p^*$, and the user picks random $r_1', r_2', \in \mathbb{Z}_p^*$. Now the blind key derivation protocol can be implemented using standard techniques as a secure two party protocol in which the input of the user is $r_1', r_2'$ and the input of the KGC is $\omega, t_1, t_2, t_3, t_4, \hat{r}_1, \hat{r}_2$. The user's output should be a secret key:

$$[sk_{id} = (h^{r_1 t_1 t_2 + r_2 t_3 t_4}, h^{-\omega t_2}(h_0 h_1^{id})^{-r_1 t_2}, h^{-\omega t_1}(h_0 h_1^{id})^{-r_1 t_1}, (h_0 h_1^{id})^{-r_2 t_4}, (h_0 h_1^{id})^{-r_2 t_3})]$$

with $r_1 = \hat{r}_1 r_1'$ and $r_2 = \hat{r}_2 r_2'$, while the KGC does not learn any valuable information.

By decomposing this protocol into subprotocols whose results only require simple arithmetic operations (addition and multiplication), we are able to obtain an efficient protocol.

1. The KGC picks random $\hat{r}_1, \hat{r}_2 \in \mathbb{Z}_p^*$, and the user picks random $r_1', r_2', u_0 \in \mathbb{Z}_p^*$ and $u_1, u_2, u_3 \in \mathbb{Z}_p$. Now the KGC and the user have to run an efficient two-party computation protocol for simple arithmetic computations. The input of the user is $r_1', r_2'$ and the blinding values $u_0, u_1, u_2, u_3$, whereas the input of the KGC is $\omega, t_1, t_2, t_3, t_4, \hat{r}_1, \hat{r}_2$.

   In addition, the user provides a commitment $C_{u_0}$ to $u_0$, and the KGC provides commitments $C_{\hat{r}_1}$ and $C_{\hat{r}_2}$ to $\hat{r}_1$ and $\hat{r}_2$ respectively. Both parties prove that their input corresponds to the committed values. Moreover, the KGC proves that $\omega, t_1, t_2, t_3, t_4$ correspond to the master secret key.

The KGC obtains

$$x_1 = (\hat{r}_1 r'_1 t_1 t_2 + \hat{r}_2 r'_2 t_3 t_4) + u_1 \; mod \; p$$
$$x_2 = -((u_0/r'_1 \cdot \omega t_2) + u_2) \; mod \; p$$
$$x_3 = -((u_0/r'_1 \cdot \omega t_1) + u_3) \; mod \; p,$$

and the user obtains commitments $C_{x_1}$, $C_{x_2}$ and $C_{x_3}$ to these values.

We give an instantiation for these protocols based on homomorphic encryption in Section 4.3.3. Jarecki and Shmatikov [JS07] give a protocol for two-party computation on committed input.

2. The user uses the blinding value $u_0$ to compute $ID' \leftarrow (h_0 h_1^{id})^{u_0}$. The user executes a proof of knowledge with the KGC to show that the value was constructed correctly and that the $id$ in $ID'$ corresponds to the $id$ in $C$. Details on this proof can be found in Section 4.3.3.

3. If the proof fails, the KGC returns $\perp$. Otherwise it computes its reply as follows

$$d'_0 = h^{x_1}$$
$$d'_1 = h^{x_2}(ID')^{-\hat{r}_1 t_2}$$
$$d'_2 = h^{x_3}(ID')^{-\hat{r}_1 t_1}$$
$$d'_3 = (ID')^{-\hat{r}_2 t_4}$$
$$d'_4 = (ID')^{-\hat{r}_2 t_3}.$$

4. The KGC sends the vector of $d'_i$ values, and engages the user in a proof of knowledge that $d'_0, \ldots, d'_4$ were constructed correctly. Details on this proof can be found in Section 4.3.3.

5. If the proof fails the user returns $\perp$. Otherwise she computes

$$[sk_{id} = (d_0, d_1, d_2, d_3, d_4) = (d'_0/h^{u_1}, (d'_1 h^{u_2})^{r'_1/u_0},$$
$$(d'_2 h^{u_3})^{r'_1/u_0}, {d'_3}^{r'_2/u_0}, {d'_4}^{r'_2/u_0})]$$

**Theorem 6** *The protocol ComBlindExtract provides a leak free and selective-failure blind committed issuing protocol for the Boyen and Waters anonymous-IBE scheme.*

*Proof. leak freeness:* No efficient distinguisher $D$ can distinguish Game Real (where $A$ is interacting with an honest $P$ running the *ComBlindExtract* protocol) from Game Ideal (where S is given access to a trusted party that executes the *Extract* algorithm). $S$ works as follows:

**1.** On input *params* from the trusted party, $S$ hands *params* to a copy of $A$ it runs internally.

**2.** Using its rewinding capabilities, $S$ extracts the adversary's input $r'_1, r'_2, u_0, u_1, u_2, u_3$ to the two party computation protocol.

**3.** The adversary sends $ID' = (h_0 h_1^{id})^{u_0}$ together with a proof of knowledge of a correct representation of $ID'$ and that $id$ in $ID'$ corresponds to $id$ in $C$. Using again its rewinding capabilities, $S$ extracts the values $id$, $open_{id}$ and $u_0$.

**4.** $S$ submits $id$ to the trusted party, who returns the valid secret key for this identity $sk_{id} = (d_0, d_1, d_2, d_3, d_4)$.

**5.** Finally, $S$ computes $(d_0 \cdot h^{u_1}, d_1^{u_0/r'_1}/h^{u_2}, d_2^{u_0/r'_1}/h^{u_3}, d_3^{u_0/r'_2}, d_4^{u_0/r'_2})$ and returns these values to the adversary.

We note that the responses of $S$ are correctly formed and drawn from the same distribution as those of $P$, so the distinguisher cannot distinguish between the two games.

*Selective-failure blindness:* The adversary provides *params*, and two identities $id_0, id_1$. Now the game chooses a random bit $b$. The adversary has blackbox access to two oracles $Oracle_{BlindObtain}(params, id_b)$ and $Oracle_{BlindObtain}(params, id_{1-b})$.

Note that once an oracle $Oracle_{BlindObtain}$ is activated, the adversary can run a two party protocol with the oracle, the result of which are three randomly distributed values in $\mathbb{Z}_p$. In the next step the oracle provides a randomly distributed value in $G_2$, $ID'$, to the adversary. Then it performs a zero-knowledge proof with the adversary.

Suppose that $A$ runs one or both of the oracles up to this point. Up to now the distributions of the two oracles are computationally indistinguishable. (Otherwise we could break the security of the two-party computation protocol, the semantic security of the commitment scheme or the witness indistinguishability of the zero-knowledge proof. The latter is implied by the zero-knowledge property of the proof system.)

Now $A$ must provide values $(d'_0, d'_1, d'_2, d'_3, d'_4)$ and a proof that these values were correctly computed. We can assume that $A$ chooses these values using an arbitrary complex strategy. Now we show that any adversary $A$ can predict the output $sk_{id_b}$ of $Oracle_{BlindObtain}(params, id_b)$ without further interaction with the oracles:

1. If the proof fails, it records $sk_{id_0} = \perp$. Otherwise, the adversary temporarily records $sk_{id_0} = Extract(params, s_k, id_0)$.

2. In turn, $A$ generates different $(d'_0, d'_1, d'_2, d'_3, d'_4)$ and executes a second proof of knowledge, now for the second oracle. It performs the same checks and recordings for $sk_{id_1}$ and $id_1$.

3. Finally the adversary predicts $(sk_{id_0}, sk_{id_1})$, if both $sk_{id_0} \neq \perp$ and $sk_{id_1} \neq \perp$; $(\epsilon, \perp)$, if only $sk_{id_1} = \perp$; $(\perp, \epsilon)$, if only $sk_{id_0} = \perp$; and $(\perp, \perp)$, if $sk_{id_0} = sk_{id_1} = \perp$.

These predictions result in the same distributions as the one returned by the oracle, as the same checks are performed. Moreover, note that for the case that keys are returned by $Oracle_{Aux}$ they are in both cases equally distributed random keys because of the random values $r'_1$ and $r'_2$ contributed by the oracles. □

### 4.3.3 Subprotocols for the Blind Key Derivation Protocol

In this section we detail some protocols used as building blocks in the construction of the blind key derivation protocol presented in the previous section. First we show the two-party protocols used in Step 1 and after that we detail the proofs of knowledge that should be run in Step 2 and in Step 4.

#### Two-Party Protocol for Simple Arithmetics

These protocols use an additive homomorphic encryption scheme with encryption and decryption functions $Enc$ and $Dec$. Thus $Enc(x) \otimes y = Enc(xy)$ and $Enc(x) \oplus Enc(y) = Enc(x + y)$. The key pair is generated by the KGC and is made available to the user. The protocols depicted in Figure 4.3 and in Figure 4.4 are used to compute the values $x_1$ and $x_2$ respectively. The protocol for $x_3$ is the same as the protocol for $x_2$ but changing $u_2$ for $u_3$ and exchanging $t_1$ for $t_2$.

#### Proofs of Correct Key Derivation

**Proof for Step 2.** The KGC has commitment $C_{u_0}$ to $u_0$, and $C$ to the user's choice of $id$.

$$\begin{aligned} PK\{&(id, u_0, id \cdot u_0, open_{id}, open_{u_0}, open_{id} \cdot u_0) : \\ &C = h_0^{id} h_1^{open_{id}} \wedge C_{u_0} = h_0^{u_0} h_1^{open_{u_0}} \wedge \\ &1 = C^{u_0} (1/h_0)^{id \cdot u_0} (1/h_1)^{open_{id} \cdot u_0} \wedge ID' = h_0^{u_0} h_1^{id \cdot u_0} \} \end{aligned}$$

The user proves that $id$ is correctly encoded in $ID'$. This is the step in which the user proves that the identity she is submitting to the KGC is the same identity that was used to compute the commitment. The way by which the user shows that she has been authorized to obtain a secret key for that identity depends on the application. We show a way to achieve this in Section 4.4.

| User$(r'_1, r'_2, u_1, v_1, \ldots, v_4, C_{\hat{r}_1}, C_{\hat{r}_2})$ | | KGC$(\hat{r}_1, \hat{r}_2, t_1, t_2, t_3, t_4)$ |
|---|---|---|
| | | $e_1 = Enc(\hat{r}_1 t_1 t_2)$ |
| | $\xleftarrow{e_1, e_2}$ | $e_2 = Enc(\hat{r}_2 t_3 t_4)$ |
| | $\xleftrightarrow{PK_1}$ | |
| $e_{x_1} = (e_1 \otimes r'_1) \oplus (e_2 \otimes r'_2) \oplus Enc(u_1)$ | $\xrightarrow{e_{x_1}}$ | |
| | $\xleftrightarrow{PK_2}$ | |
| | | $x_1 = Dec(e_{x_1}) =$ |
| | | $\hat{r}_1 r'_1 t_1 t_2 + \hat{r}_2 r'_2 t_3 t_4 + u_1$ |
| | | $open_{x_1} \leftarrow \mathbb{Z}_p$ |
| | $\xleftarrow{C_{x_1}}$ | $C_{x_1} = h_0^{x_1} h_1^{open_{x_1}}$ |
| | $\xleftrightarrow{PK_3}$ | |
| $PK_1 = PK\{(\rho_1, open_{\rho_1}, \rho_2, open_{\rho_2}, \tau_1, \tau_2, \tau_3, \tau_4) :$ | | |
| $v_1 = g^{\tau_1} \wedge v_2 = g^{\tau_2} \wedge v_3 = g^{\tau_3} \wedge v_4 = g^{\tau_4} \wedge$ | | |
| $C_{\hat{r}_1} = Commit(\rho_1, open_{\rho_1}) \wedge C_{\hat{r}_2} = Commit(\rho_2, open_{\rho_2}) \wedge$ | | |
| $e_1 = Enc(\rho_1 \tau_1 \tau_2) \wedge e_2 = Enc(\rho_2 \tau_3 \tau_4)\}$ | | |
| $PK_2 = PK\{(\rho'_1, \rho'_2, \mu_1) : e_{x_1} = (e_1 \otimes \rho'_1) \oplus (e_2 \otimes \rho'_2) \oplus Enc(\mu_1)\}$ | | |
| $PK_3 = PK\{(\chi, open_\chi) : e_{x_1} = Enc(\chi) \wedge C_{x_1} = h_0^\chi h_1^{open_\chi}\}$ | | |

Figure 4.3: Protocol for deriving $x_1$ Step 1

| User$(r'_1, u_0, u_2, v_1, v_2, \Omega)$ | | KGC$(\omega, t_1, t_2)$ |
|---|---|---|
| | $\xleftarrow{e}$ | $e = Enc(\omega t_2)$ |
| | $\xleftrightarrow{PK_1}$ | |
| $e_{x_2} = ((e \otimes u_0 / r'_1) \oplus Enc(u_2)) \otimes -1$ | $\xrightarrow{e_{x_2}}$ | |
| | $\xleftrightarrow{PK_2}$ | |
| | | $x_2 = Dec(e_{x_2}) =$ |
| | | $-((u_0/r'_1)\omega t_2 + u_2)$ |
| | | $open_{x_2} \leftarrow \mathbb{Z}_p$ |
| | $\xleftarrow{C_{x_2}}$ | $C_{x_2} = h_0^{x_2} h_1^{open_{x_2}}$ |
| | $\xleftrightarrow{PK_3}$ | |
| $PK_1 = PK\{(\omega', \tau_1, \tau_2) :$ | | |
| $v_1 = g^{\tau_1} \wedge v_2 = g^{\tau_2} \wedge \Omega = e(g,h)^{t_1 t_2 \omega'} \wedge e = Enc(\omega' t_2)\}$ | | |
| $PK_2 = PK\{(\rho'_1, \mu_0, \mu_2) : e_{x_2} = -((\mu_0 / \rho'_1 \otimes e) \oplus Enc(\mu_2))\}$ | | |
| $PK_3 = PK\{(\chi, open_\chi) : e_{x_2} = Enc(\chi) \wedge C_{x_2} = h_0^\chi h_1^{open_\chi}\}$ | | |

Figure 4.4: Protocol for deriving $x_2$ in Step 1

**Proof for Step 4.** The user has commitments $C_{\hat{r}_1}, C_{\hat{r}_2}$ and $C_{x_1}$, $C_{x_2}$, $C_{x_3}$. The KGC does the following proof of knowledge:

$$
\begin{aligned}
PK\{&(\hat{r}_1, \hat{r}_2, open_{\hat{r}_1}, open_{\hat{r}_2}, t_1, t_2, t_3, t_4, x_1, x_2, x_3, open_{x_1},\\
&open_{x_2}, open_{x_3}, -\hat{r}_1 t_1, -\hat{r}_1 t_2, -\hat{r}_2 t_3, -\hat{r}_2 t_4) :\\
&C_{\hat{r}_1} = h_0^{\hat{r}_1} h_1^{open_{\hat{r}_1}} \wedge C_{\hat{r}_2} = h_0^{\hat{r}_2} h_1^{open_{\hat{r}_2}} \wedge\\
&v_1 = g^{t_1} \wedge v_2 = g^{t_2} \wedge v_3 = g^{t_3} \wedge v_4 = g^{t_4} \wedge\\
&C_{x_1} = h_0^{x_1} h_1^{open_{x_1}} \wedge C_{x_2} = h_0^{x_2} h_1^{open_{x_2}} \wedge C_{x_3} = h_0^{x_3} h_1^{open_{x_3}} \wedge\\
&1 = (1/v_1)^{\hat{r}_1} (1/g)^{-\hat{r}_1 t_1} \wedge 1 = (1/v_2)^{\hat{r}_1} (1/g)^{-\hat{r}_1 t_2} \wedge\\
&1 = (1/v_3)^{\hat{r}_2} (1/g)^{-\hat{r}_2 t_3} \wedge 1 = (1/v_4)^{\hat{r}_2} (1/g)^{-\hat{r}_2 t_4} \wedge\\
&d'_0 = h^{x_1} \wedge d'_1 = h^{x_2} ID'^{-\hat{r}_1 t_2} \wedge d'_2 = h^{x_3} ID'^{-\hat{r}_1 t_1} \wedge\\
&d'_3 = ID'^{-\hat{r}_2 t_4} \wedge d'_4 = ID'^{-\hat{r}_2 t_3}\}
\end{aligned}
$$

By means of this proof the KGC demonstrates to the user that it uses the correct values for $t_1, t_2, t_3, t_4, \hat{r}_1, \hat{r}_2$ when it computes $[d'_0, d'_1, d'_2, d'_3, d'_4]$.

### 4.3.4 Modifications to achieve Adaptive-id Security

So far we have presented a blind key derivation protocol for an anonymous-IBE scheme that is selective-id (IBE-IND-sID-CPA) secure. According to Boyen and Waters [BW06], a technique described in [Wat05] and in [Nac05] should be applied to achieve adaptive-id (IBE-IND-CPA) security. This technique consists in expressing any identity $id$ as a vector of subcomponents which are small integers. In other words, each $id$ is a bit string of length $N$ belonging to the identity space $I(k)$ and represented by $n$ blocks of $l$ bits each.

Therefore, we have to modify the IBE scheme to implement this technique. First of all, *SetupIBE* should output two random set of generators $\{g_{1,1}, \ldots, g_{1,n}\}$ and $\{h_{1,1}, \ldots, h_{1,n}\}$ along with the generators $g_1$ and $h_1$. After that, in algorithm *Extract* the element $ID' = h_0 h_1^{id}$ becomes $ID' = h_0 \prod_{i=1}^{n} h_{1,i}^{id_i}$, whereas in algorithm *Encrypt* the value $g_0 g_1^{id}$ becomes $g_0 \prod_{i=1}^{n} g_{1,i}^{id_i}$. The reason why this technique provides adaptive-id security is discussed in [Wat05] and in [Nac05].

In order to provide a blind key derivation protocol for this adaptive-id secure scheme, we need to modify three things in Step 2 of the protocol depicted in the previous sections. First, $ID' = (h_0 h_1^{id})^{u_0}$ becomes again $ID' = (h_0 \prod_{i=1}^{n} h_{1,i}^{id_i})^{u_0}$. In addition, the proof of knowledge in this step should be modified in order to prove that each element of the identity lies in an interval $0 \leq id_i < 2^l$. This can be achieved by using the proofs for interval checks showed in Section 2.9.3. Finally, the commitment utilized

by the user in this proof has to be of the form $C = h_0^{open_{id}} \prod_{i=1}^{n} h_{1,i}^{id_i}$. The resultant proof works as follows:

**Modified proof for Step 2.** The KGC has commitment $C_{u_0}$ to $u_0$, and $C$ to the user's choice of $id$.

$$PK\{(id_1, \ldots, id_n, u_0, id_1 \cdot u_0, \ldots, id_n \cdot u_0, open_{id}, open_{u_0}, open_{id} \cdot u_0) :$$
$$C = h_0^{open_{id}} \prod_{i=1}^{n} h_{1,i}^{id_i} \wedge \bigwedge_{i=1}^{n} 0 \leq id_i < 2^l \wedge C_{u_0} = h_0^{u_0} h_1^{open_{u_0}} \wedge$$
$$1 = C^{u_0} \prod_{i=1}^{n} ((1/h_{1,i})^{id_i \cdot u_0})(1/h_0)^{open_{id} \cdot u_0} \wedge ID' = h_0^{u_0} \prod_{i=1}^{n} h_{1,i}^{id_i \cdot u_0}\}$$

**Theorem 7** *The protocol ComBlindExtract provides a leak free and selective-failure blind committed issuing protocol for the adaptive-id secure Boyen and Waters anonymous-IBE scheme.*

*Proof. leak freeness:* This proof is very similar to the one provided for the selective-id secure scheme. $S$ again extracts $A$'s input $r_1'$, $r_2'$, and $u_0, u_1, u_2, u_3$ to the two party computation protocol. In the next steps $S$ extracts $id_1, \ldots, id_n$, reconstructs the identity $id$ and submits it to the trusted party, who returns the secret key $sk_{id} = (d_0, d_1, d_2, d_3, d_4)$. Finally, $S$ computes $(d_0 \cdot h^{u_1}, d_1^{u_0/r_1'}/h^{u_2}, d_2^{u_0/r_1'}/h^{u_3}, d_3^{u_0/r_2'}, d_4^{u_0/r_2'})$ and returns these values to the adversary.

Like in the other proof the responses of $S$ are correctly formed and drawn from the same distribution as those of $P$, so the distinguisher cannot distinguish between the two games.

*Selective-failure blindness:* In this case, we note that the prediction of $A$ can be done in the same way as in the previous proof. Therefore, $A$ must be able to distinguish the two distributions by using the two party computation protocol, the value $ID'$ or the commitments, and this would mean that $A$ is able to break the security of the two party computation protocol, the semantic security of the commitment scheme or the witness indistinguishability of the zero-knowledge proofs. □

## 4.4 Application: Data Retention

The "Directive 2006/24/EC of the European Parliament and of the Council on the retention of data generated or processed in connection with the provision of publicly available electronic communications services or of public communications networks" aims at harmonizing "Member States provisions concerning the obligations of the providers of publicly available electronic communications services or of public communications networks with respect

to the retention of certain data which are generated or processed by them, in order to ensure that the data are available for the purpose of the investigation, detection and prosecution of serious crime, as defined by each Member State in its national law"(Article 1.1). This shall be applied to "traffic and location data on both legal entities and natural persons and to the related data necessary to identify the subscriber or registered user. It shall not apply to the content of electronic communications, including information consulted using an electronic communications network"(Article 1.2).

We provide a solution for this scenario that achieves three goals. First, Internet Service Providers (ISP's) or other telecommunication service providers to which the directive mentioned above will be applied can store this data in an efficient and secure manner. Second, it is possible for an investigator to efficiently retrieve data needed for her investigation, whereas the rest of data remains unrevealed. Finally, the ISP gains no information about which data is requested by the investigator.

The first two goals can be achieved by applying PEKS, but not the third requirement. The reason is that an investigator cannot compute trapdoors herself, and PEKS does not permit to obtain trapdoors from a third party in a blind fashion. Fortunately, if we implement PEOKS, then the investigator can use the $BlindTrapdoor$ protocol to achieve this goal.

We start by describing this scenario and by showing how we apply PEOKS to it. In Section 4.4.2 we explain in detail the procedure that should be used to store the data in an efficient and secure way. Finally, we show how to build temporary keyword search in Section 4.4.3 and a possible modification to improve the efficiency in Section 4.4.4.

### 4.4.1 Data Retention Scenario

We have a scenario with five parties: the ISP, the database, the investigator, the TGC and the judge. The ISP is from now on every provider to which the directive shall apply. It stores records of data in a database that contains the information specified by the directive in a way that it is possible to search for records related with specific keywords. The investigator is interested in getting information related with one or more keywords from the database. The TGC is the party that gives to investigators trapdoors for keywords if they have authorization to search for information related to those keywords. The judge is the party that authorizes the investigator to get information related with one ore more keywords.

The scenario is depicted in Figure 4.5. At the beginning, the TGC runs algorithm $SetupPEOKS$, keeps the secret key $s_k$ and publishes the parameters of the scheme. Then, an ISP stores in the database all data in an encrypted manner, in order to provide privacy and security. It uses algorithm $PEOKS$ to obtain one ore more searchable encryptions that were computed using keywords that describe the content of the data. The way

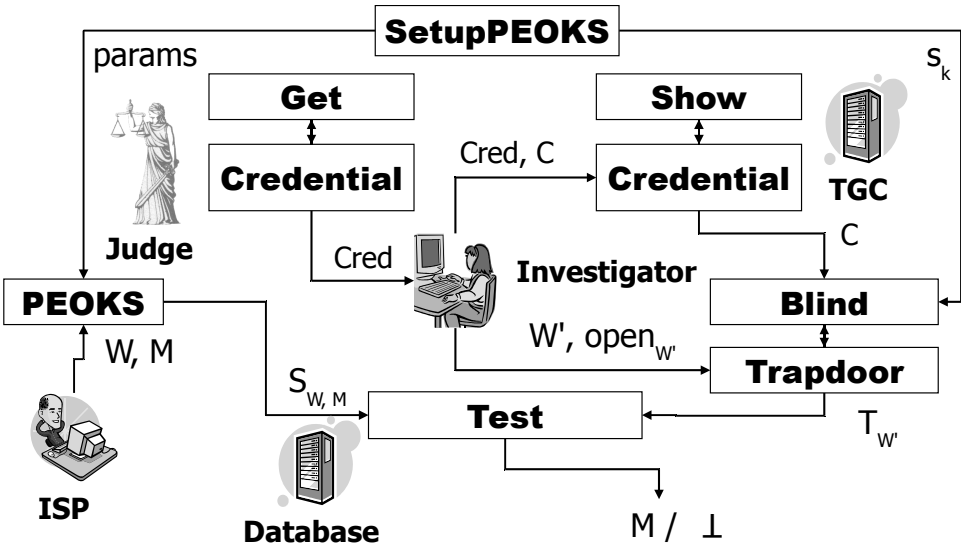


Figure 4.5: PEOKS in the data retention scenario.

by which searchable encryptions and records of data are related is specified in Section 4.4.2. Finally, an investigator who wants to get some information from the database follows these steps:

**1.** The investigator gets authorization from the judge. In practice, it means that they run a protocol *GetCredential* by means of which the investigator gets a credential $Cred$ from the judge (see Section 2.11). The messages used as input of the protocol are keywords that describe the information that the investigator wants to get from the database. Recall that in the protocol depicted in Section 2.11 some messages are known only by the owner (investigator), whereas the other messages are known by both parties. This means that there exists the possibility that the investigator could get a credential for keywords that are not revealed to the judge. The use of this feature might depend on the law of each member state of the EU. Furthermore, there exist protocols for getting credentials that make it possible that some messages be jointly chosen by the owner and the issuer, which may be interesting for this application. The reader interested is referred to [Koh04].

**2.** The investigator gets trapdoors from the TGC. For each trapdoor, the investigator should follow three steps. First, she sends a commitment $C = Commit(params_{comm}, W, open_W)$ to the keyword for which she wants to get a trapdoor. Second, she and the TGC run protocol *ShowCredential*, by means of which she proves that she owns a valid credential issued by the judge that was created by utilizing the same keyword used to compute the commitment (see Section 2.11). Finally,

she and the TGC execute protocol $BlindTrapdoor$ with $W$ and $open_W$ as inputs of the investigator and $C$ and $s_k$ as inputs of the TGC, by means of which she gets a trapdoor $T_W$.

**3.** The investigator searches for information in the database. Once the investigator is given private access to the database, she uses algorithm $Test$ to look for the searchable encryptions that match the trapdoors she has obtained. The output of algorithm $Test$ gives her the necessary information to decrypt the records of data associated with a searchable encryption. This step is also detailed in Section 4.4.2.

### 4.4.2 Storage of Data

In the following, let $Enc(K, M)$ and $Dec(K, C)$ be the encryption and decryption algorithms of any symmetric key encryption scheme, where $K$ is a fresh random key, $M$ denotes the data that is going to be encrypted and $C$ the data to be decrypted.

Whenever the ISP has to store data in order to fulfill the requirements expressed in the directive, it builds a record $R$ and encrypts it by running $Enc(K_R, R)$. This is done in order to protect the content from unauthorized parties. After that, it stores the encrypted record in the database. For the purpose of providing an efficient way to search in the database, it also selects keywords that describe the content of the record. For example, the identifier of the sender and of the receiver of a communication may act as keywords.

At this point, the traditional mechanism that was used in other scenarios to which PEKS was applied consists in generating searchable encryptions for these keywords and attaching them to the record. In this approach the investigator has to compute the $Test$ algorithm for the searchable encryptions attached to each record of data in order to obtain all the information related to a keyword, which is inefficient. Therefore, we present a new procedure in which there is only one searchable encryption per keyword, and once the investigator finds the searchable encryption that matches her trapdoor she is able to obtain all the information related to the keyword corresponding to the trapdoor without further searching. On the other hand, it also achieves better privacy properties in the sense that the investigator gains no information about the number of keywords attached to a record. In addition, this procedure also preserves the integrity of the database in such a way that if the database is compromised any attempt to alter its content can be detected.

Specifically, we propose to use linked lists whose nodes are stored in a random manner. There will be one linked list per keyword, and each node of the linked list contains the necessary information to get and decrypt a record related to this keyword. This information consists of a pointer to the record of data $P_R$ and the key $K_R$ used to encrypt the record. Apart from

that, it also stores a pointer to the next node of the list and the key that will be used to encrypt the next node of the list.

Therefore, when the ISP selects a keyword $W$ for which a searchable encryption has not been computed yet, it picks a symmetric key $K_{N_1}$ and runs algorithm $PEOKS(params, W, K_{N_1}||P_{N_1})$ to compute the searchable encryption, where $P_{N_1}$ is the pointer to the first node of the list and $K_{N_1}$ is the symmetric key used to encrypt this node. Then it builds the node $N_1 = \{P_R, K_R, P_{N_2}, K_{N_2}\}$, it computes algorithm $Enc(K_{N_1}, N_1)$ and it stores the node in the position given by $P_{N_1}$. Finally, it keeps values $P_{N_2}$ and $K_{N_2}$, which are the pointer and the key for the second node of the list, and stores in position $P_{N_2}$ some specific flag to indicate the end of the list.

Then, when the ISP selects again this keyword to describe another record $R'$, it builds the second node $N_2 = \{P_{R'}, K_{R'}, P_{N_3}, K_{N_3}\}$, runs $Enc(K_{N_2}, N_2)$ and stores the encrypted record in the position given by $P_{N_2}$. It keeps again $P_{N_3}$ and $K_{N_3}$ in order to be able to add another node to the list and it stores the flag in $P_{N_3}$. This procedure will be applied as many times as necessary to add more nodes to the list.

On the other hand, when an investigator searches for information in the database, she uses a trapdoor and the searchable encryptions to run the $Test$ algorithm. When both correspond to the same keyword, she gets the key $K_{N_1}$ and the pointer $P_1$ to access the first node of the list. Each node gives her the necessary information to get one record of data described by that keyword and also the information required to access the next node of the list. When she reaches the end of the list she has got all the records related with that keyword.

It can be seen that a malicious party (who has not got authorization to obtain trapdoors) that compromises the ISP can modify neither the records nor the content of the lists. Namely, the only not encrypted information available in the database is the position and the key to encrypt the next node of the list, which is not useful for those purposes. On the other hand, in order to prevent him from deleting contents in a secret way, mechanisms that aim at preserving the integrity of a database, such as hash chains, can be applied [WBDS04]. In conclusion, he can only add more records to the database, but this is impossible to prevent according to [SK98].

### 4.4.3 PETOKS

According to Directive 2006/24/EC, the data should be retained "for periods of not less than six months and not more than two years from the date of the communication" (Article 6). In the solution proposed in the previous section we do not provide a mechanism to efficiently delete data that should not be retained anymore. Furthermore, we do not give the possibility of authorizing searches restricted to specific periods of time, and we do not allow the investigator to retrieve only data associated with one keyword and

generated in a specific time period.

These problems can be solved by applying to PEOKS the notion of temporary keyword search that exists in PEKS (see Section 3.2.4), and thus constructing Public-key Encryption with Temporary Oblivious Keyword Search (PETOKS). Recall that in temporary keyword search the searchable encryptions and the trapdoors are related to a specific time period in such a way that, even if the keyword used to compute them is the same, they do not match if the time period is different. The simplest way to build PETOKS consists in concatenating keywords and time periods when computing searchable encryptions and trapdoors.

In order to apply this notion to our scenario, we should divide the time into periods of the same length, e.g., months. Then, one possibility would consist in concatenating the name of the month and the year to all the keywords that are used to generate searchable encryptions in that month. With this kind of keywords the judges can authorize searches in a specific time period and the investigators can also get the information that belongs to a specific period. Finally, the data that should not be retained anymore is deleted by eliminating all the searchable encryptions that were generated in the same time period as the data.

However, we note that in this solution different linked lists are used for the same keyword concatenated with different time periods. Therefore, if the investigator wants to perform searches to retrieve the data related with one keyword regardless of the time period, she has to be authorized by the judge to perform searches in each time period independently. For example, if the investigator wants to get the information generated during the last two years, she has to get a credential for twenty-four keywords in which only the time period varies, and later on get twenty-four trapdoors.

Another possibility would be to construct only one linked list per keyword, whereas the searchable encryptions are again generated by concatenating the time period to the keyword. Then, when the time period changes and a new node is added to the list, a new PEOKS element that gives access directly to this node is generated for the new time period. In this solution it is possible to get all the information related with a keyword by obtaining authorization to search for this keyword concatenated with the oldest time period, because it gives access to the whole list. However, now the searches can only be restricted to time intervals that end in the present.

In conclusion, the adoption of either the first or the second solution may depend on what is more often required by judges or investigators.

### 4.4.4 Efficiency Improvement

We recall that in our application the investigator needs to be given private access to the database. The reason is that, despite the number of possible keywords is large, there will be a subset of frequently used keywords. There-

fore, the database can compute searchable encryptions for keywords of this subset and check whether the $Test$ algorithm does not output $\perp$ on input the trapdoors sent by the investigator.

However, downloading a copy of the database may be impossible because of its size. One solution to avoid this problem is to download only the searchable encryptions. The construction of the searchable encryptions and the linked lists needs to be modified in such a way that, instead of having a pointer to a record of data and to the next node of the list, they now have an index that allows the investigator to make a PIR query in order to retrieve the record and the next node of the list. Once the investigator has downloaded the searchable encryptions and has obtained the trapdoors, she can search herself for the indexes and after that she can use PIR queries to retrieve the records without revealing the indexes.

## 4.5 Other Applications of Blind-Anonymous-IBE

In this section we propose other applications of blind-anonymous-IBE. In Section 4.5.1 we show how to build an OKS scheme in the random oracle model by using any blind-anonymous-IBE scheme. In Section 4.5.2 we explain how blind-anonymous-IBE can be utilized to reduce the trust in the KGC, and, finally, we mention that it can also be used to obtain blind signatures.

### 4.5.1 Construction of an OKS scheme

In Section 3.3.2, we shown a transformation due to [GH07] by means of which it is possible to obtain an $OT^N_{k\times 1}$ scheme by using any blind-IBE scheme. In that case it was not possible to derive an OKS scheme because the IBE scheme was not anonymous and thus the receiver could use $A_j$ to know all the keywords used by the sender (when using keywords instead of selection values).

However, thanks to the anonymity property we can provide a similar transformation to build an $OKS^N_{k\times 1}$ scheme from any blind-anonymous-IBE scheme. As in Transformation IBE-to-OT, this transformation also uses random oracles and the resultant OKS scheme is secure in the full-simulation model under static complexity assumptions.

**Transformation 4 (IBE-to-OKS) Initialization phase.** *The sender runs* $S_I((M_{W_1}, W_1), \ldots, (M_{W_N}, W_N))$ *and the receiver runs* $R_I()$*:*

1. *The sender runs* $SetupIBE(1^\kappa)$ *to get the secret key* $s_k$ *and params, the parameters of the scheme. He also chooses a function* $H$ *that is modeled as a random oracle.*

2. *Given the set of message-keyword pairs* $\{(M_{W_1}, W_1), \ldots, (M_{W_N}, W_N)\}$*, for* $j = 1, \ldots, N$ *the sender selects random values* $\{Z_{W_1}, \ldots, Z_{W_N}\}$*, runs* $A_{W_j} = Encrypt(params, W_j, Z_{W_j})$ *and creates the element* $B_{W_j} = H(j, W_j, Z_{W_j}) \oplus (0^{2\kappa} || M_{W_j})$*. Let* $C_{W_j}$ *be* $(A_{W_j}, B_{W_j})$*.*
3. *The sender conducts this proof of knowledge* $PK\{(s_k) : (params, s_k) \in SetupIBE(1^\kappa)\}$*. The receiver aborts if the proof fails.*
4. *The sender transmits* $(params, C_{W_1}, \ldots, C_{W_N})$ *to the receiver.*
5. *The sender outputs state value* $S_0 = (params, s_k)$ *and the receiver outputs* $R_0 = (params, C_{W_1}, \ldots, C_{W_N})$*.*

**Transfer phase.** *The sender runs* $S_T(S_{i-1})$ *and the receiver runs* $R_T(R_{i-1}, \hat{W}_i)$ *during the ith phase* $(i \in \{1, \ldots, k\})$*:*

1. *The receiver engages with the sender in protocol ComBlindExtract* $(S(params, s_k, C_{\hat{W}_i}),\ R(params, \hat{W}_i, open_{\hat{W}_i}))$ *in order to obtain the secret key* $sk_{\hat{W}_i}$ *for the keyword* $\hat{W}_i$*. If the ComBlindExtract protocol fails the receiver sets* $M'_{\hat{W}_i}$ *to* $\perp$*.*
2. *For* $j = 1, \ldots, N$*, the receiver executes* $Z'_{W_j} = Decrypt(sk_{\hat{W}_i}, A_{W_j})$ *and computes* $X = H(j, \hat{W}_i, Z'_{W_j}) \oplus B_{W_j}$*. When* $W_j = \hat{W}_i$*,* $Z'_{W_j} = Z_{W_j}$ *and thus the first* $2\kappa$ *bits of* $X$ *are 0. Then she stops searching and she sets* $M'_{\hat{W}_i}$ *in such a way that* $X = (0^{2\kappa} || M'_{\hat{W}_i})$*. If the search is not successful for all* $j$*, she sets* $M'_{\hat{W}_i}$ *to* $\perp$*.*
3. *The sender outputs state value* $S_i = S_{i-1}$ *and the receiver outputs* $R_i = R_{i-1}$ *and the message* $M'_{\hat{W}_i}$*.*

We refer to [CNS07] for the reason why it is necessary to pad the messages with $2\kappa$ zeroes instead of $\kappa$. Now we prove that this construction fulfills the definition of security for OKS given in Section 3.4.

**Theorem 8** *If the IBE scheme is blind-IBE-IND-ANO-CPA secure and the proof of knowledge that is run in the initialization phase fulfills the zero-knowledge property, then the OKS scheme described in Transformation 4 is sender secure in the random oracle model.*

*Proof.* We have to demonstrate that for any real-world cheating receiver $\tilde{R}$ we can construct an ideal-world receiver $\tilde{R}'$ such that no probabilistic polynomial time algorithm $D$ can distinguish the distributions $Real_{S,\tilde{R}}(N, k, (M_{W_1}, W_1), \ldots, (M_{W_N}, W_N), \hat{W}_1, \ldots, \hat{W}_k)$ and $Ideal_{S',\tilde{R}'}(N, k, (M_{W_1}, W_1), \ldots, (M_{W_N}, W_N), \hat{W}_1, \ldots, \hat{W}_k)$ with non-negligible probability.

$\tilde{R}'$ interacts with $\tilde{R}$ and the trusted party as follows. $\tilde{R}'$ first runs $SetupIBE(1^\kappa)$ to generate the scheme parameters *params* and the secret

key $s_k$. Then $\tilde{R}'$ computes $(C_{W_1}, \ldots, C_{W_N})$ formed by setting $(A_{W_1}, \ldots, A_{W_N})$ and $(B_{W_1}, \ldots, B_{W_N})$ to be random bit strings. Finally, $\tilde{R}'$ proves knowledge of $s_k$ and hands $(params, C_{W_1}, \ldots, C_{W_N})$ as input to $\tilde{R}$ to obtain initial state $R_0$.

In the $i$th transfer phase, when $\tilde{R}$ queries a secret key for a keyword $\hat{W}_i$, $\tilde{R}'$ simulates the honest sender side of the *ComBlindExtract* protocol to output a correct secret key $sk_{\hat{W}_i}$. $\tilde{R}'$ uses the knowledge extractor to extract $\hat{W}_i$ and queries the trusted party to obtain $M_{\hat{W}_i}$. To answer random oracle queries of the form $H(j, W_j, Z_{W_j})$, if the trusted third party states that there is no message attached to keyword $W_j$, $\tilde{R}'$ answers randomly. Otherwise $\tilde{R}'$ follows this procedure. First $\tilde{R}'$ computes a secret key $sk_{W_j}$ for keyword $W_j$. Then $\tilde{R}'$ computes $Z'_{W_j} = Decrypt(sk_{W_j}, A_{W_j})$ on input the element $A_{W_j}$ corresponding to the index $j$. If $Z_{W_j} = Z'_{W_j}$ and there was not any previous query on index $j$ or all of them were answered randomly, $\tilde{R}'$ programs the oracle so that $H(j, W_j, Z_{W_j}) = B_{W_j} \oplus (0^{2\kappa} || M_{W_j})$. If this condition is not fulfilled, $\tilde{R}'$ answers randomly. If the condition is fulfilled, but $\tilde{R}$ did not request a secret key $sk_{W_j}$ for $W_j$, then $\tilde{R}'$ aborts.

When eventually $\tilde{R}$ outputs the final state $R_k$, $\tilde{R}'$ halts with the same output $R_k$. The running time of $\tilde{R}'$ is that of $\tilde{R}$ plus the time of $SetupIBE(1^\kappa)$. Therefore, if $\tilde{R}'$ does not abort, $\tilde{R}'$ provides a perfect simulation of $Real_{S,\tilde{R}}(N, k, (M_{W_1}, W_1), \ldots, (M_{W_N}, W_N), \hat{W}_1, \ldots, \hat{W}_k)$ and so no distinguisher can distinguish the real game from the ideal game with non-negligible probability.

It is clear that, if $\tilde{R}$ is able to cause $\tilde{R}'$ to abort with non-negligible probability, then it must be the case that $\tilde{R}$ can break the blind-IBE-ANO-IND-CPA security of the IBE scheme or that the proof of knowledge that is run in the initialization phase is not zero-knowledge.

□

**Theorem 9** *If the ComBlindExtract protocol is selective-failure blind (with commitment), then the OKS scheme described in Transformation 4 is receiver secure in the random oracle model.*

*Proof.* We have to show that for any real-world cheating sender $\tilde{S}$ we can construct an ideal-world sender $\tilde{S}'$ such that no probabilistic polynomial time algorithm can distinguish the distributions $Real_{\tilde{S},R}(N, k, (M_{W_1}, W_1), \ldots, (M_{W_N}, W_N), \hat{W}_1, \ldots, \hat{W}_k))$ and $Ideal_{\tilde{S}',R'}(N, k, (M_{W_1}, W_1), \ldots, (M_{W_N}, W_N), \hat{W}_1, \ldots, \hat{W}_k))$ with non-negligible probability.

$\tilde{S}'$ interacts with $\tilde{S}$ and the trusted party as follows. On input the message-keyword pairs $(M_{W_1}, W_1), \ldots, (M_{W_N}, W_N)$ it runs $\tilde{S}((M_{W_1}, W_1), \ldots, (M_{W_N}, W_N))$. First, $\tilde{S}'$ answers the random oracle queries made by $\tilde{S}$ by returning random values. $\tilde{S}$ proves knowledge of the secret key $s_k$ and $\tilde{S}'$ uses the knowledge extractor to obtain $s_k$. Then, $\tilde{S}$ sends the vector

$(params, C_{W_1}, \ldots, C_{W_N})$. For all random oracle queries $H(j, W_j, Z_{W_j})$, $\tilde{S}'$ computes a secret key $sk_{W_j}$ for keyword $W_j$. Then $\tilde{S}'$ checks whether $Z_{W_j} = Decrypt(sk_{W_j}, A_{W_j})$ on input the element $A_{W_j}$ corresponding to index $j$, and whether $X = H(j, W_j, Z_{W_j}) \oplus B_{W_j}$ is such that $X = (0^{2\kappa}||M)$. If both conditions are fulfilled, $\tilde{S}'$ sets $M'_{W_j} = M$. Otherwise $\tilde{S}'$ assigns a random value to $M'_{W_j}$. Finally $\tilde{S}'$ sends $((M'_{W'_1}, W'_1), \ldots, (M'_{W'_N}, W'_N))$ to the trusted party.

In order to run the transfer phase, $\tilde{S}'$ sets the state value $R_0$ to $(params, C_{W_1}, \ldots, C_{W_N})$. In the $i$th transfer $\tilde{S}'$ simulates the environment of $\tilde{S}$ by running $R_T(R_{i-1}, W_1)$ to obtain state value $R_i$. $\tilde{S}$ uses $W_1$ as keyword because he is not given the keywords $(\hat{W}_1, \ldots, \hat{W}_k)$. If the output of $R_T$ is $\perp$, then $\tilde{S}'$ sends $b_i = 0$ to the trusted party; otherwise he sends $b_i = 1$. Random oracle queries of the form $H(j, W_j, Z_{W_j})$ are answered by returning $B_{W_j} \oplus (0^{2\kappa}||M'_{W_j})$, whereas the rest of queries are answered with random values.

At the end of the $k$th query, $\tilde{S}$ outputs state $S_k$ and $\tilde{S}'$ outputs the same string $S_k$. It is clear that the only difference between the real game and the ideal game is that in the real game $R_T$ receives the selection keywords $(\hat{W}_i, \ldots, \hat{W}_k)$. We define an hybrid game Game-$i$ in which $R_T$ receives $W_1$ for the first $i$ transfers, and $\hat{W}_i$ for the rest of the transfers. Then, if a distinguisher $D$ is able to distinguish the real game from the ideal game with probability at least $\nu$, we have that $D$ must to be able to distinguish Game-$i$ from Game-$(i+1)$ with probability at least $\nu/k$.

Given this distinguisher, we show how to construct an adversary $A$ that succeeds in breaking the selective-failure blindness of the protocol *ComBlindExtract*. $A$ behaves as $\tilde{S}'$ during the initialization phase, and until the $i+1$ transfer $A$ uses $R_T(\cdot, W_1)$, setting the message that is output in the $i$th transfer phase to be $M'_{\hat{W}_i}$ if the transfer succeeds or $\perp$ if it does not succeed. In the $(i+1)$th transfer phase, $A$ outputs two challenge messages $M_0 = \hat{W}_{i+1}$ and $M_1 = W_1$. Given oracle $Oracle_{R_T}(params, M_b)$ for some random $b \in \{0, 1\}$, $A$ relays messages between $\tilde{S}$ and the oracle. With the oracle $Oracle_{R_T}(params, M_{1-b})$ $A$ interacts in an arbitrary way, most likely causing it to fail. Finally it receives the secret keys $(sk_{M_0}, sk_{M_1})$. If $sk_{M_0}$ is $\perp$ then it sets the message of the $(i+1)$th transfer phase to $\perp$; otherwise, it sets it to $M_{\hat{W}_i}$. For the remaining transfers, it uses $R_T(\cdot, \hat{W}_i)$ to simulate the receiver, setting the messages to $M_{\hat{W}_i}$ if the transfer succeeds and to $\perp$ if it does not succeed.

Finally, $A$ gives the final state $(S_k, (M'_{W_1}, \ldots, M'_{W_k}))$ to the distinguisher $D$. We note that if the bit $b$ is zero then this state is distributed according to Game-$i$, and if $b = 1$ then it this distributed according to Game-$(i+1)$. Therefore, if the distinguisher can distinguish between the two games, then $A$ can use the output of the distinguisher to break the selective-failure blindness of the protocol with probability at least $\nu/k$, because he can guess bit $b$ with

at least this probability.

In addition, we note that $A$ is able to break the semantic security of the commitment scheme. According to Definition 67, $Oracle_{R_T}(params, M_b)$ sends a commitment $C_{M_b}$ to $M_b$. Since $A$ can use the distinguisher to guess bit $b$, $A$ can know which message was used to compute commitment $C_{M_b}$. □

### 4.5.2 Reducing Trust in the KGC

Since in not blind-IBE the KGC learns the secret key corresponding to any identity when deriving the key, it has to be completely trusted. Namely, it can perform several malicious activities such as decrypting messages intended for other users or distributing private keys for any identity without any risk of being confronted in a court of law. The reason for the latter is that if the user finds someone who has the same secret key as her, she cannot demonstrate that it was not her who revealed her secret key. However, if this third person has a secret key for her identity different from hers, she can prove that the KGC has derived two secret keys for the same identity. Therefore, we should avoid that the KGC learns the secret keys when deriving them.

For this purpose, so far two approaches have been taken. One of them consists in employing multiple KGC's [BF01]. The master secret key is distributed to multiple KGC's in a way that nobody knows it completely, and for deriving the secret keys for identities more than one KGC is needed. This approach avoids placing the trust in a single KGC. However, it increases the costs of communication and infrastructure.

The second approach is called accountable-authority-IBE [Goy07]. It consists in using a secure key generation protocol between the user and the KGC, by means of which the KGC does not learn the secret key for the identity of the user.

We note that in blind-IBE the KGC does not learn the secret key either. The problem now is that the KGC does not learn the identity, so everybody could obtain a secret key for every identity. However, in our case we can make the user open the commitment before running the protocol, and thus we can achieve the goals that were described in [Goy07] for the Boyen and Waters IBE scheme.

### 4.5.3 Blind Signature Scheme

Each adaptive-id secure IBE scheme implies an existentially unforgeable signature scheme. Therefore, an adaptive-id secure blind-IBE scheme implies an unforgeable, selective-failure blind signature scheme. This result also applies to the selective-id secure scheme when this scheme is instantiated with appropriately-sized parameters and a hash function [BB04a].

# Chapter 5

# Priced Oblivious Transfer

As explained in Section 3.3, an Oblivious Transfer (OT) scheme allows a receiver to get data of her choice from a sender without the sender learning anything about the receiver's choices, whereas the sender is guaranteed that the receiver learns nothing about the rest of the data. Now we want to extend this primitive in order to provide a solution by means of which the sender is able to charge for the data that he provides, while the receiver can pay without the sender learning how much she has paid.

This primitive is called Priced Oblivious Transfer (POT). The existing POT scheme aims at utilizing the specific characteristics of the problem to give an efficient solution. However, in this chapter we use a different approach. Namely, we provide what we call an Oblivious Payment (OP) scheme, which can be combined with any Committed OT (COT) [JS07] scheme in order to construct a POT scheme.

First, we introduce POT and we review the previous scheme. Then we define the concept of OP, of COT and of POT. In Section 5.3 we show our OP scheme, and in Section 5.4 we modify it in order to provide extra features. Finally, in Section 5.5 we give a concrete POT scheme that utilizes our OP scheme and a COT scheme based on the OT scheme secure in the standard model due to [CNS07].

## 5.1 Introduction

Priced Oblivious Transfer (POT) is a primitive that extends the concept of Oblivious Transfer in order to apply it to the buying and the selling of digital goods. We have a scenario with two parties: a vendor and a buyer. Like in OT, POT provides a solution in which the vendor does not learn the item being bought, whereas the buyer does not get any information about the items that she has not requested. However, now we have an additional requirement: the buyer should pay for the item without letting the vendor know the price of the item she has bought.

The usual approach to fulfill this requirement consists in making the buyer pay an initial deposit. After that, she will be able to retrieve any item of her choice and her current balance will be debited by the item's price. However, she cannot retrieve an item if she has not enough funds. On the other hand, the vendor will not be able to learn anything apart from the initial deposit and the amount of interaction.

At this point, we note that, if all the items have the same price, OT can be successfully applied. Namely, having the initial deposit, the vendor can know the number of items that are bought by means of the amount of interaction, and therefore he can decrease the balance of the buyer. However, OT cannot be used in the realistic scenario in which the items can have different prices. Moreover, POT can provide extra privacy properties. For example, by having a dummy item with price zero, the buyer can hide when she is buying something.

From now on we will show the main characteristics of the Priced Oblivious Transfer scheme proposed by [AIR01], in order to compare it with the scheme proposed in this chapter. In [AIR01] they show a basic scheme and after that they propose important efficiency improvements. Here we will only talk about their best solution, which can only be applied to sell keys.

In this solution they use three main tools: homomorphic encryption, a method for conditional disclosure of secrets described in [GIKM98] and a technique for symmetrically private information retrieval described in [NP99a] and in [NP99b]. It is assumed that each item has a different price (if it is not the case, this can be achieved by adequately scaling the prices and the deposit of the buyer).

In the initialization phase, the buyer sends the vendor an initial deposit and a public key of the homomorphic encryption scheme. Then the buyer conducts a proof of knowledge to ensure that she knows the corresponding secret key and that the public key is well-formed[1]. Finally, the vendor computes an encryption of the deposit under this public key.

Each transfer phase involves only two rounds of communication. In the $i$th transfer phase, the buyer sends a message composed of three elements: a separate encryption of the bits that form the current value of the account $\{E_k(account_{l-1}), \ldots, E_k(account_0)\}$, a separate encryption of the bits that form the price $\{E_k(p_{l-1}), \ldots, E_k(p_0)\}$ and an encryption $E_k(u^i)$. The last encrypted value is used by the vendor to ensure that the buyer has behaved honestly in former transactions.

Upon receiving the message, the vendor creates an encryption of the price $E_k(p) = \sum_j E_k(p_j)2^j$ and updates the account $E_k(account^i) = E_k(account^{i-1}) \ominus E_k(p)$, where $\ominus$ is an operator that produces the ciphertext

[1] In order to avoid this proof the authors propose to use ElGamal encryption scheme because then it is possible to perform a check to ensure the validity of the public key, although they note that in this case the vendor does not know if the buyer has the corresponding secret key.

$E_k(account^{i-1} - p)$. Then he discloses the values $v^i$ and $u^i$ under the condition $(E_k(account^{i-1}) = \sum_j E_k(account_j)2^j) \wedge (0 \leq p \leq account^{i-1}) \wedge (u = u^{i-1})$. We refer to [AIR01] in order to learn how this conditional disclosure is performed[2], but we note that the condition $0 \leq p \leq account^{i-1}$ involves a monotone formula of size $O(l)$ which uses the separate encryption of bits sent by the buyer. The value $v^i$ is used to mask the item.

In addition, the vendor uses the symmetrically private information retrieval technique to disclose the item. Let $G_q$ be a group with a generator $g$ where the DDH assumption holds. The vendor has a sequence of pseudo-random items $(M^{p_0}, \ldots, M^{p_{n-1}})$, where each $M^{p_j} = g^{\prod_{j=0}^{l-1} s_j^{p_j}}$ and $\{(s_0^0, s_0^1), \ldots, (s_{l-1}^0, s_{l-1}^1)\}$ are keys. In order to let the buyer get the item, the vendor discloses $s_j^{p_j} r_j$ of each pair of keys, where $p_j$ is the $j$th bit of the price and $r_j$ is a random value. He also sends the value $g^{1/(r_0 \ldots r_{l-1})}$.

Finally, the buyer uses her secret key to obtain the values $s_j^{p_j} r_j$, $u^i$ and $v^i$. She uses the values $s_j^{p_j} r_j$ to raise $g^{1/(r_0 \ldots r_{l-1})}$ and obtain $M^{p_j}$. The final item is $M^{p_j} + v^i$. After that, the vendor can get the data encrypted under this key by using a PIR query. For further explanations we refer to [AIR01].

The fact that this scheme is only useful to sell keys is not an important limitation, because the more common application for Oblivious Transfer is also the provision of keys.

As it can be seen, this scheme requires only two rounds of communication in each transfer phase. This is optimal since even without providing privacy the buyer needs to specify the item and the vendor has to send it. Both vendor and buyer have to send $O(l)$ encryptions. On the other hand, the buyer needs to perform $O(l)$ public key operations and at most $nl$ secret key operations, which can be a problem if the number of items is big[3].

The scheme proposed in this chapter follows a different approach, which consists in constructing a POT scheme by combining an OP scheme with any COT scheme. The OP scheme will ensure that the buyer updates her account properly and that she pays the right price for the item she wants to obtain.

In terms of amount of interaction we do not achieve the optimal solution, and in terms of computation and communication complexity the efficiency will depend mainly on the COT scheme selected, specially when using the simple OP scheme (see Section 5.4.1). Apart from that, we note that the scheme due to [AIR01] is secure in the half-simulation model (see Section 3.3.1), whereas the security of our scheme will depend on the COT scheme chosen and can thus be in the full-simulation model.

[2]For example, a conditional disclosure of the value x under the condition $u = u^{i-1}$ involves picking a random $\alpha$ and computing $E_k(\alpha(u - u^{i-1}) + x))$. If the buyer knows the secret key and the condition is fulfilled, she can get the value x.

[3]According to the authors, this can be improved by arranging the prices in a TRIE structure.

Another difference is that in their scheme the vendor does not know if the buyer misbehaves, i.e., the vendor cannot know whether she sends a price that does not fulfill $0 \leq p \leq account$, but if she misbehaves once then she cannot get data from the vendor anymore. In our solution the vendor knows when a buyer misbehaves, but the buyer can still get data from the vendor in future transfers if after that she behaves properly. This is contrived for practical reasons, e.g, avoiding denial of service attacks.

In addition, in the standard version of our scheme (see Section 5.3) we let different items have the same price. In Section 5.4.2, we extend this scheme in order to allow the vendor to charge different prices for the same item depending on the buyer.

## 5.2 Definition

In this section we define the concepts of OP and of COT, and we show how to construct a POT scheme by combining an OP scheme and a COT scheme.

An OP scheme must ensure that the buyer can pay to the vendor in such a way that the vendor learns nothing about the amount of money that she has paid and nothing about the item she wants to obtain. At the same time, the vendor is guaranteed that the buyer pays the right price for the corresponding item.

For the following definition, let $p_j$ be the price of the $j$th item, and let $account_0$ be the initial deposit. In addition, let $C_{\sigma_i}$ be a commitment to the selection value in the $i$th transfer phase and $open_{\sigma_i}$ be the randomness used to compute this commitment, and let $params_{comm}$ be the parameters that are used to compute commitments.

**Definition 69 ($OP^N_{k\times 1}$)** *An $OP^N_{k\times 1}$ scheme is a tuple of algorithms $(V_I, B_I, V_T, B_T)$ used in matched pairs. During the initialization phase the vendor runs $V_I(p_1, \ldots, p_N)$, obtains state value $V_0$ and outputs $account_0$ and $params_{comm}$, or $\perp$ indicating failure. The buyer runs $B_I(account_0)$, obtains state value $B_0$ and outputs $params_{comm}$ or $\perp$ indicating failure.*

*During the transfer phase vendor and buyer execute $k$ times $(V_T, B_T)$. In the $i$th transfer, $1 \leq i \leq k$, the vendor runs $V_T(V_{i-1})$, obtains state value $V_i$ and outputs commitment $C_{\sigma_i}$ or $\perp$ indicating failure. The buyer runs $B_T(B_{i-1}, \sigma_i)$, obtains state value $B_i$ and outputs $open_{\sigma_i}$ or $\perp$ indicating failure.*

In this definition, for all prices $(p_1, \ldots, p_N)$ and for all selection values $(\sigma_1, \ldots, \sigma_k) \in \{1, \ldots, N\}$, correctness requires that the buyer pays the price $p_{\sigma_i}$, $1 \leq i \leq k$, for the item $\sigma_i$, that $C_{\sigma_i}$ is a commitment to $\sigma_i$ and that $open_{\sigma_i}$ is the randomness used to compute this commitment.

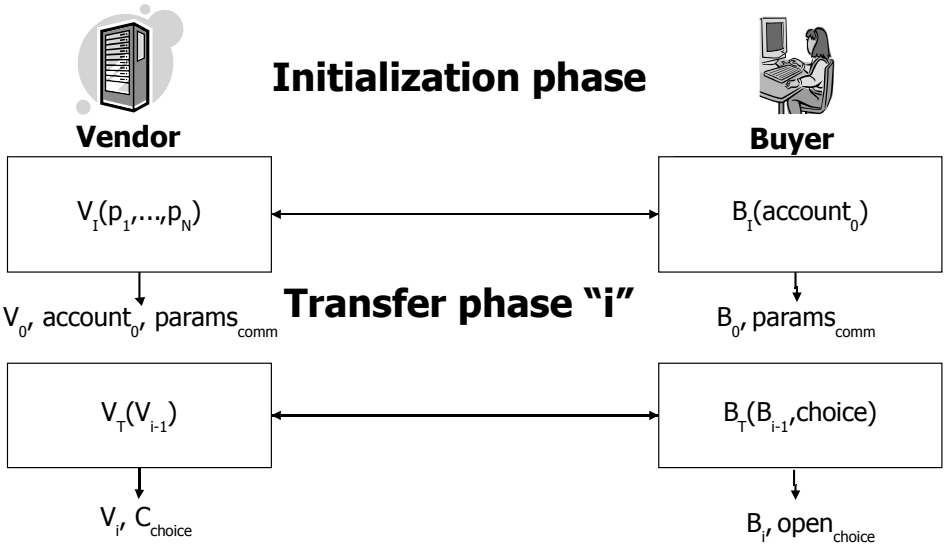


Figure 5.1: Oblivious Payment scheme.

An OP scheme is depicted in Figure 5.1. The commitment to the selection value $C_{\sigma_i}$, which is output by algorithm $V_T$, and the randomness for this commitment $open_{\sigma_i}$, which is output by algorithm $B_T$, are used in the COT scheme in order to ensure that the buyer obtains the same item for which she has paid. The parameters $params_{comm}$, that are output by algorithms $V_I$ and $B_I$, are used in the COT scheme for the same purpose.

In order to make the buyer prove this statement, the COT scheme acts as follows. In the initialization phase both algorithms $S_I$ and $R_I$ of the COT scheme receive as input $params_{comm}$. After that, in each transfer phase, algorithm $S_T$ receives $C_{\sigma_i}$ as input, whereas algorithm $R_T$ receives $open_{\sigma_i}$. Then the receiver conducts a proof by means of which she convinces the sender that her selection value is the same as the one that was used to compute the commitment. Therefore, we define COT as follows:

**Definition 70** ($COT^N_{k\times 1}$) *An $COT^N_{k\times 1}$ scheme is a tuple of algorithms $(S_I, R_I, S_T, R_T)$ used in matched pairs. During the initialization phase the sender runs $S_I(M_1, \ldots, M_N, params_{comm})$ and obtains state value $S_0$, and the receiver runs $R_I(params_{comm})$ and obtains $R_0$. During the transfer phase sender and receiver execute $k$ times $(S_T, R_T)$. During the $i$th transfer, $1 \leq i \leq k$, the sender runs $S_T(S_{i-1}, C_{\sigma_i})$ to obtain $S_i$, and the receiver runs $R_T(R_{i-1}, \sigma_i, open_{\sigma_i})$ to obtain state value $R_i$ and the message $M'_{\sigma_i}$ (or $\perp$ indicating failure).*

In this definition, correctness requires that, for all messages $(M_1, \ldots, M_N)$ and for all selection values $(\sigma_1, \ldots, \sigma_k) \in \{1, \ldots, N\}$, $M'_{\sigma_i} = M_{\sigma_i}$ if $C_{\sigma_i} = Commit(params_{comm}, \sigma_i, open_{\sigma_i})$, and $M'_{\sigma_i} = \perp$ otherwise.

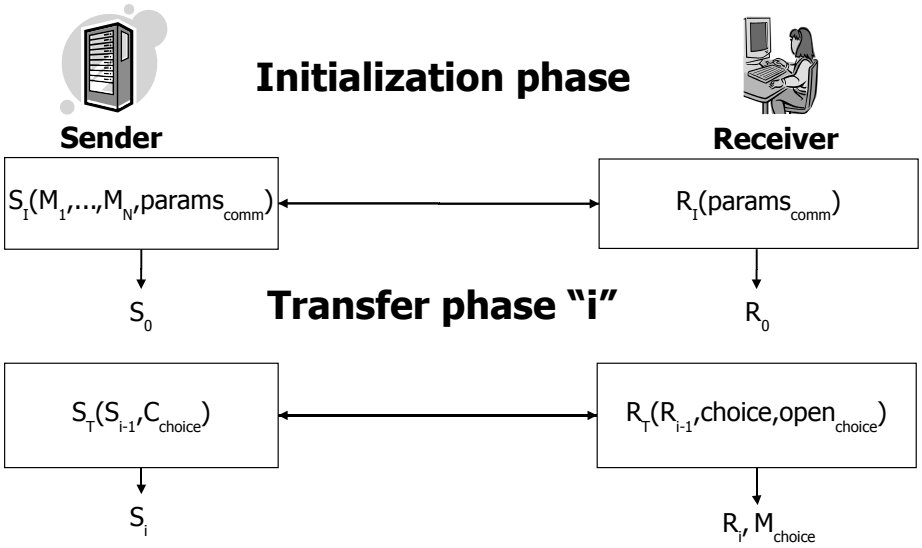


Figure 5.2: Committed Oblivious Transfer scheme.

A generic COT scheme is depicted in Figure 5.2. The way by which the buyer proves that she is requesting the same value for which she has paid when running the $B_T$ algorithm of the OP scheme will be different for each concrete COT scheme. For example, if we use the transformation IBE-to-OT described in Section 3.3.2 to build a COT scheme, and we use for this transformation the blind key derivation protocol for the Boyen and Waters anonymous-IBE scheme that is provided in Section 4.3, this can be done in a straightforward manner. Namely, as in this protocol we have already included a commitment as input for the KGC and the randomness of the commitment as input for the user in order to prove that the identity for which the user requests a secret key is the same as the identity utilized to compute the commitment, we can use them now to prove in the COT scheme that the receiver requests an item whose selection value is the same as the selection value that was utilized to compute the commitment. In addition, in Section 5.5 we show how to modify the OT scheme secure in the standard model due to [CNS07] in order to construct a COT scheme.

Now we will show how to construct a POT scheme by using an OP scheme and a COT scheme. Intuitively, it consists in running, during the initialization phase, algorithms $(V_I, B_I)$ and $(S_I, R_I)$, where $params_{comm}$ is output by both $V_I$ and $B_I$ and is used as input of $S_I$ and $R_I$. In the transfer phase vendor and buyer run $(V_T, B_T)$ and $(S_T, R_T)$, where $C_{\sigma_i}$ is output by $V_T$ and is used as input of $S_T$, and $open_{\sigma_i}$ is output by $B_T$ and is used as input of $R_T$.

**Construction 4** ($POT^N_{k\times 1}$) *Given an* $OP^N_{k\times 1}$ *scheme with algorithms* $(V_I,$

*$B_I, V_T, B_T$), and an $COT^N_{k\times 1}$ scheme with algorithms $(S_I, R_I, S_T, R_T)$, an $POT^N_{k\times 1}$ scheme can be built as follows.*

*During the initialization phase the vendor runs algorithm $\{V_0, account_0, params_{comm}\} \leftarrow V_I(p_1, \ldots, p_N)$ and the buyer runs $\{B_0, params_{comm}\} \leftarrow B_I(account_0)$ as specified in Definition 69. After that, the vendor runs $S_0 \leftarrow S_I(M_1, \ldots, M_N, params'_{comm})$ and the buyer runs $R_0 \leftarrow R_I(params'_{comm})$ as specified in Definition 70.*

*During the transfer phase vendor and buyer execute $k$ times $(V_T, B_T)$ and $(S_T, R_T)$. First, the vendor runs $\{V_i, C_{\sigma_i}\} \leftarrow V_T(V_{i-1})$ and the buyer runs $\{B_i, open_{\sigma_i}\} \leftarrow B_T(B_{i-1}, \sigma_i)$ as specified in Definition 69. After that, if there was no failure in $(V_T, B_T)$, the vendor runs $S_i \leftarrow S_T(S_{i-1}, C'_{\sigma_i})$ and the buyer runs $\{R_i, M'_{\sigma_i}\} \leftarrow R_T(R_{i-1}, \sigma'_i, open'_{\sigma_i})$ as specified in Definition 70. If there was a failure, the vendor sets $S_i = S_{i-1}$, and the buyer sets $R_i = R_{i-1}$ and $M'_{\sigma_i} = \perp$.*

In this definition, correctness, in addition to the requirements made for the underlying $OP^N_{k\times 1}$ and $COT^N_{k\times 1}$ schemes, requires, in the initialization phase, that $params_{comm} = params'_{comm}$, and in the transfer phase, that $C'_{\sigma_i} = C_{\sigma_i}$, that $\sigma_i = \sigma'_i$ and that $open_{\sigma_i} = open'_{\sigma_i}$.

## 5.3 A Standard Oblivious Payment Scheme

In this section we provide a concrete OP scheme. In the initialization phase the buyer sends an initial deposit to the vendor and she also commits to this deposit. After that, in each transfer phase, the buyer updates the commitment to her account by debiting the price of the item she wishes to retrieve. In addition, she has to prove that the commitment is correctly updated and that the new account is non-negative. If the proof fails, the vendor aborts and does not allow her to retrieve the item.

For its construction, we use Damgård-Fujisaki commitments (see Section 2.8.3), and the credential scheme that employs the CL signature scheme (see Section 2.11.3). Note that in the description of the credential scheme given in that section Pedersen commitments (see Section 2.8.2) were used, but it can be easily adapted to use Damgård-Fujisaki commitments.

The main idea behind our scheme is that, in the initialization phase, the vendor issues credentials to the buyer that relate each item with its price. These values are known by the vendor. After that, in each transfer phase the buyer sends the vendor a commitment to her selection value and a commitment to its price, and then she proves that the committed values are equal to the values contained in one of the credentials. Therefore, as this credential was provided by the vendor, the vendor is convinced that the commitments contain a valid item-price pair and thus the buyer can use the commitment to the price to update her account and the commitment

to the item to prove that she is requesting the right value when executing algorithms $(S_T, R_T)$ of the COT scheme.

The OP scheme works as follows:

**Initialization phase.** The vendor runs $V_I(p_1, \ldots, p_N)$ and the buyer runs $B_I(account_0)$:

1. The vendor runs algorithm $SetupCommit(1^k)$ of the Damgård-Fujisaki commitment scheme. Namely, he chooses $\mathfrak{n} = pq$ a special RSA modulus and picks two elements $\mathfrak{g}, \mathfrak{h} \in QR_{\mathfrak{n}}$. He sends $params_{comm} = (\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$ to the buyer and runs the following proof of knowledge with the buyer: $PK\{(\alpha) : \mathfrak{g} = \mathfrak{h}^{\alpha}\}$.
2. The vendor runs $SetupCred(1^k)$ of the credential scheme that uses the CL signature scheme (see Section 2.11). He sends the buyer the public parameters $params_{sig} = (\mathfrak{n}, r_1, r_2, s, z, l_m)$ and provides the following non-interactive proof:

   $$NIPK\{(\rho_1, \rho_2, \alpha) : r_1 = s^{\rho_1} \bmod \mathfrak{n} \wedge r_2 = s^{\rho_2} \bmod \mathfrak{n} \wedge z = s^{\alpha} \bmod \mathfrak{n}\}$$

   Note that for efficiency we can use the same group $QR_{\mathfrak{n}}$ for both $SetupCred(1^k)$ and $SetupCommit(1^k)$. The vendor keeps the secret key $s_k = (p, q)$.
3. For $j = 1$ to $N$, the vendor and the buyer execute protocol $GetCredential(Vendor(input_{vendor}), Buyer(input_{buyer}))$. The input of the vendor is $input_{vendor} = \{params, s_k, M_k, C_s\}$, where $params = (params_{sig}, params_{comm})$, $s_k = (p, q)$, $M_k = \{j, p_j\}$ and $C_s = \emptyset$. The input of the buyer is $input_{buyer} = \{params, M_k, M_s, Open_s\}$, where $M_s = \emptyset$ and $Open_s = \emptyset$. The protocol is depicted in Construction 3. The buyer obtains a set of credentials $\{Cred(1, p_1), \ldots, Cred(N, p_N)\}$.
4. The buyer computes a commitment $C_{account_0} = \mathfrak{g}^{account_0} \mathfrak{h}^{open_{account_0}}$ and sends values $account_0$ and $C_{account_0}$ to the vendor. Then she conducts a proof of knowledge $PK\{(open_{account_0}) : C_{account_0}/\mathfrak{g}^{account_0} = \mathfrak{h}^{open_{account_0}}\}$.
5. The vendor obtains state value $V_0 = (params, s_k, C_{account_0})$ and outputs $account_0$ and $params_{comm}$, whereas the buyer obtains state value $B_0 = (params, Cred(1, p_1), \ldots, Cred(N, p_N), account_0, open_{account_0})$ and outputs $params_{comm}$.

**Transfer phase.** The vendor runs $V_T(V_{i-1})$ and the buyer runs $B_T(B_{i-1}, \sigma_i)$ during the $i$th phase ($i \in \{1, \ldots, k\}$):

1. The buyer computes commitments $C_{p_{\sigma_i}} = \mathfrak{g}^{p_{\sigma_i}} \mathfrak{h}^{open_{p_{\sigma_i}}}$ and $C_{\sigma_i} = \mathfrak{g}^{\sigma_i} \mathfrak{h}^{open_{\sigma_i}}$ and sends them to the vendor. Then they run protocol

$ShowCredential(Vendor(input_{vendor}),\ Buyer(input_{buyer}))$. The input of the vendor is $input_{vendor}$=$\{params,\ C_u,\ M_r\}$, where $C_u = \{C_{p_{\sigma_i}}, C_{\sigma_i}\}$ and $M_r = \emptyset$. The input of the buyer is $input_{buyer}$ =$\{params, Cred, M_r, M_u, Open_u\}$, where $Cred = Cred(\sigma_i, p_{\sigma_i})$, $M_u = \{\sigma_i, p_{\sigma_i}\}$ and $Open_u = \{open_{\sigma_i}, open_{p_{\sigma_i}}\}$. The protocol is depicted in Construction 3. If the protocol fails the vendor aborts.

2. The buyer computes a commitment $C_{account_i} = \mathfrak{g}^{account_i}$ $\mathfrak{h}^{open_{account_i}}$ and sends it to the vendor. Then buyer and vendor run the following proof of knowledge: $PK\{(account_i, open_{account_i}, \alpha) : C_{account_i} = \mathfrak{g}^{account_i}\mathfrak{h}^{open_{account_i}} \wedge C_{account_{i-1}}/(C_{p_{\sigma_i}}C_{account_i}) = h^{\alpha} \wedge account_i \geq 0\}$. If the proof fails the vendor aborts.
3. The vendor outputs state value $V_i = (params, s_k, C_{account_i})$ and the commitment $C_{\sigma_i}$, whereas the buyer outputs $B_i = (params, Cred(1, p_1), \ldots, Cred(N, p_N), account_i, open_{account_i})$ and the value $open_{\sigma_i}$.

Intuitively, in the proof of knowledge that is run in the second step of the transfer phase the buyer can succeed only if $account_i = account_{i-1} - p_{\sigma_i}$ and if $account_i \geq 0$. Therefore, it is ensured that the buyer updates her account correctly and that she has enough funds to buy the item.

## 5.4 Variants of the Oblivious Payment Scheme

In this section we present three possible variants of the OP scheme described in the previous section. The first one aims at improving the efficiency, the second one allows the vendor to provide extra features and the third one provides anonymity.

### 5.4.1 A Simple Oblivious Payment Scheme

As can be seen, two resource-intensive parts of the OP scheme are the issuing and the showing of credentials. These credentials allow us to provide extra features (see below), but if this is not necessary we can apply a simpler solution that avoids the use of credentials.

For this purpose, we change the COT scheme in order to make the price be the selection value, and thus we need that all items have a different price. With this setting, in the standard OP scheme Step 2 and Step 3 of the initialization phase are not needed. In Step 1 of the transfer phase, only one commitment is needed, because now the commitment to the price is also a commitment to the selection value. The rest of this step is also unnecessary. Finally, algorithm $V_T$ should output the commitment to the price and $B_T$ should output the randomness of this commitment. With these changes we construct an OP scheme whose complexity is independent of the number of items that the vendor sells.

### 5.4.2 An Extended Oblivious Payment Scheme

The standard OP scheme allows the vendor to provide extra features. The key observation is that, when issuing the credentials in the initialization phase, the vendor can use a different price for the same item depending on the particular buyer.

This is difficult in the simple OP scheme described above because there the price identifies the item. The only possibility will be to combine it with a COT scheme in which during the initialization phase a different set of commitments is sent to each receiver[4], and thus the sender cannot precompute the commitments. He can have several sets of commitments with different prices each, but to achieve the same level of flexibility it would be necessary to have a different copy for each receiver. In addition, it is necessary to ensure that the receivers do not share the commitments. On the other hand, it also seems difficult to adapt the scheme due to [AIR01] to enable this feature, because in this case the price is used to compute the item, and so the vendor needs to have different Naor-Pinkas pseudorandom sequences $\{(s_0^0, s_0^1), \ldots, (s_{l-1}^0, s_{l-1}^1)\}$ for each buyer such that they produce the same keys.

Therefore, when issuing the credentials, the vendor should also include the identifier of the buyer in the credential in order not to let the buyers share their credentials. Then, during the transfer phase he can check that the credential was issued to that buyer when she shows it. Another possibility would consist in making different groups of buyers. In this case the buyer must prove that she is member of a particular group.

Thanks to this possibility, the vendor can apply some marketing techniques to his business. For example, he can charge different prices for the same item depending on the age of the buyer, or he can offer lower prices to buyers that have made a big initial deposit. For the former, the buyer can show a credential issued by another organization that relates her identity and her age in Step 2 of the initialization phase, and she can conduct a proof of knowledge that ensures that her age belongs to a specific interval (see "proofs of knowledge of the inclusion of a secret key in an interval" in Section 2.9.3).

### 5.4.3 An Anonymous Oblivious Payment Scheme

Another possible feature would be to provide anonymity to the buyers. One could think that, if anonymity is provided, then it is not important to hide the items that are being bought. However, there might be applications in which providing both features can be interesting. For example, in the USA there are companies that sell information about people. If a private

[4]For example, in the OT scheme described in Section 3.3.2 this means to compute $Encrypt(params, j, W_j)$, where j is the price, for each receiver.

detective is looking for a person and thus she wants to buy the information sold by a company, she probably wants to remain anonymous, but in addition she would want that the company does not learn that someone has bought information about that person.

In order to provide anonymity, one straightforward solution would be to integrate the extended OP protocol in a general framework provided by an anonymous credential system [CH02]. In this case, the buyer will be identified by using a pseudonym and to prove, for example, that she is of a particular age she will show a credential issued by another organization during the initialization phase. In order to discourage the sharing of credentials and to ensure their consistency it is possible to use methods provided by the anonymous credential system [CL01].

## 5.5 A Priced Oblivious Transfer Scheme

In this section we provide a POT scheme that employs our simple OP scheme and a COT scheme based on the OT scheme secure in the standard model due to Camenisch, Neven and shelat [CNS07]. Here we review this OT scheme and we also show the modifications that are needed to transform it into a COT scheme. This scheme is used in Chapter 6 for the implementation of a POT scheme.

The COT scheme based on the OT scheme due to [CNS07] is depicted in Figure 5.3. The changes that were needed to transform this OT scheme into a COT scheme were few. In the initialization phase, the sender and the receiver obtain the parameters of the commitment scheme used in the simple OP scheme. In each transfer phase, the vendor obtains the commitment to the selection value and the buyer gets the randomness for this commitment both from the output of the simple OP scheme. Then, in the second proof of knowledge, the sender proves that the selection value and the value used to compute the commitment are the same. The rest of this proof was part of the original OT scheme.

It can be seen that the messages are elements of the target group $G_T$. Therefore, to implement the scheme the authors suggest to extract a random pad from the element in the target group $(e(h, A))$ and xor it with the message.

Finally, by combining the simple OP scheme described in Section 5.4.1 and the COT scheme depicted in this section, a POT scheme can be built in a straightforward manner by following Construction 4.

| INITIALIZATION PHASE | | |
|---|---|---|
| $S_I(1^k, M_1, \ldots, M_N, params_{comm})$ | | $R_I(params_{comm})$ |
| $(q, G_1, G_T, g, h, e) \leftarrow BilinearSetup(1^k)$ | | |
| $H = e(g,h)$; $x \leftarrow \mathbb{Z}_q$; $y \leftarrow g^x$; $params \leftarrow (g, H, y)$ | | |
| For $i = 1, \ldots, N$ do | | |
| $A_i \leftarrow g^{1/(x+i)}$ | | |
| $B_i \leftarrow e(h, A_i) \cdot M_i$ | | |
| $C_i = (A_i, B_i)$ | $\xrightarrow{M = (\ldots)}$ | |
| | $\xleftrightarrow{PK_1\{\ldots\}}$ | |
| $S_0 \leftarrow (h, params, params_{comm})$ | | $R_0 \leftarrow (M, params_{comm})$ |
| **TRANSFER PHASE** | | |
| $S_T(S_{i-1}, C_{\sigma_i})$ | | $R_T(R_{i-1}, \sigma_i, open_{\sigma_i})$ |
| | $\xleftarrow{V}$ | $v \leftarrow \mathbb{Z}_q; V \leftarrow (A_{\sigma_i})^v$ |
| | $\xleftrightarrow{PK_2\{\ldots\}}$ | |
| $W \leftarrow e(h, V)$ | $\xrightarrow{W}$ | |
| | $\xleftrightarrow{PK_3\{\ldots\}}$ | |
| | | $\mathrm{M} \leftarrow B_{\sigma_i}/W^{(1/v)}$ |
| $S_i = S_{i-1}$ | | $R_i = R_{i-1}, M_{\sigma_i}$ |
| $M = (params, C_1, \ldots, C_N)$ | | |
| $PK_1 = PK\{(h) : H = e(g,h)\}\}$ | | |
| $PK_2 = PK\{(\sigma_i, v) : e(V,y) = e(V,g)^{-\sigma_i} e(g,g)^v \wedge C_{\sigma_i} = \mathfrak{g}^{\sigma_i} \mathfrak{h}^{open_{\sigma_i}}\}$ | | |
| $PK_3 = PK\{(h) : H = e(g,h) \wedge W = e(h,V)\}$ | | |

Figure 5.3: The COT scheme based on the OT scheme due to [CNS07].

# Chapter 6

# Implementation of a Priced Oblivious Transfer Scheme

We describe here the implementation of a Priced Oblivious Transfer (POT) scheme that follows the ideas explained in Chapter 5. In particular, this scheme is formed by the simple Oblivious Payment (OP) scheme, which is shown in Section 5.4.1, and the Committed Oblivious Transfer (COT) scheme depicted in Section 5.5, which is based on the Oblivious Transfer (OT) scheme due to [CNS07].

In this chapter we focus on the description of the functions that form the core of the OP and of the COT scheme, which are divided into two libraries, one for the vendor and one for the buyer. Other parts of the implementation, such as an application for the buyer, an interface to communicate with this application, an interface to communicate with the socket's layer, and a client for the buyer and a server for the vendor that were made mainly to test the library, are explained briefly. However, this description can be complemented with the comments that are given in the source code.

To start with, we talk about the tools that were used to construct the implementation. After that, we describe its general structure and we give the necessary instructions to install and run it. In Section 6.3, we focus on the OP scheme and in Section 6.4 we describe the implementation of the COT scheme.

## 6.1 Framework

Two main tools were used to build the implementation: the GNU Multiple Precision arithmetic (GMP) library and the Pairing-Based Cryptography (PBC) library. The OP scheme needs only the GMP library, whereas the COT scheme needs both.

The GMP library is a free library for precision arithmetic that operates

with integers, rational numbers and floating point numbers. According to its authors, it has a number of advantages: it is faster than any other library for operating with large numbers, there is no limit to the precision except the one implied by the hardware resources,... In addition, it has been designed specially for its use in cryptography and internet security applications. For further explanations about the library, we refer to its website[1], where it can be downloaded and a manual and information about its different releases are provided.

In our implementation, we use only the functions that operate with integers and some number theory algorithms, like the Miller-Rabin algorithm to test probabilistically whether a number is prime. The release that we use is the Version 4.2.2.

On the other hand, we use the PBC library to compute bilinear pairings, which is needed for the COT scheme. This is a recent library that was constructed by utilizing the GMP library and whose main purpose is the computation of pairings. It offers a pairing computation time between eleven milliseconds and several hundreds of milliseconds, depending on the parameters used to define the pairing. For further explanations we refer to its website[2], where it can be downloaded along with a manual.

We mainly use the routines for initializing and computing pairings, but also some arithmetic functions. The release that we utilize is the Version 0.4.12.

Since both libraries use C as programming language, we have used C to build our implementation.

## 6.2 General Structure

In this section we give a general overview on how this implementation is structured. We describe its different parts and the content of each file of source code in Section 6.2.1. After that, we explain how to install and run it in Section 6.2.2. Finally, in Section 6.2.3 we show the different types of functions that are part of the libraries of the buyer and of the vendor and the nomenclature used in both libraries.

### 6.2.1 Organization of the Source Code

As mentioned at the beginning of the chapter, the implementation, apart from the libraries for the buyer and for the vendor, has several parts that help us to check their correctness and that can be utilized to build practical applications. In the following we relate the different parts along with the name of their corresponding files and a description of their content.

[1] http://gmplib.org/
[2] http://crypto.stanford.edu/pbc/

**Vendor's library.** It contains the functions that are needed to implement the algorithms $(V_I, V_T)$ of the OP scheme and the algorithms $(S_I, S_T)$ of the COT scheme. The file "VendorLibrary.c" contains the body of the functions and "VendorLibrary.h" contains their respective heads. See Section 6.3 and Section 6.4 for more details.

**Buyer's library.** It contains the functions that are needed to implement the algorithms $(B_I, B_T)$ of the OP scheme and the algorithms $(R_I, R_T)$ of the COT scheme. The file "BuyerLibrary.c" contains the body of the functions and "BuyerLibrary.h" contains their respective heads. See Section 6.3 and Section 6.4 for more details.

**Client and Server.** They are a client for the buyer, in file "BuyerMain.c", and a server for the vendor, in file "VendorMain.c". Both use the libraries mentioned above to build a POT scheme.

**Application.** It contains an application that can be used by a buyer to purchase digital goods. It communicates with the client of the buyer through an interface and it can order initializations and transfers. "BuyerAplic.c" is the main file of the application, whereas "BuyerAplicLibrary.c" and "BuyerAplicLibrary.h" form a library used by the application.

**Interface for the transport layer.** It contains functions that mask the C functions to create and use TCP sockets. These functions are used by the libraries to communicate buyer and vendor. The file "communication.c" contains the body and "communication.h" the heads.

**Interface for the application.** It contains functions that mask the C functions to create and use datagram sockets. They are used by the application and the buyer's library to communicate with each other. The file "interface.c" contains the body of the functions and "interface.h" the heads.

**Hash function.** The files "md5.c" and "md5.h" contain the implementation of the hash function md5 (RFC 1321). It is used to extract a random pad from the element in the target group in the COT scheme (see Section 5.5). The concrete implementation that was utilized is a free and widely used one that can be found in several websites[3].

**Pairing Generation.** It contains an application that creates a file with pairing parameters of Type A (see Section 6.4.5). The file "TypeA-PairingGeneration.c" contains this application.

[3]For example, http://www.fourmilab.ch/md5/

In addition, the file "types.h" contains the types of data used in both libraries along with macros that define the security parameters and other constants.

### 6.2.2 Installation and Execution

In order to install all this source code, it is necessary to install first the GMP library and then the PBC library. After that, by means of the "Makefile" file, which uses gcc as compiler, it is possible to install the vendor, the buyer and the application by typing "make vendor", "make buyer" and "make aplic" respectively. The application to create pairing parameters can also be compiled with "make paramgen".

The vendor needs his library and the interface for the transport layer, the hash function and the file "types.h". The buyer needs her library and the interface for the transport layer, the interface for the application, the hash function and the file "types.h". Finally, the application needs its library, the interface for the application and the file "types.h".

To execute the vendor, the buyer and the application of the buyer it is necessary to type the following commands:

./vendor -fm messages -fp prices [-p port] < params

./buyer [-f socket] [-p port] < params

./aplic [-f socket]

*messages* is a file that contains the messages that the vendor offers to the buyer. Each line contains one message, a blank and the price of the message[4]. The $i$th file contains the message corresponding to the $i$th item. *prices* is a file that contains the price of the $i$th item in the $i$th file. *port* is the number of the listening port that the vendor uses to receive connection queries from buyers. *socket* is the name of the file of the socket that is used to communicate the client of the buyer with the application of the buyer. Finally, *params* is the name of a file that contains the pairing parameters. This file can be created by typing:

./paramgen [-f params] [-r rbits] [-q qbits]

*rbits* is the bit-length of the group order and *qbits* is the bit-length of the base field (see Section 6.4.5).

[4]The prices are also included in this file because in the simple OP scheme the items are identified by the price. If the standard OP scheme is implemented, then only the message is needed, because the selection value is specified by the position of the message in the file.

### 6.2.3 Function Classes and Nomenclature

Both the library of the vendor and the library of the buyer use the same notation in order to facilitate the comprehension of the purpose of a function or of a type of data. There are three generic types of functions in the libraries:

**Main Functions.** These are the functions called by the client of the buyer and the server of the vendor. They offer an interface that implements the algorithms $(V_I, B_I, V_T, B_T)$ of the OP scheme and the algorithms $(S_I, R_I, S_T, R_T)$ of the COT scheme. The functions $BI$, $VT$ and $BT$ of the OP scheme and $ST$ and $RT$ of the COT scheme implement their respective algorithms, whereas algorithms $V_I$, $S_I$ and $R_I$ are split into two functions, *Setup_V* and $VI$ for $V_I$, *Setup_S* and $SI$ for $S_I$, and *Setup_R* and $RI$ for $R_I$. The setup functions contain the parts of the initialization algorithms that need to be computed only once.

**I/O Functions.** These are functions used to exchange information between the buyer and the vendor. The functions used to send information are referred to as *send_message_**, whereas the ones utilized to receive information are called *receive_message_**. $*$ indicates the Main Function that calls a particular function. For example, *receive_message_VT* is called by the Main Function $VT$, and its counterpart, *send_message _BT*, is called by $BT$.

**PoK Functions.** These functions are used to run a proof of knowledge and are named by the tag *PK_**, where $*$ has the same meaning as before. After an I/O Function pair *receive_message_-send_message_*, a PoK function pair is run, in which the party who sends data proves knowledge of it according to the description of the OP and of the COT schemes.

Each *PK_** Function calls four functions that represent the four steps of a proof. First, one out of the pair *send_commitment_*-receive _commitment_**, then *receive_challenge-send_challenge*, after that *send _response_*-receive_response_** and finally *receive_verification-send _verification*. A prover executes the functions on the left of the hyphen and a verifier the ones on the right. Here $*$ stands for the name of the *PK_** Function that invokes the function. Note that the functions for exchanging the challenge and the result of the verification are the same for all proofs.

All the functions return an integer that indicates an error when it is negative. This error is not related with the result of the computation, i.e., it indicates a failure in the execution of the function. For example, the fact

that a prover does not succeed is indicated through a parameter that is output by a PoK Function, not by the returned value.

The general structure of a Main Function consists in invoking first an I/O function and after that a PoK Function. If the proof succeeds, it proceeds with the next I/O-PoK function pair, and otherwise it returns an error. We should note that these functions use the functions of the interface with the transport layer. However, it is possible to change the body of the functions of the interface in order to use a different transport layer.

Apart from that, there are some auxiliary functions that implement some specific algorithms, like, for example, a function to output Sophie Germain primes that is used during the vendor's setup. These functions will be detailed in Section 6.3 and in Section 6.4.

On the other hand, the types of data that were created for both libraries are structures referred to as $s\_*$, where $*$ is a description of the content of the structure. The corresponding variables are called $*$. If there is more than one variable of the same type, a figure is concatenated to $*$.

The main idea of this design is that the only functions that the server and the client need to call in order to construct a POT scheme are the Main Functions. Nevertheless, it is possible to access directly the other kinds of functions, which can be used to implement a different POT scheme.

Most of the parameters that are given as input to the functions of both libraries are pointers to structures or to arrays of structures. We should note that these functions do not reserve memory for these variables, and thus the calling functions are the ones that have to do it.

In order to build a server for the vendor that handles multiple buyers the usual approach of creating threads should be taken. Each thread will call one of the Main Functions, and thus an initialization phase or a subphase of a transfer phase will be completely managed by the thread.

## 6.3 Implementation of an OP Scheme

In this section we explain the implementation of the simple OP scheme that is described in Section 5.4.1. We begin by talking about the data structures and after that we depict the functions that are needed to implement the setup, the initialization and the transfer phase for both the vendor and the buyer. Finally, in Section 6.3.5 we talk about the security parameters of the scheme and in Section 6.3.6 we discuss its efficiency.

As a general remark, we note that, in order to let different items have the same price, we scale the value of the account by multiplying it with a value contained in the macro *scale* of the file "types.h". Then the prices of the items become $scale \cdot p_i - i$, where $p_i$ is the price of the item and $i$ is its corresponding selection value.

### 6.3.1 Data Structures

All the types defined for the implementation of the OP scheme are structures. In the following we show the name of the type and its function:

**s_commitment_DF.** It contains the values $params_{comm} = (\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$, i.e., the public parameters of the Damgård-Fujisaki commitment scheme.

**s_secret_key_V.** It contains the secret key $s_k = (p, q)$, i.e., the two primes such that $\mathfrak{n} = pq$.

**s_alpha.** It contains the value $\alpha$ such that $\mathfrak{g} = \mathfrak{h}^{\alpha}$.

**s_price.** It contains the price of an item.

**s_payment.** It contains the commitments $C_{account_i}$ and $C_{p_{\sigma_i}}$ that the buyer sends to the vendor at the $i$th transfer phase.

**s_payment_buyer.** It contains the values $account_i$, $open_{account_i}$, $price$ and $open_{price}$ that were used to compute the commitments mentioned above.

**s_item.** It contains the commitment $C_{\sigma_i}$ that the buyer sends to the vendor at the $i$th transfer phase.

**s_item_receiver.** It contains the values $\sigma_i$ and $open_{\sigma_i}$ that were used to compute the commitment mentioned above.

**s_commitment_FS.** It contains the commitments $C_{w_1}, C_{w_2}, C_{w_3}, C_{w_4}$ to the four values $w_1, w_2, w_3, w_4$ such that $account_i = w_1^2 + w_2^2 + w_3^2 + w_4^2$. They are used to prove that the account is non-negative (see Section 2.9.3).

**s_FS.** It contains the four values $w_1, w_2, w_3, w_4$ and the values $open_{w_1}$, $open_{w_2}, open_{w_3}, open_{w_4}$ that were used to compute the commitments mentioned above.

**s_commitmentPK.** It contains one commitment that is sent in the first step of a proof of knowledge.

**s_randomnessPK.** It contains one random value that was used to compute a commitment in the first step of a proof of knowledge.

**s_challengePK.** It contains the challenge that is sent in the second step of a proof of knowledge.

**s_responsePK.** It contains one response that is sent in the third step of a proof of knowledge.

For each value, each structure stores two fields: one of type *unsigned char[]* and the other one of type *unsigned int*. The former contains the value that we want to store. It is an array and not an integer because there we store the output of the Export functions of the GMP library. These functions also output the size of the element in bytes, which is stored in the latter field. Therefore, when we want to use a value contained in this structure we need to use the Import function that is provided by the GMP library in order to get back the integer.

The last structure, *s_responsePK*, contains a field "negative", which is used to indicate whether the number is negative or not. This is necessary because the import/export functions of the GMP library use the absolute value. The other structures do not include this field because they never store a negative value.

### 6.3.2 Setup

The setup of the OP scheme includes all the parts of the initialization phase that can be precomputed, i.e., that do not need to be computed anew for each initialization phase. It takes place only in the vendor's side.

The functions that are run in this phase are the Main Function *Setup_V* and the auxiliary functions *setup_commit_DF*, *sophie_germain_prime_generation* and *get_prices*. The function *Setup_V*, on input the name of the file that contains the prices of the items in **file_prices*, calls first the function *setup_commit_DF*, whose behavior is described in detail below. This function runs the function *sophie_germain_prime_generation* and outputs the parameters $(\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$ of the Damgård-Fujisaki commitment scheme in **commitment_DF*, the values $p, q$ such that $\mathfrak{n} = pq$ in **secret_key* and the value $\alpha$ such that $\mathfrak{g} = \mathfrak{h}^{\alpha}$ in **alph*.

Second, *Setup_V* calls *get_prices* on input **file_prices*. This function reads the prices from the file and outputs an array of prices in **price* and the number of elements in **number_prices*. In order to have protection against big files, the macro *size_comm* limits the maximum number of lines that can be read.

Finally, *Setup_V* outputs **commitment_DF*, **secret_key*, **alph*, **price* and **number_prices*.

#### Setup of the Damgård-Fujisaki Commitment Scheme

The setup of the Damgård-Fujisaki commitment scheme (see Section 2.8.3) corresponds to the first step of the initialization algorithm of the OP scheme described in Section 5.3. Recall that this $SetupCommit(1^k)$ algorithm, on input a security parameter $k$, outputs a special RSA modulus $\mathfrak{n}$ of bit-length given by $k$ (see Section 2.5) and two generators $\mathfrak{g}, \mathfrak{h}$ of the group of quadratic residues $QR_{\mathfrak{n}}$.

The first step consists in generating two Sophie Germain primes $p', q'$ and is implemented in function *sophie_germain_prime_generation*. For this purpose, a procedure described in [CS00] has been followed:

1. Generate a random odd number in the interval $2^{\frac{k}{2}-1} \leq p' < 2^{\frac{k}{2}}$.
2. Test if either $p'$ or $p = 2p' + 1$ are divisible by any primes up to some bound B. If so, go back to Step 1.
3. Test if 2 is a Miller-Rabin witness to the compositeness of $p'$. This involves the following steps:
   (a) Express $p' - 1$ as $2^s d$.
   (b) Check if $2^d = 1 \ mod \ p'$.
   (c) For $i = 0$ to $s - 1$, check if $2^{2^i d} = p' - 1 \ mod \ p'$.

   If none of the checks succeed, then 2 is a witness and so go back to Step 1.
4. Test if $2^{p'} = \pm 1 \ mod \ p$. If not, go back to Step 1.
5. Apply the Miller-Rabin algorithm as many times as necessary with randomly selected bases in order to ensure an error probability lower than $\epsilon$.

This procedure is applied twice for the generation of both $p'$ and $q'$.

In Step 2, we use $B = (k/2)^4$. According to [CS00], the probability that a random number passes this test is approximately $0.416/(log \ B)^2$ when $k$ tends to infinity.

On the other hand, according to [CS00] it is desirable to have an error probability $\epsilon < 2^{-80}$. Let $p_{k/2,t}$ be the probability that a composite $p'$ passes the Miller-Rabin algorithm when applying it $t$ times. The formula that relates this probability with the error probability is:

$$p_{k/2,t} \leq \frac{\epsilon}{\epsilon + 2(k/2)}$$

In [DLP93], there is a table that gives the number of times $t$ depending on $k/2$ and the probability $p_{k/2,t}$. For example, for $k/2 = 512$, we need $p \approx 2^{-90}$. Then, following that table we see that $t = 6$ gives a probability $p_{k/2,t} \leq 2^{-91}$ for $k/2 = 500$ and $p_{k/2,t} \leq 2^{-96}$ for $k/2 = 550$.

Therefore, we decided to use, with $k/2 = 512$, a number of times $t = 6$ to ensure a negligible error probability. For further explanations we refer to [CS00].

The second step, which is implemented in function *setup_commit_DF*, consists in generating $(\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$ such that $\mathfrak{n} = pq$, where $p = 2p' + 1$ and $q = 2q' + 1$, and $\mathfrak{g}, \mathfrak{h} \in QR_{\mathfrak{n}}$ such that $g \in \langle h \rangle$.

In order to generate $h \in QR_{\mathfrak{n}}$, we need to pick a random number $h_*$ in $Z_n^*$ and check whether it belongs both to the set of quadratic residues modulo $p$ ($QR_p$) and to the set of quadratic residues modulo $q$ ($QR_q$). This is done by checking both if $h_*^{p'} = 1 \ mod \ p$ and if $h_*^{q'} = 1 \ mod \ q$. If both checks succeed, then $h_*$ is a quadratic residue and we set $\mathfrak{h} = h_*$.

To compute $g$, we recall that in $QR_n$ almost all the elements ($\phi(p'q')$) are generators. Thus, we pick a random $\alpha$ such that $1 < \alpha < |QR_{\mathfrak{n}}|$, where $|QR_{\mathfrak{n}}| = p'q'$, and we compute $g_* = \mathfrak{h}^{\alpha} \ mod \ \mathfrak{n}$. Finally, we set $\mathfrak{g} = g_*$.

### 6.3.3 Initialization Phase

The implementation of the initialization of the OP scheme contains all the parts of the $(V_I, B_I)$ algorithms that need interaction between buyer and vendor. First, the vendor sends the buyer the parameters of the Damgård-Fujisaki commitment scheme and proves knowledge of the value $\alpha$. After that, the buyer sends him the deposit and a commitment to it, and also proves that the commitment is correctly formed.

First we describe the functions of the vendor and after that we depict the functions of the buyer.

#### Vendor Initialization Phase

The functions that are run in this phase on the vendor's side are the Main Function *VI*, the I/O Functions *receive_message_VI* and *send_message_VI* and the PoK Functions *PK_VI_1* and *PK_VI_2*, along with the functions that implement these proofs of knowledge.

The function *VI* is run on input the parameters $(\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$ of the Damgård-Fujisaki commitment scheme in **commitment_DF*, the value $\alpha$ in **alph*, the array of the prices of the items in **price* and the number of elements of the array in **number_prices*. First, *VI* runs *send_message_VI* on input **commitment_DF*, **price* and **number_prices* in order to send to the receiver the parameters of the commitment scheme and the prices of the items.

After that, it calls *PK_VI_1* on input **commitment_DF* and **alph* in order to prove to the buyer that $\mathfrak{g} = \mathfrak{h}^{\alpha}$. For this purpose, *PK_VI_1* invokes *send_commitment_PK_VI_1* to send a value $t = \mathfrak{h}^r$. After receiving the challenge $c$ by means of *receive_challenge*, it runs *send_response_VI_1* in order to send a response $s = r - c\alpha$. Finally, it calls *receive_verification* to get the result, which is output by *PK_VI_1* in result.

If the result is not zero, *VI* returns and outputs **result*. Otherwise it starts the second step by calling *receive_message_VI* in order to receive

the deposit in **deposit* and a commitment $C_{account}$ to the deposit in **payment*. Then, in order to check that this commitment was correctly computed by the buyer, *VI* invokes *PK_VI_2* on input **commitment_DF*, **payment* and **deposit*. After receiving the commitment $t$ by means of *receive_commitment_PK_VI_2*, *PK_VI_2* runs *send_challenge* in order to send a challenge $c$, which is output in **challenge*. It runs *receive_response_PK_VI_2* in order to receive the response $s$ and check whether $t \stackrel{?}{=} \mathfrak{h}^s(C_{account}/\mathfrak{g}^{deposit})^c$. Finally it sends the result by means of *send_verification*, and it outputs the result in **result*.

In the end, *VI* outputs the deposit in **deposit*, $C_{account}$ in **payment* and the result in **result*.

### Buyer Initialization Phase

The functions that are run in this phase in the buyer's side are the Main Function *BI*, the I/O Functions *receive_message_BI* and *send_message_BI* and the PoK Functions *PK_BI_1* and *PK_BI_2*, along with the functions that implement these proofs of knowledge.

The function *BI* receives as input the deposit in *deposit*. First of all, *BI* invokes *receive_message_BI* in order to get the parameters $(\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$ in **commitment_DF*, the array of prices of the items in *price* and the number of elements in the array in **number_prices*. After that, *BI* invokes *PK_BI_1* in order to verify if $(\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$ were computed correctly by the vendor. For this purpose, first *PK_BI_1* calls *receive_commitment_PK_BI_1* in order to get the commitment $t = \mathfrak{h}^r$ in **commitment*, and then it sends the challenge $c$ by invoking *send_challenge*. After that, it receives the response $s = r - c\alpha$ and verifies if $t \stackrel{?}{=} \mathfrak{h}^s\mathfrak{g}^c$ by means of *receive_response_PK_BI_1*. It sends the result that is output by this function to the vendor by using *send_verification*. Finally, *PK_BI_1* outputs the result in **result*. If it is 0, *BI* proceeds with the second step; otherwise it returns and outputs **result*.

The second step begins when *BI* calls *send_message_BI* and gives in *deposit* the deposit $account_0$ as input in order to send both this value and a commitment to the deposit $C_{account_0}$ to the vendor. The value $account_0$ and the randomness of the commitment $open_{account_0}$ are output in **payment_buyer*. Afterwards, it invokes *PK_BI_2* in order to prove that the commitment was computed correctly. First *PK_BI_2* uses *send_commitment_PK_BI_2* to send a commitment $t = \mathfrak{h}^r$, and then it calls *receive_challenge* in order to receive a challenge $c$. After that, it invokes *send_response_PK_BI_2* in order to compute $s = r - open_{account}$ and send $s$ to the vendor, and it receives the result of the verification by running *receive_verification*. It outputs the result in **result*. Finally, *BI* outputs $account_0$ and the randomness of the commitment $open_{account_0}$ in **payment_buyer*, and the result in **result*.

### 6.3.4 Transfer Phase

The implementation of the transfer phase of the OP scheme contains everything necessary to implement the algorithms $(V_T, B_T)$. First we describe the functions of the vendor and after that we depict the functions of the buyer.

#### Vendor Transfer Phase

The functions that are run in this phase in the vendor's side are the Main Function *VT*, the I/O Function *receive_message_VT* and the PoK Function *PK_VT*, along with the functions that implement this proof of knowledge.

First of all, *VT*, which receives as input the parameters $(\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$ in **commitment_DF* and the commitment to the account $C_{account_{i-1}}$ in **payment*, invokes *receive_message_VT* in order to get the commitment $C_{account_i}$ to the new value of the account and the commitment to the price $C_{p_{\sigma_i}}$ both in **new_payment*, and also a commitment to the item[5] $C_{\sigma_i}$ in **item*. In addition, it receives in **commitment_FS* commitments $(C_{w_1}, C_{w_2}, C_{w_3}, C_{w_4})$ to four values $(w_1, w_2, w_3, w_4)$ such that $account_i = w_1^2 + w_2^2 + w_3^2 + w_4^2$.

After that, *VT* runs *PK_VT* to verify that the new commitment to the account was computed correctly, i.e., that the value $account_i$ is not lower than zero and that it equals $account_i = account_i - p_{\sigma_i}$. First, *PK_VT* calls *receive_commitment_PK_VT* to obtain an array of seven commitments in **commitment*. These commitments are of the form $t_{w_1} = \mathfrak{g}^{r_{w_1}} \mathfrak{g}^{r_{open_{w_1}}}$ and similar for $t_{w_2}$, $t_{w_3}$ and $t_{w_4}$, $t_{account_i} = \mathfrak{g}^{r_{account_i}} \mathfrak{h}^{r_{open_{account_i}}}$, $t_{notneg} = C_{w_1}^{r_{w_1}} C_{w_2}^{r_{w_2}} C_{w_3}^{r_{w_3}} C_{w_4}^{r_{w_4}} \mathfrak{h}^{r_\alpha}$, where $\alpha = open_{account_i} - open_{w_1} w_1 - open_{w_2} w_2 - open_{w_3} w_3 - open_{w_4} w_4$, and $t_\beta = \mathfrak{h}^{r_\beta}$, where $\beta = open_{account_{i-1}} - open_{account_i} - open_{p_{\sigma_i}}$. These commitments are mentioned in the same order as they appear in the array.

Second, *PK_VT* runs *send_challenge* in order to send a challenge $c$ to the buyer. Then it invokes *receive_response_PK_VT* in order to receive the twelve responses of the buyer and verify them (see the next subsection for the order of the responses in the array and for their computation). In the input of this function, **commitment* has to be the array of seven structures that was output by *receive_commitment_PK_VT*, whereas the other parameters are pointers to one structure as usual. It verifies whether $t_{w_1} \stackrel{?}{=} \mathfrak{g}^{s_{w_1}} \mathfrak{h}^{s_{open_{w_1}}} C_{w_1}^c$ and it performs a similar check for $t_{w_2}$, $t_{w_3}$ and $t_{w_4}$. It also verifies that $t_{account_i} \stackrel{?}{=} \mathfrak{g}^{s_{accoun_i}} \mathfrak{h}^{s_{open_{account_i}}} C_{account_i}^c$, that $t_{notneg} \stackrel{?}{=} C_{w_1}^{s_{w_1}} C_{w_2}^{s_{w_2}} C_{w_3}^{s_{w_3}} C_{w_4}^{s_{w_4}} \mathfrak{h}^{s_\alpha} C_{account_i}^c$, and that $t_\beta \stackrel{?}{=} \mathfrak{h}^{s_\beta} (C_{account_{i-1}} / (C_{account_i} C_{p_{\sigma_i}}))^c$.

[5] In the simple scheme the commitment to the price and the commitment to the item have the same value. Different variables are used in order to facilitate the extension of this implementation to construct the standard scheme. In order to ensure that the commitment to the price is used as commitment to the item in the transfer phase of the COT scheme, this function copies $C_{p_{\sigma_i}}$ in $C_{\sigma_i}$.

Finally, if all the checks are true, *receive_response_PK_VT* sets **result* to 0 and outputs it. After that, *PK_VT* runs *send_verification* to send the result to the buyer and outputs **result*. Finally, *VT* outputs **result*.

### Buyer Transfer Phase

The functions that are run in this phase in the vendor's side are the Main Function *BT*, the I/O Function *send_message_BT* and the PoK Function *PK_BT*, along with the functions that implement this proof of knowledge.

First of all, *BT* receives as input the selection value in *choice*, the array of the prices of the items in **price*, the number of elements of the array in *number_prices*, the parameters $(\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$ in **commitment_DF* and the value of the account in the last transfer phase $account_{i-1}$ and the random value $open_{account_{i-1}}$ that was used to compute the last commitment to the account both in **payment_buyer*.

*BT* invokes *send_message_BT* with the same inputs. If the selection value is such that $0 \leq \sigma_i < number\_prices$ and the new account $account_i \geq 0$, this function computes the new account value $account_i = account_{i-1} - p_{\sigma_i}$, a commitment $C_{account_i} = \mathfrak{g}^{account_i}\mathfrak{h}^{open_{account_i}}$ to it, a commitment to the price $C_{p_{\sigma_i}} = \mathfrak{g}^{p_{\sigma_i}}\mathfrak{h}^{open_{p_{\sigma_i}}}$, a commitment to the item[6] $C_{\sigma_i} = \mathfrak{g}^{\sigma_i}\mathfrak{h}^{open_{\sigma_i}}$ and four commitments $(C_{w_1}, C_{w_2}, C_{w_3}, C_{w_4})$ to four values $(w_1, w_2, w_3, w_4)$ such that $account_i = w_1^2 + w_2^2 + w_3^2 + w_4^2$, and **result* is set to 0. Otherwise it sets result to another value and returns.

The values $(w_1, w_2, w_3, w_4)$ are computed by means of the Rabin-Shallit algorithm. We have used an improved version proposed in [Lip03]. It works as follows:

**Step 1** Find $t$ and $k \geq 0$ such that $account_i = 2^t(2k+1)$.

**Step 2** Pick random values $w_1 \leq \sqrt{account_i}$ and $w_2 \leq \sqrt{account_i - w_1^2}$ where $w_1$ is even and $w_2$ is odd, and compute $p = account_i - w_1^2 - w_2^2$. Now $p = 1 \ mod\ 4$. Hoping that $p$ is prime, try to express $p$ as $w_3^2 + w_4^2$.

1. Find a solution to the equation $u^2 = -1 \ mod\ p$. If there is no solution for this $p$, go back to the beginning of Step 2.
2. Apply the Euclidean algorithm to $(p, u)$ and take the first two remainders that are less than $\sqrt{p}$ to be $w_3$ and $w_4$ (if $u \leq \sqrt{p}$, use $u$ as the first remainder).
3. If $p \neq w_3^2 + w_4^2$ then $p$ was not prime and thus go back to the beginning of Step 2. If not, return $(w_1, w_2, w_3, w_4)$ as a representation.

**Step 3** If $t$ is odd but not 1, find a representation for $2(2k+1)$ following Step 2. After that, return $(sw_1, sw_2, sw_3, sw_4)$, where $s = 2^{(t-1)/2}$.

[6] In the simple scheme these two commitments are the same.

**Step 4** If $t$ is even find a representation for $2(2k+1)$ following Step 2. After that, if $w_3$ is even, exchange $w_2$ and $w_3$. Otherwise exchange $w_2$ and $w_4$. Then return $(s(w_1+w_2), s(w_1-w_2), s(w_3+w_4), s(w_3-w_4))$, where $s = 2^{t/2-1}$.

The function *compute_representation* implements Step 2. It receives as input a number of the form $2(2k+1)$ in **number* and outputs in **value1*, **value2*, **value3* and **value4* a representation $(w_1, w_2, w_3, w_4)$.

The function *algorithm_Rabin_Shallit*, with the same input and output as *compute_representation*, implements Step 1, Step 3 and Step 4 and calls *compute_representation* in order to implement Step 2. It is called by the function *send_message_BT*.

When all the commitments are computed, *send_message_BT* sends the vendor $C_{account_i}$, $C_{p_\sigma}$, $C_{\sigma_i}$ and the four commitments $(C_{w_1}, C_{w_2}, C_{w_3}, C_{w_4})$. Apart from **result*, it returns the values $account_i$, $open_{account_i}$, $p_{\sigma_i}$ and $open_{p_{\sigma_i}}$ in **new_payment_buyer* and also the last two values in **item*. It also returns the commitments $(C_{w_1}, C_{w_2}, C_{w_3}, C_{w_4})$ in **commitment_FS* and the values $(w_1, w_2, w_3, w_4)$ and the random values $(open_{w_1}, open_{w_2}, open_{w_3}, open_{w_4})$ that were used to compute those commitments in **FS*.

After that, if **result* is not 0 $BT$ returns. Otherwise it runs $PK\_VT$ in order to prove that the commitment $C_{account_i}$ was computed correctly and that $account_i \geq 0$ and $account_i = account_{i-1} - p_{\sigma_i}$. For this purpose, $PK\_VT$ calls *send_commitment_PK_BT*, which sends the vendor seven commitments as described in the previous subsection. It receives as input the values $(\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$ and the commitments $(C_{w_1}, C_{w_2}, C_{w_3}, C_{w_4})$ that were output by *send_message_BT* in **commitment_FS*. It outputs in **randomness* an array of twelve structures that contains the twelve random values that were used to compute the seven commitments. The first position contains $r_{w_1}$, the second $r_{open_{w_1}}$ and the next positions follow the same structure for $w_2$, $w_3$ and $w_4$. The ninth position contains $r_{account_i}$, the tenth $r_{open_{account_i}}$, the eleventh $r_\alpha$ and the twelfth $r_\beta$.

Then $PK\_BT$ invokes *receive_challenge* to get the challenge $c$ from the vendor. Afterwards, it calls *send_response_PK_BT* in order to send twelve responses of the form $s = r - cx$, where $x$ stands for the secret key. It receives as input the challenge $c$ in **challenge*, the array of twelve random values in **randomness*, the values $(w_1, w_2, w_3, w_4)$ and $(open_{w_1}, open_{w_2}, open_{w_3}, open_{w_4})$ in **FS*, the value $open_{account_{i-1}}$ in **payment_buyer* and the values $account_i$, $open_{account_i}$ and $open_{p_{\sigma_i}}$ in **new_payment_buyer*. The responses are stored in the array of responses that is sent to the vendor in the same position that was used to store the random value corresponding to each response in the array of random values.

Finally, $PK\_VT$ runs *receive_verification* and outputs the result in **result*. Then, $BT$ outputs **result*.

| bit-length of $\mathfrak{n}$ | $\bar{t}(s)$ | $S^2(s^2)$ |
|---|---|---|
| 256 | 0.047215 | 0.000730 |
| 512 | 0.262045 | 0.020609 |
| 1024 | 3.839378 | 8.515296 |

Table 6.1: Running time of *setup_commit_DF*

### 6.3.5 Security Parameters

There are three security parameters in this scheme: the bit-length $k$ of the special RSA modulus $\mathfrak{n}$, the bit-length $k'$ of the challenge $c \in \{0,1\}^{k'}$ that is used in the proofs of knowledge and the bit-length $k''$ that is used to define the interval from which the random values $r \in ]-\mathfrak{n}2^{-2+k'+k''}..\mathfrak{n}2^{-2+k'+k''}[$ that are picked for running proofs of knowledge are taken.

These values are defined by three macros, *leng_rsa*, *leng_challenge* and *param_indistin*, in the file "types.h". The current values of these macros are 1024, 80 and 80 respectively.

### 6.3.6 Efficiency

The most resource-intensive functions of the OP scheme are *setup_commit_DF*, which is run in the setup of the vendor, and *algorithm_Rabin_Shallit*, which is run in the transfer phase in the buyer's side. Therefore, we have measured the efficiency of these functions by calculating the average running time after twenty executions.

The running time of *setup_commit_DF* is probabilistic and depends on the value of the macro *leng_rsa*, which gives the bit-length of the special RSA modulus $\mathfrak{n}$. The average running time and the variance are depicted in Table 6.1, in which we can see that the running time of this function grows polynomially with the length of the security parameter.

The running time of *algorithm_Rabin_Shallit* is also probabilistic and depends on the size of the number for which a representation needs to be computed. In our case this number is the product of the account of the buyer in each phase and the bound used to build a scheme in which different items can have the same price. This bound is given by the macro *scale*. In Table 6.2 we show the average running time and the variance for numbers whose order of magnitude ranges from 3 to 9.

As can be seen, the variance of the running time of these functions is big. This is specially undesirable for *algorithm_Rabin_Shallit*, because this function is run in every transfer phase. Therefore it is advisable not to multiply the prices by a large *scale*, although this would mean that prices that in theory are the same in practice are a little bit different.

| Order of magnitude | $\bar{t}(s)$ | $S^2(s^2)$ |
|---|---|---|
| 3 | 0.003984 | 0.000001 |
| 4 | 0.004623 | 0.000002 |
| 5 | 0.045366 | 0.001993 |
| 6 | 0.181229 | 0.074129 |
| 7 | 4.233847 | 35.226078 |
| 8 | 10.868670 | 138.414597 |
| 9 | 227.213165 | 78382.828125 |

Table 6.2: Running time of *algorithm_Rabin_Shallit*

## 6.4 Implementation of a COT Scheme

In this section we explain the implementation of the COT scheme that is described in Section 5.5. We begin by talking about the data structures and after that we depict the functions that are needed to implement the setup, the initialization and the transfer phase for both the sender and the receiver. Finally, in Section 6.4.5 we talk about the security parameters of the scheme and in Section 6.4.6 we discuss its efficiency.

The functions of the implementation of the COT scheme are of the same types and follow the same structure as the functions of the OP scheme. The proofs of knowledge that are run in this scheme are depicted in [CNS07].

### 6.4.1 Data Structures

As for the OP scheme, all the data types that are defined for the COT scheme are structures. In the following we show the name of the type and its function:

**s_public_key.** It contains the values $(g, y, H)$ that are generated during the setup of the COT scheme.

**s_secret_key.** It contains the value $h$ such that $H = e(g, h)$.

**s_commitment.** It contains the values $(A_{p_i}, B_{p_i})$, i.e., the commitments for the $i$th item.

**s_parameter_RT.** It contains the values $(v, V)$ computed by the receiver at the beginning of each transfer phase.

**s_parameter_ST.** It contains the value $V$ that the receiver sends to the sender.

**s_key.** It contains the value $W$ that the sender sends to the receiver at the end of each transfer phase.

In addition, the structures *s_item*, *s_item_receiver*, *s_commitment_DF*, *s_commitmentPK*, *s_randomnessPK*, *s_challengePK* and *s_responsePK*, that are used in the implementation of the OP scheme (see Section 6.3.1), are also used in this scheme. For each value the structure stores an array of type *unsigned char[]* for the value and the length of the value in a field of type *unsigned int* (see Section 6.3.1).

### 6.4.2 Setup

The setup of the COT scheme includes all the parts of the initialization phase that can be precomputed, i.e., that do not need to be computed anew for each initialization phase. It takes place both in the sender's side and in the receiver's side.

The function that is run in this phase in the sender's side is the Main Function *setup_S*. It receives as input the pairing parameters in a file descriptor **stdin* and the messages that are offered in a file **file_messages* (this parameter indicates the name of the file).

Although the PBC library provides functions that generate pairing parameters, we note that these functions are different for each type of pairings. However, the function that initializes pairings works for all the types. Therefore, in order to build a scheme that can be used with all these types[7] there were two approaches: one that consisted in implementing all the possible generations and choose one through a parameter given as input, and the other that consisted in giving a file with the parameters as input. We chose the latter, which means that the generation of the parameters (of any type) should be done before executing the implementation. Another reason is that the library does not provide a way to export the parameters and, since the receiver needs to have the same parameters as the sender, the most natural approach is that the sender publishes the parameters in a file and the receiver also takes this file as input.

The file with the messages contains in each line the message and the price[8]. These elements are separated by a blank.

With this input, first *setup_S* initializes the global variable *pairing* of type *pairing_t*. Then it computes the secret values $(x, h)$ and the parameters $(y, g, H)$. After that, it reads the file and, for each line, it computes a pair of commitments $(A_{p_i}, B_{p_i})$, where the position of the line in the file indicates the index $i$ of the commitment, but the price is used to compute $A_{p_i} = g^{1/(x+p_i)}$. For the computation of $B_{p_i}$ it computes the md5 hash of $e(h, A_{p_i})$ and after that it xores the hash with the message, according to the solution explained in Section 5.5. In order to have protection against big

[7] Note that this scheme uses symmetric pairings.

[8] Recall that we need to introduce the price because in the simple OP scheme the item is identified by its price.

files, the macro *size_comm* limits the maximum number of lines that can be read.

Finally, it outputs the number of commitments that have been created in **number_commitments*, an array of commitments of size given by **number_commitments* in **commitments*, in which the commitment for the $i$th item is stored in the $i$th position of the array, the parameters $(y, g, H)$ in **pk* and the secret value $h$ in **sk*.

In the receiver's side, the setup consists in running the Main Function *setup_R*, which, on input the name of a file that describes the pairing parameters, initializes the global variable *pairing*.

### 6.4.3 Initialization Phase

The implementation of the initialization of the COT scheme contains all the parts of the $(S_I, R_I)$ algorithms that need interaction between sender and receiver. First we describe the functions of the sender and after that we depict the functions of the receiver.

#### Sender Initialization Phase

The functions that are run in this phase in the sender's side are the Main Function *SI*, the I/O Function *send_message_SI* and the PoK Function *PK_SI*, along with the functions that implement this proof of knowledge.

First of all, *SI* runs *send_message_SI* on input the array of commitments in **commitment*, the number of commitments in *number_commitments* and the parameters $(y, g, H)$ in **public_key*. This function sends the receiver these three sets of values.

After that, *SI* calls *PK_SI* on input $(y, g, H)$ in **public_key* and $h$ in **secret_key* in order to prove knowledge of $h$. First, *PK_SI* runs *send_commitment_PK_SI* to compute a commitment $a = e(r, g)$ and send it to the verifier. It outputs the random value $r$ in **randomness*.

Upon receiving the challenge $c$ by means of *receive_challenge*, it invokes *send_response_PK_SI* on input the challenge $c$ in **challenge*, the random value $r$ in **randomness* and the secret value $h$ in **secret_key*. This function computes the response $z = rh^{-c}$ and sends it to the receiver. Then, *PK_SI* calls *receive_verification* to obtain the result and outputs it in **result*. Finally, *SI* also outputs result in **result*.

#### Receiver Initialization Phase

The functions that are run in this phase in the receiver's side are the Main Function *RI*, the I/O Function *receive_message_RI* and the PoK Function *PK_RI*, along with the functions that implement this proof of knowledge.

First of all, *RI* runs *receive_message_RI* in order to obtain an array of commitments of the form $(A_{p_i}, B_{p_i})$ in **commitments*, the size of the array in **number_commitments* and the parameters $(y, g, H)$ in **public_key*.

After that, *RI* calls *PK_RI* to verify that the vendor has knowledge of a value $h$ such that $H = e(h, g)$. For this purpose, *PK_RI* first calls *receive_commitment_PK_RI* to get the commitment $a = e(r, g)$ in **commitment*. Then it sends the challenge $c$ by means of *send_challenge*, and afterwards it calls **receive_response_PK_RI* in order to receive and verify the response sent by the sender. This function checks whether $a \stackrel{?}{=} e(z, g)H^c$ and, if it is the case, sets **result* to 0. Otherwise, it sets **result* to a different value.

Finally, *RI* outputs the result in **result*, the array of commitments of the form $(A_{p_i}, B_{p_i})$ in **commitments*, the size of the array in **number_commitments* and the parameters $(y, g, H)$ in **public_key*.

### 6.4.4 Transfer Phase

The implementation of the transfer phase of the COT scheme contains everything necessary to implement the algorithms $(S_T, R_T)$. First we describe the functions of the sender and after that we depict the functions of the receiver.

#### Sender Transfer Phase

The functions that are run in this phase in the sender's side are the Main Function *ST*, the I/O Functions *receive_message_ST* and *send_message_ST* and the PoK Functions *PK_ST_V* and *PK_ST_W*, along with the functions that implement these proofs of knowledge.

First of all, *ST* receives as input the values $(\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$ in **commitment_DF*, $(y, g, H)$ in **public_key*, $h$ in **secret_key* and the commitment to the item[9] $C_{\sigma_i}$ in **item*. Then it calls *receive_message_ST* in order to get the value $V$ in **parameter_ST*.

After that, *ST* runs *PK_ST_V* with the same inputs as *ST* plus *parameter_ST* in order to verify that the receiver knows a value $v$ such that $V = (A_{p_{\sigma_i}})^v$, that she is requesting an item such that[10] $\sigma_i \in \{0, \ldots, N-1\}$ and that this value is the same as the one used to compute the commitment $C_{\sigma_i}$. For this purpose, *PK_ST_V* runs *receive_commitment_PK_ST_V* in order to get commitments $a = e(V, g)^{-r_1} e(g, g)^{r_2}$ and $t = \mathfrak{g}^{r_1}\mathfrak{h}^{r_3}$ in **commitment1* and **commitment2* respectively.

[9] We use here $C_{\sigma_i}$ instead of $C_{p_{\sigma_i}}$ because the behavior is the same when the price identifies the item and when it is not the case.

[10] The items are indexed from 0 to N-1 in order to facilitate the access to the arrays of commitments. When using prices instead of selection values, it is verified that the receiver uses a price that the sender utilized to compute the commitments.

After sending the challenge $c$ by means of *send_challenge*, *PK_ST_V* invokes *receive_response_PK_ST_V* on input $(\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$ in **commitment_DF*, $(y, g, H)$ in **public_key*, $V$ in **parameter_ST*, $a$ in **commitment1*, $t$ in **commitment2*, $c$ in **challenge* and $C_{\sigma_i}$ in **item*. This function receives the responses $z_1, z_2, s$ (see the next subsection) and verifies whether $a \stackrel{?}{=} e(V, g)^{-z_1} e(g, g)^{z_2} e(V, y)^c$ and whether $t \stackrel{?}{=} \mathfrak{g}^{z_1} \mathfrak{h}^s C_{\sigma_i}^c$. If it is the case it sets **result* to 0, and otherwise to another value. It outputs **result*. Finally, *PK_ST_V* runs *send_verification* in order to send the result to the receiver and outputs the result in **result*.

Then *ST* returns and outputs result in **result* if it is not zero; otherwise it proceeds with the second step. In the latter case, it calls *send_message_ST* on input $h$ in **secret_key* and the value $V$ in **parameter_ST* in order to compute a value $W = e(h, V)$ and send it to the receiver. *send_message_ST* outputs $W$ in **key*.

Afterwards, *ST* invokes *PK_ST_W* on input $(y, g, H)$ in **public_key*, $h$ in **secret_key*, $W$ in **key* and $V$ in **parameter_ST*. This function proves that $W$ was correctly computed. For this purpose, it first runs *send_commitment_PK_ST_W* in order to compute two commitments $a_1 = e(r, g)$ and $a_2 = e(r, V)$ and send them in this order to the receiver. It outputs the random value $r$ in **randomness*.

After receiving the challenge $c$ by means of *receive_challenge*, *PK_ST_W* calls *send_response_PK_ST_W* on input $h$ in **secret_key*, the challenge in **challenge* and the random value $r$ in **randomness*. This function computes the response $z = rh^{-c}$ and sends it to the receiver.

Finally, it receives the result by means of *receive_verification* and it outputs it in **result*. Then *ST* outputs **result*.

### Receiver Transfer Phase

The functions that are run in this phase in the receiver's side are the Main Function *RT*, the I/O Functions *send_message_RT* and *receive_message_RT* and the PoK Functions *PK_RT_V* and *PK_RT_W*, along with the functions that implement these proofs of knowledge. The auxiliary function *obtain_item* is also used.

First of all, *RT* receives as input in **item_receiver* the item $\sigma_i$ and the random value $open_{\sigma_i}$ that were used to compute the commitment $C_{\sigma_i}$ that is used as input of *ST*. It also receives the values $(y, g, H)$ in **public_key*, $(\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$ in **commitment_DF* and the values $(A_{p_{\sigma_i}}, B_{p_{\sigma_i}})$ in **commitment* (it does not receive the whole array of commitments).

Then *RT* calls *send_message_RT* on input $(A_{p_{\sigma_i}}, B_{p_{\sigma_i}})$ in **commitment* in order to pick a random $v$, compute the value $V = A_{p_{\sigma_i}}^v$ and send it to the sender. This function outputs $(v, V)$ in **parameter_RT*.

After that, *RT* calls *PK_RT_V* in order to prove that it knows $v$, that the

selection value is such that $\sigma_i \in \{0, \ldots, N-1\}$ and that $C_{\sigma_i}$ is a commitment to this selection value. For this purpose, *PK_RT_V* calls **send_commitment_PK_RT_V* on input $(y, g, H)$ in **public_key*, $(\mathfrak{n}, \mathfrak{g}, \mathfrak{h})$ in **commitment_DF* and $(v, V)$ in **parameter_RT*. This function picks random values $r_1, r_2, r_3$, computes the commitments $a = e(V, g)^{-r_1} e(g, g)^{r_2}$ and $t = \mathfrak{g}^{r_1} \mathfrak{h}^{r_3}$ and sends them in this order to the sender. It outputs, $r_1$, $r_2$ and $r_3$ in **randomness1*, **randomness2* and **randomness3* respectively.

Upon receiving the challenge $c$ by means of *receive_challenge*, it invokes *send_response_PK_RT_V* on input the challenge in **challenge*, the above-mentioned random values, the values $(v, V)$ in **parameter_RT* and $\sigma_i$ and $open_{\sigma_i}$ in **item_receiver*. This function computes the responses $z_1 = r_1 - \sigma_i c$, $z_2 = r_2 - vc$ and $s = r_3 - open_{\sigma_i} c$ and sends them in this order to the sender.

Finally, *PK_RT_V* receives the result by means of *receive_verification* and outputs it in **result*. Then *RT* returns if result is not zero. Otherwise it proceeds with the second step.

*RT* runs *receive_message_RT* in order to receive the value $W$ in **key*. After that, it runs *PK_RT_W* in order to check that the value $W$ was computed correctly. For this purpose, *PK_RT_W* calls *receive_commitment_PK_RT_W* in order to get commitments $a_1$ and $a_2$ in *commitment1* and *commitment2* respectively. Then it invokes *send_challenge* to send the challenge $c$.

Afterwards, *PK_RT_W* calls *receive_response_PK_RT_W* on input $(y, g, H)$ in **public_key*, $W$ in **key*, $(v, V)$ in **parameter_RT* and the challenge and the commitments mentioned above. This function receives the response $z$ from the sender and verifies whether $a_1 \stackrel{?}{=} e(z, g) H^c$ and $a_2 \stackrel{?}{=} e(z, V) W^c$. If it is the case, it sets the result to zero; otherwise it sets the result to another value. It outputs the result in **result*.

Then *PK_RT_W* sends the result by means of *send_verification* and outputs it. If it is not zero, *RT* returns and outputs it in **result*. Otherwise, it calls *obtain_item*, on input $(A_{p_{\sigma_i}}, B_{p_{\sigma_i}})$ in **commitment*, $(v, V)$ in **parameter_RT* and $W$ in **key*. This function computes the operation $M_{\sigma_i} = B_{p_{\sigma_i}} \oplus hash$, where *hash* is the result of applying md5 to $W^{1/v}$. Finally, it outputs $M_{\sigma_i}$ in the string **item* and result in **result*.

### 6.4.5 Security Parameters

In order to implement pairings, the PBC library offers several pairing types that have different security and efficiency properties. We have chosen the Type A pairing because this is the most efficient one.

The PBC library has one function that generates Type A pairings. As input of this function it is necessary to specify the bit-length of the group order in *rbits* and the bit-length of the base field order in *qbits*.

Therefore, we have to select these bit-lengths in order to ensure the security of the scheme. According to [KM05], to achieve a level of security equivalent to AES-80 we need that $rbits = 160$ and $qbits = 1024$, and

| Number of items | Security Level | $\bar{t}(s)$ | $S^2(s^2)$ |
|---|---|---|---|
| 100 | AES-80 | 6.212601 | 0.007440 |
| 100 | AES-128 | 73.409019 | 0.224027 |
| 200 | AES-80 | 12.320795 | 0.003043 |
| 200 | AES-128 | 145.644363 | 0.260003 |

Table 6.3: Running time of *setup_S*

to achieve a level equivalent to AES-128 we need that $rbits = 256$ and $qbits = 3072$.

By means of the application *TypeAPairingGeneration* it is possible to obtain a file that specifies new pairing parameters for the bit-lengths *rbits* and *qbits* that are given as input. In addition, the files *paramsOT80* and *paramsOT128* contain Type A pairing parameters that provide a level of security equivalent to AES-80 and AES-128 respectively.

### 6.4.6 Efficiency

The most resource-intensive function of the COT scheme is *setup_S*. We have measured its efficiency by calculating the average running time after twenty executions.

The running time of *setup_S* is almost deterministic and depends on the number of items that the vendor wants to sell and on the parameters that are used to compute pairings. The average running time and the variance are depicted in Table 6.3, in which we can see that the running time of this function grows linearly with the number of commitments, and that the pairing parameters that are needed to achieve a security level equivalent to AES-128 lead to a pairing computation time far larger than the ones that offer a security level equivalent to AES-80 (see the previous subsection).

# Chapter 7

# Conclusion and Future Work

In this thesis we have described privacy preserving solutions for two applications: the search by keywords on encrypted data and the buying of digital goods.

For the former, we have defined a new concept, Public-key Encryption with Oblivious Keyword Search (PEOKS), that extends the existing concept of Public-key Encryption with Keyword Search and that offers similar properties to Oblivious Keyword Search. In addition, we have shown how to build a PEOKS scheme from any anonymous Identity-Based Encryption (IBE) scheme with blind key derivation and we have proven that the PEOKS scheme is secure if the blind-anonymous-IBE scheme is secure.

In order to describe an instantiation of a PEOKS scheme, we have built the first blind-anonymous-IBE scheme. Furthermore, we have applied a PEOKS scheme to provide a privacy-enhancing solution for the data retention scenario that is specified by the Directive 2006/24/EC of the EU. Finally, we have proposed other applications of blind-anonymous-IBE.

For the latter, we have constructed a new Priced Oblivious Transfer (POT) scheme following an approach different from previous works. Instead of designing a POT scheme from scratch, we have defined the concept of Oblivious Payment (OP) scheme and we have shown how to combine it with any Committed Oblivious Transfer (COT) scheme in order to build a POT scheme.

Moreover, we have described a simple OP scheme and some extensions that allow us to provide some new features that were not covered in other schemes. In particular, we have shown a solution in which the vendor can ask different buyers to pay different prices for the same item, which allows the vendor to apply marketing techniques to his business. For example, he can offer lower prices to underage buyers or to buyers that made a big initial deposit.

There are still many things that could be done in order to provide a better solution to the search by keywords on encrypted data. For example, we note that when a user searches for information, it is usual that she wants to obtain only the information attached to two or more keywords specified by her, or even to obtain only the information that was attached to some keywords but not to others. This can be formalized by allowing her to specify a predicate.

In the solution presented for the data retention scenario, if the investigator wanted to get the information attached to a predicate of her choice, she would need to obtain all the pointers to records of data for each keyword that belongs to the predicate, and after that to compare all the pointers in order to know if the record of data corresponding to the pointer was attached to all the keywords specified in the predicate.

On the other hand, in the traditional solution, in which a different searchable encryption is computed for each record of data that is attached to the same keyword, this would mean that the searchable encryptions attached to the same record would need to be tested more than once in order to know if the predicate is fulfilled (and this has to be done for each record of data).

Therefore, it would be interesting to have a way to construct searchable encryptions that encrypt predicates of keywords. With these searchable encryptions, the $Test$ algorithm would output the message if the predicate was fulfilled. It might be possible to build such a scheme by utilizing Predicate Encryption [KSW07].

This can also be suitable to improve OKS schemes. Since in OKS every message is attached to only one keyword, now it would be attached to a predicate.

Apart from that, we note that the anonymous IBE scheme we have used to build blind-anonymous-IBE is not as efficient as others. Therefore, it would be convenient to construct a more efficient anonymous IBE scheme for which a blind key derivation protocol is possible. It would also be desirable that in this anonymous IBE scheme the KGC cannot use the master secret key in order to decrypt every message, which is a property that the scheme we have used does not meet. It would help in the design of an IBE scheme that does not need to trust the KGC as much as now.

For the Priced Oblivious Transfer scheme, we note that the extensions of the simple OP scheme have a complexity in the initialization phase that grows linearly with the amount of messages that the vendor offers. Therefore, it seems convenient to find a more efficient scheme with the same level of flexibility, i.e., one in which it is still possible to offer different prices for each user.

On the other hand, the implementation of the simple OP scheme can be extended to the other variants of OP schemes, and it is also possible to combine it with a different COT scheme. In addition, it would be interesting

to integrate the libraries provided here in an e-commerce web service.

Finally, we should note that the two solutions presented in this thesis employ bilinear pairings. It was observed that two important challenges for pairing-based cryptography will be to create more efficient implementations of bilinear pairings and to study in-depth the assumptions under which a pairing-based cryptosystem is supposed to be secure. The former will lead to scalability, whereas the latter will increase the confidence in using this kind of cryptography.

# Bibliography


[ABC+05] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2005.

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In Pfitzmann [Pfi01], pages 119–135.

[Bab85] László Babai. Trading group theory for randomness. In *STOC* [DBL85], pages 421–429.

[BB04a] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In Cachin and Camenisch [CC04], pages 223–238.

[BB04b] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Cachin and Camenisch [CC04], pages 56–73.

[BBDP01] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Boyd [Boy01], pages 566–582.

[BC86] Gilles Brassard and Claude Crépeau. Non-transitive transfer of confidence: A perfect zero-knowledge interactive protocol for sat and beyond. In *FOCS* [DBL86], pages 188–195.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[BCL04] Endre Bangerter, Jan Camenisch, and Anna Lysyanskaya. A cryptographic framework for the controlled release of certified

data. In Bruce Christianson, Bruno Crispo, James A. Malcolm, and Michael Roe, editors, *Security Protocols Workshop*, volume 3957 of *Lecture Notes in Computer Science*, pages 20–42. Springer, 2004.

[BCR86a] Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In Odlyzko [Odl87], pages 234–238.

[BCR86b] Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. Information theoretic reductions among disclosure problems. In *FOCS* [DBL86], pages 168–173.

[BDOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Cachin and Camenisch [CC04], pages 506–522.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Brickell [Bri93], pages 390–420.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In Boyd [Boy01], pages 514–532.

[BM88] László Babai and Shlomo Moran. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.*, 36(2):254–276, 1988.

[BM89] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and spplications. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 547–557. Springer, 1989.

[BN00] Daniel Bleichenbacher and Phong Q. Nguyen. Noisy polynomial interpolation and noisy chinese remaindering. In *EUROCRYPT*, pages 53–69, 2000.

[BNPS01] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The power of rsa inversion oracles and the security of chaum's rsa-based blind signature scheme. In Paul F. Syverson, editor, *Financial Cryptography*, volume 2339 of *Lecture Notes in Computer Science*, pages 309–328. Springer, 2001.

[BNPS03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *J. Cryptology*, 16(3):185–215, 2003.

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In Yvo Desmedt, editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003.

[Boo67] Taylor L. Booth. *Sequential Machines and Automata Theory.* John Wiley and Sons, Inc., New York, 1967.

[Bou00] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*, pages 431–444, 2000.

[Boy01] Colin Boyd, editor. *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*. Springer, 2001.

[Bri93] Ernest F. Brickell, editor. *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, volume 740 of *Lecture Notes in Computer Science*. Springer, 1993.

[BSNS] Joonsang Baek, Reihaneh Safiavi-Naini, and Willy Susilo. Public key encryption with keyword search revisited.

[BW06] Xavier Boyen and Brent Waters. Anonymous hierarchical identity-based encryption (without random oracles). In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 290–307. Springer, 2006.

[Cam98] Jan Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem.* 1998.

[CC04] Christian Cachin and Jan Camenisch, editors. *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*. Springer, 2004.

[CCMLS06] Liqun Chen, Zhaohui Cheng, John Malone-Lee, and Nigel Smart. Efficient id-kem based on the sakai-kasahara key construction. *IEE Proceedings - Information Security*, 153(1):19–26, March 2006.

[CCT07] Sébastien Canard, Iwen Coisel, and Jacques Traoré. Complex zero-knowledge proofs of knowledge are easy to use. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec*, volume 4784 of *Lecture Notes in Computer Science*, pages 122–137. Springer, 2007.

[CDM00] Ronald Cramer, Ivan Damgard, and Philip D. MacKenzie. Efficient zero-knowledge proofs of knowledge without intractability assumptions. In *Public Key Cryptography*, pages 354–372, 2000.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, pages 41–50, 1995.

[CH02] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 21–30. ACM, 2002.

[Cha82] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.

[Cha86] David Chaum. Demonstrating that a public predicate can be satisfied without revealing any information about how. In Odlyzko [Odl87], pages 195–199.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 255–271. Springer, 2003.

[CL] Jung Hee Cheon and Dong Hoon Lee. Diffie-hellman problems and bilinear maps.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Pfitzmann [Pfi01], pages 93–118.

[CL02] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer, 2002.

[CM99] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *EUROCRYPT*, pages 107–122, 1999.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT*, pages 402–414, 1999.

[CNS07] Jan Camenisch, Gregory Neven, and Abhi Shelat. Simulatable adaptive oblivious transfer. In Naor [Nao07], pages 573–590.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Jr. [Jr.97], pages 410–424.

[CS00] Ronald Cramer and Victor Shoup. Signature schemes based on the strong rsa assumption. *ACM Trans. Inf. Syst. Secur.*, 3(3):161–185, 2000.

[CT05] Cheng-Kang Chu and Wen-Guey Tzeng. Efficient -out-of- oblivious transfer schemes with adaptive and non-adaptive queries. In Serge Vaudenay, editor, *Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 2005.

[DBL85] *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, 6-8 May 1985, Providence, Rhode Island, USA*. ACM, 1985.

[DBL86] *27th Annual Symposium on Foundations of Computer Science, 27-29 October 1986, Toronto, Ontario, Canada*. IEEE, 1986.

[DF02] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Yuliang Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142. Springer, 2002.

[DLP93] Ivan Damgård, Peter Landrock, and Carl Pomerance. Average case error estimates for the strong probable prime test. 61(203):177–194, July 1993.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *J. Cryptology*, 1(2):77–94, 1988.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Jr. [Jr.97], pages 16–30.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.

[Gen06] Craig Gentry. Practical identity-based encryption without random oracles. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 445–464. Springer, 2006.

[GH07] Matthew Green and Susan Hohenberger. Blind identity-based encryption and simulatable oblivious transfer. In *ASIACRYPT*, pages 265–282, 2007.

[GIKM98] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *STOC*, pages 151–160, 1998.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC* [DBL85], pages 291–304.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

[GO92] Shafi Goldwasser and Rafail Ostrovsky. Invariant signatures and non-interactive zero-knowledge proofs are equivalent (extended abstract). In Brickell [Bri93], pages 228–245.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptology*, 7(1):1–32, 1994.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[Gol01] Oded Goldreich. *The foundations of cryptography.* Cambridge University Press, 2001.

[Goy07] Vipul Goyal. Reducing trust in the pkg in identity based cryptosystems. In Alfred Menezes, editor, *CRYPTO*, volume 4622

of *Lecture Notes in Computer Science*, pages 430–447. Springer, 2007.

[Jou00] Antoine Joux. A one round protocol for tripartite diffie-hellman. In Wieb Bosma, editor, *ANTS*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer, 2000.

[Jr.97] Burton S. Kaliski Jr., editor. *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*. Springer, 1997.

[JS07] Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In Naor [Nao07], pages 97–114.

[KM05] Neal Koblitz and Alfred Menezes. Pairing-based cryptography at high security levels. In Nigel P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 13–36. Springer, 2005.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *FOCS*, pages 364–373, 1997.

[Koh04] Markulf Kohlweiss. *Towards Anonymous Digital Credentials. Integrating Idemix with Access Control Products.* Master Thesis, 2004.

[KSW07] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. Cryptology ePrint Archive, Report 2007/404, 2007.

[Lip03] Helger Lipmaa. On diophantine complexity and statistical zero-knowledge arguments. In Chi-Sung Laih, editor, *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 398–415. Springer, 2003.

[Lyn07] Ben Lynn. *On the implementation of pairing-based cryptosystems.* 2007.

[Maa04] Martijn Maas. *Pairing-Based Cryptography.* 2004.

[MvOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[MW98] Ueli M. Maurer and Stefan Wolf. Lower bounds on generic algorithms in groups. *Lecture Notes in Computer Science*, 1403:72–??, 1998.

[MW99] Ueli M. Maurer and Stefan Wolf. The relationship between breaking the Diffie-Hellman protocol and computing discrete logarithms. *SIAM Journal on Computing*, 28(5):1689–1721, 1999.

[MY96] Ueli M. Maurer and Yacov Yacobi. A non-interactive public-key distribution system. *Des. Codes Cryptography*, 9(3):305–316, 1996.

[Nac05] David Naccache. Secure and practical identity-based encryption. *CoRR*, abs/cs/0510042, 2005.

[Nao07] Moni Naor, editor. *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*. Springer, 2007.

[NP99a] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *STOC*, pages 245–254, 1999.

[NP99b] Moni Naor and Benny Pinkas. Oblivious transfer with adaptive queries. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 573–590. Springer, 1999.

[NP05] Moni Naor and Benny Pinkas. Computationally secure oblivious transfer. *J. Cryptology*, 18(1):1–35, 2005.

[Odl87] Andrew M. Odlyzko, editor. *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*. Springer, 1987.

[OK04] Wakaha Ogata and Kaoru Kurosawa. Oblivious keyword search. *J. Complexity*, 20(2-3):356–371, 2004.

[OP01] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In *PKC '01: Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, pages 104–118, London, UK, 2001. Springer-Verlag.

[Ost] Rafail Ostrovsky. Software protection and simulation on oblivious rams.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[Pfi01] Birgit Pfitzmann, editor. *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*. Springer, 2001.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *EUROCRYPT*, pages 387–398, 1996.

[Rab81] Michael O. Rabin. How to exchange secrets by oblivious transfer. 1981.

[RS85] J. O. Rabin and Jeffrey Shallit. Randomized algorithms in number theory. Technical report, Chicago, IL, USA, 1985.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.

[Sha92] Adi Shamir. Ip = pspace. *J. ACM*, 39(4):869–877, 1992.

[SK98] Bruce Schneier and John Kelsey. Cryptographic support for secure logs on untrusted machines. In *SSYM'98: Proceedings of the 7th conference on USENIX Security Symposium, 1998*, pages 4–4, Berkeley, CA, USA, 1998. USENIX Association.

[SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, 2005.

[WBDS04] Brent R. Waters, Dirk Balfanz, Glenn Durfee, and Diana K. Smetters. Building an encrypted and searchable audit log. In *NDSS*. The Internet Society, 2004.

# Index