KUNGL
TEKNISKA
HÖGSKOLAN

On Equality Testing Protocols and their Security

Anna Redz

TRITA-NA-03-17

Licentiate Thesis

Royal Institute of Technology

Department of Numerical Analysis and Computer Science

Akademisk avhandling som med tillstånd av Kungl Tekniska Högskolan framlägges till offentlig granskning för avläggande av teknologie licentiatexamen 26 september 2003 i sal D3, Lindstedtsv 5, KTH.

ISBN 91-7283-569-9
TRITA-NA-03-17
ISSN 0348-2952
ISRN KTH/NA/R--03/17--SE

Universitetsservice, US AB, Stockholm, 2003

Abstract

This thesis is written for the Swedish degree Licentiate of Science, *Teknisk Licentiat.* It is a university degree, between that of master and that of doctor.

The main focus of the thesis is on the construction of secure protocols for comparing the underlying plain-texts in ElGamal encryptions. The protocols make use of the malleability of the ElGamal encryption scheme. More specifically they use the multiplicative homomorphic property of ElGamal.

We present fully verifiable protocols for both the two-party setting and the multi-party setting. These protocols are built on sub-protocols, which are specially constructed to fit the present setting. We also present full proofs for completeness, soundness, and zero-knowledge for all the given protocols, in the random oracle model.

ISBN 91-7283-569-9 • TRITA-NA-03-17 • ISSN 0348-2952 • ISRN KTH/NA/R--03/17--SE

Acknowledgments

Orakel spektakel, brahmin vitamin
Var kan jag köpa billig bensin?!
Fysikalen, 1996

Are you surprised you're holding a thesis written by me? So am I.

First of all, I want to express my gratitude to my brilliant advisor Johan Håstad. You have always been helpful and most generous with your time and comments on my work.

To all members of the theoretical computer science group, and specially all of my current and previous room mates (Mats Näslund, Staffan Ulfberg, Jonas Holmerin, Gustav Hast, Mårten Trolin, Rafael Pass, and Magnus Rosell) as well as Mikael Goldmann and Lars Engebretsen, a special thanks. You have all been very tolerant, listened to me patiently, answered my questions patiently and discussed a lot of various topics with me, both relevant and irrelevant.

I would like to acknowledge the research group at Microsoft Research in Mountain View for housing me for three months, and specially Cynthia Dwork, Lidong Zhou and Mike Burrows for their support, never ending flow of ideas and enthusiasm.

What would life be without friends? Thanks Hedvig Sidenbladh, Olof Kjellström, Olle & Katarina Bälter, Maria Ögren, Fredrik Zetterling, Dennis Tell, Olov Engwall, Mats Blomqvist for being great friends and for still being a bit childish and mischievous. Lill-Gustaf, the brain behind *Sub Rosa*, has added a touch of well needed humanistic recreational discussion to my technical environment. I also want to thank Malin Andersson and Louise Banning for distracting me and making me climb around on walls, and all the members of Fysikalen for keeping me busy with student farce theater for two entire weeks every second year.

To my parents, sister with family and to Staffan I owe my profoundest thanks, deeper than the deepest crack in the ocean.

All of you who I have forgotten to mention, thank you for helping and encouraging me.

Contents

List of Figures

Chapter 1

Introduction

Imagine two people who want to communicate secrets with each other. They have many options of how to proceed. They may decide to meet and talk to each other or maybe buy a box with a lock, take a key each and pass messages to each other in the box. Another option is for the two people to use a *cryptosystem* or *encryption scheme*, which can be seen as a kind of box. The key is used to encrypt the message, i.e., to put the message in the box, as well as to decrypt the message, to open up the box. The encrypted message is a string of characters which should not make any sense to anybody who does not have the key.

Cryptosystems can roughly be divided into two categories, namely *symmetric* cryptosystems and asymmetric or *public-key* cryptosystems. In a symmetric cryptosystem the same key is used for both encryption and decryption, as in the example above. This means that anybody with access to the key can encrypt a message, and also decrypt any message encrypted with this key. A consequence of this is that each pair of people or players who want to communicate in secret must agree on a key. Not only must each player keep track of a lot of keys, but also the process of exchanging keys becomes non-trivial. A public-key cryptosystem takes care of this drawback. Here each participant has two keys, a public one and a private one. The public key is used for encryption and is thus distributed freely, whereas the private key is used for decryption and kept secret as its name indicates. Anybody can send a message to the player without first going through the procedure of agreeing on a key.

Public-key encryption schemes can subsequently be divided into two categories, deterministic and random. In a deterministic public-key cryptosystem, encrypting a plain-text with a specific key always gives the same encryption, whereas in a random public-key cryptosystem the encryption depends on the key and a random number chosen by the encrypting party. Examples of such random encryption schemes are RSA [RSA78], Paillier [Pai99], ElGamal [ElG85] and the McEliece cryptosystem [McE78]. In this work we focus on the ElGamal cryptosystem.

There are several security objectives for an encryption scheme apart from the obvious one: an encryption should not leak any information about the underlying plain-text. Different cryptosystems are resilient against different kinds of attacks. One interesting notion is that of malleability, which has been discussed in [DDN91]. This is both a desirable and undesirable property. With a malleable encryption scheme a player can change the underlying plain-text of an encryption in a deterministic way, without having any knowledge of the plain-text. At the same time, a malleable encryption scheme gives the possibility to build interesting protocols which are based on manipulating encryptions. In this work we exploit the malleability property of the ElGamal encryption scheme.

1.1 Threshold Systems and Multi-Party Computation

Cryptography is not only about sending arbitrary messages back and forth between two distant players. Instead it may involve many players simultaneously and also involve all aspects of secure or secretive interaction.

Consider a company in place of a player, and imagine that all e-mail to its employees is encrypted using the company's public key. The storage of the company's private key is then a single point of attack, since we put all power and all trust in one place (this is a short version of an example from [Cac01]).

To overcome this predicament, the trust, i.e., the key, is distributed. Distributed trust [Cac01] is a powerful paradigm that creates a trusted unit from untrusted parts. In *threshold cryptography* this trust is spread among a large set of players, each holding one piece of the key to the box, i.e., to the cryptosystem. To reconstruct the key, it is enough to hold over some threshold of the key shares. Another benefit with a threshold scheme is that it is fault tolerant. The idea of threshold cryptography was introduced in the papers [Des88, Des89].

The existence of threshold schemes is implied by *secure multi-party computation.* To explain multi-party computation we give the following example. Imagine a group of people who each have some information which they unwillingly want to disclose, but still, as a group, they want to jointly compute some function based on all of their secret inputs. This group could consist of unreliable, but technically advanced, art collectors who participate in a sealed bid electronic auction for a Picasso painting. The secret information each collector has is the price at which they value the paining and the function they want to compute is the electronic auction. Jointly computing a function on the players' secret inputs is usually referred to as *multi-party computation.*

One obvious solution is to involve an independent trusted party who collects all the necessary information and computes the function. But in many cases there is no trusted party, or no guarantee that a so called trusted party really is honest and is not collaborating with any of the players. To prevent this, the players can encrypt the bids before sending them to the trusted party. This means that the trusted party does not get information in a form from which it can deduce anything useful. In order to be able to actually compute the function, the players may need to submit some additional information to the trusted party, possibly giving the trusted party more information than one would really want.

A better solution is for the players to jointly compute the function without involving other parties. The players follow a *protocol* which specifies their actions in every step.

Multi-party computation was introduced in [Yao82]. Goldreich, Micali and Wigderson [GMW87] gave a general solution to the multi-party computation problem, presenting a protocol for computing any imaginable function (that is possible to compute in reasonable time). The problem was subsequently studied in a number of works, [BOGW88, CCD88, CFGN96, BOCG93, CDD$^+$99, Can00] among others. They discuss different kinds of adversarial behaviors and increase efficiency for different models. Due to the generality of the problem, many of the given solutions are not very efficient. Therefore, given specific functions, there may be more efficient solutions to the problem.

Whatever method we choose to compute the function, it is imperative that the solution can be trusted in the way that is claimed. The only way to guarantee this is to design the protocol in such a way that we can write down proofs ensuring some kind of security. Today many protocols are published without complete proofs of security, or even without proofs at all. We are left to trust our intuition or to construct the proofs on our own. History has proven that trusting your intuition can be dangerous. There are many examples of protocols in cryptographic literature that at some point were believed secure, that later have turned out to be the opposite.

We want the result of the protocol to really be an evaluation of the specified function on the given data. This is usually referred to as *completeness*. Also, the protocol should be *sound*:, the outcome should be a correct evaluation of the given function even if some of the players in the group try to cheat. We want to guarantee that the protocol does not disclose any information other than the apparent, i.e., the result of the protocol. This is usually referred to as *zero-knowledge*.

The proofs for the protocols are not necessarily absolute, that is, we can not always guarantee that some corrupt players never succeed in cheating. Instead we are satisfied with showing that it is extremely unlikely that players can cheat without detection. We quantify this unlikeliness in terms of a security parameter which we call k. The cryptographic community accepts a proof in which the probability of a failure is negligible, meaning that it approaches 0 faster than the inverse of any polynomial $poly(k)$ in the security parameter. Also, we cannot always guarantee that a protocol does not leak any information at all, but we can quantify that it leaks so little information that an adversary can not detect it. Moreover, the proofs have to be based on an assumption that certain problems are difficult to solve; these assumptions have not been proved, but are believed true.

When composing a protocol one must have these notions in mind to ensure that it is possible to prove correctness of some sort for the protocol. All details must be filled in and specified. All this has been done for the general solution for multi-party computation mentioned above.

Even though the general ideas and techniques for producing rigorous proofs are well established, each protocol must be treated and considered individually and the proofs designed for it specifically. This is at times tedious work, including numerous changes to the protocol in order to be able to prove the desired traits of the protocol. During protocol design, when the focus is on security and provability, the efficiency suffers, but not to such an extent that it matches the inefficiency of the general solution for multi-party computation.

1.2 Thesis Topics

The main focus of this thesis is to construct provably secure protocols and to write down the proofs. We give proofs of completeness, soundness and zero-knowledge for all protocols.

This thesis does not introduce any new techniques in protocol design, but combines known methods to build new protocols. All of the protocols we present here build on manipulating cipher-texts. To this end, we use the malleability property of the ElGamal encryption scheme. More specifically, we use the fact that it is homomorphic. In 1993 Franklin and Haber [FH94] used the technique of manipulating cipher-texts to construct a secure multi-party computation method. Since then several works have been based on this technique, e.g. [JJ00].

Most of the protocols presented in this thesis make use of the same trick to avoid disclosing information when decrypting messages. This trick is to blind the value which is going to be decrypted, and to blind it in such a way that the piece of information relevant to the current problem is not destroyed. Blinding was first proposed in [Cha83, Cha85, Cha88] in combination with digital signatures in order to hide what is being signed from the signer. Blinding has also been used to hide what is being decrypted from the player or players who are decrypting [GGJR00].

The first protocol we present, called EGO, is a fully verifiable two-party plain-text equality test, i.e., a comparison of the underlying plain-texts of two encryptions, encrypted using different keys. This problem is described in [Sal90] and [BST01]. We assume that the encryptions are computed by a third party, and given to players together with the claim that they are equal. Each player has one of the private keys. In [BST01] a fully secure version of the two-party cipher-text comparison problem is given. However, they assume that the players themselves compute the cipher-texts. Also, the solution they present is not verifiable to observers. The computational complexities of the two solutions are equivalent.

In a multi-party setting, both messages are encrypted with the same public key and the involved players share the private key in a threshold fashion. We call this the Multi-party Plain-text Equality Test, or the PET protocol for short. In [JJ00] Jakobsson and Juels present a solution for general multi-party computation using multi-party plain-text equality testing as a building block. They give an outline of the protocol in the paper. But, that outline lacks formal proofs. In order to be able to construct satisfactory proofs, we needed to fill in all the details of the protocol.

Using the PET protocol sequentially makes it possible to construct an efficient electronic auction which does not disclose the bids of the losing players in a highest bid auction. We call this sequential use the SEQ protocol. It is also possible to implement a second price auction using the SEQ protocol. Electronic auctions are a case of multi-party computation, computing a very well defined function. In Chapter 7.1 we give a short overview of the work done on electronic auctions.

All protocols in this thesis make use of one or two sub-protocols to prove correctness of published data. These sub-protocols are constructed to fit the setting of the proofs of the protocols in which they are used. The basic ideas for the construction of the sub-protocols were presented in [CP92, Oka92, Sch91].

It turns out that the difficult part of protocol construction lies in the proofs of soundness and zero-knowledge.

In theory the notion of zero-knowledge is simple. All zero-knowledge proofs are based on constructing a simulator which on public and adversarial input produces output with probability distribution indistinguishable from the probability distribution of the output of a real run of the protocol. But, actually constructing the protocol and the simulators giving all necessary probability distributions is a non-trivial and tiresome task. The main obstacle is showing the indistinguishability of the probability distributions. This may be done by constructing additional simulators whose outputs define probability distributions lying between the two mentioned above. In this manner we construct a chain of constant length of indistinguishable probability distributions.

All proofs given in this thesis are in the random oracle model [BR93], where players have oracle access to truly random function. In the corresponding real setting the truly random function is replaced with hash functions (such as MD5). The main benefit of the random oracle model is the possibility of proving the security of efficient cryptographic protocols in an ideal setting. It has been believed that a proof of security in the random oracle model means that a protocol does not contain any structural flaws. But [CGH98] show that there exist protocols that are secure in the random oracle model, and for which any implementation of the random oracle leads to an insecure protocol, which is something we should have in mind. Lacking better tools, we nevertheless use the random oracle model in this thesis, since it asserts some kind of security.

As mentioned above, the PET protocol was outlined in [JJ00], though without any formal proofs. A similar, but more complicated, formulation of the PET protocol is presented in [MSJ02]. Our work was done independently of and prior to the work of [MSJ02]. Their paper contains proof sketches which, due to their more complicated protocol, are more laborious than the ones we present.

1.3 Organization of this Thesis

Chapters 2 and 3 serve as background to the thesis and provide definitions and notations which are necessary for the problems addressed, together with a brief review of the ElGamal cryptosystem and some of its properties. The notions are standard, but formulated for our purposes. The chapters also present the setting in which the protocols are intended to be used, and how we model players and adversaries.

Chapter 4 provides protocols for two generic zero-knowledge proofs used for proving correctness of data. They are used in the following chapters.

The following chapters present the EGO protocol (Chapter 5) for plain-text comparison in a two-player setting, the PET protocol (Chapter 6) for plain-text comparison in a multi-party setting, and the SEQ protocol (Chapter 6.5) which is the sequential composition of the PET protocol. The auction protocol is presented in Chapter 7.

Apart from the PET protocol, all protocols are entirely new constructions. The protocols are given in detail and are presented with full proofs for completeness, soundness and zero-knowledge.

Chapter 2

Preliminaries

Here we present some notations and some formal definitions used throughout the thesis.

2.1 Notation

- For a set A let $|A|$ denote its size.
- For integer n, let $|n|_2$ be the length of the binary representation of n.
- Let $\log_g(h)$ be the discrete logarithm of h in the base g, i.e., the number x such that $g^x = h \mod p$, where p is defined by the group we are working in.
- Let $d \in_R \mathcal{G}$ be a variable d that is randomly chosen and uniformly chosen from a group $\mathcal{G}$.
- Let $\mathcal{G}_q$ denote a group of size q.
- For values $d, g \in \mathcal{G}_q$, d/g is equivalent to writing dg^{-1}, where g^{-1} is the multiplicative inverse of g.
- Let $V_1||V_2$ denote the concatenation of strings V_1 and V_2.

2.2 Basic Definitions

First we define the notion of a *negligible* function, which is a function approaching 0 faster than the inverse of any polynomial $poly(k)$ in the security parameter.

Definition 2.1. A function $f(k) > 0$ is negligible if, for every polynomial $poly(k) > 0$, $\exists k_0 > 0$ such that $\forall k > k_0$, $f(k) < 1/poly(k)$.

A problem that is considered easy to solve is such that the amount of computational power needed to solve the problem grows polynomially in some security parameter k. A hard, difficult or intractable problem, can not be computed by any circuit of polynomial size in the same security parameter.

The focus of this work is showing security for protocols based on the discrete-logarithm problem, i.e., given a value y find the value x such that $g^x = y$. For this purpose we define a *discrete-log instance*. This definition is an altered version of the definition found in [Jar01].

Definition 2.2. A discrete-log instance with security parameter k is a triple of the form $(p, q, g)_k$, where p and q are primes such that q divides $p - 1$, $|q|_2 = k$, and $|p|_2 \leq \mathit{poly}(k)$. The value g is a generator of the multiplicative subgroup $\mathbb{Z}_p^*$ of order q.

For all discrete-logarithm instances we use from now on, we assume that $p = 2q + 1$ and that $\mathcal{G}_q$ is the subgroup of quadratic residues of $\mathbb{Z}_p^*$. The discrete-logarithm problem is believed to be intractable in such $\mathcal{G}_q$. [Jar01] presents a multi-party threshold protocol for jointly computing this kind of discrete-log instances. (A prime p such that $p = 2q + 1$, for a prime q, is called a Sophie Germain prime. It is not known if there are infinitely many such primes, but it is strongly believed to be the case. This does not affect our results.)

All modular calculations are implicit from now on, i.e., ′ mod p' is omitted when computing exponentiations and discrete logarithms and ' mod q' is omitted when performing computation on exponents.

2.3 The Decisional Diffie-Hellman Assumption

The decisional Diffie-Hellman (DDH) assumption is frequently used in literature on cryptography, here likewise. We use the definition suggested in the survey [Bon98] and define the non-uniform DDH assumption.

Definition 2.3. A DDH algorithm B is an algorithm running in probabilistic polynomial time satisfying, for some polynomial $\mathit{poly}(k)$ and discrete-log instance $(p, q, g)_k$

$$\begin{aligned}&\big|\mathsf{Pr}\left[B(g, g^a, g^b, g^{ab}, p, q, aux) = \text{“true”}\right] - \\ &\mathsf{Pr}\left[B(g, g^a, g^b, g^c, p, q, aux) = \text{“true”}\right]\big| > \frac{1}{\mathit{poly}(k)},\end{aligned} \tag{2.1}$$

where aux is non-uniform auxiliary input. The probability is over the random choice of the discrete-log instance, the values a, b, c, and over the random bits of B.

Note that standard non-uniformity is in terms of polynomial circuits. This is replaced by aux, that might encode any circuit.

The *decisional Diffie-Hellman assumption* is that B does not exist for a discrete-log instance $(p, q, g)_k$ with $p = 2q + 1$ [Bon98].

A triple (g^a, g^b, g^c) is in the set Δ_{DDH} if $a, b \in_R \mathbb{Z}_q$ and $c = ab$. If $a, b, c \in_R \mathbb{Z}_q$ then we write that $(g^a, g^b, g^c) \in \Delta_{rand}$.

2.4 The Computational Diffie-Hellman Assumption

The computational Diffie-Hellman problem for a discrete-log instance $(p, q, g)_k$ is the following: given the generator g and the group elements g^a and g^b find g^{ab}. The *computational Diffie-Hellman assumption* is that solving this problem is hard in the group $\mathcal{G}_q$ as defined by the discrete-log instance.

The DDH assumption implies the computational Diffie-Hellman assumption.

2.5 Generating New Independent Triples

We have a triple (g^a, g^b, g^c) where either $c = ab$ or $c \neq ab$ and we intend to generate a set of new triples. We require that these new triples are all independent and random, and that they are of the same sort as (g^a, g^b, g^c), i.e., $(g^a, g^b, g^c) \in \Delta_{DDH}$ then each new triple is in Δ_{DDH} and if $(g^a, g^b, g^c) \notin \Delta_{DDH}$ then all new triples are not in Δ_{DDH}.

To compute a new triple (g^u, g^v, g^w), select three values r_1, r_2 and r_3 randomly and independently from $\mathbb{Z}_q$. The new triple is then

$$\begin{aligned}
&(g^u, g^v, g^w) = \\
&(g^a g^{r_1}, (g^b)^{r_2} g^{r_3}, (g^c)^{r_2} (g^a)^{r_2} (g^b)^{r_1 r_2} g^{r_1 r_3}) = \\
&(g^{a+r_1}, g^{br_2+r_3}, g^{cr_2+br_1r_2+r_1r_3}).
\end{aligned} \tag{2.2}$$

We note that $w = uv + (c - ab)r_2$. If the original tuple is in Δ_{DDH}, then $c - ab = 0$. This gives $w = uv$, where u, v are random and independent of a and b. This is clearly seen by fixing r_2. Random choices of r_1 and r_3 give all possible values for u and v.

If instead the original triple is not in Δ_{DDH}, then we can compute each possible triple (g^u, g^v, g^w) in exactly one way; $r_1 = u - a$, $r_2 = (w - uv)/(c - ab)$, and $r_3 = v - br_2$. This implies that (g^u, g^v, g^w) is random.

Computing several new random and independent triples means selecting new random and independent values $r_1, r_2, r_3 \in_R \mathbb{Z}_q$ for each additional triple.

2.6 Indistinguishability

Intuitively, two objects are considered indistinguishable when it is infeasible to distinguish between them in any efficient way. This is a central notion in cryptography, first introduced in [GM84]. There are several different flavors of indistinguishability depending on the limitations set upon the distinguishing algorithm. They are nicely formulated in [GMR89], but here we state them reformulated for our purposes.

Let a family of probability distributions be defined as $U = \{U_k\}_{k \in \{0,1\}^*}$, where U_k is a probability distribution whose random variable ranges over strings of length k. Two families of probability distributions are obviously indistinguishable if the families are equal. If instead their statistical difference is negligible for all sufficiently large indices, then we say that they are *statistically indistinguishable*.

Definition 2.4. Two families of probability distributions U and V are *statistically indistinguishable* if

$$\sum_{\alpha} |\mathsf{Pr}[U_k = \alpha] - \mathsf{Pr}[V_k = \alpha]| \text{ is negligible as a function of } k, \tag{2.3}$$

the sum taken over all possible outputs α.

Two probability distributions are considered computationally indistinguishable if no efficient algorithm can tell them apart.

Definition 2.5. Let U and V be two families of probability distributions. We say that U_k and V_k are *computationally indistinguishable* if, for all polynomial algorithms B

$$|\mathsf{Pr}[B(u_k) = 1] - \mathsf{Pr}[B(v_k) = 1]| \text{ is negligible as a function of } k, \tag{2.4}$$

the probability taken over random samples u_k and v_k of U_k and V_k.

When U and V are statistically indistinguishable we write $U \overset{s}{\approx} V$, and when they are computationally indistinguishable we write $U \overset{c}{\approx} V$.

2.7 Commitments

A commitment is a way of making a player bind himself to a value, while keeping the value secret. This commitment can later be opened, and the value revealed, or the commitment can be used in itself to prove that the value to which our player committed was actually used. We make use of the latter construction.

More specifically, we use Pedersen commitments, which is a commitment scheme described in [Ped91]. The scheme works in the group $\mathcal{G}_q$ defined by a discrete-log instance (p, q, g).

Committing to a value $z \in \mathbb{Z}_q$ is done by computing $C = g^z h^r$, where h is a random generator in $\mathcal{G}_q$ and $r \in_R \mathbb{Z}_q$. This commitment scheme protects z in an information theoretic sense. For any z and random r, C is uniformly distributed in $\mathcal{G}_q$. We say that a pair (z, r) gives C if $g^z h^r = C$.

Assuming that the discrete logarithm problem is difficult in $\mathcal{G}_q$, a polynomially limited player can not compute $\log_g(h)$ efficiently. This implies that the players can not efficiently compute several different pairs (z, r) giving C, which in turn implies that the player can not open a published commitment in two different ways.

There exist multi-party threshold protocols for computing a random generator h in a multi-party setting, on input k and a discrete-log instance. Jarecki [Jar01] presents one such protocol.

2.8 The ElGamal Encryption Scheme

ElGamal is a public key encryption scheme [ElG85] in the subgroup $\mathcal{G}_q$ defined by a discrete log instance (p, q, g). We recall that the private key x, used for decryption, is a randomly chosen element from $\mathbb{Z}_q$. The public key, $y = g^x$, is used for encryption. Plain-texts can be chosen from $\mathbb{Z}_p$. Plain-texts not in $\mathcal{G}_q$ can be mapped onto $\mathcal{G}_q$ through multiplication by a predetermined non-residue. Encrypting a plain-text message m means choosing a blinding value $R \in_R \mathbb{Z}_q$ and computing the pair $(\alpha, \beta) = (my^R, g^R)$. Decrypting is done by computing α/β^x. The encryption scheme is malleable, i.e., given an encryption $E(m)$ of a message m, it is possible to compute a random encryption of a function of m without knowing m.

The malleability property in an encryption scheme is generally an area of concern in protocol design. Unless the protocol is carefully designed, malleability gives an adversary the possibility of producing correct encryptions that are functions of other encrypted values. This implies that the adversary may be able to bias the output.

On the other hand, malleability can be considered a good feature. It can be used in protocols to compute functions of the encrypted plain-texts without the need to decrypt. The protocols we present exploit the malleability property of ElGamal in order to compute random exponentiation of encrypted plain-texts.

In order to compute a random re-encryption of an arbitrary encryption $E(m) = (my^R, g^R)$, select $R' \in_R \mathbb{Z}_q$ and compute a new encryption of m as $E'(m) = (my^R y^{R'}, g^R g^{R'}) = (my^{R+R'}, g^{R+R'})$. This is still an encryption of m.

Also note that ElGamal encryption is homomorphic, that is,

$$\frac{E(m_1)}{E(m_2)} = E(\frac{m_1}{m_2}). \tag{2.5}$$

Breaking the security of ElGamal, that is recovering a message m given an encryption, is equivalent to solving the computational Diffie-Hellman problem. Given y, g, α, β, p, extracting m, is equivalent to computing the value g^{xR} from g^x and g^R.

If only one player is involved then it is apparent that this lone player should have access to the private key. When $n > 1$ players are involved then the way of distributing the private key among the players decides the level of security. One may want to protect the key and the availability of the key. Giving a copy of x to each player implies that each player is trusted and can decrypt messages on his own. This also means that it is enough for an adversary to corrupt a single player to get hold of the private key.

To spread the trust and risk, the private key is “divided” into shares (one per player), such that a group of t or fewer players has no information about the private key and can not decrypt a cipher-text, whereas a group of size $> t$ can extract the private key. This implies that an adversary must corrupt $> t$ players to get the key. This is what we call a (t, n)-threshold system.

Distributing x with $t = n - 1$, meaning that all players' shares are needed, is not robust since a single player can disrupt the decryption process. We also want to prevent an adversary from disrupting decryption, which means that at least $t+1$ players must remain honest. Thus letting an adversary corrupt up to a minority, $t < \lfloor (n-1)/2 \rfloor$, the threshold system withstands attacks from groups.

We use Shamir's idea [Sha79] to distribute the private key. A polynomial f unknown to the players and of degree t hides the private key as $f(0) = x$. The private share of player P_i is $x_i = f(i)$. With $t+1$ private shares it is possible to reconstruct the polynomial and the private key. Jarecki [Jar01] suggests a protocol for jointly constructing and sharing ElGamal keys. The protocol reveals y, the public key and the public shares $\{y_i = g^{x_i} : i = 1, \ldots, n\}$ in the process.

Using distributed keys affects only the decryption. The Lagrange interpolation formula can be used to reconstruct the private key:

$$x = f(0) = \sum_{P_i \in T} \lambda_i x_i, \tag{2.6}$$

where T is a set of $t+1$ players whose private shares we know, and λ_i are the Lagrange coefficients computed as

$$\lambda_i = \frac{\prod_{P_j \in T_i} j}{\prod_{P_j \in T_i} (j - i)}, \tag{2.7}$$

where $T_i = T - \{i\}$.

A set T of $t+1$ players can decrypt (α, β) without revealing their shares. If each player $P_i \in T$ publishes β^{x_i} (and proves its correctness) then each player can compute

$$\beta^x = \beta^{\sum_{P_i \in T} \lambda_i x_i} = \prod_{P_i \in T} (\beta^{x_i})^{\lambda_i}, \tag{2.8}$$

and the decryption α/β^x on his own without learning the private shares.

Also, given a set T of $t+1$ players P_i and their β^{x_i} values, we can compute β^{x_j} for each $P_j \notin T$ as follows.

Let T_1 be the set $t+1$ players for which we know β^{x_i}. Define the set T_2 by taking T_1 and replacing player $P_k \in T_1$ by player $P_l \notin T_1$. Let $\lambda_i^{(1)}$ be the Lagrange coefficient for player P_i when in set T_1. Let $\lambda_i^{(2)}$ be the Lagrange coefficient for player P_i when in set T_2. We can compute β^x in two different ways using the two sets T_1 and T_2 as

$$\beta^x = \prod_{P_i \in T_1} \beta^{x_i \lambda_i^{(1)}}$$

and

$$\beta^x = \prod_{P_j \in T_2} \beta^{x_j \lambda_j^{(2)}}.$$

Using two above equations we can compute $\beta^{x_l \lambda_l^{(2)}}$ as

$$\beta^{x_l \lambda_l^{(2)}} = \frac{\prod_{P_i \in T_1} \beta^{x_i \lambda_i^{(1)}}}{\prod_{P_j \in T_2 \setminus P_l} \beta^{x_j \lambda_j^{(2)}}} = \beta^{x_k \lambda_k^{(1)}} \prod_{P_i \in T_1 \cap T_2} (\frac{\beta^{\lambda_i^{(1)}}}{\beta^{\lambda_i^{(2)}}})^{x_i}.$$

This gives

$$\beta^{x_l} = (\beta^{x_k \lambda_k^{(1)}} \prod_{P_i \in T_1 \cap T_2} (\frac{\beta^{\lambda_i^{(1)}}}{\beta^{\lambda_i^{(2)}}})^{x_i})^{-\lambda_l^{(2)}}, \tag{2.9}$$

which can be done for all $P_l \notin T_1$.

We can also do Lagrange interpolation to compute $g^{f(i)}$, player P_i's public key share.

Chapter 3

The General Setting

An interactive protocol is a set of communication rounds between a number of participants, which we call *players*, that are supposed to follow some well-defined rules. In literature two-party protocols are usually presented with the two following participants: the *prover* and the *verifier*. The purpose is for the prover to convince the verifier of the validity of some statement. The protocol generally consists of letting the prover answer some "hard questions", something he must do convincingly in order for the verifier to be convinced. The protocol should allow the verifier to always accept a correct statement when interacting with a honest prover (completeness) and to reject an incorrect statement with at least non-negligible probability when interacting with an arbitrary prover (soundness).

The protocols we present in the following chapters are run by two or more players. We model the players by probabilistic Turing Machines with an internal random string, an auxiliary input, and running in time polynomial in the security parameter k and input length.

Generally in the definitions of interactive protocols such as that presented in [Gol02], the verifier is polynomially bounded whereas the prover is unbounded. In our protocols though, regardless of the number of participants, each verifier doubles as a prover and vice versa, which forces us to put a polynomial computational restriction on the prover.

A proof where a prover asserts knowledge of some fact is usually referred to as a *proof of knowledge*. This inevitably leads to the question of what knowledge is for a Turing Machine. We choose not to discuss this here and instead refer to [Gol01] for a thorough discussion on this topic. The intuition of a zero-knowledge proof of knowledge is that it is a protocol in which the probability that the verifier outputs 1 (is convinced) is inversely proportional to the difficulty of extracting the secret of the prover, when using the prover as a sub-routine.

3.1 Communication

All players are connected to a bulletin board on which they can publish messages and read what others have published. A bulletin board can be seen as a public broadcast channel with a memory. All broadcast information is public and can be seen even by observers. Players can only append information, not erase from the bulletin board. Also, the bulletin board is authenticated, i.e., a player can not publish messages in any other player's name. Eavesdroppers can read but not write on the bulletin board. We refer to eavesdroppers as observers, since they eavesdrop legitimately. Apart from the bulletin board, the players have no means of communication with each other.

The communication in all of the following protocols proceeds in synchronized rounds. There is no synchronization within a round and we can assume that corrupt players publish last. The current round ends after a fixed time bound or when all players have published an answer. If a player fails to publish during a round, then it is not possible for him to publish later.

3.2 Modeling Faulty Players

Players may be faulty, that is they either willingly or by mistake publish erroneous messages. We assume the worst case, i.e., that we have a set collaborating players trying to cheat. We model this scenario by an additional Turing Machine $\mathcal{A}$ with a uniformly distributed random string $ran_{\mathcal{A}}$ and with access to some auxiliary input $aux_{\mathcal{A}}$. We refer to $\mathcal{A}$ as the adversary. The adversary corrupts players and controls their actions. When corrupted by $\mathcal{A}$, a player reveals all its internal data to $\mathcal{A}$: its random string, auxiliary input, and possible secret inputs. We assume that the adversary is static, i.e, it corrupts players before the start of the protocol. $\mathcal{A}$'s decision whether to corrupt a player or not does not depend on information published by the players. A corrupted player stays corrupted.

When we have a group of n players sharing ElGamal keys in a threshold manner, we want to be able to tolerate up to any minority of faulty players. Therefore we allow $\mathcal{A}$ to corrupt up to $t = \lfloor (n-1)/2 \rfloor$ players. Non-corrupt players are assumed honest; they are in majority and can proceed without the corrupt players if necessary.

Henceforth let A denote the set of players that are corrupted by adversary $\mathcal{A}$. Let H be the set of remaining players assumed honest. Let ran_H be the set of all the random strings of the honest players.

3.3 Zero-Knowledge

The notion of zero-knowledge was introduced by [GMR89]. Roughly speaking, when we say that a protocol is *zero-knowledge* we want to capture the fact that a player or a coalition of players get no additional information by running the protocol, other than the protocol's output.

The intuition when showing that a protocol is zero-knowledge is to show that everything an adversary can feasibly compute from running a zero-knowledge protocol is also feasible to compute from the input and the result, i.e., without interacting with the honest players. This is supposed to be true independent of strategy of the adversary.

This gave rise to the simulation paradigm [GMR89]. To prove zero-knowledge we construct a simulator, a probabilistic polynomial time Turing Machine, simulating the actions of the honest players running P on the same input as $\mathcal{A}$ has access to, both public and private.

We can define the the simulator's output to be the set of values produced by the simulator. This is called the simulated view. The values $\mathcal{A}$ sees when running P with the honest players we call the adversarial view. If the probability distribution of these two views is computationally indistinguishable then $\mathcal{A}$ learns nothing from running P, since it could compute the same information on his own. As more formally stated by [Gol01]:

Definition 3.1. Let $P(H, A, x, z)$ be the probability distribution of the output from the interactive protocol P run by H and A on input x and auxiliary input z.

We say that P is *zero-knowledge* if for every A there exists a probabilistic Turing Machine S using x, z as input and generating output with the probability distribution $S(x, z)$ (statistically, computationally) indistinguishable from $P(H, A, x, z)$.

This definition requires us to build one simulator for each adversary in order to show zero-knowledge. Instead we strengthen the definition of zero-knowledge by constructing a universal simulator that get access to the program of the adversary. The simulators' algorithm should be a function of the adversarial program so that we can plug in any adversarial program and let the simulator make use of the adversary as a *black-box*. This is sometimes referred to as black-box simulation; it was introduced in [GK96].

If a protocol asserting knowledge is a zero-knowledge protocol, the resulting protocol is referred to as a zero-knowledge proof of knowledge. These are often used to force adversarial players to behave according to a protocol, by requiring parties to provide zero-knowledge proofs of correctness of their secret-based actions, without revealing these secrets.

3.4 Modeling Hash-functions

We use the random oracle model. This method, suggested by [BR93], uses an ideal random oracle in the place of a hash-function. The random oracle always outputs a truly random value to each question.

We use the oracle to produce a random challenge when proving the correctness of published output. The oracle's output will therefore be referred to as *the oracle challenge*.

The random oracle can be modeled as a Turing Machine with access to a long list. Each question identifies a position in the list. Each time an oracle question is asked, the Turing Machine looks at this position and sees if there is an entry at this position. If there is, this is the oracle answer. If the entry is empty, the oracle tosses a coin k times and registers the answer in the position in the list and outputs this answer. We assume that asking a question to the oracle and getting an answer takes constant time.

Chapter 4

The Generic Sub-Protocols

This chapter describes two generic protocols, DLOG and COM. They are used, with different inputs, as sub-protocols of protocols presented in the subsequent chapters.

We prove security, i.e., completeness, soundness, and zero-knowledge of DLOG and COM. The proofs are secure in the random oracle model [BR93]. Since the protocols are zero-knowledge proofs of knowledge, we occasionally refer to them as "proofs" rather than "protocols".

The DLOG protocol allows a prover to prove the existence and knowledge of an element $x \in \mathbb{Z}_q$ satisfying $y_1 = g^x$ and $y_2 = h^x$, where g and h are publicly known generators of $\mathcal{G}_q$, but where $\log_g(h)$ is unknown, and the values y_1 and y_2 are public. This kind of protocol is due to Chaum and Pedersen [CP92]. These are similar to the protocol Schnorr presented in [Sch91] that allows a prover to prove to another player that she knows the discrete logarithm (ex. x such that $g^x = y$, where y is public).

The COM protocol lets a prover prove that $D = d^z$, where $z \in_R \mathbb{Z}_q$ is private and D, d are public values, and convince the verifier that z is the value to which the prover committed earlier. The commitment is a Pedersen commitment, $C = g^z h^r$, where g and h are random generators for which $\log_g(h)$ is unknown, and $r \in_R \mathbb{Z}_q$ is private. This protocol is similar to the DLOG protocol and to Okamoto's protocol from [Oka92]. The Okamoto protocol is used when a prover commits to the value z by publishing $C = g^z h^r$, and publishes the value $G = g^z$. The prover is then able to convince the verifier that $\log_g(G) = \log_g(C/h^r)$ using a random oracle. The COM protocol never reveals G.

The normal setting for a proof of knowledge, such as DLOG and COM, is with two players, a prover claiming knowledge, and a verifier verifying the fact. We use these protocols in a multi-party setting in which each player acts as both a prover and verifier, having different roles for different statements.

Many protocols published today prefer to give proofs of knowledge as non-interactive proofs of knowledge using the Fiat-Shamir heuristic [FS87]. The heuristic is to replace the query from the verifier to the prover by an output from a random oracle. The prover evaluates the random oracle in a position depending on his previous output. This is stated in a two-party setting, with a prover and a verifier. The number of communication rounds is reduced by a constant factor.

In our protocols, if we had a verifier that we could guarantee was honest, we could let him produce random output when needed. Unfortunately, we do not have any such guarantees. Provers double as verifiers and vice versa and can not be considered honest. To come around this we follow the Fiat-Shamir heuristic and let the provers, i.e., all the players, contribute to the query to the random oracle. In this way we get a query to the random oracle that is sufficiently random as long as at least one player is honest.

4.1 The DLOG Protocol

The goal of the DLOG protocol is for each player P_i to convince all other n participating players that the given tuple $(g, g^{a_i}, g^{b_i}, g^{c_i}) \in \Delta_{DDH}$. Reformulated, given a tuple $(g, g^{a_i}, g^{b_i}, g^{b_i a_i'})$, a random oracle $\mathcal{O}$, and the value a_i, P_i shows that $a_i = a_i'$, revealing neither a_i nor a_i'. We assume that a majority of the verifiers are honest.

The technique is to let P_i compute an answer w_i to a challenge v_i from the random oracle $\mathcal{O}$. This challenge is obtained by letting P_i ask $\mathcal{O}$ a question consisting of a concatenation of some data published by player P_i (values U_i and Q_i) and of one random input b_j of length k from each participating player. This construction prevents P_i from asking this question earlier, except with negligible probability. To prevent players from asking each others' questions, they include a random string to the input to the oracle, thus diversifying the question to the oracle. This random input is revealed after all players have asked their oracle question. The values U_i and Q_i are used to avoid revealing a_i when w_i is published. The full protocol is presented in Figure 4.1. To capture the fact that the values a_i and b_i are not known in the public input to DLOG, we write the tuple as (g, A_i, B_i, C_i). All communication is via the bulletin board. The output of DLOG consists of the set M_{DLOG} of complaints filed by the verifying players. If $M_{\mathsf{DLOG}} = \emptyset$, we say that DLOG accepts, i.e., outputs ACC.

4.1.1 The Completeness of DLOG

Lemma 4.1. If $w_i = s_i + v_i a_i$, $C_i = B_i^{a_i}$, and $Q_i = B_i^{s_i}$, then P_i's proof is always accepted on input (g, A_i, B_i, C_i).

Proof. With $w_i = s_i + v_i a_i$, (4.1) is always true by definition. Equation 4.2 is always true when $Q_i = B_i^{s_i}$ and $C_i = B_i^{a_i}$. Therefore a correct proof will always be accepted. ∎

Public input: $(p,q,g)_k$, (g, A_i, B_i, C_i) for $i = 1, \ldots, n$
P_i**'s private input:** $a_i = log_g(A_i)$
DLOG**'s output:** The set $M_{\mathsf{DLOG}} = \{F_{j,i}\}$ for $j, i = 1, \ldots, n$, $j \neq i$.

1. Each player P_i selects $s_i \in_R \mathbb{Z}_q$, computes and publishes

$$U_i = g^{s_i} \text{ and } Q_i = (B_i)^{s_i}.$$

2. Each player P_i selects and publishes $d_i \in_R \{0,1\}^k$
3. Each player P_i does the following:
4.
   - Selects $e_i \in \{0,1\}^k$.
   - Computes $v_i = \mathcal{O}(e_i d_1 \ldots d_n U_i Q_i i)$.
   - Computes $w_i = s_i + v_i a_i$.
   - Publishes w_i and e_i.
5. Each player P_j checks the values published by all other players P_i by checking the validity of

$$g^{w_i} = U_i A_i^{v_i} \tag{4.1}$$

and

$$B_i^{w_i} = Q_i C_i^{v_i}. \tag{4.2}$$

If P_j's check of P_i's proof fails, P_j publishes a complaint $F_{j,i}$.

Figure 4.1. The DLOG protocol

4.1.2 The Soundness of DLOG

Lemma 4.2. The probability is negligible that the DLOG protocol outputs ACC when $(g, A_i, B_i, C_i) \in \Delta_{rand}$.

Proof. We have $(g, A_i, B_i, C_i) = (g, g^{a_i}, g^{b_i}, g^{b_i a_i'})$. If $a_i \neq a_i'$ and $Q_i = B_i^{s_i'}$, and $s_i' = s_i$, where s_i is defined by U_i, no value w_i can satisfy both (4.1) and (4.2). Therefore we assume that $s_i' \neq s_i$. We also assume that w_i is such that (4.1) is true, i.e., $w_i = s_i + v_i a_i$. To make (4.2) hold too, P_i must find v_i such that $w_i = s_i' + v_i a_i'$, since the values s_i, s_i' and a_i' already are fixed at this point in the protocol. This can only be done by asking different oracle questions, i.e., trying different values for e_i.

At best the probability of acceptance of DLOG for player P_i is $p(k)/q$ when $a_i' \neq a_i$, where $p(k)$ is the polynomial number of different oracle questions (choices of e_i) P_i (or $\mathcal{A}$) can try out before deciding upon one. ∎

4.1.3 The Zero-knowledge of DLOG

In order to show zero-knowledge of DLOG, we define the simulator $\mathsf{SIM}_{\mathsf{DLOG,i}}$ running the DLOG protocol on behalf of P_i, without access to P_i's private input. The simulator $\mathsf{SIM}_{\mathsf{DLOG}}$ runs the simulators $\mathsf{SIM}_{\mathsf{DLOG,i}}$ for all players. These protocols are shown in Figure 4.3 and Figure 4.2. The public input to $\mathsf{SIM}_{\mathsf{DLOG}}$ consists of a tuple $(g^{a_i}, g^{b_i}, g^{c_i}) = (A_i, B_i, C_i)$ and a random oracle.

Let V_{DLOG} be the random variable that is defined by the adversarial view of an execution of DLOG, and let D_{DLOG} be the probability distribution of V_{DLOG}. Let $V_{\mathsf{SIM}_{\mathsf{DLOG}}}$ be the random variable defined by the output of $\mathsf{SIM}_{\mathsf{DLOG}}$ and let $D_{\mathsf{SIM}_{\mathsf{DLOG}}}$ be the probability distribution of $V_{\mathsf{SIM}_{\mathsf{DLOG}}}$. The probability in both $D_{\mathsf{SIM}_{\mathsf{DLOG}}}$ and D_{DLOG} is taken over $ran_A, ran_H, \mathcal{O}$, and the input. To prove zero-knowledge we show that $\mathsf{SIM}_{\mathsf{DLOG}}$ produces an output in polynomial time and that D_{DLOG} is statistically indistinguishable from $D_{\mathsf{SIM}_{\mathsf{DLOG}}}$.

Lemma 4.3. $\mathsf{SIM}_{\mathsf{DLOG}}$ runs in time polynomial in the security parameter k.

Proof. $\mathsf{SIM}_{\mathsf{DLOG}}$ contains no loops, and each of the constant number of steps takes at most polynomial time to complete. ∎

Lemma 4.4. The probability is negligible that $V_{\mathsf{SIM}_{\mathsf{DLOG}}}$ contains an ERR-message.

Proof. ERR-messages can only be published in $\mathsf{SIM}_{\mathsf{DLOG}}$, and only when the oracle is asked a question that has been asked earlier. In the worst case only the value e_i in player P_i's question to the oracle can be trusted to be random and independent. Therefore the probability is $\leq p(k)/2^k$ that a specific question has been asked at any stage, where $p(k)$ is the polynomial number of questions all players can ask. As p is polynomial the lemma follows. ∎

Lemma 4.5. $D_{\mathsf{SIM}_{\mathsf{DLOG}}} \overset{s}{\approx} D_{\mathsf{DLOG}}$

Public input: $(p, q, g)_k$, (g, A_i, B_i, C_i)

1. For player P_i select $w_i, v_i \in_R Z_q$ and compute and publish

$$U_i = \frac{g^{w_i}}{A_i^{v_i}}, \text{ and } Q_i = \frac{B_i^{w_i}}{C_i^{v_i}}.$$

2. For player P_i select and publish $d_i \in_R \{0,1\}^k$.
3. For player P_i select and publish $e_i \in_R \{0,1\}^k$.
   - If $\mathcal{O}(e_i d_1, \ldots, d_n U_i Q_i i)$ is unasked let $\mathcal{O}(e_i d_1, \ldots, d_n U_i Q_i i) = v_i$ and publish w_i and e_i.
   - If the question $\mathcal{O}(e_i d_1, \ldots, d_n U_i Q_i i)$ has been asked earlier publish ERR.

Figure 4.2. The protocol for $\mathsf{SIM}_{\mathsf{DLOG,i}}$

Public input: $(p, q, g)_k$, (g, A_i, B_i, C_i) for $i = 1, \ldots, n$.
$\mathcal{A}$'s private input: a_i for all corrupt P_i
$\mathsf{SIM}_{\mathsf{DLOG}}$**'s output:** $V_{\mathsf{SIM}_{\mathsf{DLOG}}}$

1. For each player $P_i \in H$ run $\mathsf{SIM}_{\mathsf{DLOG,i}}$ synchronized and interact with $\mathcal{A}$ when necessary.
2. If an ERR message has been published let $V_{\mathsf{SIM}_{\mathsf{DLOG}}} = ERR$, otherwise let $V_{\mathsf{SIM}_{\mathsf{DLOG}}} = \{U_i, Q_i, d_i, e_i, w_i, v_i : i = 1, \ldots n\}$.

Figure 4.3. The protocol for $\mathsf{SIM}_{\mathsf{DLOG}}$

Proof. Both $\mathsf{SIM}_{\mathsf{DLOG}}$ and D_{DLOG} output the variables U_i, Q_i, v_i, and w_i. The values v_i and w_i are random and independent values. Together with values U_i and Q_i they satisfy equations 4.1 and 4.2.

Let NOC be the event that there are no oracle collisions in $\mathsf{SIM}_{\mathsf{DLOG}}$. Let OC be the event that there are oracle collisions in $\mathsf{SIM}_{\mathsf{DLOG}}$. The two events NOC and OC are exclusive:

$$\begin{aligned}\mathsf{Pr}[V_{\mathsf{SIM}_{\mathsf{DLOG}}} = \alpha] = \quad & \mathsf{Pr}[V_{\mathsf{SIM}_{\mathsf{DLOG}}} = \alpha | NOC]\mathsf{Pr}[NOC] + \\ & \mathsf{Pr}[V_{\mathsf{SIM}_{\mathsf{DLOG}}} = \alpha | OC]\mathsf{Pr}[OC], \end{aligned} \tag{4.3}$$

where α is a view. The same is valid for $\mathsf{Pr}[V_{\mathsf{DLOG}} = \alpha]$. Statistical indistinguishability be expressed as

$$\begin{aligned}&\mathsf{Pr}[NOC]\sum_{\alpha} |(\mathsf{Pr}[V_{\mathsf{SIM}_{\mathsf{DLOG}}} = \alpha|NOC] - \mathsf{Pr}[V_{\mathsf{DLOG}} = \alpha|NOC])| + \\ &\mathsf{Pr}[OC]\sum_{\alpha} |(\mathsf{Pr}[V_{\mathsf{SIM}_{\mathsf{DLOG}}} = \alpha|OC] - \mathsf{Pr}[V_{\mathsf{DLOG}} = \alpha|OC])| < \frac{1}{p(k)}. \end{aligned} \tag{4.4}$$

The probability is taken over ran_A, ran_H, the input, and the random oracle $\mathcal{O}$. We call this ran. Let ran_{NOC} be the set of all ran that do not give rise to an oracle collision in $\mathsf{SIM}_{\mathsf{DLOG}}$. Let ran_{OC} be the set of all ran that do give rise to an oracle collision in $\mathsf{SIM}_{\mathsf{DLOG}}$.

In both cases v_i, w_i are random and independent numbers and U_i and Q_i satisfy (4.1) and (4.2). This means that

$$|(\mathsf{Pr}[V_{\mathsf{SIM}_{\mathsf{DLOG}}} = \alpha|NOC] - \mathsf{Pr}[V_{\mathsf{DLOG}} = \alpha|NOC])| = 0, \tag{4.5}$$

for all possible views α that ran can produce, that do not contain any oracle collisions. The second sum in (4.4) leaves us with

$$\mathsf{Pr}[OC]\sum_{\alpha} |(\mathsf{Pr}[V_{\mathsf{SIM}_{\mathsf{DLOG}}} = \alpha|OC] - \mathsf{Pr}[V_{\mathsf{DLOG}} = \alpha|OC])| \leq 2\mathsf{Pr}[OC]. \tag{4.6}$$

For a fixed ran, i.e., for a fixed view, the probability is $\leq p(k)/2^k$ that we get an oracle collision (see Lemma 4.4). The probability is $\leq |H||A|p^2(k)/2^k$ that $\mathsf{SIM}_{\mathsf{DLOG}}$ asks a question on behalf of the honest players that already has been asked. Since this fraction is negligible, we conclude that $\mathsf{Pr}[OC]$ is negligible, giving $D_{\mathsf{SIM}_{\mathsf{DLOG}}} \overset{s}{\approx} D_{\mathsf{DLOG}}$. ∎

Lemma 4.6. Running $\mathsf{SIM}_{\mathsf{DLOG}}$ on input $(g^a, g^b, g^c) \in \Delta_{rand}$ gives output whose probability distribution, $D'_{\mathsf{SIM}_{\mathsf{DLOG}}}$, is computationally indistinguishable from that of a real run of DLOG.

Proof. If $D'_{\mathsf{SIM}_{\mathsf{DLOG}}}$ is computationally distinguishable from $D_{\mathsf{SIM}_{\mathsf{DLOG}}}$, then we can use $\mathcal{A}$ to solve the decisional Diffie-Hellman assumption in polynomial time, and hence $D_{\mathsf{SIM}_{\mathsf{DLOG}}} \overset{c}{\approx} D'_{\mathsf{SIM}_{\mathsf{DLOG}}}$. From Lemma 4.5 we have that $D_{\mathsf{SIM}_{\mathsf{DLOG}}} \overset{s}{\approx} D_{\mathsf{DLOG}}$, which implies $D_{\mathsf{SIM}_{\mathsf{DLOG}}} \overset{c}{\approx} D_{\mathsf{DLOG}}$. On this basis we conclude that $D'_{\mathsf{SIM}_{\mathsf{DLOG}}} \overset{c}{\approx} D_{\mathsf{DLOG}}$. ∎

4.2 The COM Protocol

The goal of the COM protocol is for each player P_i to convince the others that he actually used the value z_i to which he committed earlier, using a Pedersen commitment to compute d_i. More specifically, given C_i and $d_i (= d^{z'_i})$, each P_i shows that he knows a pair (z_i, r_i) giving C_i such that $z_i = z'_i$, without revealing (z_i, r_i). This is done by computing w_i and $\bar{w}_i$ in response to an oracle challenge v_i. The value w_i is also used to prove that P_i knows z'_i, thus also showing that $z_i = z'_i$. The COM protocol is shown in Figure 4.4. The values U_i and Q_i are published to avoid revealing z_i and r_i. Note that the value s_i is uniquely defined by U_i, the value t_i by Q_i and the value z'_i by d_i.

The output of COM is the set M_{COM} that is the set of all published complaints.

4.2.1 The Completeness of COM

Lemma 4.7. If (z_i, r_i) gives C_i, (s_i, t_i) gives Q_i, $w_i = s_i + z_i v_i$, and $\bar{w}_i = t_i + r_i v_i$ then P_i's proof is always accepted on input d, C_i, and $d_i = d^{z_i}$, when all players are honest.

Proof. With $w_i = s_i + z_i v_i$, (4.7) will always be true. Let $\bar{w}_i = t_i + r_i v_i$. Then (4.8) will always be true when (s_i, t_i) give Q_i and (z_i, r_i) give C_i. Therefore a correct proof will always be accepted. ∎

4.2.2 The Zero-knowledge of COM

To show zero-knowledge for COM, we build a simulator $\mathsf{SIM}_{\mathsf{COM,i}}$ (Figure 4.5) running the COM protocol on behalf of player P_i, without access to P_i's private input (z_i, r_i). Let $\mathsf{SIM}_{\mathsf{COM}}$ (Figure 4.6) be the simulator using $\mathsf{SIM}_{\mathsf{COM,i}}$ for all $P_i \in H$ to simulate the actions of the honest players running the COM protocol interacting with $\mathcal{A}$.

Let V_{COM} be the random variable defined by the adversarial view of an execution of COM, and let D_{COM} be the probability distribution of V_{COM}. Let $V_{\mathsf{SIM}_{\mathsf{COM}}}$ be the random variable defined by the output of $\mathsf{SIM}_{\mathsf{COM}}$ and let $D_{\mathsf{SIM}_{\mathsf{COM}}}$ be the probability distribution of $V_{\mathsf{SIM}_{\mathsf{COM}}}$. The probability in both $D_{\mathsf{SIM}_{\mathsf{COM}}}$ and D_{COM} is taken over $ran_A, ran_H, \mathcal{O}$, d, d_i, and C_i. To prove zero-knowledge we show that $\mathsf{SIM}_{\mathsf{COM}}$ produces an output in polynomial time and that D_{COM} is statistically indistinguishable from $D_{\mathsf{SIM}_{\mathsf{COM}}}$.

Lemma 4.8. $\mathsf{SIM}_{\mathsf{COM}}$ runs in time polynomial in the security parameter k.

Proof. $\mathsf{SIM}_{\mathsf{COM}}$ contains no loops, and each of the constant number of steps takes at most polynomial time to complete. ∎

Public input: $(p, q, g)_k$, h, d, d_i, Pedersen commitments C_i for $i = 1, \ldots, n$.
P_i **'s private input**: z_i, r_i
COM**'s output:** M_{COM}

1. Each player P_i selects $s_i, t_i \in_R \mathbb{Z}_q$, computes and publishes

$$U_i = d^{s_i} \text{ and } Q_i = g^{s_i} h^{t_i}.$$

2. Each player P_i selects and publishes $b_i \in_R \{0,1\}^k$.
3. Each player P_i does the following:
   (a) Selects $c_i \in_R \{0,1\}^k$.
   (b) Computes $v_i = \mathcal{O}(c_i b_1 \ldots b_n U_i Q_i i)$.
   (c) Computes $w_i = s_i + v_i z_i$ and $\bar{w}_i = t_i + v_i r_i$.
   (d) Publishes $(w_i,\, \bar{w}_i,\, c_i)$.
4. Each player P_j checks the values published by all other players P_i by checking the validity of

$$d^{w_i} = U_i d_i^{v_i} \tag{4.7}$$

and

$$g^{w_i} h^{\bar{w}_i} = Q_i (C_i)^{v_i}. \tag{4.8}$$

If P_j's check of P_i's proof fails then P_j publishes a complaint $F_{j,i}$. Let M_{COM} be the set of all published complaints.

Figure 4.4. The COM protocol

Public input: $(p, q, g)_k$, h, d, d_i and Pedersen commitment C_i.

For player P_i do the following:

1. Select $w_i, \bar{w}_i, v_i \in_R \mathbb{Z}_q$ and compute and publish
$$Q_i = \frac{g^{w_i} h^{\bar{w}_i}}{C_i^{v_i}} \text{ and } U_i = \frac{d^{w_i}}{d_i^{v_i}}.$$
2. Select and publish $b_i \in_R \{0,1\}^k$.
3. Select $c_i \in_R \{0,1\}^k$.
   - If $\mathcal{O}(c_i b_1 \dots b_n U_i Q_i i)$ is unasked then let $\mathcal{O}(c_i b_1 \dots b_n U_i Q_i i) = v_i$ and publish $w_i, \bar{w}_i, c_i$.
   - If the question $\mathcal{O}(c_i b_1 \dots b_n U_i Q_i i)$ has been asked earlier then publish *ERR*.

Figure 4.5. The protocol for $\mathsf{SIM}_{\mathsf{COM,i}}$

Public input: $(p, q, g)_k$, h, d, Pedersen commitments C_i and values d_i for $i = 1, \dots, n$.
$\mathsf{SIM}_{\mathsf{COM}}$**'s output:** $V_{\mathsf{SIM}_{\mathsf{COM}}}$.

1. For each player $P_i \in H$ run $\mathsf{SIM}_{\mathsf{COM,i}}$ synchronized and interact with $\mathcal{A}$ when needed.
2. If no *ERR* message has been published then let $V_{\mathsf{SIM}_{\mathsf{COM}}} = \{U_i, Q_i, b_i, w_i, \bar{w}_i, v_i, c_i : i = 1 \dots n\}$, otherwise let $V_{\mathsf{SIM}_{\mathsf{COM}}} = ERR$.

Figure 4.6. The protocol for $\mathsf{SIM}_{\mathsf{COM}}$

Lemma 4.9. The probability is negligible that $V_{\mathsf{SIM}_{\mathsf{COM}}}$ contains the message ERR.

Proof. ERR-messages can only be published in $\mathsf{SIM}_{\mathsf{COM,i}}$, and only when the oracle is asked a question that has been asked earlier. Only the honest players contribution to the oracle question can be trusted to be random. We refer to the proof of Lemma 4.4, which analyses an identical case and computes the probability. ∎

Lemma 4.10. $D_{\mathsf{SIM}_{\mathsf{COM}}} \overset{s}{\approx} D_{\mathsf{COM}}$

Proof. Both $\mathsf{SIM}_{\mathsf{COM}}$ and D_{COM} output the variables U_i, Q_i, v_i, w_i, and $\bar{w}_i$. The values v_i, w_i, and $\bar{w}_i$ are random and independent. Together with values U_i and Q_i, they satisfy equations 4.7 and 4.8. The proof of this lemma is analogous to that of Lemma 4.5.

∎

4.2.3 The Soundness of COM

The Pedersen commitment scheme is an information theoretically secure commitment scheme, i.e., it does not reveal any information about the pair (z_j, r_j) used to compute the commitment C_j, apart from the value $e_j = z_j + \log_g(h)r_j$, where $C_j = g^{e_j}$. In fact any pair (z, r) where $e_j = z + \log_g(h)r$ is a possible candidate pair.

The value d_j determines some z_j uniquiely, since $z_j = \log_d(d_j)$. If P_j passes the COM protocol this means that P_j can efficiently compute a pair (z_j, r_j) such that $d_j = d^{z_j}$ and $e_j = z_j + \log_g(h)r_j$. Even if we extract (z_j, r_j), we still have no means of determining whether the pair we extracted was the pair used to compute the commitment C_j in the begining. We can therefore not determine whether P_j is cheating from a single round of COM.

Only when P_j is capable of passing the COM protocol with non-negligible probability for different values of d_j while using the same commitment can we say that P_j is cheating in one of the cases, i.e., $\log_d(d_j)$ not being the value used to compute the commitment. If P_j passes for two different values, $g^{z_j}h^{r_j}$ and $g^{z_j'}h^{r_j'}$, then he can compute $\log_g(h) = (z' - z)/(r - r')$ and vice versa.

From this informal discussion we see that the soundness of COM depends on P_j not being able to compute $\log_g(h)$ efficiently. This prevents an adversary from committing to one value and opening it as another, except with negligible probability.

In order to show that the COM protocol is sound we run first address the case when we have a single corrupt player, namely P_j. This is formalized in Lemma 4.11, where we use P_j as a black-box to extract $\log_g(h)$. The general case, in which an adversary corrupts up to t players is not much more complicated. Lemma 4.12 addresses this case.

The COM protocol is used in a setting where players first compute and publish commitments C_j. At a later stage they publish the values d_j and claim their consistency with the commitment. We assume this setting when showing the soundness of COM.

Let $PASS_j$ be the event that player P_j runs COM successfully. Also let d_j be the event that P_j published the value d_j and let 2^T be the time it takes to compute a discrete logarithm in $\mathcal{G}_q$.

Lemma 4.11. Given C_j, if $\exists\ d_j$, d'_j, where $d'_j \neq d_j$, such that $\mathsf{Pr}[d_j] > 1/poly(k)$, $\mathsf{Pr}[d'_j] > 1/poly(k)$, $\mathsf{Pr}[PASS_j|d_j] > 1/poly(k)$, and $\mathsf{Pr}[PASS_j|d'_j] > 1/poly(k)$, then we can use P_j to compute the discrete logarithm of h in base g in polynomial time, assuming all other players' values fixed.

Proof. We define an extractor $\mathsf{ECOM_1}$ (Figure 4.7), using P_j as a black-box. $\mathsf{ECOM_1}$ is a Turing Machine with an internal random string $ran_{\mathsf{ECOM_1}}$.

To compute $\log_g(h)$, $\mathsf{ECOM_1}$ needs to extract two different pairs (z_j, r_j) and (z'_j, r'_j) giving C_j. To compute (z_j, r_j) (or (z'_j, r'_j)) P_j must pass COM twice with different oracle challenges (v_j and v'_j). This gives the values $w_j = s_j + v_j z_j$, $w'_j = s_j + v'_j z_j$, $\bar{w}_j = t_j + v_j r_j$, and $\bar{w}'_j = t_j + v'_j r_j$, where w_j, w'_j, $\bar{w}_j$, $\bar{w}'_j$, and the oracle challenges are known. The values z_j and r_j are computed as

$$z_j = \frac{w_j - w'_j}{v_j - v'_j} \text{ and } r_j = \frac{\bar{w}_j - \bar{w}'_j}{v_j - v'_j}. \tag{4.9}$$

Therefore $\mathsf{ECOM_1}$ rewinds the protocol to get a new oracle challenge for P_j until P_j has passed COM twice. Though $\mathsf{ECOM_1}$ rewinds no more than 2^T times, in which case we let $\mathsf{ECOM_1}$ compute $\log_g(h)$ from g and h. When extracting (z'_j, r'_j), if $\mathsf{SIM}_{\mathsf{COM},i}$ publishes ERR then rewind.

In each step in $\mathsf{ECOM_1}$'s algorithm, $\mathsf{ECOM_1}$ must compute the output of the honest players. Before rewinding, this does not impose any problems in steps 1 to 3d, since the honest players' output only depends on their random bits. $\mathsf{ECOM_1}$ chooses a pair (z_i, r_i) for each player $P_i \in H$ and computes $d_i = d^{z_i}$, and can therefore always run COM successfully.

Step 4 in Figure 4.7 deals with the problem of getting two different pairs (z_j, r_j) and (z'_j, r'_j) giving C_j. $\mathsf{ECOM_1}$ rewinds the protocol and publishes values d_i for the honest players. If $\mathsf{ECOM_1}$ keeps all the d_i's, then everything P_j sees is exactly as in the first steps of $\mathsf{ECOM_1}$'s algorithm, including what P_j reads from ran_j causing P_j to publish the same d_j. Consequently, the extractor $\mathsf{ECOM_1}$ must output $d'_i \neq d_i$ for at least one $P_i \in H$. Let us assume that $\mathsf{ECOM_1}$ outputs $d'_i \neq d_i \forall P_i \in H$. As long as P_j does not output $d'_j \neq d_j$, rewind and try a new set of d_i values for the honest players.

When P_j outputs $d'_j \neq d_j$ we have a situation where $\mathsf{ECOM_1}$ does not know the pair (z'_i, r'_i) giving C_i, such that $d'_i = d^{z'_i}$. To compute output for each player $P_i \in H$, $\mathsf{ECOM_1}$ uses $\mathsf{SIM}_{\mathsf{COM},i}$, which produces output for player P_i with exactly the same probability distribution as is P_i's real output, in all but a negligible number of cases (Lemma 4.10).

Below we show that ECOM_1 runs in expected polynomial time. In an expected polynomial number of tries, ECOM_1 succeeds with extracting (z_j, r_j), making P_j output $d'_j \neq d_j$ and extracting (z'_j, r'_j), and running $\mathsf{SIM}_{\mathsf{COM},i}$ successfully for all honest players P_i. Let RT_{ECOM_1} be the time ECOM_1 needs to terminate, i.e., extract $\log_g(h)$. Let $RT_{(z_j,r_j)}$ be the time needed to complete step 3 and $RT_{(z'_j,r'_j)\neq(z_j,r_j)}$ be the time needed to complete steps 4 and 5. The total expected running time for ECOM_1 is $\mathsf{E}[RT_{\mathsf{ECOM}_1}] \leq \mathsf{E}[RT_{(z_j,r_j)}] + \mathsf{E}[RT_{(z'_j,r'_j)\neq(z_j,r_j)}]$.

Let us first define the following: Let U be the random variable defined by the value of the random bits used by ECOM_1 and P_j from start and up to step 2 in Figure 4.7. Let M_{d_j} be the subset of all u such that P_j publishes d_j. Let V be all the bits used from start (when selecting z_j and r_j for the commitment) to the end of COM, let v be an instance of V and let $\mathcal{V}_j = \{v | 2^{-(j+1)} < \mathsf{Pr}[PASS_j|v] \leq 2^{-j}\}$, be the subset of all bit strings resulting in $PASS_j$. Now let us look at $\mathsf{E}[RT_{(z_j,r_j)}]$ and $\mathsf{E}[RT_{(z'_j,r'_j)\neq(z_j,r_j)}]$ separately.

$\mathsf{E}[RT_{(z_j,r_j)}]$:

The running time to find the pair (z_j, r_j) ($RT_{(z_j,r_j)}$) only depends on the success rate of P_j passing COM, which P_j must do twice with different and independent challenges.

We take a closer look at the expected running time of one execution from start to $PASS_j$, given that we rewind to get a new oracle challenge, but with the restriction that we rewind at most 2^T times. Bayes theorem together with the definition of $\mathcal{V}_j$ gives

$$\begin{aligned} \mathsf{Pr}[PASS_j \cap V \in \mathcal{V}_j] &= \sum_v \mathsf{Pr}[PASS_j | v \in \mathcal{V}_j]\mathsf{Pr}[v \in \mathcal{V}_j] \\ &\leq 2^{-j} \sum_v \Pr[v \in \mathcal{V}_j] \\ &\leq 2^{-j}. \end{aligned} \tag{4.10}$$

When we have independent tries, as we have when P_j is trying to answer a random oracle's challenge, the expected number of oracle challenges needed to $PASS_j$ is as follows. Let p be the probability of $PASS_j$, giving $\mathsf{Pr}[PASS_j$ after i tries$] = (1-p)^{i-1}p$.

$$\mathsf{E}[\#tries] \leq \sum_{i=1}^{\infty} i(1-p)^{i-1}p = \sum_{i=1}^{\infty}(-p)\frac{d}{dp}(1-p)^i = \frac{1}{p}. \tag{4.11}$$

Hence the average number of tries needed to $PASS_j$ using $v \in \mathcal{V}_j$ is $< 2^{(j+1)}$. Recall that the maximum number of tries is $\min\{2^j, 2^T\}$. For $V = v$ the expected running time over all j is:

$$\begin{aligned} & \sum_{j=1}^{\infty} \mathsf{E}[\#tries\ with\ v \in \mathcal{V}_j]\mathsf{Pr}[PASS_j \cap (v \in \mathcal{V}_j)] \\ < \quad & \sum_{j=1}^{T-1} 2^{j+1}2^{-j} + \sum_{j=T}^{\infty} 2^{T+1}2^{-j} \\ = \quad & 2(T+1). \end{aligned} \tag{4.12}$$

With two independent tries we get that $E[RT_{(z_j,r_j)}] < 4(T+1)$.

$\mathsf{E}[\boldsymbol{RT_{(z'_j,r'_j)\neq(z_j,r_j)}}]$ **:**
The running time $RT_{(z'_j,r'_j)\neq(z_j,r_j)}$, depends on u being such that Pj publishes $d'_j \neq d_j$, and on the following success rate of passing COM. We assume that P_j published d_j in step 2. Let $RT_{u\notin M_{d_j}}$ be the time needed to find $u \notin M_{d_j}$. Recall that for a fixed C_j, $\exists d'_j \neq d_j$ such that $\mathsf{Pr}[d_j] < 1/poly(k)$, where $poly(k)$ is a polynomial. Using the same analysis as in (4.11) we get that $\mathsf{E}[RT_{u\notin M_{d_j}}] < poly(k)$. The analysis of step 5 in $\mathsf{ECOM_1}$ is similar to that of step 3 in $\mathsf{ECOM_1}$, the difference being that $\mathsf{SIM}_{\mathsf{COM},i}$ may output ERR for such a v on which P_j would pass COM. Even in the case when we run 2^T iterations of step 5, the probability of getting ERR is negligible, since $2^T << 2^{-(k+3-t)}$.

We can conclude that $\mathsf{E}[RT_{z'_j,r'_j}] < 4(T+1)$. This gives $\mathsf{E}[RT_{\mathsf{ECOM_1}}] < 8(T+1) + poly(k)$, which is polynomial in k. ∎

We now address the case of an adversary $\mathcal{A}$ corrupting t players and adapting the corrupt players d'_j-values to the honest players values. For this purpose we build an extractor $\mathsf{ECOM_t}$ (Figure 4.8). Recall that this lemma is used to show the soundness of the COM protocol. Soundness only makes sense if we get an answer (run to the end) with at least non-negligible probability. This implies that all players pass COM with non-negligible probability.

Lemma 4.12. Given C_j, if for at least one player $P_j \in A$ $\exists d_j, d'_j$, where $d_j \neq d'_j$, such that $\mathsf{Pr}[d_j] > 1/poly(k)$, $\mathsf{Pr}[d'_j] > 1/poly(k)$, $\mathsf{Pr}[PASS_j|d_j] > 1/poly(k)$, and $\mathsf{Pr}[PASS_j|d'_j] > 1/poly(k)$, then we can use P_j to compute the discrete logarithm of h in base g in polynomial time.

Proof. Here we define an extractor $\mathsf{ECOM_t}$ (Figure 4.8), a Turing Machine with an internal random string $ran_{\mathsf{ECOM_t}}$ using $\mathcal{A}$ as a black-box.

This proof is very similar to the proof of Lemma 4.11. The difference is, here we do not have a specific player to focus on. Instead we have a set of players of which at least one has the behavior exploited in Lemma 4.11. Consequently the extractor $\mathsf{ECOM_t}$ is very similar to $\mathsf{ECOM_1}$.

Since the probability is non-negligible that the COM protocol terminates, then $\mathsf{Pr}[PASS_j] > 1/poly(k)$ for some polynomial $poly(k)$ and for all players $P_j \in A$.

Public input: $(p, q, g)_k$, h
Public output: $\log_g(h)$

1. Select values values $z_i, r_i \in_R \mathbb{Z}_q \forall P_i \in H$ and compute C_i.
2. Compute d_i for all $P_i \in H$.
3. Run the COM protocol to get values w_j and $\bar{w}_j$.

   Rewind to step 2 in the COM protocol and rerun the remaining steps with a new oracle challenge. This gives w'_j and $\bar{w}'_j$. Compute

   $$z_j = \frac{w_j - w'_j}{v_j - v'_j} \text{ and } r_j = \frac{\bar{w}_j - \bar{w}'_j}{v_j - v'_j}.$$

   If we do not have (z_j, r_j) after rewinding 2^T times, then compute $\log_g(h)$.
4. Output $d'_i \neq d_i$, for all $P_i \in H$. Repeat until P_j outputs $d'_j \neq d_j$.
5. Run $\mathsf{SIM}_{\mathsf{COM},i}$ for all $P_i \in H$. Run and rewind until P_j has passed COM twice and compute (z'_j, r'_j) giving C_j as above.

   If COM outputs $M_{\mathsf{COM}} \neq \emptyset$ or $\mathsf{SIM}_{\mathsf{COM},i}$ outputs ERR, then rewind COM to get a new oracle question and a new challenge for P_j. If we do not have (z'_j, r'_j) after rewinding 2^T times, compute $\log_g(h)$.
6. Compute

   $$\log_g(h) = \frac{z_j - z'_j}{r'_j - r_j}. \tag{4.13}$$

Figure 4.7. The extractor ECOM_1 interacting with P_j

Therefore, for some player $P_j \in A$ who publishes d_j and $d'_j (\neq d_j)$ with non-negligible probability, we have $\mathsf{Pr}[PASS_j | d_j] > 1/poly(k)$ and $\mathsf{Pr}[PASS_j | d'_j] > 1/poly(k)$.

For the analysis of $\mathsf{ECOM_t}$ we refer to that of extractor $\mathsf{ECOM_1}$, since the two are almost identical. The expected running time is thus polynomial in the security parameter. ∎

Public input: $(p, q, g)_k$, h
Public output: $\log_g(h)$

1. Select $z_i, r_i \in_R \mathbb{Z}_q$ and compute C_i, $\forall P_i \in H$. Each player $P_j \in A$ produces a value for C_j.
2. Compute d_i for each $P_i \in H$. Each $P_j \in A$ produces d_j.
3. Rewind and repeat step 2 until some $P_j \in A$ produces $d'_j \neq d_j$. Let each $P_i \in H$ output $d'_i \neq d_i$.
4. Run the COM protocol with values C_j, d, d_j for P_j and rewind the protocol to get a new oracle challenge. Rewind until P_j has passed twice and compute (z_j, r_j) giving C_j.

   If COM outputs REJ, then rewind COM to get a new oracle question and a new challenge for P_j. If after 2^T rewindings we do not have (z_j, r_j) compute $\log_g(h)$.
5. Run $\mathsf{SIM}_{\mathsf{COM},i}$ for all $P_i \in H$. Run the COM protocol with values C_j, d, d'_j for P_j and rewind the protocol to get a new oracle challenge. Rewind until P_j has passed twice and compute (z'_j, r'_j) giving C_j.

   If COM outputs REJ or $\mathsf{SIM}_{\mathsf{COM},i}$ outputs ERR, then rewind COM to get a new oracle question and a new challenge for P_j. If after 2^T rewindings we do not have (z'_j, r'_j) compute $\log_g(h)$.
6. Now compute

$$\log_g(h) = \frac{z_j - z'_j}{r'_j - r_j}. \tag{4.14}$$

Figure 4.8. The extractor $\mathsf{ECOM_t}$ interacting with $\mathcal{A}$

Chapter 5

EGO - A Verifiable Solution to the Socialist Millionaires Problem

In the millionaires problem, two millionaires, Alice and Bob, want to find out which is the richer without revealing any information other than the result. The socialist millionaires problem is similar: Alice and Bob only want to find out if they are equally rich or not.

Independently of our work, Boudot, Schoenmakers and Traoré [BST01] published a protocol which does not disclose any other information than whether Alice's private input equals Bob's private input and what can be implicated by it. Apart from being zero-knowledge, the protocol is also fair and efficient, requiring $O(k)$ exponentiations for security parameter k. The scenario they work with is that the two players each determine their inputs.

We consider the case when the players publicly receive their inputs from a third party, or as output from another protocol. The players are not able to alter their given input in any way. The inputs are randomly encrypted using the players' public keys. Our protocol, which we call the EGO protocol, lets the two players publicly determine if two such inputs encrypt the same plain-text. Any observer can verify the correctness of the result of the protocol. The EGO protocol reveals nothing about the underlying messages apart from what can be deduced from the result of the protocol. The security is proved using the random oracle model and under the condition that the computation is fair. Fairness means that a player does not stop after learning the result before the other, and thereby preventing the other player from getting the result.

Just as [BST01], our protocol requires $O(k)$ exponentiations for security parameter k. The main difference between the two protocols lies is that the EGO protocol is publicly verifiable.

5.1 The EGO Protocol

The EGO protocol is designed for two players, which we call P_1 and P_2. These players are modeled as we previously described in Chapter 3. Their only means of communication with each other and the external world is via a bulletin board, on which they publish messages.

To start with P_1 and P_2 jointly compute a discrete-log instance $(p, q, g)_k$ and a random generator h of $\mathcal{G}_q$. Based on these values they can each compute an ElGamal key-pair, $(x_1, y_1 = g^{x_1})$ for P_1, and $(x_2, y_2 = g^{x_2})$ for P_2, after which they publish their public keys y_1 and y_2, keeping their private keys to themselves.

The players get two ElGamal encryptions $(\gamma_1, \delta_1) = (m_1 y_1^{R_1}, g^{R_1})$ and $(\gamma_2, \delta_2) = (m_2 y_2^{R_2}, g^{R_2})$. These are encryptions of two plain-texts m_1 and m_2 which are encrypted under the players' public keys using the random blinding values R_1 and R_2. These blinding values are given to the respective player. Each player can easily verify that he obtained the correct blinding value by checking that $g^{R_i} = \delta_i$. Transferring the blinding values can be done by encrypting R_1 and R_2 with the players' public keys, y_1 and y_2 and publishing these encryptions on the bulletin board. The players can decrypt their own blinding values, but learn nothing about the other player's blinding value.

The main idea of the EGO protocol is to compute the value of $m = m_1/m_2$ which equals 1 when $m_1 = m_2$. To this end we define $(\epsilon, \varphi) = (\gamma_1/\gamma_2, \delta_1/\delta_2) = (mg^{x_1R_1 - x_2R_2}, g^{R_1 - R_2})$. This is not a regular ElGamal encryption of m. We refer to this as an EGO-encryption. Decrypting an EGO-encryption means jointly computing $g^{x_1R_1 - x_2R_2}$, which gives $m = \epsilon / g^{x_1R_1 - x_2R_2}$. We refer to this kind of decryption as EGO-decryption.

We could simply let P_1 and P_2 publish $g^{x_1R_1}$ and $g^{x_2R_2}$, but this would mean publishing all the information needed to decrypt each message on its own. Both P_1, P_2, and all observers would then be able to decrypt both messages. The goal of the EGO protocol is to reveal nothing apart from whether the messages are equal or not. This solution obviously reveals more than that.

A better solution is to let P_1 and P_2 jointly EGO-decrypt (ϵ, φ). When this decryption results in a value not equal to 1, the decryption reveals information about the messages. Given the result of the EGO-decryption and a player's own message, both players can easily compute the other player's message even when the messages are not equal. Observers do not necessarily deduce the exact values of the two messages, but they still learn a great deal about them.

In order to solve these problems we make the following observations. Using the values (ϵ, φ) and $z \in \mathbb{Z}_q$, we notice that $(\alpha, \beta) = (\epsilon^z, \varphi^z)$ is an EGO-encryption of m^z, with g^z as generator. As $\mathcal{G}_q$ is a group of prime size, g^z is a generator of $\mathcal{G}_q$. If the blinding factor z is randomly chosen and $m \in \mathcal{G}_q$ then $m^z \in_R \mathcal{G}_q$. If $m = 1$ then $m^z = 1$. As long as z is unknown to both players, EGO-decrypting (ϵ^z, φ^z) gives information only about equality.

The main part of the EGO protocol is dedicated to computing (ϵ^z, φ^z) without any of the players knowing z or being able to affect its randomness. Pedersen commitments and the sub-protocols EPSI and PHI are used in this process.

To be able to EGO-decrypt the players need to jointly compute $g^{x_1R_1-x_2R_2}$ without revealing any additional information. We use the fact that $g^{x_1R_1-x_2R_2} = g^{(x_1+x_2)(R_1-R_2)}g^{-x_1R_2}g^{x_2R_1}$, which consists of values that the players can compute together. In the process the players compute $g^{x_1R_2} = G_{12}$ and $g^{x_2R_1} = G_{21}$. The sub-protocols $\mathsf{EGO-O}$, $\mathsf{EGO-Q}$, $\mathsf{EGO-T}$ and BETA are used to ensure that (ϵ^z, φ^z) are correctly EGO-decrypted. Observers learn the result and P_1 and P_2 learn also what is inferred from the result, but nothing more. The sub-protocols mentioned are described in more detail in Section 5.2. The decryption is computed as

$$\frac{\alpha T}{\beta_1\beta_2 Q} = \frac{\epsilon^z G_{21}^z}{\varphi^{zx_1}\varphi^{zx_2}G_{12}^z} = \frac{m^z g^{(x_1R_1-x_2R_2)z}g^{x_2R_1z}}{g^{(R_1-R_2)x_1z}g^{(R_1-R_2)x_2z}g^{x_1R_2z}} = \frac{m^z g^{x_1R_1z+x_2R_1z-x_2R_2z}}{g^{x_1R_1z+x_2R_1z+x_1R_2z-x_1R_2z-x_2R_2z}} = m^z. \tag{5.1}$$

Figure 5.1 shows the EGO protocol in detail. Recall that the execution of the protocol is in synchronized communication rounds, i.e., the two players are expected to await the publishing of the results in the current step before proceeding with publishing in the next step. The result of EGO, to which we refer as $res_{\mathsf{EGO}}(m)$, can be one of the following:

E	when (α, β) EGO-decrypts to 1.
N	when (α, β) does not EGO-decrypt to 1.
F_ϵ	when the output of EPSI is REJ.
F_φ	when the output of PHI is REJ.
F_O	when the output of $\mathsf{EGO-O}$ is REJ.
F_Q	when the output of $\mathsf{EGO-Q}$ is REJ.
F_T	when the output of $\mathsf{EGO-T}$ is REJ.
F_β	when the output of BETA is REJ.
VIO	when a player published prematurely.

Let $FAIL$ be the event when one of F_ϵ, F_φ, F_O, F_Q, F_T, F_β, VIO occurs.

5.2 The Sub-Protocols in EGO

As mentioned above, the EGO protocol uses sub-protocols to make sure that the values published by players are computed as intended. These sub-protocols are EPSI, PHI, $\mathsf{EGO-O}$, $\mathsf{EGO-Q}$, $\mathsf{EGO-T}$, and BETA.

Public input: $(p, q, g)_k$, h y_1, y_2 (ϵ, φ), (δ_1, γ_1), (δ_2, γ_2)
Private input: Private keys corresponding to the public keys.

Each player P_i, for $i \in \{1, 2\}$ does the following:

Round 1.

- Select $z_i, r_i \in_R \mathbb{Z}_q$ and let $C_i = g^{z_i} h^{r_i}$.
- Publish C_i.

Round 2.

- Let $\epsilon_i = \epsilon^{z_i}$ and publish ϵ_i.
- Run EPSI. If EPSI outputs ACC then continue, otherwise if EPSI outputs REJ then publish F_ϵ and terminate.

Round 3.

- Let $\varphi_i = \varphi^{z_i}$ and publish φ_i.
- Run PHI. If PHI outputs ACC then continue, otherwise if PHI outputs REJ then publish F_φ and terminate.

Round 4.

- Let $G_{ij} = \delta_j^{x_i}$ and publish G_{ij}, for $j \in \{1, 2\}$, $j \neq i$.
- Run $\mathsf{EGO} - \mathsf{O}$. If $\mathsf{EGO} - \mathsf{O}$ outputs ACC then continue, otherwise if $\mathsf{EGO} - \mathsf{O}$ outputs REJ then publish F_O and terminate.

Round 5.

- Let $Q_i = G_{12}^{z_i}$ and publish Q_i.
- Run $\mathsf{EGO} - \mathsf{Q}$. If $\mathsf{EGO} - \mathsf{Q}$ outputs ACC then continue, otherwise if $\mathsf{EGO} - \mathsf{Q}$ outputs REJ then publish F_Q and terminate.

Round 6.

- Let $T_i = G_{21}^{z_i}$ and publish T_i.
- Run $\mathsf{EGO} - \mathsf{Q}$. If $\mathsf{EGO} - \mathsf{T}$ outputs ACC then continue, otherwise if $\mathsf{EGO} - \mathsf{T}$ outputs REJ then publish F_T and terminate.

Round 7.

- Compute $\beta = \varphi_1 \varphi_2$.
- Let $\beta_i = \beta^{x_i}$ and publish β_i.
- Run BETA. If BETA outputs ACC then continue, otherwise if $\mathsf{EGO} - \mathsf{Q}$ outputs REJ then publish F_β and terminate.

Round 8. Compute $\alpha = \epsilon_1 \epsilon_2$, $Q = Q_1 Q_2$ and $T = T_1 T_2$, and finally compute

$$m^z = \alpha T / (\beta_1 \beta_2 Q). \tag{5.2}$$

Figure 5.1. The EGO protocol

All of these, except EPSI, are executions of the DLOG protocol run on different inputs. We give them different names depending on their input to tell them apart. The EPSI protocol is an execution of the generic COM protocol. Section 5.2.1 gives the details of the EPSI protocol, whereas Section 5.2.2 specifies the components of the remaining protocols.

Sections 4.1 and 4.2 contain the DLOG protocol and the COM protocol together with proofs showing their is completeness, soundness and zero-knowledge. From these proofs we have that EPSI, PHI, EGO − O, EGO − Q, EGO − T, and BETA have the same properties.

5.2.1 The EPSI Protocol

The EPSI protocol is an execution of the the COM protocol on public input $(p, q, g)_k$, h, ϵ, ϵ_i, Pedersen commitments C_i, and private input $z_i = \log_{\epsilon_i}$, and r_i such that $C_i = g^{z_i} h^{r_i}$. The protocol shows that a player uses the value z, to which he committed earlier, as exponent. The values (z_i, r_i) are known only to player P_i. If both players proofs are accepted then EPSI outputs ACC. When using $\mathsf{SIM}_{\mathsf{COM}}$ on the input specified for EPSI we call the simulator SIM_ϵ.

5.2.2 The PHI, EGO − O, EGO − Q, EGO − T, and BETA Protocols

The DLOG protocol is a protocol proving that a tuple $(g, g^a, g^b, g^c) \in \Delta_{DDH}$, where a is input known only to the prover. The PHI, EGO − O, EGO − Q, EGO − T, and BETA protocols all do the same, but for different tuples, as follows :

EGO − O:	Public input: $(g, y_i, \delta_j, G_{ij})$ for $j \neq i$ Private input: $x_i = \log_g y_i$
EGO − Q:	Public input: $(\varphi, \varphi_i, G_{12}, Q_i)$ Private input: $z_i = \log_\varphi \varphi_i$
EGO − T:	Public input: $(\varphi, \varphi_i, G_{21}, T_i)$ Private input: $z_i = \log_\varphi \varphi_i$
BETA:	Public input: (g, y_i, β, β_i) Private input: $x_i = \log_g y_i$
PHI:	Public input: $(\epsilon, \epsilon_i, \varphi, \varphi_i)$ Private input: $z_i = \log_\epsilon \epsilon_i$

All protocols have the discrete-log instance $(p, q, g)_k$ as public input. The DLOG protocol outputs the set M_{DLOG}. The output of the protocols is ACC when $M_{\mathsf{DLOG}} = \emptyset$, or REJ otherwise.

When running the simulator $\mathsf{SIM}_{\mathsf{DLOG}}$ to simulate the output of honest players running one of the copies of DLOG mentioned here we write SIM_{φ}, SIM_{O}, SIM_{Q}, SIM_{T}, and SIM_{β} subsequently.

When the players have access to their own blinding factors (R_1 or R_2) they are able to compute both $G_{12} = g^{x_1 R_2}$ and $G_{21} = g^{x_2 R_1}$ on their own using private data. Player P_1 uses her private key x_1 and the public value $\delta_2 = g^{R_2}$ to compute $g^{x_1 R_2}$, whereas P_2 uses this private random value R_2 and A's public key etc. Observers, on the other hand can not compute G_{12} and G_{21} on their own, since computing these values would mean solving the computational Diffie-Hellman problem. In order to make $\mathsf{EGO}-\mathsf{Q}$ and $\mathsf{EGO}-\mathsf{T}$ verifiable to observers, the players P_1 and P_2 must publish G_{12} and G_{21} and run $\mathsf{EGO}-\mathsf{O}$ to prove their correctness.

5.3 Completeness, Soundness and Zero-Knowledge of EGO

Theorem 5.1. The EGO protocol as presented in Figure 5.1 is complete, sound and zero-knowledge if one player is honest, under the non-uniform DDH-assumption, in the random oracle model and assuming that players do not stop after learning the result in round 7.

Proof. To simplify presentation we have divide this proof into its natural components, and treat them separately; the completeness in Lemma 5.2, soundness in Lemma 5.3 and zero-knowledge Lemma 5.4. ∎

5.3.1 The Completeness of EGO

Lemma 5.2. If both players follow the EGO-protocol, when $m = 1$, $res_{\mathsf{EGO}} = E$. When $m \neq 1$, then $res_{\mathsf{EGO}} = N$ with probability $(q-1)/q$.

Proof. The result of EGO is based on the result of $\alpha T/\beta_A \beta_B Q$. This equals m^z as verified by equation 5.1. If both players are honest then res_{EGO} never equals $FAIL$.

> If $m = 1$, then for any number $z \in \mathbb{Z}_q$, $m^z = 1 \mod p$. Therefore res_{EGO} always equals E when $m = 1$.
>
> If $m \neq 1$, then $res_{\mathsf{EGO}} = N$ for all z except $z = 0$, when q is prime. With $z = 0$, $m^z = 1 \mod p$, and $res_{\mathsf{EGO}} = E$. Since z is random and uniformly distributed in $\mathbb{Z}_q$ the probability that $z = 0$ is $1/q$.

∎

5.3.2 The Soundness of EGO

Soundness in this setting is the probability that a player changes the outcome without it being detected. Changing the outcome of EGO means changing the result of $\alpha T(\beta_A \beta_B Q)^{-1}$ from 1 or to 1, and still passing all the sub-protocols.

Lemma 5.3. The probability is negligible that P_1 or P_2 efficiently changes the output of EGO from E to N or from N to E successfully.

Proof. The protocols PHI, EGO − O, EGO − Q, EGO − T, and BETA are executions of the DLOG protocol. By Lemma 4.2 we know that the probability is negligible that a player P_i passes with an incorrect triple. We can therefore assume that PHI, EGO − O, EGO − Q, EGO − T, and BETA output ACC, which means that $\log_\epsilon(E_i) = \log_\varphi(F_i)$, $\log_\varphi(F_i) = \log_{G_{21}}(T_i)$, $\log_\varphi(F_i) = \log_{G_{12}}(Q_i)$, and $\log_g(y_i) = \log_\beta(\beta_i)$ for all $i = \{1, 2\}$, except with negligible probability.

> *Case* $m = 1$*: (*$res_{\mathsf{EGO}} = E$ *to be changed to* $res_{\mathsf{EGO}} = N$*)*
> We have the fact that $m^z = 1$ for all a if $m = 1$. This implies that EGO's output can not be changed from E to N.
>
> *Case* $m \neq 1$*: (*$res_{\mathsf{EGO}} = N$ *to be changed to* $res_{\mathsf{EGO}} = E$*)*
> We make the observation that if $m \neq 1$ then $m^z = 1 \mod p$ iff $z = 0$. In other words, to change EGO's output, either z_1 or z_2 must be adapted so that $z_1 + z_2 = 0$.
>
> In Lemma 4.11 we proved an adversarial player not use his commitment in several different ways with a non-negligible probability. Consequently if a player can make a$z_1 + z_2 = 0$ with non-negligible probability, we can use that player to compute $\log_g(h)$ in expected polynomial time. Since this is assumed impossible, the probability is negligible that the adversarial player succeeds, as long as $\log_g(h)$ is unknown. Lemma 4.11 formalizes this for the n-player scenario, and we refer this lemma using $n = 2$.

Therefore we can conclude that the probability of changing the output of EGO from or to 1 is negligible. ∎

5.3.3 The Zero-Knowledge of EGO

To show that the EGO protocol is zero-knowledge, we prove that an adversarial player does not learn more than the output of the protocol. We therefore look at the case when one of P_1 and P_2 is arbitrary, and the other is honest. To make the analysis more clear, we let P_1 denote the adversarial player, and P_2 denote the honest player.

Let $\mathcal{I}_E(m_1, m_2, k)$ be the set of all possible inputs, both public and private, for messages m_1 and m_2, and fixed security parameter k. We do in fact assume worst case message m_1, m_2. The set is defined by all possible choices of discrete-log instances $(p, q, g)_k$ and values h giving public and private keys to the two players. We let the security parameter k and the messages, m_1 and m_2, be implicit from now on. Let in_E be a randomly and uniformly chosen element from $\mathcal{I}_E$.

Define V_{EGO} as the random variable taking on the adversarial view, i.e., what the adversarial player P_1 sees during an execution of EGO together with P_2, running on input in_E. Then D_{EGO} is the probability distribution of V_{EGO}, where the probability is taken over ran_1, aux_1, ran_2, and in_E.

We show that no polynomially bounded probabilistic adversary can gain any information by running EGO, apart from the result and what may be implied by it. To this end we construct the simulator $\mathsf{S}_{\mathsf{EGO}}$ with random string $ran_{\mathsf{S}_{\mathsf{EGO}}}$, outputting an adversarial view $V_{\mathsf{S}_{\mathsf{EGO}}}$ with probability distribution $D_{\mathsf{S}_{\mathsf{EGO}}}$ indistinguishable from the probability distribution of the adversarial view of a real execution of EGO. The probability is taken over ran_1, aux_1, $ran_{\mathsf{S}_{\mathsf{EGO}}}$, and in_E. More formally stated:

Lemma 5.4. There exists a polynomially bounded probabilistic oracle Turing Machine $\mathsf{S}_{\mathsf{EGO}}$ such that for all adversarial behaviors of P_1 and $\forall m_1, m_2$, $\mathsf{S}_{\mathsf{EGO}}$ outputs a random variable $V_{\mathsf{S}_{\mathsf{EGO}}}$ with probability distribution $D_{\mathsf{S}_{\mathsf{EGO}}}$ such that $D_{\mathsf{S}_{\mathsf{EGO}}} \overset{c}{\approx} D_{\mathsf{EGO}}$, using the non-uniform DDH assumption and in the random oracle model.

Proof. In this proof we present the simulator $\mathsf{S}_{\mathsf{EGO}}$, we show that $\mathsf{S}_{\mathsf{EGO}}$ runs in expected polynomial time and show that the probability distribution of $\mathsf{S}_{\mathsf{EGO}}$'s output, i.e., $D_{\mathsf{S}_{\mathsf{EGO}}}$ is computationally indistinguishable from D_{EGO}. Let 2^T be the time it takes to compute $\log_g(h)$, as we will need this in the proof.

The simulator $\mathsf{S}_{\mathsf{EGO}}$*:*

$\mathsf{S}_{\mathsf{EGO}}$ is a probabilistic Turing Machine with an internal random string $ran_{\mathsf{S}_{\mathsf{EGO}}}$ and with access to the random oracle $\mathcal{O}$ and to P_1's private input and auxiliary input aux_1. $\mathsf{S}_{\mathsf{EGO}}$ runs the algorithm described in Figure 5.2 and Figure 5.3 using P_1 as a black-box and the simulators SIM_ϵ, SIM_φ, SIM_O, SIM_Q and SIM_β. Using the same P_1 in both EGO and $\mathsf{S}_{\mathsf{EGO}}$ means that P_1's output has the same probability distribution, as long as the probability distributions of the views created by EGO and $\mathsf{S}_{\mathsf{EGO}}$ computationally indistinguishable.

Input to $\mathsf{S}_{\mathsf{EGO}}$ is $in_E \in_R \mathcal{I}_E$, res_{EGO}, and the private key of P_1. Note that we solely look at such outcomes of res_{EGO} that have a non-negligible probability of occurring.

Each of the simulators of the sub-protocols outputs a view which is incorporated in $V_{\mathsf{S}_{\mathsf{EGO}}}$. The random variable $res_{\mathsf{S}_{\mathsf{EGO}}}$ contains the result of the protocol computed by running $\mathsf{S}_{\mathsf{EGO}}$ on random input in_E.

The running time of $\mathsf{S}_{\mathsf{EGO}}$*:*

On input $res_{\mathsf{EGO}} \in \{E, FAIL\}$, $\mathsf{S}_{\mathsf{EGO}}$'s algorithm contains no internal loops and one run through the algorithm is in time polynomial in k. $\mathsf{S}_{\mathsf{EGO}}$ runs the simulators SIM_O and SIM_β which can with negligible probability output ERR (Lemma 4.4). If we recall that $\mathsf{S}_{\mathsf{EGO}}$ only runs on such res_{EGO} that have

Public input: $(p,q,g)_k$, h y_1, y_2 (γ_1,δ_1), (γ_2,δ_2), $res_{\mathsf{EGO}} \in \{E, FAIL\}$
Private input: Private keys corresponding to the public keys.

Let $(\epsilon,\varphi) = (\gamma_1/\gamma_2, \delta_1/\delta_2)$.

1. For $res_{\mathsf{EGO}} \in \{E, FAIL\}$ do the following, otherwise run the algorithm in Figure 5.3:
   (a) Select $z_2, r_2 \in_R \mathbb{Z}_q$, and compute $C_2 = g^{z_2}h^{r_2}$.
   (b) Compute $\epsilon_2 = \epsilon^{z_2}$ and run EPSI. If EPSI outputs REJ then let $res_{\mathsf{S_{EGO}}} = F_\epsilon$, otherwise continue.
   (c) Compute $\varphi_2 = \varphi^{z_2}$ and run PHI. If PHI outputs REJ then let $res_{\mathsf{S_{EGO}}} = F_\varphi$, otherwise continue.
   (d) Compute $G_{21} = y_2^{R_1}$ and run $\mathsf{SIM}_{O,2}$. If it outputs ERR let $res_{\mathsf{S_{EGO}}} = ERR$.
   (e) Compute $Q_2 = G_{12}^{z_2}$ and run EGO − Q. If EGO − Q outputs REJ then let $res_{\mathsf{S_{EGO}}} = F_Q$, otherwise continue.
   (f) Compute $T_2 = G_{21}^{z_2}$ and run EGO − T. If EGO − T outputs REJ then let $res_{\mathsf{S_{EGO}}} = F_T$, otherwise continue.
   (g) Compute $\beta_2 = \alpha T/\beta_1 Q$ and run $\mathsf{SIM}_{\beta,2}$. If it outputs ERR let $res_{\mathsf{S_{EGO}}} = ERR$.
   (h) If $res_{\mathsf{EGO}} = E$ and $res_{\mathsf{S_{EGO}}} = E$ then $V_{\mathsf{S_{EGO}}} = \{res_{\mathsf{S_{EGO}}}, C_i, \epsilon_i, \varphi_i, G_{21}, G_{12}, Q_i, T_i, \beta_i : i \in \{1,2\}\}$ together with the views from EPSI, PHI, EGO − O, $\mathsf{SIM}_{O,2}$, EGO − Q, EGO − T, BETA and $\mathsf{SIM}_{\beta,2}$.
   If $res_{\mathsf{EGO}} = FAIL$, let $V_{\mathsf{S_{EGO}}}$ contain all information up to the point of failure in EGO, if $\mathsf{S_{EGO}}$ ran that far. Otherwise restart $\mathsf{S_{EGO}}$.

Figure 5.2. The $\mathsf{S_{EGO}}$ protocol on input $res_{\mathsf{EGO}} = \{E, FAIL\}$

Public input: $(p,q,g)_k$, h y_1, y_2 (γ_1,δ_1), (γ_2,δ_2), $res_{\mathsf{EGO}} = N$
Private input: Private keys corresponding to the public keys.

Let $(\epsilon,\varphi) = (\gamma_1/\gamma_2, \delta_1/\delta_2)$.

1. For $res_{\mathsf{EGO}} = N$ do the following, otherwise run the algorithm in Figure 5.2:
   (a) Select $s \in_R \mathbb{Z}_q$ and compute g^s and y_2^s.
   (b) Select $r_2, z_2 \in_R \mathbb{Z}_q$ and compute $C_2 = g^{z_2}h^{r_2}$.
   (c) Compute $\epsilon_2 = \epsilon^{z_2}$ and run EPSI.
   (d) Rewind EPSI to get a new oracle question and a new challenge. Compute z_1 and r_1 from P_1's two outputs.
   (e) Let $\varphi_2 = g^s/\varphi_1$, where φ_1 is computed using the value computed for z_1 from the previous step. Run $\mathsf{SIM}_{\varphi,2}$. If the simulator produces an *ERR* message let $res_{\mathsf{S_{EGO}}} = ERR$.
   (f) Compute $\beta = \varphi_1\varphi_2$ and $\alpha = \epsilon_1\epsilon_2$. If $\beta \neq g^s$ let $res_{\mathsf{S_{EGO}}} = BAD$.
   (g) Compute $G_{21} = y_2^{R_1}$ and run $\mathsf{SIM}_{O,2}$. If it outputs *ERR* let $res_{\mathsf{S_{EGO}}} = ERR$.
   (h) Compute $Q_2 = (g^{z_2R_1}/\varphi_2)^{x_1}$ and run the $\mathsf{SIM}_{Q,2}$ protocol. If it outputs *ERR* let $res_{\mathsf{S_{EGO}}} = ERR$.
   (i) Compute $T_2 = G_{21}^{z_2}$ and run $\mathsf{EGO-T}$.
   (j) Compute $\beta_2 = y_2^s$. Run $\mathsf{SIM}_{\beta,2}$ and let $res_{\mathsf{S_{EGO}}} = ERR$ if $\mathsf{SIM}_{\beta,2}$ outputs *ERR*.
2. If $res_{\mathsf{EGO}} = N$ and $res_{\mathsf{S_{EGO}}} \in \{E,N\}$ then $V_{\mathsf{S_{EGO}}} = \{res_{\mathsf{S_{EGO}}}, C_i, \epsilon_i, \varphi_i, G_{21}, G_{12}, Q_i, T_i, \beta_i : i \in \{1,2\}\}$ together with the views from EPSI, PHI, $\mathsf{EGO-O}$, $\mathsf{SIM}_{O,2}$, $\mathsf{EGO-Q}$, $\mathsf{SIM}_{Q,2}$, $\mathsf{EGO-T}$, BETA, and $\mathsf{SIM}_{\beta,2}$.

   If $res_{\mathsf{S_{EGO}}} \in \{FAIL, ERR\}$, restart $\mathsf{S_{EGO}}$'s algorithm. If $res_{\mathsf{S_{EGO}}} = BAD$, let $V_{\mathsf{S_{EGO}}} = BAD$.

Figure 5.3. The $\mathsf{S_{EGO}}$ protocol on input $res_{\mathsf{EGO}} = N$

a non-negligible probability of occurring, then it is clear that $\mathsf{S}_{\mathsf{EGO}}$ will be restarted at most a polynomial number of times.

For input $res_{\mathsf{EGO}} = N$, $\mathsf{S}_{\mathsf{EGO}}$'s running time is more complex. Each step runs in polynomial time, but the algorithm contains a loop. We need to look closer at the expected number of iterations in this loop. Here follow some helpful definitions. Let RW be the number of times $\mathsf{S}_{\mathsf{EGO}}$ rewinds EPSI. Let $RW_{(z_1,r_1)}$ be the number of rewindings needed to extract (z_1, r_1). Let the expected number of rewindings be $\mathsf{E}[RW]$. Let $PASS_1$ be the event that player P_1 passes EPSI. We know that the probability is non-negligible that P_1 passes all proofs. This follows from the fact that the probability is non-negligible that $res_{\mathsf{EGO}} = N$. This means that $\mathsf{Pr}[PASS_1] > 1/poly(k)$.

Let V be the random variable taking on the value of the random bits used in $\mathsf{S}_{\mathsf{EGO}}$ up to the oracle challenge in EPSI. Let $\mathcal{V}_j = \{v | 2^{-(j+1)} < \mathsf{Pr}[PASS|v] \leq 2^{-j}\}$, be the subset of all bit strings resulting in $PASS$. The average number of tries P_1 needs to pass EPSI using $v \in \mathcal{V}_j$ is $< 2^{j+1}$. If the number of iterations is larger than 2^T then $\mathsf{S}_{\mathsf{EGO}}$ computes (z_1, r_1). This gives the expected number of iterations taken over all j's is

$$\begin{aligned}
\mathsf{E}[\# \text{ of iterations}] &= \sum_{j=1}^{\infty} \mathsf{E}[\#tries\ with\ v \in \mathcal{V}_j] \mathsf{Pr}[PASS_1 \cap V \in \mathcal{V}_j] \\
&\leq \sum_{i=1}^{T} (2^{j+1} 2^{-j}) + \sum_{j=T+1}^{\infty} 2^{T+1} 2^{-j} \\
&= \sum_{1}^{T} 2 + \frac{2^{T+1} 2^{-(T+1)}}{1 - 2^{-1}} \\
&= 2T + 2. \qquad (5.3)
\end{aligned}$$

The expected number of iterations needed to extract (z_1, r_1) for player P_1 is $\mathsf{E}[RW]$. After this, the adversarial player P_1 may fail in one of the remaining proofs, but we know that P_1 runs to the end with non-negligible probability. Consequently $\mathsf{S}_{\mathsf{EGO}}$ will be restarted at most a polynomial number of times on the average.

The conclusion of this reasoning is that $\mathsf{S}_{\mathsf{EGO}}$ runs in expected polynomial time on input $res_{\mathsf{EGO}} = N$ and in polynomial time on input $res_{\mathsf{EGO}} \in E$ and $FAIL$.

The probability distribution of $\mathsf{S}_{\mathsf{EGO}}$*'s output versus* $\mathsf{D}_{\mathsf{EGO}}$*:*

The next step in the proof is to show that $D_{\mathsf{S}_{\mathsf{EGO}}}$ is computationally indistinguishable from D_{EGO} $\forall m_1, m_2 \in G_q$. Let us take a closer look at D_{EGO}. This distribution can be divided into the following disjoint parts:

$$\begin{aligned} D_{\mathsf{EGO}} = \quad & D_{\mathsf{EGO}}(res_{\mathsf{EGO}} = E)||D_{\mathsf{EGO}}(res_{\mathsf{EGO}} = N)|| \\ & D_{\mathsf{EGO}}(res_{\mathsf{EGO}} = FAIL). \end{aligned} \tag{5.4}$$

A similar partitioning can be done to $D_{\mathsf{S_{EGO}}}$:

$$\begin{aligned} D_{\mathsf{S_{EGO}}} = \quad & D_{\mathsf{S_{EGO}}}(res_{\mathsf{S_{EGO}}} = E)||D_{\mathsf{S_{EGO}}}(res_{\mathsf{S_{EGO}}} = N)|| \\ & D_{\mathsf{S_{EGO}}}(res_{\mathsf{S_{EGO}}} = FAIL)||D_{\mathsf{S_{EGO}}}(res_{\mathsf{S_{EGO}}} = ERR). \end{aligned} \tag{5.5}$$

Let res be the variable taking on one of the values E, N or $FAIL$. To show that $D_{\mathsf{S_{EGO}}} \overset{c}{\approx} D_{\mathsf{EGO}}$ it is enough to show $D_{\mathsf{S_{EGO}}}(res) \overset{c}{\approx} D_{\mathsf{EGO}}(res)$ for each m_1 and m_2, and for each value of res where $\mathsf{Pr}[res_{\mathsf{EGO}} = res]$ is non-negligible. Recall that the values m_1 and m_2 are implicit. If $D_{\mathsf{S_{EGO}}}$ and D_{EGO} are distinguishable then $D_{\mathsf{S_{EGO}}}(res)$ and $D_{\mathsf{EGO}}(res)$ must be distinguishable for at least one value of res.

– $D_{\mathsf{S_{EGO}}}(FAIL) = D_{\mathsf{EGO}}(FAIL)$: When $res_{\mathsf{EGO}} = FAIL$ then $\mathsf{S_{EGO}}$'s algorithm for player P_1 is identical to EGO up to and including the place of failure. None of these steps involve any private input, they only involve public input and randomly chosen and uniformly distributed elements. The probability distribution of $\mathsf{S_{EGO}}$'s output is therefore identical to the probability distribution of the output from EGO.

– $D_{\mathsf{S_{EGO}}}(E) \overset{s}{\approx} D_{\mathsf{EGO}}(E)$: We show computational indistinguishability separately for $m_1 = m_2$ and for $m_1 \neq m_2$.

  Case $m_1 = m_2$*:* In the case when $res_{\mathsf{EGO}} = E$ it is possible to show $D_{\mathsf{S_{EGO}}}(E) \overset{s}{\approx} D_{\mathsf{EGO}}(E)$, and statistical indistinguishability implies computational indistinguishability.

  The only difference between a real run of EGO and $\mathsf{S_{EGO}}$ run on input $in(m_1 = m_2)$ and $res_{\mathsf{EGO}} = E$, lies in the difference between the adversarial view of $\mathsf{EGO} - \mathsf{O}$ and BETA and the view SIM_O and SIM_β output. Lemma 4.5 states that these views are statistically indistinguishable from that of runs of the real sub-protocol. Therefore we can conclude that $D_{\mathsf{S_{EGO}}}(E) \overset{s}{\approx} D_{\mathsf{EGO}}(E)$, which implies $D_{\mathsf{S_{EGO}}}(E) \overset{c}{\approx} D_{\mathsf{EGO}}(E)$.

  Case $m_1 \neq m_2$*:* Recall that we show computational indistinguishability only for such res_{EGO} that occur with non-negligible probability. The proof of soundness of EGO shows that $\mathsf{Pr}[res_{\mathsf{EGO}} = E | m_1 \neq m_2]$ is negligible.

– $D_{\mathsf{S_{EGO}}}(N) \overset{c}{\approx} D_{\mathsf{EGO}}(N)$: Once again we will show computational indistinguishability separately for $m_1 = m_2$ and for $m_1 \neq m_2$.

Case $m_1 = m_2$: As shown in the proof of soundness of EGO $\Pr[res_{\mathsf{EGO}} = N]$ is negligible. Therefore this case will never be simulated.

Case $m_1 \neq m_2$: To show $D_{\mathsf{S_{EGO}}}(N) \overset{c}{\approx} D_{\mathsf{EGO}}(N)$ for $m_1 \neq m_2$, we introduce an additional Turing Machine $\mathsf{SI_{EGO}}$, which defines the intermediate probability distributions $D_{\mathsf{SI_{EGO}}}$ and $D'_{\mathsf{SI_{EGO}}}$. These distributions are computationally indistinguishable. $\mathsf{SI_{EGO}}$ is described in detail in Section 5.3.3. Lemma 5.5, 5.6 and 5.8 give that $D_{\mathsf{EGO}}(N)$ is computationally indistinguishable from $D_{\mathsf{S_{EGO}}}(N)$ for all $m_1 \neq m_2$.

Conclusion: For each result of EGO that occurs with non-negligible probability, the distributions $D_{\mathsf{S_{EGO}}}(res)$ and $D_{\mathsf{EGO}}(res)$ are computationally indistinguishable. We can therefore conclude that $D_{\mathsf{S_{EGO}}} \overset{c}{\approx} D_{\mathsf{EGO}}$.

■

The Intermediate Simulator $\mathsf{SI_{EGO}}$

This section contains a detailed description of the simulator $\mathsf{SI_{EGO}}$ used in the proof of Lemma 5.4. $\mathsf{SI_{EGO}}$ is a probabilistic Turing Machine with an internal, uniformly distributed, random string $ran_{\mathsf{SI_{EGO}}} \in \{0,1\}^*$, with access to the random oracle $\mathcal{O}$ and with black-box access to P_1.

The only purpose of $\mathsf{SI_{EGO}}$ is to help prove that $D_{\mathsf{S_{EGO}}} \overset{c}{\approx} D_{\mathsf{EGO}}$. The idea is to build a chain of four probability distributions, including $D_{\mathsf{S_{EGO}}}$ and D_{EGO}, where each neighboring pair is indistinguishable.

The simulator runs the algorithm described in Figure 5.4. The input to this algorithm consists of a random discrete-log instance $(p, q, g)_k$, a random generator h of $\mathcal{G}_q$, values $\hat{m}_1 = \log_g(m_1)$ and $\hat{m}_2 = \log_g(m_2)$, and a triple (g^a, g^b, g^c) either in Δ_{DDH} or in Δ_{rand}.

For all random choices of $(p, q, g)_k$, h, a, b, m_1, m_2, and $ran_{\mathsf{SI_{EGO}}}$, the first steps of $\mathsf{SI_{EGO}}$'s algorithm compute a random instance $in_E \in \mathcal{I}_E$ giving $\mathsf{SI_{EGO}}$ access to both private keys. The random instance in_E contains random encryptions of m_1 and m_2 which we obtain by setting $g^a = \varphi$ and $g^b = g^{z_2}$. $\mathsf{SI_{EGO}}$ computes δ_1 by selecting $R_1 \in_R \mathbb{Z}_q$ and calculating $\delta_1 = g^{R_1}$. The value for ϵ_1 follows directly. Since $\varphi = \delta_1/\delta_2$, we set $\varphi = g^a$ here. This gives $\delta_2 = \delta_1/g^a$ and from this follows that $\gamma_2 = g^{m_2} g^{x_2 R_1}/(g^a)^{x_2}$. The remaining rounds in $\mathsf{SI_{EGO}}$'s algorithm compute the remaining values. In particular we let $\mathsf{SI_{EGO}}$ use $g^c = \varphi_2$. Since our simulator has both private keys the other values follow easily. For values ϵ_2, φ_2, Q_2, and T_2 $\mathsf{SI_{EGO}}$ can not run their proof of knowledge, but must run the corresponding simulator. This simulator may output ERR, though from lemmas 4.4 and 4.9 we see that the probability is negligible that $\mathsf{SIM_{DLOG,2}}$ or $\mathsf{SIM_{COM,2}}$ registers ERR, i.e., that any of the simulators $\mathsf{SI_{EGO}}$ uses registers ERR. The variable $res_{\mathsf{SI_{EGO}}}$ takes on the result computed by $\mathsf{SI_{EGO}}$.

The output is the random variable $V_{\mathsf{SI_{EGO}}}$. We call the probability distribution of $V_{\mathsf{SI_{EGO}}}$ $D_{\mathsf{SI_{EGO}}}$ when $(g^a, g^b, g^c) \in \Delta_{rand}$. Here the probability is taken over $ran_{\mathsf{SI_{EGO}}}$,

ran_1, aux_1, $(p,q,g)_k$, h, $\hat{m}_1$, $\hat{m}_2$, a, b, and $\mathcal{O}$. Let $D'_{\mathsf{SI}_{\mathsf{EGO}}}$ be the probability distribution of the random variable $V_{\mathsf{SI}_{\mathsf{EGO}}}$, when $(g^a, g^b, g^c) \in \Delta_{DDH}$, where the probability is taken over $ran_{\mathsf{SI}_{\mathsf{EGO}}}$, ran_1, aux_1, $(p,q,g)_k$, h, $\hat{m}_1$, $\hat{m}_2$, (a,b,c), and $\mathcal{O}$.

With $(g^a, g^b, g^c) \in \Delta_{DDH}$ the probability distribution of $\mathsf{SI}_{\mathsf{EGO}}$'s output is indistinguishable from $D_{\mathsf{S}_{\mathsf{EGO}}}$ (Lemma 5.8) and when $(g^a, g^b, g^c) \in \Delta_{rand}$ $D'_{\mathsf{SI}_{\mathsf{EGO}}}$ is indistinguishable from D_{EGO} (Lemma 5.5). Also by Lemma 5.6 $D'_{\mathsf{SI}_{\mathsf{EGO}}}$ is computationally indistinguishable from $D_{\mathsf{SI}_{\mathsf{EGO}}}$, under the non-uniform DDH assumption. Since $\mathsf{SI}_{\mathsf{EGO}}$ gets extra information ($\hat{m}_1$, $\hat{m}_2$) we must use the non-uniform DDH assumption to show this indistinguishability. If we consider random m_1, m_2 or m_1, m_2 produced such that their discrete logarithms are easy to compute, then we have uniformity.

Lemma 5.5. $D_{\mathsf{EGO}} \overset{s}{\approx} D'_{\mathsf{SI}_{\mathsf{EGO}}}$

Proof. EGO computes $\epsilon_2 = \epsilon^{z_2} = (mg^{x_1R_1 - x_2R_2})^{z_2}$, $\varphi_2 = \varphi^{z_2}$ and $Q_2 = g^{x_1R_2z_2}$, whereas in $\mathsf{SI}_{\mathsf{EGO}}$ these we have $\epsilon_2 = (g^b)^{\hat{m}}(g^b)^{R_1(x_1-x_2)}(g^c)^{x_2}$, $\varphi_2 = g^c$ and $Q_2 = ((g^b)^{R_1}/g^c)^{x_1}$. When $(g^a, g^b, g^c) \in \Delta_{DDH}$ then $\log_g(\varphi_n) = \log_g(\varphi)\log_\varphi(\varphi_n)$ and $\log_g(\epsilon_n) = \log_g(\epsilon)\log_\epsilon(\epsilon_n)$ for values both in $\mathsf{SI}_{\mathsf{EGO}}$ and in EGO. The value T_2 from $\mathsf{SI}_{\mathsf{EGO}}$ does not depend on g^c. In both $\mathsf{SI}_{\mathsf{EGO}}$ and EGO the values are the same, only g^b is used in place of g^{z_2} in $\mathsf{SI}_{\mathsf{EGO}}$ as is the case for C_2. All values are thus correlated to each other in the same way in both EGO and $\mathsf{SI}_{\mathsf{EGO}}$.

Since $\mathsf{SI}_{\mathsf{EGO}}$ does not know the exponents for many of the values it computes it must run a simulator in place of a real execution of a proof of knowledge to produce output with the correct probability distribution. The output from these simulators is exactly the same as that of real executions , as long as there are no oracle collisions,i.e., none of the simulators for the sub-protocols output ERR. This happens with negligible probability. By Lemma 4.5 the sub-protocols and their simulators produce output with statistically indistinguishable probability distributions. From this we conclude that $D'_{\mathsf{SI}_{\mathsf{EGO}}} \overset{s}{\approx} D_{\mathsf{EGO}}$. ∎

Lemma 5.6. The distribution $D'_{\mathsf{SI}_{\mathsf{EGO}}}$ is computationally indistinguishable from $D_{\mathsf{SI}_{\mathsf{EGO}}}$ under the non-uniform decisional Diffie-Hellman assumption.

Proof. We start by assuming the opposite, that $D'_{\mathsf{SI}_{\mathsf{EGO}}}$ is computationally distinguishable from $D_{\mathsf{SI}_{\mathsf{EGO}}}$. This implies the existence of a probabilistic polynomial time Turing Machine Q that has the ability to distinguish between $D'_{\mathsf{SI}_{\mathsf{EGO}}}$ and $D_{\mathsf{SI}_{\mathsf{EGO}}}$ in polynomial time. If $\mathsf{SI}_{\mathsf{EGO}}$ runs in polynomial time, we can construct a machine M that solves the non-uniform DDH-assumption in polynomial time by using Q as a subroutine. As input M gets a random discrete-log instance $(p,q,g)_k$, a triple (g^a, g^b, g^c) either in Δ_{DDH} or in Δ_{rand}, $\hat{m}_1 = \log_g(m_1)$ and $\hat{m}_2 = \log_g(m_2)$, access to $\mathsf{SI}_{\mathsf{EGO}}$ and to the adversarial player.

M chooses a random generator h and runs $\mathsf{SI}_{\mathsf{EGO}}$ on this input. Using $\mathsf{SI}_{\mathsf{EGO}}$'s output, Q can break the non-uniform DDH-assumption efficiently. On this basis, we state that $D'_{\mathsf{SI}_{\mathsf{EGO}}}$ is computationally indistinguishable from $D_{\mathsf{SI}_{\mathsf{EGO}}}$. ∎

Lemma 5.7. $\mathsf{SI}_{\mathsf{EGO}}$ terminates in time polynomial in k

Proof. $\mathsf{SI}_{\mathsf{EGO}}$'s algorithm contains no loops and no individual step is more than polynomial in k. Consequently one run through the algorithm runs in polynomial time. We will restart at most a polynomial number of times, since $\mathsf{SI}_{\mathsf{EGO}}$ is used with such $\mathcal{A}$, m_1 and m_2 that give res_{EGO} as result with non-negligible probability. ∎

Lemma 5.8. $D_{\mathsf{SI}_{\mathsf{EGO}}} \overset{s}{\approx} D_{\mathsf{S}_{\mathsf{EGO}}}$

Proof. $D_{\mathsf{SI}_{\mathsf{EGO}}}$ is the probability distribution when $(g^a, g^b, g^c) \in \Delta_{rand}$. Recall that $\mathsf{SI}_{\mathsf{EGO}}$ and $\mathsf{S}_{\mathsf{EGO}}$ are only used when $res_{\mathsf{EGO}} = N$, which means that $m_1 \neq m_2$. The first steps of $\mathsf{SI}_{\mathsf{EGO}}$ compute a random input in_E. These steps are independent of the value g^c and g^b. The random value g^a is used for the difference between the blinding factors R_1 and R_2. This difference is also random in $\mathsf{S}_{\mathsf{EGO}}$. Therefore, $in_E(m)$ as computed by $\mathsf{SI}_{\mathsf{EGO}}$ has same probability distribution as the input to $\mathsf{S}_{\mathsf{EGO}}$.

The simulator $\mathsf{SI}_{\mathsf{EGO}}$ uses g^b in place of g^{z_2}, giving $\epsilon_2 = (g^b)^{\hat{m}}(g^b)^{R_1(x_1-x_2)}(g^c)^{x_2}$. Both b and z_2 are random. The value z_2 is used to compute ϵ_2 in $\mathsf{S}_{\mathsf{EGO}}$. It is also used to compute φ_2, whereas in $\mathsf{SI}_{\mathsf{EGO}}$ g^c is used. In both simulators the value ϵ_2 is independent from φ_2. Also Q_2 depends in φ_2 in the same way in both simulators, which is also true for the remaining values.

The only difference between $\mathsf{S}_{\mathsf{EGO}}$ and $\mathsf{SI}_{\mathsf{EGO}}$ lies in $\mathsf{S}_{\mathsf{EGO}}$ rewinding the EPSI protocol. We can look at this rewinding as two parallel executions of $\mathsf{S}_{\mathsf{EGO}}$ instead, using the same random strings and inputs, but different oracles. All information published up to the oracle's question is the same in both executions. Therefore rewinding does not affect the probability distribution of $V_{\mathsf{S}_{\mathsf{EGO}}}$, providing that P_1 passes EPSI with non-negligible probability. ∎

5.4 Comments to the EGO Protocol

So far we have assumed that the two players are fair. That is, a player does not terminate the protocol prematurely after it gets enough information to compute the answer, but before the other player has enough information to easily compute the answer.

If we assume that a player is not fair, then the critical point in the EGO protocol is when publishing β_i, round 7 in the EGO protocol. Before this round the messages being published are all about computing (α, β), the message we intend to ego-decrypt. They do not give any player an advantage in decrypting and getting hold of the answer before the other player.

If we still want the resulting protocol to be as fair as possible, i.e., a corrupt player should not get a bigger advantage than a single bit at any point, we need to replace round 7. Here follows an outline of what to do.

Instead of publishing β_i, player P_i publishes some commitment to β_i. We want the commitment scheme to be such that a commitment can be used to prove that β_i

is computed as the protocol indicates, i.e., that $\log_\beta \beta_i = \log_g(y_i)$, without revealing neither x_i nor β_i. We also want the commitment scheme to be such that you can open the commitment slowly, giving the opposing player an advantage of at most a one bit.

On an even more informal note, we can replace the BETA proof with a circuit which takes the bits of β_i as input. If we let player P_i commit to each bit in β_i separately (using a commitment key for each bit), then for each bit it is possible to build a circuit that computes the committed bit, using the commitment key as input. Consequently, it is possible to construct one circuit computing the whole problem. Player P_i can prove that he has commitment keys for which the circuit produces accepting output. Since there exists a zero-knowledge proof for every $\mathcal{NP}$ statement, and the commitment scheme and the proof of correctness of β_i can be reduced to an $\mathcal{NP}$ statement, then we can state that there exists a commitment scheme that fulfills our requirements. The players can now open their bits, bit by bit.

We see here that there is a tradeoff between fairness and efficiency. Making the protocol fair, forces us to use to the general solution of multi-party computation which is inefficient.

$\mathsf{SI}_{\mathsf{EGO}}$'s input: $(p,q,g)_k$, h, (g^a, g^b, g^c), $\hat{m}_1$, $\hat{m}_2$.
$\mathsf{SI}_{\mathsf{EGO}}$'s output: $V_{\mathsf{SI}_{\mathsf{EGO}}}$

1. Compute public and private keys for P_1 and P_2. This gives $\mathsf{SI}_{\mathsf{EGO}}$ access to both private keys, x_1 and x_2.
2. Select $R_1 \in_R \mathbb{Z}_q$ and compute and encryption of $m_1 = g^{\hat{m}_1}$ as
$$(\gamma_1, \delta_1) = (g^{\hat{m}_1} y_1^{R_1}, g^{R_1}).$$
Compute an encryption of m_2 as
$$(\gamma_2, \delta_2) = (g^{\hat{m}_2} g^{x_2 R_1}/(g^a)^{x_2}, g^{R_1 - a}).$$
Let
$$(\epsilon, \varphi) = (\gamma_1/\gamma_2, \delta_1/\delta_2) = (g^{\hat{m}} g^{x_1 R_1 - x_2 R_1 + x_2 a}, g^a),$$
where $\hat{m} = \hat{m}_1 - \hat{m}_2$.
3. Select $r_2 \in_R \mathbb{Z}_q$ and compute C_2.
4. Let $\epsilon_2 = (g^b)^{\hat{m}} (g^b)^{R_1(x_1 - x_2)} (g^c)^{x_2}$ and run $\mathsf{SIM}_{\epsilon,2}$. If $\mathsf{SIM}_{\epsilon,2}$ outputs ERR let $res_{\mathsf{SI}_{\mathsf{EGO}}} = ERR$.
5. Let $\varphi_2 = g^c$ and run $\mathsf{SIM}_{\varphi,2}$. If $\mathsf{SIM}_{\varphi,2}$ outputs ERR let $res_{\mathsf{SI}_{\mathsf{EGO}}} = ERR$.
6. Let $G_{21} = g^{x_2 R_1}$ and run $\mathsf{EGO-O}$.
7. Let $Q_2 = ((g^b)^{R_1}/g^c)^{x_1}$ and run $\mathsf{SIM}_{Q,2}$. If $\mathsf{SIM}_{Q,2}$ outputs ERR let $res_{\mathsf{SI}_{\mathsf{EGO}}} = ERR$.
8. Let $T_2 = (g^b)^{x_2 R_1}$ and $\mathsf{SIM}_{T,2}$. Let $res_{\mathsf{SI}_{\mathsf{EGO}}} = ERR$ if $\mathsf{SIM}_{T,2}$ outputs ERR.
9. Compute $\beta = \varphi_1 \varphi_2$ and $\beta_2 = \beta^{x_2}$ and run BETA.

Let $V_{\mathsf{SI}_{\mathsf{EGO}}} = res_{\mathsf{SI}_{\mathsf{EGO}}}, \{y_i, x_i, (\gamma_i, \delta_i), C_i, \epsilon_i, \varphi_i, G_{ij}, Q_i, T_i, \beta_i : i,j = \{1,2\}, i \neq j\}$, and all views from $\mathsf{EPSI}, \mathsf{PHI}$, $\mathsf{EGO-O}, \mathsf{EGO-Q}$, $\mathsf{EGO-T}, \mathsf{BETA}$, and their simulators.

Figure 5.4. The $\mathsf{SI}_{\mathsf{EGO}}$ protocol

Chapter 6

Multi-party Plain-text Equality Testing

In the previous chapter we looked at the case where two players publicly compare the underlying plain-texts of given encrypted inputs using their own public keys for encryption. We extend the protocol and let n players compare the underlying plain-texts of two encryptions. In this chapter we assume that both messages are encrypted using the same public key, and that each of the n players holds a share of the private key. We call this comparison protocol the *plain-text equality test*, or PET. We present all the details of this protocol together with proofs of its correctness. We also study the sequential composition of PET, which we call SEQ, and show its correctness.

The problem of testing plain-text equality was outlined in [JJ00], in the context of electronic auctions. The protocol in [JJ00] lacks in details, and no formal proofs of correctness is given.

Our contribution is to fill in the necessary details of the PET-protocol and to prove that the resulting protocol is complete, sound, and computational zero-knowledge in the presence of a static adversary corrupting up to a minority of the players. This is done in the random oracle model and assuming the hardness of the decisional Diffie-Hellman problem. We also prove that sequential composition of PET using the same set of encryption keys in each execution of PET is complete, sound, and zero-knowledge.

Independent of our work, [MSJ02] presented a more complicated version of the PET-protocol as a building block for threshold key-exchange. Their paper gives a proof sketch of computational zero-knowledge of the threshold key-exchange protocol. That proof sketch and our proof are similar in structure, our proof being essentially simpler. Briefly, proving computational zero-knowledge of a protocol means showing computational indistinguishability between two probability distributions: one produced by a real run of the protocol and one produced by simulating the protocol [GMR89]. In some cases, it is necessary to introduce intermediate probability distributions, to form a sequence with the above distributions at its ends. One proves then that each pair of neighboring distributions is computationally indistinguishable.

In our proof, we use two such intermediate distributions, while [MSJ02] need six. Their longer and more laborious proof is partially due to their protocol being more intricate.

6.1 The Current Setting

The PET protocol is run by n players $P_1, \ldots, P_n$. These are modeled as described in Chapter 3. When started, the players get the security parameter k as input. The number n is polynomially related to k. The players communicate via a bulletin board.

An adversary $\mathcal{A}$ corrupts up to $t = \lfloor n/2 \rfloor - 1$ players, which always gives a majority of honest players. For simplicity, we assume that $\mathcal{A}$ corrupts t players in the rest of the chapter. The proofs can easily be extended to cases where $\mathcal{A}$ corrupts $< t$ players. As the protocol is symmetric, we can assume that the corrupt players are $P_1, \ldots, P_t$, without loss of generality. Consequently, players $P_{t+1}, \ldots, P_n$ are assumed honest. Recall from Chapter 3 that A denotes the set of players corrupted by $\mathcal{A}$, and H denotes the set of honest players.

6.2 The Plain-text Equality Test Protocol

The main input to the PET protocol are two ElGamal encryptions (γ_1, δ_1) and (γ_2, δ_2) of two unknown plain-text messages m_1 and m_2, using the public key y. The plain-text equality test takes these encryptions and determines whether $m_1 = m_2$ or not. The protocol reveals the result to each of the n players and to all observers.

Before being able to run PET, the participating players must jointly produce a random discrete-log instance, a random generator h and ElGamal keys and key-shares on input k and t.

The homomorphic property (2.5) gives that $(\gamma_1/\gamma_2, \delta_1/\delta_2)$ is an ElGamal encryption of m_1/m_2. From now on $(\epsilon, \varphi) = (\gamma_1/\gamma_2, \delta_1/\delta_2)$ denotes the encryption of $m = m_1/m_2$. We have the fact that $m = 1$ if and only if $m_1 = m_2$. Therefore, decrypting (ϵ, φ) reveals if we have equality or not. Unfortunately, when $m_1 \neq m_2$, m reveals more information than only non-equality.

To hide this information, we proceed as in Chapter 5.1. That is we compute and decrypt (ϵ^z, φ^z) instead, where $z \in \mathbb{Z}_q$ is random and unknown to the players. For $z \in \mathbb{Z}_q$, the pair $(\alpha, \beta) = (\epsilon^z, \varphi^z)$, is an encryption of m^z. We observe that given an element $x \in \mathcal{G}_q$ and $z \in_R \mathbb{Z}_q$, we have $x^z \in_R \mathcal{G}_q$. This means that if z is randomly chosen then $m^z \in_R \mathcal{G}_q$ for $m \neq 1$. If $m = 1$ then $m^z = 1$. As long as z is unknown, decrypting (α, β) gives information only about equality.

Public input: $(p,q,g)_k$, h, t, ElGamal encryption (ϵ,φ), public ElGamal key y, public shares y_i of a distributed ElGamal key, for all players $P_1,\dots,P_n$.
P_i**'s private input:** x_i such that $g^{x_i}=y_i$.
PET's output: E if $\frac{\alpha}{\beta^x}=1$, N if $\frac{\alpha}{\beta^x}\neq 1$, or one of F_ϵ, F_φ, F_β, VIO

The protocol for player P_i runs as follows:

Round 1.

- Choose randomly z_i, $r_i \in \mathbb{Z}_q$.
- Let $C_i \leftarrow g^{z_i}h^{r_i}$, the Pedersen commitment to z_i.
- Publish C_i.

Round 2.

- Let $\epsilon_i \leftarrow \epsilon^{z_i}$.
- Publish ϵ_i.
- Run EPSI. If EPSI outputs ACC then continue, otherwise if EPSI outputs REJ then publish F_ϵ and terminate.

Round 3.

- Let $\varphi_i \leftarrow \varphi^{z_i}$.
- Publish φ_i.
- Run PHI. If PHI outputs ACC then continue, otherwise if PHI outputs REJ then publish F_φ and terminate.

Round 4.

- Compute $\alpha=\prod_{i=1}^n \epsilon_i$ and $\beta=\prod_{i=1}^n \varphi_i$.
- Let $\beta_i \leftarrow \beta^{x_i}$.
- Publish β_i.
- Run BETA. If BETA outputs $F3$ such that $|F3| > n-(t+1)$ then publish F_β and terminate, otherwise continue.

Round 5. Compute β^x using $t+1$ values β_j where $P_j \notin F3$, and compute α/β^x

Figure 6.1. The PET protocol

The main issue of the protocol is to compute (ϵ^z, φ^z) without any player or coalition of players knowing z or being able to affect its randomness. Pedersen commitments and the sub-protocols from Chapter 4 are used in this process and during the decryption of (α, β). We call these sub-protocols EPSI, PHI and BETA. They are described in Section 6.3. Figure 6.1 shows the PET protocol in detail. Recall that the execution is synchronous, i.e., players are expected to await the publishing of all the players' results before proceeding with the publishing in the next step. If a player observes another player violating this convention, then he publishes VIO and the execution of PET is terminated prematurely. The result of PET, to which we refer as res_{PET}, can be one of the following:

E	when (α, β) decrypts to 1.
N	when (α, β) does not decrypt to 1.
F_ϵ	when the output of EPSI is REJ.
F_φ	when the output of PHI is REJ.
F_β	when the output of BETA $\geq n - (t+1)$.
VIO	when a player publishes prematurely.

Let $FAIL$ be the event when one of F_ϵ, F_φ, F_β, VIO occurs. If a player $FAIL$S to either publish a result in time or to pass a proof he is disqualified. If $FAIL$ occurs in the BETA protocol we simply exclude the erroneous player and proceed. As long as at most t players fail, the result is recoverable. It is not possible to bias it by excluding values. In all other cases we abort the execution and restart the protocol without the disqualified players.

6.3 The Sub-Protocols used in PET

The PET protocol executes the two sub-protocols described in Chapter 4 on different inputs. We choose to give these executions different names to simplify the presentation and the proofs of the PET protocol. We call these executions EPSI, PHI and BETA.

The EPSI protocol is an execution of the COM protocol described in Chapter 4.2. It deals with making sure that players use the values to which they committed without explicitly opening the commitments. The EPSI protocol is discussed in Section 6.3.1.

The two other sub-protocols, PHI and BETA are executions of the DLOG protocol from Chapter 4.1. These ensure that published data are correct tuples in Δ_{DDH}.

Their details are specified in sections 6.3.2 for PHI, and 6.3.3 for BETA.

The DLOG and COM protocols are presented in Chapter 4 together with proofs of completeness, soundness, and zero-knowledge. From these proofs it follows that EPSI, PHI and BETA are complete, sound, and zero-knowledge.

6.3.1 The EPSI Protocol

The EPSI protocol is an execution of the the COM protocol on public input $(p, q, g)_k$, $h, \epsilon, \epsilon_i, C_i$ and private input $z_i = \log_{\epsilon_i}$, and r_i such that $C_i = g^{z_i} h^{r_i}$. The output of COM is the set of all complaints. When this set is empty we say that EPSI outputs ACC, otherwise it outputs REJ.

To show that COM is zero-knowledge we built the simulator $\mathsf{SIM}_{\mathsf{COM}}$ (Figure 4.6) simulating the output of the honest players, and the simulator $\mathsf{SIM}_{\mathsf{COM,i}}$ (Figure 4.5) simulating the output of a single honest player. When we simulate honest players running the EPSI protocol we refer to the simulators as SIM_ϵ and $\mathsf{SIM}_{\epsilon,i}$.

6.3.2 The PHI Protocol

The PHI protocol is an execution of the DLOG protocol run on public input $(p, q, g)_k$, $(\epsilon, \epsilon_i, \varphi, \varphi_i)$ and private input $z_i = \log_\epsilon \epsilon_i$. The goal of PHI is for each player P_i to convince all other participating players that both ϵ_i and φ_i were constructed using the same exponent. More specifically, given $\epsilon_i = \epsilon^{z_i}$ and $\varphi_i = \varphi^{z_i'}$ each P_i shows that $z_i' = z_i$ without revealing z_i and z_i'. The output of DLOG consists of the set M_{DLOG}, which is the set of complaints filed by the verifying players. We say that PHI outputs ACC if $M_{\mathsf{DLOG}} = \emptyset$, otherwise PHI outputs REJ.

To show that DLOG is zero-knowledge we built the simulator $\mathsf{SIM}_{\mathsf{DLOG}}$ (Figure 4.3) simulating the output of the honest players, and the simulator $\mathsf{SIM}_{\mathsf{DLOG,i}}$ (Figure 4.2) simulating the output of a single honest player. When we simulate honest players running the PHI protocol we refer to the simulators as SIM_φ and $\mathsf{SIM}_{\varphi,i}$.

6.3.3 The BETA Protocol

The BETA protocol is an execution of the DLOG protocol run on public input $(p, q, g)_k$, (g, y_i, β, β_i), and private input $x_i = \log_g(y_i)$, we refer to the protocol as BETA. The goal of BETA is for each player to convince all other participants that the value β_i is correctly constructed. This means showing $\log_\beta \beta_i = \log_g y_i$ without revealing x_i. The output of DLOG consists of the set M_{DLOG} of complaints filed by the verifying players. We define the output of BETA to be the set of players who have one or more complaints published against them.

To show that DLOG is zero-knowledge we built the simulator $\mathsf{SIM}_{\mathsf{DLOG}}$ (Figure 4.3) simulating the output of the honest players, and the simulator $\mathsf{SIM}_{\mathsf{DLOG,i}}$ (Figure 4.2) simulating the output of a single honest player. When we simulate honest players running the BETA protocol we refer to the simulators as SIM_β and $\mathsf{SIM}_{\beta,i}$.

6.4 Completeness, Soundness and Zero-knowledge for PET

Theorem 6.1. The PET protocol is complete, sound and zero-knowledge in the presence of a static adversary corrupting up to a minority, under the decisional Diffie-Hellman assumption and in the random oracle model.

Proof. To simplify presentation we divide this proof into three lemmas which we treat separately. These lemmas are: Lemma 6.2 showing completeness, Lemma 6.3 showing soundness, and Lemma 6.4 showing zero-knowledge. ∎

6.4.1 The Completeness of PET

The PET protocol is complete if players learn whether $m = 1$ or $m \neq 1$ with overwhelming probability, under the condition that all players follow the protocol.

Lemma 6.2. If all players follow the protocol, and if $m = 1$, then $res_{\mathsf{PET}} = E$ with probability 1. If $m \neq 1$, then $res_{\mathsf{PET}} = N$ with probability $(q-1)/q$.

Proof. The result of PET is based on the result of the division α/β^x, where (α, β) is an ElGamal encryption of m^z, for $z \in_R \mathbb{Z}_q$.

> If $m = 1$, then for any number $z \in \mathbb{Z}_q$, $m^z = 1 \mod p$. Therefore res_{PET} is always E when $m = 1$.
>
> If $m \neq 1$, then $res_{\mathsf{PET}} = N$ for all z except $z = 0$, when p is prime. With $z = 0$, $m^z = 1 \mod p$, and $res_{\mathsf{PET}} = E$. Since z is random and uniformly distributed in $\mathbb{Z}_q$ the probability that $z = 0$ is $1/q$.

If all players are honest, i.e., follow the protocol, then res_{PET} never equals $FAIL$. ∎

6.4.2 The Soundness of PET

The PET protocol is sound if a group of players cannot change the output of PET and get away with it, except with negligible probability. Changing the outcome of PET means changing the result of α/β^x from 1 or to 1.

Lemma 6.3. The probability is negligible that an adversary $\mathcal{A}$ efficiently changes the output of PET from E to N or from N to E.

Proof. We assume that EPSI and PHI output ACC and that BETA outputs an empty set. This means that we have $\log_\epsilon(\epsilon_i) = \log_\varphi(\varphi_i)$ and $\log_g(y_i) = \log_\beta(\beta_i)$ for all $i = 1, \ldots, n$, except with negligible probability.

The PET protocol gives us $\alpha = \prod_i \epsilon_i = (mg^{Rx})^z$ and $\beta = \prod_i \varphi_i = (g^R)^z$, where $z = \sum_i z_i$ and R is the random number used for encryption. The output of PET depends on the result of $\alpha/\beta^x = m^z$.

Case $m = 1$ *(*$res_{\mathsf{PET}} = E$ *to be changed to* $res_{\mathsf{PET}} = N$*):*
We have the fact that $m^z = 1$ for all a if $m = 1$. This implies that PET's output can not be changed from E to N.

Case $m \neq 1$ *(*$res_{\mathsf{PET}} = N$ *to be changed to* $res_{\mathsf{PET}} = E$*):*
We make the observation that if $m \neq 1$ then $m^z = 1 \mod p$ iff $z = 0$. In other words, to change the PET protocol's result the z_i-values must be adapted so that $\sum_i z_i = 0$.

We proved in Lemma 4.12 that if an adversary can change its z_i values and pass the EPSI protocol with non-negligible probability, we can use this adversary to compute $\log_g(h)$ in expected polynomial time. This is assumed impossible, which implies that the probability an adversary succeeds in adapting his z_j values is negligible, as long as $\log_g(h)$ remains unknown.

■

6.4.3 The Zero-knowledge of PET

Let $\mathcal{I}(m, t, k, n)$ be the set of all possible inputs, both public and private, for a fixed message m, fixed polynomial of degree t, fixed security parameter k and fixed number of players n. We can in fact assume a worst case m, since we are using the non-uniform decisional Diffie-Hellman assumption. The set is defined by all possible choices of discrete-log instance $(p, q, g)_k$ and random generator h, all possible polynomials of degree t with coefficients whose encoding is of length $\leq k$, giving the public and private key and private key-shares to n players. The values t, n, and k are implicit from now on. Let $in(m)$ be a randomly and uniformly chosen element from $\mathcal{I}(m)$. Let $res_{\mathsf{PET}}(m)$ be the variable defined by the result of a run of PET when run on input $in(m) \in_R \mathcal{I}(m)$.

For an adversarial Turing Machine $\mathcal{A}$, let $V_{\mathsf{PET}}(m)$ be the random variable defined by the view of $\mathcal{A}$ during an execution of PET with H on input $in(m)$. We call this the adversarial view of a real run. Let $D_{\mathsf{PET}}(m)$ be the probability distribution of $V_{\mathsf{PET}}(m)$, where the probability is taken over ran_A, aux_A, ran_H, and $in(m)$.

We show that no polynomially bounded probabilistic adversary $\mathcal{A}$ can gain any information running PET, except the result. To this end we construct a simulator $\mathsf{S}_{\mathsf{PET}}$ (Figure 6.2 and 6.3), outputting a view indistinguishable from the adversarial view. More formally stated:

Lemma 6.4. There exists a polynomially bounded probabilistic oracle Turing Machine $\mathsf{S}_{\mathsf{PET}}$ such that $\forall \mathcal{A}$ corrupting t or less players, and $\forall m$, $\mathsf{S}_{\mathsf{PET}}$ outputs a random variable $V_{\mathsf{S}_{\mathsf{PET}}}(m)$ with probability distribution $D_{\mathsf{S}_{\mathsf{PET}}}(m)$ such that $D_{\mathsf{S}_{\mathsf{PET}}}(m) \overset{c}{\approx} D_{\mathsf{PET}}(m)$, under the non-uniform DDH assumption and in the random oracle model.

Proof. In this proof we present the simulator $\mathsf{S}_{\mathsf{PET}}$, show that $\mathsf{S}_{\mathsf{PET}}$ runs in expected polynomial time and show that the probability distribution of $\mathsf{S}_{\mathsf{PET}}$'s output is

Public input: $(p, q, g)_k, h, \epsilon, \varphi, y, \{y_i : i = 1, \ldots, n\}, res_{\mathsf{PET}}(m) \in \{E, F_\epsilon, F_\varphi\}$
$\mathsf{S}_{\mathsf{PET}}$**'s private input:** $\{x_i : i = 1, \ldots, t\}$
Public output: $V_{\mathsf{S}_{\mathsf{PET}}}(m)$

1. When $res_{\mathsf{PET}} \in \{E, F_\epsilon, F_\varphi\}$ run the following algorithm, otherwise run the algorithm presented in Figure 6.3.
   (a) For each player $P_i \in H$ select $z_i, r_i \in_R Z_q$, compute $C_i = g^{z_i} h^{r_i}$.
   (b) For each player $P_i \in H$ compute $\epsilon_i = \epsilon^{z_i}$ and run EPSI. If EPSI outputs ACC then continue, otherwise if EPSI outputs REJ then let $res_{\mathsf{S}_{\mathsf{PET}}}(m) = F_\epsilon$.
   (c) For each player $P_i \in H$ compute $\varphi_i = \varphi^{z_i}$ and run PHI. If PHI outputs ACC then continue, otherwise if PHI outputs REJ then let $res_{\mathsf{S}_{\mathsf{PET}}}(m) = F_\varphi$.
   (d) For each player $P_i \in H$ compute and publish $\beta_i \forall P_i \in H$ and run SIM_β. If SIM_β's output contains ERR then let $res_{\mathsf{S}_{\mathsf{PET}}}(m) = ERR$. If $\alpha/\beta^x = 1$ then $res_{\mathsf{S}_{\mathsf{PET}}}(m) = E$.
   (e) If $res_{\mathsf{PET}}(m) = E$ and $res_{\mathsf{S}_{\mathsf{PET}}}(m) = E$ then $V_{\mathsf{S}_{\mathsf{PET}}}(m) = res_{\mathsf{S}_{\mathsf{PET}}}(m)$, y, (ϵ, φ), $\{y_i,\ C_i,\ \epsilon_i,\ \varphi_i,\ \beta_i\ :\ 1 \leq i \leq n\}$ together with the outputs from SIM_ϵ, SIM_φ, and SIM_β. If $res_{\mathsf{PET}}(m) = F_\epsilon$ and $res_{\mathsf{S}_{\mathsf{PET}}}(m) \in \{F_\epsilon, F_\varphi, E\}$ then let $V_{\mathsf{S}_{\mathsf{PET}}}(m) =$ contain all information up to the point of failure in PET. If $res_{\mathsf{PET}}(m) = F_\varphi$ and $res_{\mathsf{S}_{\mathsf{PET}}}(m) \in \{F_\varphi, E\}$ then output all information up to the point of failure in PET. Otherwise restart $\mathsf{S}_{\mathsf{PET}}$ (when $res_{\mathsf{PET}}(m) = F_\varphi$ and $res_{\mathsf{S}_{\mathsf{PET}}}(m) = F_\epsilon$ or when $res_{\mathsf{S}_{\mathsf{PET}}}(m) = ERR$).

Figure 6.2. The protocol for $\mathsf{S}_{\mathsf{PET}}$ on input $res_{\mathsf{PET}} \in \{E, F_\epsilon, F_\varphi\}$

Public input: $(p, q, g)_k$, h, ϵ, φ, y, $\{y_i : i = 1, \ldots, n\}$, $res_{\mathsf{PET}}(m) = N$
$\mathsf{S}_{\mathsf{PET}}$**'s private input:** $\{x_i : i = 1, \ldots, t\}$
Public output: $V_{\mathsf{S}_{\mathsf{PET}}}(m)$

1. When $res_{\mathsf{PET}} = N$ run the following algorithm, otherwise run the algorithm presented in Figure 6.2.
   (a) For each player $P_i \in H$ select $s \in_R Z_q$ and compute $g^s, y^s, \{y_i^s : i = 1 \ldots n\}$.
   (b) For each player $P_i \in H$ select $z_i, r_i \in_R Z_q$, compute $C_i = g^{z_i} h^{r_i}$.
   (c) For each player $P_i \in H$ compute $\epsilon_i = \epsilon^{z_i}$ and run EPSI. If EPSI outputs REJ then let $res_{\mathsf{S}_{\mathsf{PET}}}(m) = F_\epsilon$.
   (d) Rewind EPSI to get a new question to the oracle and a new oracle challenge and compute (z_j, r_j) for all $P_j \in A$.
   (e) For each $P_i \in H \backslash P_n$ let $\varphi_i = \varphi^{z_i}$. For P_n let $\varphi_n = g^s / \prod_{i=1}^{n-1} \varphi_i$, where for all $P_j \in A$ use $\varphi_j = \varphi^{z_j}$ using the computed z_j.
   (f) For players $P_i \in H \backslash P_n$ run PHI. For player P_n run $\mathsf{SIM}_{\varphi,n}$. If PHI outputs REJ then let $res_{\mathsf{S}_{\mathsf{PET}}}(m) = F_\varphi$. If $\mathsf{SIM}_{\varphi,n}$ produces an ERR-message then let $res_{\mathsf{S}_{\mathsf{PET}}}(m) = ERR$. If $\prod_{j=0}^{n} \varphi_i \neq g^s$ let $res_{\mathsf{S}_{\mathsf{PET}}}(m) = BAD$. Otherwise compute α and β as defined in PET. For all $P_i \in H$ $\beta_i = y_i^s$, and $\beta^x = y^s$.
   (g) Run SIM_β. If SIM_β's output contains ERR then let $res_{\mathsf{S}_{\mathsf{PET}}}(m) = ERR$. If $\alpha/\beta^x \neq 1$ then $res_{\mathsf{S}_{\mathsf{PET}}}(m) = N$, if $\alpha/\beta^x = 1$ then $res_{\mathsf{S}_{\mathsf{PET}}}(m) = E$.
   (h) If $res_{\mathsf{PET}}(m) = N$ and $res_{\mathsf{S}_{\mathsf{PET}}}(m) = E, N$ then $V_{\mathsf{S}_{\mathsf{PET}}} = res_{\mathsf{S}_{\mathsf{PET}}}(m)$, y, (ϵ, φ), $\{y_i, C_i, \epsilon_i, \varphi_i, \beta_i : 1 \leq i \leq n\}$ together the outputs from SIM_ϵ, SIM_φ and SIM_β. If $res_{\mathsf{S}_{\mathsf{PET}}}(m) \in \{F_\epsilon, F_\varphi, ERR\}$ then restart $\mathsf{S}_{\mathsf{PET}}$. If $res_{\mathsf{S}_{\mathsf{PET}}}(m) = BAD$ then let $V_{\mathsf{S}_{\mathsf{PET}}} = BAD$.

Figure 6.3. The protocol for $\mathsf{S}_{\mathsf{PET}}$ on input $res_{\mathsf{PET}} = N$

computationally indistinguishable from $D_{\mathsf{PET}}(m)$. Let 2^T be the time it takes to compute $\log_g(h)$.

The simulator $\mathsf{S}_{\mathsf{PET}}$*:*
$\mathsf{S}_{\mathsf{PET}}$ is a probabilistic Turing Machine with an internal random string ran_S and with access to the random oracle $\mathcal{O}$ and to $\mathcal{A}$'s private input and auxiliary input $aux_{\mathcal{A}}$. $\mathsf{S}_{\mathsf{PET}}$ runs the algorithm described in Figure 6.2 and Figure 6.3, using $\mathcal{A}$ as a black-box and the simulators SIM_ϵ, SIM_φ, and SIM_β. Using the same $\mathcal{A}$ in both PET and $\mathsf{S}_{\mathsf{PET}}$ means that $\mathcal{A}$'s output has the same probability distribution, as long as the probability distributions of the views created by PET and $\mathsf{S}_{\mathsf{PET}}$ are computationally indistinguishable.

Input to $\mathsf{S}_{\mathsf{PET}}$ is $in(m) \in_R \mathcal{I}(m)$, $res_{\mathsf{PET}}(m)$, and the shares of the private key for the corrupt players. Note that we solely look at such $res_{\mathsf{PET}}(m)$ that have a non-negligible probability of occurring. We write res_{PET} when m is implicit.

Let $V_{\mathsf{S}_{\mathsf{PET}}}(m)$ be the random variable denoting the output of $\mathsf{S}_{\mathsf{PET}}$ when run on random input $in(m)$ and let $D_{\mathsf{S}_{\mathsf{PET}}}(m)$ be the probability distribution of $V_{\mathsf{S}_{\mathsf{PET}}}(m)$, where the probability is taken over $ran_{\mathcal{A}}$, $aux_{\mathcal{A}}$, $ran_{\mathsf{S}_{\mathsf{PET}}}$, and $in(m)$. The simulators of the sub-protocols each output a view which is incorporated in $V_{\mathsf{S}_{\mathsf{PET}}}(m)$.

Depending on the value of res_{PET}, $\mathsf{S}_{\mathsf{PET}}$ runs one of two different algorithms. With $res_{\mathsf{PET}} \in \{E, F_\epsilon, F_\varphi\}$, $\mathsf{S}_{\mathsf{PET}}$ runs the algorithm in Figure 6.2. The simulation is straightforward, the only obstacle being in step 1d: the computation of β_i for all $P_i \in H$, without having access to their private shares x_i. To compute $\beta_{t+1}, \ldots, \beta_n$ we use (2.9) with $\beta_1, \ldots, \beta_t$, (which are computed using the private shares of $P_1, \ldots P_t$ to which $\mathsf{S}_{\mathsf{PET}}$ has access) and using the fact that $\beta^x = \alpha$.

On input $res_{\mathsf{PET}} = N$ $\mathsf{S}_{\mathsf{PET}}$'s algorithm is more complex. This algorithm is shown in Figure 6.3. The main idea is for $\mathsf{S}_{\mathsf{PET}}$ to adapt its output so that $\sum z_i$ is known to $\mathsf{S}_{\mathsf{PET}}$. To do so $\mathsf{S}_{\mathsf{PET}}$ must rewind EPSI to find (z_j, r_j) for all $P_j \in A$. If a player fails during the first execution of EPSI then $\mathsf{S}_{\mathsf{PET}}$ outputs F_ϵ and restarts. When we restart $\mathsf{S}_{\mathsf{PET}}$ we do so with a new random string $ran_{\mathsf{S}_{\mathsf{PET}}}$. When all players have passed EPSI the rewinding process starts. This process continues until all corrupt players have passed EPSI twice. If, after 2^T tries a player P_j still has not passed EPSI twice then we let $\mathsf{S}_{\mathsf{PET}}$ compute (z_j, r_j) from ϵ_j and C_j. The probability distribution of the output (the view) is not affected by this rewinding since we never discard a run of $\mathsf{S}_{\mathsf{PET}}$ because of the rewinding. Also $\mathsf{S}_{\mathsf{PET}}$ may fail to terminate if some $P_j \in A$ publishes $\varphi_j \neq \varphi^{z_j}$ and succeeds with PHI (which happens with negligible probability, Lemma 4.2). In such a case $\mathsf{S}_{\mathsf{PET}}$ registers BAD and terminates.

The running time of $\mathsf{S}_{\mathsf{PET}}$*:*
On input $res_{\mathsf{PET}}(m) \in \{E, F_\epsilon, F_\varphi\}$ $\mathsf{S}_{\mathsf{PET}}$'s algorithm contains no internal

loops. One round in the algorithm runs in time polynomial in k. From Lemma 4.4 we have $res_{\mathsf{S_{PET}}}(m) = ERR$ by the BETA protocol with negligible probability. If we recall that $\mathsf{S_{PET}}$ only runs on such $res_{\mathsf{PET}}(m)$ that have a non-negligible probability of occurring, then it is clear that $\mathsf{S_{PET}}$ is restarted at most a polynomial number of times.

When run on input $res_{\mathsf{PET}}(m) = N$ $\mathsf{S_{PET}}$'s running time is more complex. Each step runs in polynomial time, but the algorithm contains a loop. We need to look closer at the expected number of iterations in this loop. Here follow some helpful definitions. Let RW be the number of times $\mathsf{S_{PET}}$ rewinds EPSI. Let $RW_{(z_i,r_i)}$ be the number of rewindings needed to extract (z_i, r_i). The expected number of rewindings $\mathsf{E}[RW] = \max\{\mathsf{E}[RW_{(z_i,r_i)}] : \forall P_i \in A\}$. Let $PASS_i$ be the event that player P_i passes EPSI. We know that the probability is non-negligible that $\mathcal{A}$ passes all proofs. This follows from the fact that the probability is non-negligible that $res_{\mathsf{PET}}(m) = N$. This means that $\mathsf{Pr}[PASS_i] > 1/poly(k)$.

Let V be the random variable defined by the value of the random bits used in $\mathsf{S_{PET}}$ up to the oracle challenge in EPSI, and let v be an instance of V. Let $\mathcal{V}_j = \{v | 2^{-(j+1)} < \mathsf{Pr}[PASS|v] \leq 2^{-j}\}$, be the subset of all bit strings resulting in $PASS$. The average number of tries P_i needs to pass EPSI using $v \in \mathcal{V}_j$ is $< 2^{j+1}$. If the number of iterations is larger than 2^T then $\mathsf{S_{PET}}$ computes (z_i, r_i). This gives the expected number of iterations taken over all j's is

$$\begin{aligned}
\mathsf{E}[\#\text{ of iterations}] &= \sum_{j=1}^{\infty} \mathsf{E}[\#tries\ with\ v \in \mathcal{V}_j]\mathsf{Pr}[PASS_i \cap V \in \mathcal{V}_j] \\
&\leq \sum_{j=1}^{T}(2^{j+1}2^{-j}) + \sum_{j=T+1}^{\infty} 2^{T+1}2^{-j} \\
&= \sum_{1}^{T} 2 + \frac{2^{T+1}2^{-(T+1)}}{1-2^{-1}} \\
&= 2T + 2. \qquad (6.1)
\end{aligned}$$

The expected number of iterations needed to extract (z_i, r_i) for all corrupt players P_i is $\mathsf{E}[RW] = \max\{\mathsf{E}[RW_{(z_i,r_i)}] : \forall P_i \in A\} < 2T + 2$. After this a player in $\mathcal{A}$ may fail in one of the remaining proofs, but we know that $\mathcal{A}$ runs to the end with non-negligible probability. Consequently $\mathsf{S_{PET}}$ is restarted at most a polynomial number of times on the average.

The conclusion of this reasoning is that $\mathsf{S_{PET}}$ runs in expected polynomial time on input N and in polynomial time on input E, F_ϵ, and F_φ.

The probability distributions:
The next step in the proof is to show that $D_{\mathsf{S_{PET}}}(m)$ is computationally indistinguishable from $D_{\mathsf{PET}}(m)$ $\forall m \in G_q$. Let us take a closer look at $D_{\mathsf{PET}}(m)$. This distribution can be divided into the following disjoint parts:

$$\begin{aligned} D_{\mathsf{PET}}(m) = \quad & D_{\mathsf{PET}}(m, res_{\mathsf{PET}}(m) = E) || D_{\mathsf{PET}}(m, res_{\mathsf{PET}}(m) = N) || \\ & D_{\mathsf{PET}}(m, res_{\mathsf{PET}}(m) = F_\epsilon) || D_{\mathsf{PET}}(m, res_{\mathsf{PET}}(m) = F_\varphi). \end{aligned} \tag{6.2}$$

A similar partitioning can be done to $D_{\mathsf{S_{PET}}}(m)$:

$$\begin{aligned} D_{\mathsf{S_{PET}}}(m) = \quad & D_{\mathsf{S_{PET}}}(m, res_{\mathsf{S_{PET}}}(m) = E) || D_{\mathsf{S_{PET}}}(m, res_{\mathsf{S_{PET}}}(m) = N) || \\ & D_{\mathsf{S_{PET}}}(m, res_{\mathsf{S_{PET}}}(m) = F_\epsilon) || D_{\mathsf{S_{PET}}}(m, res_{\mathsf{S_{PET}}}(m) = F_\varphi) || \\ & D_{\mathsf{S_{PET}}}(m, res_{\mathsf{S_{PET}}}(m) = ERR). \end{aligned} \tag{6.3}$$

Let res be the variable defined by one of the values E, N, F_ϵ, or F_φ. To show that $D_{\mathsf{S_{PET}}}(m) \overset{c}{\approx} D_{\mathsf{PET}}(m)$ it is enough to show $D_{\mathsf{S_{PET}}}(m, res) \overset{c}{\approx} D_{\mathsf{PET}}(m, res)$ for each m and for each value of res where $\mathsf{Pr}[res_{\mathsf{PET}}(m) = res]$ is non-negligible. If $D_{\mathsf{S_{PET}}}(m)$ and $D_{\mathsf{PET}}(m)$ are distinguishable then $D_{\mathsf{S_{PET}}}(m, res)$ and $D_{\mathsf{PET}}(m, res)$ must be distinguishable for at least one value of res.

– $D_{\mathsf{S_{PET}}}(m, F_\epsilon) = D_{\mathsf{PET}}(m, F_\epsilon)$ and $D_{\mathsf{S_{PET}}}(m, F_\varphi) = D_{\mathsf{PET}}(m, F_\varphi)$: Recall that F_ϵ means that one or more players fails in EPSI and that F_φ means that one or more players fails in PHI. When $res_{\mathsf{PET}}(m) = F_\epsilon$ ($res_{\mathsf{PET}}(m) = F_\varphi$) then $\mathsf{S_{PET}}$'s algorithm for player P_i is identical to PET up to and including EPSI (PHI). None of these steps involve any private input, they only involve public input and randomly chosen and uniformly distributed elements. The probability distribution of $\mathsf{S_{PET}}$'s output is therefore identical to the probability distribution of the output from PET.

– $D_{\mathsf{S_{PET}}}(m, E) \overset{s}{\approx} D_H(m, E)$: We show computational indistinguishability separately for $m = 1$ and for $m \neq 1$.

Case $m = 1$: When $res_{\mathsf{PET}}(1) = E$ is in fact $D_{\mathsf{S_{PET}}}(1, E)$ is statistically indistinguishable from $D_{\mathsf{PET}}(1, E)$. Statistical indistinguishability implies computational indistinguishability.

The only difference between a real run of PET and $\mathsf{S_{PET}}$ run on input $in(1)$ and $res_{\mathsf{PET}}(1) = E$, lies in the difference between the adversarial view of BETA and the view SIM_β outputs. Lemma 4.5 states that the view computed in SIM_β is statistically indistinguishable from that of BETA. Therefore we can conclude that $D_{\mathsf{S_{PET}}}(1, E) \overset{s}{\approx} D_{\mathsf{PET}}(1, E)$.

Case $m \neq 1$: Recall that we show computational indistinguishability only for such $res_{\mathsf{PET}}(m)$ that occur with non-negligible probability. The proof of soundness of PET shows that $\Pr[res_{\mathsf{PET}}(m) = E | m \neq 1]$ is negligible.

– $D_{\mathsf{S_{PET}}}(m, N) \overset{c}{\approx} D_{\mathsf{PET}}(m, N)$: Once again we show computational indistinguishability separately for $m = 1$ and for $m \neq 1$.

Case $m = 1$: As shown in the proof of soundness of PET $\Pr[res_{\mathsf{PET}}(1) = N]$ is negligible. Therefore this case is never simulated.

Case $m \neq 1$: To show $D_{\mathsf{S_{PET}}}(m, N) \overset{c}{\approx} D_H(m, N)$ for $m \neq 1$, we introduce an additional Turing Machine $\mathsf{SI_{PET}}$, which defines the intermediate probability distributions $D_{\mathsf{SI_{PET}}}(m)$ and $D'_{\mathsf{SI_{PET}}}(m)$. $\mathsf{SI_{PET}}$ is described in detail in Section 6.4.4. Lemma 6.5, 6.7 and 6.8 show that

$$D_{\mathsf{PET}}(m, N) \overset{s}{\approx} D'_{\mathsf{SI_{PET}}}(m, N) \overset{c}{\approx} D'_{\mathsf{SI_{PET}}}(m, N) \overset{c}{\approx} D_{\mathsf{S_{PET}}}(m, N)$$

for $m \neq 1$. Thus, $D_{\mathsf{PET}}(m, N)$ is computationally indistinguishable from $D_{\mathsf{S_{PET}}}(m, N)$ for all $m \neq 1$.

Conclusion: For each result of PET that occurs with non-negligible probability, the distributions $D_{\mathsf{S_{PET}}}(m, res)$ and $D_{\mathsf{PET}}(m, res)$ are computationally indistinguishable. We can therefore conclude that $D_{\mathsf{S_{PET}}}(m) \overset{c}{\approx} D_{\mathsf{PET}}(m)$. ∎

6.4.4 The Intermediate Simulator $\mathsf{SI_{PET}}$

This section is a detailed description of the probabilistic Turing Machine $\mathsf{SI_{PET}}$ used in the proof of Lemma 6.4. $\mathsf{SI_{PET}}$ has an internal uniformly distributed random string $ran_{\mathsf{SI_{PET}}} \in \{0, 1\}^*$, access to the random oracle $\mathcal{O}$, and black-box access to $\mathcal{A}$ and to $\mathcal{A}$'s auxiliary input aux_A, and runs the algorithm described in Figure 6.4. Input to $\mathsf{SI_{PET}}$ is a random discrete-log instance $(p, q, g)_k$, a random generator h, a value $\hat{m}$ such that $m = g^{\hat{m}}$, and a triple (g^a, g^b, g^c) in Δ_{DDH} or in Δ_{rand}.

Our goal is to show $D_{\mathsf{S_{PET}}}(m, N) \overset{c}{\approx} D_{\mathsf{PET}}(m, N)$. For this purpose we build a chain of four probability distributions, including $D_{\mathsf{S_{PET}}}(m, N)$ and $D_{\mathsf{PET}}(m, N)$, where each neighboring pair is indistinguishable. The simulator $\mathsf{SI_{PET}}$ produces the remaining two probability distributions. $\mathsf{SI_{PET}}$ uses $\varphi = g^a$, $g^{z_n} = g^b$ and $\varphi_n = g^c$. The simulator outputs the random variable $V_{\mathsf{SI_{PET}}}(m)$.

When $c = ab$, we call the probability distribution of $\mathsf{SI_{PET}}$'s output $D'_{\mathsf{SI_{PET}}}(m, N)$. The probability is taken over $ran_{\mathcal{A}}$, $ran_{\mathsf{SI_{PET}}}$, $(p, q, g)_k$, h, $aux_{\mathcal{A}}$, $\hat{m}$, and (a, b, c). The distribution $D'_{\mathsf{SI_{PET}}}(m, N)$ is indistinguishable from $D_{\mathsf{PET}}(m, N)$ by Lemma 6.5.

When $c \in_R \mathbb{Z}_q$, we call the probability distribution of $V_{\mathsf{SI_{PET}}}(m, N)$ $D_{\mathsf{SI_{PET}}}(m, N)$. $D_{\mathsf{SI_{PET}}}(m, N)$ is indistinguishable from both $D'_{\mathsf{SI_{PET}}}(m, N)$ and from $D_{\mathsf{S_{PET}}}(m, N)$ by lemmas 6.7 and 6.8.

Since $\mathsf{SI_{PET}}$ gets extra information ($\hat{m}$) we must use the non-uniform DDH assumption to show computational indistinguishability. If we consider random m

Public input: $(p, q, g)_k$, h
$\mathsf{SI}_{\mathsf{PET}}$**'s private input:** $(g^a, g^b, g^c), \hat{m}$
Public output: $V_{\mathsf{SI}_{\mathsf{PET}}}(m)$

1. Compute a set of keys and key-shares by running the simulator for the joint key-sharing protocol used. It gives $\mathsf{SI}_{\mathsf{PET}}$ access to $x_1, \ldots, x_n$ and it outputs $y, y_1, \ldots, y_n$.
2. Compute an encryption (ϵ, φ) of m by letting $\varphi = g^a$ and $\epsilon = m\varphi^x = m(g^a)^x$.
3. For players $P_i \in H \backslash P_n$ run PET. To produce player P_n's output do the following:
   (a) Select $r_n \in_R \mathbb{Z}_q$. Let $C_n = g^b h^{r_n}$.
   (b) Let $\epsilon_n = (g^b)^{\hat{m}}(g^c)^x$, and run $\mathsf{SIM}_{\epsilon,n}$. If $\mathsf{SIM}_{\epsilon,n}$'s output contains ERR then let $res_{\mathsf{SI}_{\mathsf{PET}}}(m) = ERR$.
   (c) Let $\varphi_n = g^c$ and run $\mathsf{SIM}_{\varphi,n}$. If $\mathsf{SIM}_{\varphi,n}$'s output contains ERR then let $res_{\mathsf{SI}_{\mathsf{PET}}}(m) = ERR$.
   (d) Run round 4 and 5 as described in PET. $\mathsf{SI}_{\mathsf{PET}}$ has the private shares of all players and can compute β_i for all honest players and run BETA.
4. If F_ϵ or F_φ is registered then restart the protocol.
   Otherwise let $V_{\mathsf{SI}_{\mathsf{PET}}}(m) = res_{\mathsf{SI}_{\mathsf{PET}}}$, y, (ϵ, φ), $\{y_i, C_i, \epsilon_i, \varphi_i, \beta_i : 1 \leq i \leq n\}$, and the views from the sub-protocols and their simulators.

Figure 6.4. The protocol for $\mathsf{SI}_{\mathsf{PET}}$

or m produced such that $\hat{m}$ is easy to compute, then we have uniformity. For all random choices of $(p, q, g)_k$, h, $a, b \in \mathbb{Z}_q$, and $ran_{\mathsf{SI}_{\mathsf{PET}}} \in \{0, 1\}^*$, the first steps of $\mathsf{SI}_{\mathsf{PET}}$ compute a random instance $in(m) \in \mathcal{I}(m)$.

From lemmas 4.9 and 4.4 we see that the probability is negligible that $\mathsf{SIM}_{\epsilon,n}$ or $\mathsf{SIM}_{\varphi,n}$ registers ERR, i.e., that $\mathsf{SI}_{\mathsf{PET}}$ registers ERR.

Lemma 6.5. $D_{\mathsf{PET}}(m, N)$ is statistically indistinguishable from $D'_{\mathsf{SI}_{\mathsf{PET}}}(m, N)$.

Proof. The output from $\mathsf{SI}_{\mathsf{PET}}$ for players $P_i \in H \backslash P_n$ is distributed exactly as in PET, since it is generated by running PET.

We have $\epsilon_n = \epsilon^{z_n} = (g^{\hat{m}}\varphi^x)^{z_n} = (g^b)^{\hat{m}}(g^c)^x$ and $\varphi_n = g^c$. With $c = ab$, we have $\log_g(\varphi_n) = \log_g(\varphi)\log_\varphi(\varphi_n)$ and $\log_g(\epsilon_n) = \log_g(\epsilon)\log_\epsilon(\epsilon_n)$, which agrees with their construction in PET and are thus correlated to each other just as in PET. Since $\mathsf{SI}_{\mathsf{PET}}$ does not know a or b it runs $\mathsf{SIM}_{\epsilon,n}$ and $\mathsf{SIM}_{\varphi,n}$ to produce output for P_n

with correct probability distribution. The probability distribution for P_n's output is exactly the same as given by PET, as long as there are no oracle collisions, i.e., $\mathsf{SIM}_{\epsilon,n}$ or $\mathsf{SIM}_{\varphi,n}$ output ERR, which occurs with negligible probability. We conclude that $D'_{\mathsf{SI_{PET}}}(m,N) \overset{s}{\approx} D_{\mathsf{PET}}(m,N)$. ∎

Lemma 6.6. $\mathsf{SI_{PET}}$ terminates in time polynomial in k.

Proof. $\mathsf{SI_{PET}}$'s algorithm contains no loops and no individual step is more than polynomial in k. Consequently one run through the algorithm runs in polynomial time. We restart at most a polynomial number of times, since $\mathsf{SI_{PET}}$ is used with such $\mathcal{A}$ and m that give N as result with non-negligible probability. ∎

Lemma 6.7. The distribution $D'_{\mathsf{SI_{PET}}}(m)$ is computationally indistinguishable from $D_{\mathsf{SI_{PET}}}(m)$ under the non-uniform decisional Diffie-Hellman assumption and in the presence of a static adversary $\mathcal{A}$ of size $\leq t$ and $\forall m \neq 1$.

Proof. Let us assume the opposite, i.e., that distribution $D_{\mathsf{SI_{PET}}}(m)$ is computationally distinguishable from distribution $D'_{\mathsf{SI_{PET}}}(m)$. This implies that there exists a probabilistic polynomial time TM M that can distinguish between $D'_{\mathsf{SI_{PET}}}(m)$ and $D_{\mathsf{SI_{PET}}}(m)$. If $\mathsf{SI_{PET}}$ runs in polynomial time, we can construct a machine $M1$, using M as a subroutine, that solves the non-uniform DDH-assumption in polynomial time. As input $M1$ gets a random discrete log instance $(p,q,g)_k$, a triple (g^a, g^b, g^c) in either Δ_{DDH} or in Δ_{rand}, $\hat{m}$ such that $m = g^{\hat{m}}$ and access to $S1$ and to an adversary $\mathcal{A}$ of size t.

$M1$ chooses a random generator h and runs $S1$ on this input. Using $S1$'s output, M can break the non-uniform DDH-assumption efficiently. On this basis, we state that $D_{S1}(m)$ is computationally indistinguishable from distribution $D'_{S1}(m)$. ∎

Lemma 6.8. $D_{\mathsf{SI_{PET}}}(m,N) \overset{c}{\approx} D_{\mathsf{S_{PET}}}(m,N)$

Proof. As mentioned in Lemma 6.4, if $res_{\mathsf{PET}}(m) = N$ then $\mathsf{S_{PET}}$ only runs simulations on input $m \neq 1$, since the probability that PET outputs N with $m = 1$ is negligible. The first steps of $\mathsf{SI_{PET}}$ produce $in(m) \in_R \mathcal{I}(m)$ for fixed m. The probability of the remaining steps in $\mathsf{SI_{PET}}$ is taken over $in(m)$, ran_{PET}, ran_A, (g^a, g^b, g^c), and the random oracle $\mathcal{O}$.

For all players $P_i \in H \backslash P_n$ the values for C_i, ϵ_i, and φ_i are computed in the same way in both $\mathsf{S_{PET}}$ and $\mathsf{SI_{PET}}$, i.e., by running the algorithm described in PET. Both simulators also run EPSI and PHI. Up to this point, for players , $P_i \in H \backslash P_n$ the only difference between $\mathsf{S_{PET}}$ and $\mathsf{SI_{PET}}$'s algorithm lies in $\mathsf{S_{PET}}$ in the rewinding in EPSI. We can see this rewinding as two parallel executions of $\mathsf{S_{PET}}$ instead. Each thread using the same ran_A, ran_{PET}, and $in(m)$, but different random oracles. Therefore all information published up to the oracle question is the same in both executions. Only one of the executions is run to the end. The view of $\mathsf{S_{PET}}$ consists of this run. Therefore the rewinding process does not affect the probability distribution of $V_{\mathsf{S_{PET}}}(m)$ under the condition that $\mathcal{A}$ passes EPSI with non-negligible probability.

For player P_n in simulator $\mathsf{SI}_{\mathsf{PET}}$, the exponent $z_n (= b)$ is unknown when computing C_n. But still it is random in $\mathbb{Z}_q$. This gives the same probability distribution as C_n has in $\mathsf{S}_{\mathsf{PET}}$. Next the value for ϵ_n is published and in both simulators it consists of ϵ and an exponent random in $\mathbb{Z}_q$. In both cases this is the only place this exponent is used alone. In $\mathsf{SI}_{\mathsf{PET}}$ this exponent, $z_n = b\hat{m} + cx$, is unknown, therefore it is necessary to run $\mathsf{SIM}_{\epsilon,n}$. By Lemma 4.9 we have that $\mathsf{S}_{\mathsf{PET}}$ and $\mathsf{SI}_{\mathsf{PET}}$'s output views, for player P_n, are statistically indistinguishable up to this point.

In $\mathsf{S}_{\mathsf{PET}}$ the value φ_n is uncorrelated with ϵ_n. In $\mathsf{SI}_{\mathsf{PET}}$ the variable φ_n is computed by using the random g^c. This value is uncorrelated with $\epsilon_n^{=} g^{cx+b\hat{m}}$ because b is also random in $\mathbb{Z}_q$. Both simulators make use of $\mathsf{SIM}_{\varphi,n}$. These outputs differ only when $\mathsf{S}_{\mathsf{PET}}$ outputs a BAD message, which occurs only with negligible probability.

The β_i -values are, in both simulators, consistent with the published y_i values. $\mathsf{S}_{\mathsf{PET}}$ runs SIM_β and $\mathsf{SI}_{\mathsf{PET}}$ runs PET. From Lemma 4.5 we have that these views are statistically indistinguishable.

The conclusion of the above is that the outputs of $\mathsf{SI}_{\mathsf{PET}}$ and $\mathsf{S}_{\mathsf{PET}}$ are statistically indistinguishable from one another, as long as no BAD message is registered. The probability of getting a BAD message is negligible (Lemma 4.2). Using the same arguments as in proof of Lemma 4.9, it is clear that $D_{\mathsf{S}_{\mathsf{PET}}}(m, N) \overset{c}{\approx} D_{\mathsf{SI}_{\mathsf{PET}}}(m)$. ∎

6.5 SEQ - Sequential Composition of the Plain-Text Equality Test

When using PET, we most probably want to make comparisons of several encryptions, all encrypted with the same key. In other words, we want to run PET several times using the same discrete-log instance, the same random generator h and the same set of keys and key-shares. To this end we define the sequential plain-text equality test, SEQ, which consists of K sequential executions of the PET protocol run on the same parameters, except for the encryptions being compared, in each execution. The value K is polynomial in the security parameter k. The SEQ protocol is presented in figure 6.5.

We have up to $2K$ ElGamal encrypted messages we want to compare pairwise. These are known before running the SEQ protocol. For each pair we want to compare, we divide the encryptions by each other. By the homomorphic property of ElGamal, the resulting quotient is an encryption of the quotient of the plain-text messages of the two encryptions. We call the resulting ElGamal encryption (ϵ, φ). We let $E(\bar{m}) = (\epsilon_1, \varphi_i), \ldots, (\epsilon_K, \varphi_K)$ denote the set of K such pairwise comparison encryptions, where (ϵ_i, φ_i) is an encryption of $m_i \in \mathcal{G}_q$. The vector $\bar{m} = m_1, \ldots, m_K$.

Input to SEQ consists of a discrete-log instance $(p, q, g)_k$, a random generator h, keys and key-shares $y, y_1, \ldots, y_n$, $x_1, \ldots, x_n$, and the list $E(\bar{m})$ of K ElGamal encryptions.

Public input: $(p, q, g)_k$, h, y, $\{y_i : i = 1, \ldots, n\}$, $\{(\epsilon_i, \varphi_i) : i = 1, \ldots, K\}$.
P_i**'s private input:** x_i such that $g^{x_i} = y_i$.
SEQ**'s output:** $res^{\mathsf{SEQ}}(\bar{m})$

- For $j = 1, \ldots, K$ run PET on $(p, q, g)_k$, h, y, $\{y_i : i = 1, \ldots, n\}$, the private shares and (ϵ_j, φ_j). The output is $res^{\mathsf{SEQ}}(m_j)$.

Let $res^{\mathsf{SEQ}}(\bar{m}) = res^{\mathsf{SEQ}}(m_1)|| \ldots ||res^{\mathsf{SEQ}}(m_K)$.

Figure 6.5. The SEQ protocol

For each encryption in $E(\bar{m})$ SEQ runs the PET protocol. We refer to PET_j as the j:th execution of PET in SEQ. This execution of PET is run using the encryptions (ϵ_j, φ_j). Let EPSI_j, PHI_j, and BETA_j be the sub-protocols EPSI, PHI, and BETA when executed in PET_j. Let $C_{j,i}$ be player P_i's commitment to $z_{j,i}$ in PET_j and let $\epsilon_{j,i} = \epsilon_j^{z_{j,i}}$, $\varphi_{j,i} = \varphi_j z_{j,i}$, etc. Let $res^{\mathsf{SEQ}}(m_j)$ be the output of PET_j and let $res^{\mathsf{SEQ}}(\bar{m}) = res^{\mathsf{SEQ}}(m_1)|| \ldots ||res^{\mathsf{SEQ}}(m_K) = \{N, E, FAIL\}^K$ be the output of SEQ.

Recall that each and every player is modeled by a probabilistic Turing Machine with an auxiliary input tape. The auxiliary input tape gives each player access to all public information from $\mathsf{PET}_1, \ldots, \mathsf{PET}_{j-1}$ when running PET_j and also to his own private information from these runs.

6.6 Completeness, Soundness, and Zero-Knowledge for SEQ

Lemma 6.9. The SEQ protocol is complete, sound, and zero-knowledge in the presence of a static and terminating adversary corrupting up to a minority, under the non-uniform DDH-assumption and in the random oracle model.

Proof. To simplify presentation we divide this proof into its natural components, and treat them separately; completeness in Lemma 6.10, soundness in Lemma 6.11, and zero-knowledge Lemma 6.13. ∎

6.6.1 The Completeness of SEQ

Lemma 6.10. The protocol SEQ produces outputs $res^{\mathsf{SEQ}}(m_j) = N$ for $m_j \neq 1$ and $res^{\mathsf{SEQ}}(m_j) = E$ if $m_j = 1$ with overwhelming probability, for $j = 1, \ldots, K$ and providing that all players are honest.

Proof. This proof follows from the proof of completeness of the PET protocol. In more detail, from Lemma 6.2 we have that one execution of PET is complete if all players are honest, i.e., $res^{\mathsf{SEQ}}(m) = E$ with probability 1 with $m = 1$. When

$m \neq 1$ the probability is $(q-1)/q$ that $res^{\mathsf{SEQ}}(m) = N$. In SEQ each run of PET is independent of the others when all players are honest and from this follows that all output is correct with probability $\geq ((q-1)/q)^K$, which is overwhelming when K is polynomial and q exponential in the security parameter. ∎

When we have an adversary $\mathcal{A}$, we assume that it corrupts players $P_1, \ldots, P_t$ before the protocol starts. The corrupt players remain corrupt throughout the whole execution of SEQ. The remaining players $P_{t+1}, \ldots, P_n$ are assumed honest. The adversary's output in PET_j may depend on $\mathsf{PET}_1, \ldots, \mathsf{PET}_{j-1}$, whereas the honest players remain history independent.

6.6.2 The Soundness of SEQ

Lemma 6.11. The probability is negligible that $\mathcal{A}$ efficiently changes the output of $\{res^{PET}(m_j) : j = \{1, \ldots K\}\}$ for $j = \{1, \ldots K\}$ from E to N or from N to E.

Proof. Again, this proof follows from the proof of soundness of the PET protocol. Passing EPSI_j, PHI_j, and BETA_j depends only on being able to compute a response to the current oracle challenge. Since this challenge is truly random, looking at old conversations gives no beforehand information about the oracle challenge and does not increase the probability of passing.

We can assume that both EPSI_j and PHI_j output ACC, and that BETA_j outputs an empty set for all $j = 1, \ldots, K$. Each execution of PET falls into one of two possible cases. These are presented in the proof of Lemma 6.3, we refer to that proof for the discussion. The probability is negligible that an adversary changes any $\{res^{\mathsf{SEQ}}(m_j)\}$ for $j = \{1, \ldots K\}$. ∎

6.6.3 The Zero-Knowledge of SEQ

We define $\mathcal{I}_K(\bar{m}, k, n)$ to be the set of all possible input sets $\{(p, q, g)_k, h, y, \{x_i, y_i : i = 1, \ldots n\}, E(\bar{m})$ to the PET protocol, that can be created using $\bar{m}$, k, n, and K. From now on we let the values t, k, and n be implicit. Let $in_K(\bar{m}) \in_R \mathcal{I}_K(\bar{m})$ for a fixed vector $\bar{m}$. Let $in_K(\bar{m}, j)$ be the subset of $in_K(\bar{m})$ containing only one encryption, (ϵ_j, φ_j), the encryption of m_j.

Let $V_{SPET}(\bar{m})$ be the random variable defined by the adversarial view of an execution of SEQ with H on input $in_K(\bar{m})$. Note that all executions of PET use the same discrete-log instance, the same public and private key and the same shares thereof. We let also this fact be implicit. Let $D_{\mathsf{SEQ}}(\bar{m})$ be the probability distribution of $V_{\mathsf{SEQ}}(\bar{m})$, the probability taken over $in_K(\bar{m})$, ran_A, ran_H, and over $\mathcal{O}$.

We define the simulator $\mathsf{S}_{\mathsf{SEQ}}$ in figure 6.6. It simulates an execution of SEQ on behalf of the honest players in order to show that SEQ is zero-knowledge, i.e., that no polynomially bounded $\mathcal{A}$ can gain any information from running SEQ, apart from $res^{\mathsf{SEQ}}(\bar{m})$. $\mathsf{S}_{\mathsf{SEQ}}$ uses the adversary as a black-box asking it for output when needed.

$\mathsf{S_{SEQ}}$'s input: $(p, q, g)_k$, h, y, $\{y_1, \ldots, y_n\}$, $\{(\epsilon_i, \varphi_i) : i = 1, \ldots, K\}$, $res^{\mathsf{SEQ}}(\bar{m})$, $x_i \forall P_i \in A$
$\mathsf{S_{SEQ}}$'s output: $V_{\mathsf{S_{SEQ}}}(\bar{m})$

For $j = 1, \ldots, K$ do the following:

- If $res^{\mathsf{PET}}(m_j) \in \{E, FAIL\}$ run the algorithm of $\mathsf{S_{PET}}$ given in figure 6.2 on the input given above together with the view of the previous runs.
- If $res^{\mathsf{PET}}(m_j) = N$ run the algorithm of $\mathsf{S_{PET}}$ in figure 6.3 on the input given above together with the view of the previous runs, i.e., $(1, \ldots, j-1)$.

The output $V_{\mathsf{S_{SEQ}}}(\bar{m})$ consists of all public values produced by $\mathsf{S_{SEQ}}$, the sub-protocols used in $\mathsf{S_{SEQ}}$ and their simulators, i.e.,

$V_{\mathsf{S_{SEQ}}}(\bar{m}) = \{res^{\mathsf{S_{PET}}}(m_j),\ y,\ (\epsilon_j, \varphi_j),\ y_i,\ C_{j,i},\ \epsilon_{j,i}, \varphi_{j,i},\ \alpha_j,\ \beta_j, \beta_{j,i}, : i = 1, \ldots, n, j = 1, \ldots, K\}$ together with the views of the sub-protocols.

Figure 6.6. The protocol for $\mathsf{S_{SEQ}}$

The simulator $\mathsf{S_{SEQ}}$ consists of K executions of $\mathsf{S_{PET}}$ and runs in expected polynomial time (Lemma 6.12). We refer to the i:th run of $\mathsf{S_{PET}}$ in $\mathsf{S_{SEQ}}$ as $\mathsf{S_{PET}}_i$. Input to $\mathsf{S_{PET}}_i$ is $res^{\mathsf{SEQ}}(m_i)$ and all information $\mathcal{A}$ has access to in a real run. In each run of $\mathsf{S_{PET}}_i$ we use the same adversary, who has access to the views of the previous executions of $\mathsf{S_{PET}}_i$. The output of $\mathsf{S_{SEQ}}$ is $V_{\mathsf{S_{SEQ}}}(\bar{m})$, which is the set of all public information published throughout the execution of $\mathsf{S_{SEQ}}$. $V_{\mathsf{S_{SEQ}}}(\bar{m})$ has probability distribution $D_{\mathsf{S_{SEQ}}}(\bar{m})$, where the probability is over $in_K(\bar{m})$, $ran_{\mathcal{A}}$, $\mathcal{O}$ and $ran_{\mathsf{S_{SEQ}}}$, the internal random string of $\mathsf{S_{SEQ}}$. If the output from $\mathsf{S_{PET}}_i$ does not agree with $res^{\mathsf{SEQ}}(m_i)$ then rerun the protocol. Each element in $res^{\mathsf{SEQ}}(\bar{m})$ occurs with non-negligible probability. We can say this since $\bar{m}$ is fixed from start, thus not letting the messages depend on results of previous evaluations. Since $\mathsf{S_{SEQ}}$ runs parts of $\mathsf{S_{PET}}$ which may output the message BAD, then $\mathsf{S_{SEQ}}$'s output may contain a BAD-message. The probability is negligible that $\mathsf{S_{PET}}$ outputs BAD, and therefore $\mathsf{S_{SEQ}}$'s output contains BAD with negligible probability.

Lemma 6.12. $\mathsf{S_{SEQ}}$ runs in expected polynomial time.

Proof. In each round of $\mathsf{S_{SEQ}}$ we run the algorithm of $\mathsf{S_{PET}}$. We have from Lemma 6.4 that the simulator $\mathsf{S_{PET}}$ runs in expected polynomial time. As stated above each of the K rounds in $\mathsf{S_{SEQ}}$ is restarted at most a polynomial number of times. If K is polynomial in k. We restart at most a polynomial number of times in SEQ. This gives an expected polynomial running time for $\mathsf{S_{SEQ}}$. ∎

Lemma 6.13 Zero-knowledge of SEQ. There exists a simulator $\mathsf{S}_{\mathsf{SEQ}}$ having access to the same information as $\mathcal{A}$ and to $res^{\mathsf{SEQ}}(\bar{m})$, that outputs $V_{\mathsf{S}_{\mathsf{SEQ}}}(\bar{m})$ with probability distribution $D_{\mathsf{S}_{\mathsf{SEQ}}}(\bar{m})$ computationally indistinguishable from $D_{\mathsf{SEQ}}(\bar{m})$.

Proof. In Figure 6.6 we construct the simulator $\mathsf{S}_{\mathsf{SEQ}}$ outputting $V_{\mathsf{S}_{\mathsf{SEQ}}}(\bar{m})$ with probability distribution $D_{\mathsf{S}_{\mathsf{SEQ}}}(\bar{m})$. To show that $D_{\mathsf{SEQ}}(\bar{m})$ is computationally indistinguishable from $D_{\mathsf{S}_{\mathsf{SEQ}}}(\bar{m})$ we build a chain of computationally indistinguishable distributions. For this purpose we construct the intermediate simulator $\mathsf{SI}_{\mathsf{SEQ}}$ producing these intermediate probability distributions. These are described in detail in Section 6.6.4. Lemma 6.14, 6.15, and 6.17 give that $D_{\mathsf{SEQ}}(\bar{m})$ is computationally indistinguishable from $D_{\mathsf{S}_{\mathsf{SEQ}}}(\bar{m})$. ∎

6.6.4 The Intermediate Simulator $\mathsf{SI}_{\mathsf{SEQ}}$

We aim to show that the probability distributions $D_{\mathsf{SEQ}}(\bar{m})$ and $D_{\mathsf{S}_{\mathsf{SEQ}}}(\bar{m})$ are computationally indistinguishable. For this purpose we introduce the intermediate simulator $\mathsf{SI}_{\mathsf{SEQ}}$, presented in figure 6.7. On input (g^a, g^b, g^c) it outputs the random variable $V_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$.

When $(g^a, g^b, g^c) \in \Delta_{DDH}$ we refer to the probability distribution of $V_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$ as $D'_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$. When $(g^a, g^b, g^c) \in \Delta_{rand}$ we call the probability distribution of $V_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$ $D_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$. The probability in $D'_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$ and $D_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$ is taken over $in_K(\bar{m})$, a, b, c, $ran_{\mathsf{SI}_{\mathsf{SEQ}}}$, ran_A, and $\mathcal{O}$. Lemmas 6.14, 6.15 and 6.17 state that

$$D_{\mathsf{SEQ}}(\bar{m}) \overset{s}{\approx} D'_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m}) \overset{c}{\approx} D_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m}) \overset{s}{\approx} D_{\mathsf{S}_{\mathsf{SEQ}}}(\bar{m}).$$

With help of $\mathsf{SI}_{\mathsf{SEQ}}$ we have now produced a chain consisting of four probability distribution, where each neighboring pair of distributions is computationally indistinguishable. Recall that statistical indistinguishability implies computational indistinguishability.

In more detail this is how $\mathsf{SI}_{\mathsf{SEQ}}$ works. For each $res^{\mathsf{SEQ}}(m_j) = N$, $\mathsf{SI}_{\mathsf{SEQ}}$ computes a new random and independent triple of the same kind as (g^a, g^b, g^c). The method is described in Section 2.5.

Apart from the triple (g^a, g^b, g^c), input to the simulator consists of values $m_1, \ldots, m_K$, and $\hat{m}_j = \log_g(m_j)$ for such m_j where $res^{\mathsf{SEQ}}(m_j) = N$. For all $res^{\mathsf{SEQ}}(m_j) = N$, these values and the new triples are used to compute the encryptions (ϵ_j, φ_j). The remaining values in the protocol are computed as in $\mathsf{SI}_{\mathsf{PET}}$. The encryptions for all executions where $res^{\mathsf{SEQ}}(m_j) = e$ are computed by selecting a random value $R_j \in_R \mathbb{Z}_q$ and computing $(\epsilon_j, \varphi_j) = (m_j y^{R_j}, g^{R_j})$. The remaining values are computed by running PET. This is possible since $\mathsf{SI}_{\mathsf{SEQ}}$ has access to the private key-shares of the honest players, and all other values depend only on the honest players random strings.

From lemmas 4.9 and 4.4 we see that the probability is negligible that $\mathsf{SIM}_{\epsilon,n}$ or $\mathsf{SIM}_{\varphi,n}$ registers ERR, i.e., that $\mathsf{SI}_{\mathsf{SEQ}}$ registers ERR.

Lemma 6.14. $D_{\mathsf{SEQ}}(\bar{m}) \overset{s}{\approx} D'_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$

$\mathsf{SI}_{\mathsf{SEQ}}$'s input: $(p,q,g)_k$, h, (g^a, g^b, g^c), $res^{\mathsf{SEQ}}(\bar{m})$, $m_1,\dots,m_K$, $\hat{m}_j = \log_g(m_j)$ for m_j where $res^{\mathsf{SEQ}}(m_j) = N$
$\mathsf{SI}_{\mathsf{SEQ}}$'s output: $V_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$

1. Compute a set of key and key-shares by running the simulator for the joint-key sharing protocol. It gives $\mathsf{SI}_{\mathsf{SEQ}}$ access to $x_1,\dots,x_n$, the private key x and it outputs $y, y_1,\dots,y_n$.
2. For each m_j such that $res^{\mathsf{SEQ}}(m_j) = N$ compute a random triple $(g^{a_j}, g^{b_j}, g^{c_j})$ of the same sort as the input triple.
3. For each m_j such that $res^{\mathsf{SEQ}}(m_j) = N$ compute an encryption (ϵ_j, φ_j) of m_j, by letting $\varphi_j = g^{a_j}$ and $\epsilon_j = g^{\hat{m}_j}(g^{a_j})^x$.

   For each m_j such that $res^{\mathsf{SEQ}}(m_j) = E$ compute an encryption (ϵ_j, φ_j) of m_j by selecting $R_j \in_R \mathbb{Z}_q$ and letting $\epsilon_j = m_j y^{R_j}$ and $\varphi_j = g^{R_j}$.
4. For $j = 1,\dots,K$, if $res^{\mathsf{SEQ}}(m_j) = E$ run PET_j. Otherwise, if $res^{\mathsf{SEQ}}(m_j) = N$ run the following steps:

   For players $P_i \in H \backslash P_n$ run PET_j. To produce player P_n's output do the following:

   (a) Select $r_{j,n} \in_R \mathbb{Z}_q$. Let $C_{j,n} = g^{b_j} h^{r_{j,n}}$.
   (b) Let $\epsilon_{j,n} = (g^{b_j})^{\hat{m}_j}(g^{c_j})^x$, and run $\mathsf{SIM}_{\epsilon,n}$. If $\mathsf{SIM}_{\epsilon,n}$'s output contains ERR, let $res^{\mathsf{SI}_{\mathsf{SEQ}}}(m_j) = ERR$.
   (c) Let $\varphi_{j,n} = g^{j,c}$ and run $\mathsf{SIM}_{\varphi,n}$. If $\mathsf{SIM}_{\varphi,n}$'s output contains ERR, let $res^{\mathsf{SI}_{\mathsf{SEQ}}}(m_j) = ERR$.
   (d) Run round 4 and 5 as described in PET. $\mathsf{SI}_{\mathsf{SEQ}}$ has the private shares of all players and can compute $\beta_{j,i}$ for all honest players and run BETA.

   If $res^{\mathsf{SI}_{\mathsf{SEQ}}}(m_j) \neq res^{\mathsf{SEQ}}(m_j)$ rerun the j:th execution of this protocol.

Let $V_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m}) = in_K(\bar{m})$, $res^{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$, $\{C_{j,i},\ \epsilon_{j,i},\ \varphi_{j,i},\ \beta_{j,i} : 1 \le i \le n, 1 \le j \le K\}$, and the adversarial views of the sub-protocols and their simulators.

Figure 6.7. The protocol for $\mathsf{SI}_{\mathsf{SEQ}}$ interacting with $\mathcal{A}$

Proof. When $(g^a, g^b, g^c) \in \Delta_{DDH}$, all variables in $\mathsf{SI}_{\mathsf{SEQ}}$ are distributed as they are in SEQ. The difference between the probability distribution of the adversarial view SEQ and the probability distribution of the output of $\mathsf{SI}_{\mathsf{SEQ}}$ lies in the proofs of correctness of the values $\epsilon_{j,i}$ and $\varphi_{j,i}$. For those j where $res^{\mathsf{SEQ}}(m_j) = N$, $\mathsf{SI}_{\mathsf{SEQ}}$ must run $\mathsf{SIM}_{\epsilon,n}$ and $\mathsf{SIM}_{\varphi,n}$ to simulate these proofs for player P_n. From lemmas 4.10 and 4.5 we have that the output from these sub-protocol simulators is statistically indistinguishable from the output of their real executions. Therefore the probability distribution of the whole adversarial view of $\mathsf{SI}_{\mathsf{SEQ}}$ is statistically indistinguishable from that of the adversarial view of SEQ. ∎

Lemma 6.15. The probability distribution $D'_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$ is computationally indistinguishable from $D_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$ under the non-uniform decisional Diffie-Hellman assumption and in the presence of a static adversary $\mathcal{A}$ of size $\leq t$.

Proof. The only difference between the two distributions lies in the input triple (g^a, g^b, g^c). If we assume that $D'_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$ is computationally distinguishable from $D_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$, then there exists a polynomial Turing Machine M that can distinguish between the two probability distributions. We can use M and $\mathsf{SI}_{\mathsf{SEQ}}$ as subroutines to solve the non-uniform DDH-assumption efficiently. On this basis, we state that $D'_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m}) \overset{c}{\approx} D_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m})$. ∎

Lemma 6.16. $\mathsf{SI}_{\mathsf{SEQ}}$ terminates in time polynomial in k.

Proof. $\mathsf{SI}_{\mathsf{SEQ}}$'s algorithm contains no loops and no individual step is more than polynomial in k. Consequently one run through the algorithm runs in polynomial time. We will restart at most a polynomial number of times, since $\mathsf{SI}_{\mathsf{SEQ}}$ is used with such $\mathcal{A}$ and m_j producing $res^{\mathsf{SEQ}}(m_j)$ that occurs with non-negligible probability. ∎

Lemma 6.17. $D_{\mathsf{SI}_{\mathsf{SEQ}}}(\bar{m}) \overset{s}{\approx} D_{\mathsf{S}_{\mathsf{SEQ}}}(\bar{m})$

Proof. The main step in showing statistical indistinguishability between $D_{\mathsf{SI}_{\mathsf{SEQ}}}$ and $D_{\mathsf{S}_{\mathsf{SEQ}}}$ lies in getting $\varphi_{j,n}$ random and independent of all other values, for those j where $res^{\mathsf{SEQ}}(m_j) = N$.

In subsequent executions the adversary may base his choices and outputs on outputs from previous runs, i.e., in the j:th run of $\mathsf{SI}_{\mathsf{SEQ}}$, the adversarial output may depend on ran_H and $(g^{a_1}, g^{b_1}, g^{c_1}), \ldots, (g^{a_{j-1}}, g^{b_{j-1}}, g^{c_{j-1}})$.

The value $\varphi_{j,n}^{(\mathsf{SI}_{\mathsf{SEQ}})}$ is computed as g^{c_j}, whereas $\varphi_{j,n}^{(\mathsf{S}_{\mathsf{SEQ}})} = g^{s_j} / \prod_{i=1}^{n-1} \varphi_{j,i}^{(\mathsf{S}_{\mathsf{SEQ}})}$ where the corrupt players' $\varphi_{j,i}$ values are computed from values for $z_{j,i}$ extracted in the preceding step. In $\mathsf{SI}_{\mathsf{SEQ}}$ it is clear that $\varphi_{j,n}^{(\mathsf{SI}_{\mathsf{SEQ}})}$ is random and independent as long as the triples are random and independent. In $\mathsf{S}_{\mathsf{SEQ}}$ the adversarial output $\varphi_{j,1}, \ldots, \varphi_{j,t}$ may depend on previous executions. But, since the value g^{s_j} is random and independent $\varphi_{j,n}^{(\mathsf{S}_{\mathsf{SEQ}})}$ is likewise.

For those $1 \leq j \leq K$ where $res^{\mathsf{SEQ}}(m_j) = E$ the simulator $\mathsf{S}_{\mathsf{SEQ}}$ runs the algorithm of $\mathsf{S}_{\mathsf{PET}}$ on input E. In each such round the adversary may base his input and output on the information obtained from rounds $1, \ldots, j-1$ of $\mathsf{S}_{\mathsf{SEQ}}$. This information is statistically indistinguishable from the information that the adversary has access to in a real run of SEQ. Therefore if $\mathsf{S}_{\mathsf{SEQ}}$ and $\mathsf{SI}_{\mathsf{SEQ}}$ produce outputs that are statistically indistinguishable when run on the same input, which they do by Lemma 6.4, then running them on statistically indistinguishable inputs can not change this.

Differences between the entire outputs of $\mathsf{SI}_{\mathsf{SEQ}}$ and $\mathsf{S}_{\mathsf{SEQ}}$ lie in running $\mathsf{SIM}_{\varphi,n}$ in place of PHI and SIM_{β} in place of BETA. By Lemma 4.5 the outputs of the two simulators are statistically indistinguishable in distribution. ∎

We conclude this by saying that we can compare up to a polynomial number message pairs using the same public and private key pair without leaking any information about the private key. This property can be used as a base for something useful, such as an electronic auction.

Chapter 7

PET Auctions

The Internet has given us a unique possibility of remote participation in, for example, electronic auctions. There is no longer a need to be present in person. Unfortunately, with remote participation we get reduced ability of surveillance and control of what is happening. Suppose you find a sealed bid auction for a Picasso painting, where all bids are submitted in private to an auctioneer. After lots of thought you determine to participate and submit a bid for the painting. But you have good reasons to feel concerned about the auction. Can you trust the result of the auction? How do you know that the information you submit (of how you value the painting) will not be misused in future pricing? Do the sellers actually have the painting and will they send it to you if you win?

In this chapter we present a protocol for an electronic auction which addresses some of these concerns. We discuss only the aspects that are connected with bid information. We do not discuss the parts connected with the handling of goods.

Our auction system consists of three interacting entities: *the auction service* – a group of independent servers who compute the auction, *a seller* – the party with an item to sell, and *bidders* – the players who are interested in buying the item.

Auctions can roughly be divided into two groups: *open cry auctions* and *sealed bid auctions*. Open cry auctions are the auctions we are used to seeing, with bidders crying out their bids for all to hear (or see). In terms of revealing information, the open cry auction reveals all available information, whereas sealed bid auctions hide the bid to at least all the other participants. We focus on sealed bid auctions and limit them to be single sealed bid auctions - letting players submit a bid at a single occasion.

Ideally an auction protocol should not reveal any information at all, other than the winner and the winning bid. At the same time bidders and other observers should be able to verify the correctness of the whole process. Moreover, we require the protocol to be efficient even for a larger number of players.

The protocol we present is for a single sealed bid auction and uses sequential execution of PET, the SEQ protocol to determine the outcome of the auction. It is the order in which we compare the messages which gives us an auction protocol. We implement a fully secure highest-bid auction hiding all bids but the highest, and a Vickery auction, i.e., a second price auction, revealing the winner and the two highest bids.

7.1 Related Work

The auction problem is a well studied problem. It is in fact a special case of secure multi-party computation. A general solution for secure multi-party computation was presented by [GMW87]. This solution is based on circuits, and has a polynomial running time, but because of its generality it is impractical for real use. The efficiency requirement rules out general multi-party computational schemes.

All existing solutions specially designed for electronic auctions focus on reducing the power of the auctioneer.

One solution working with a single auctioneer is that of Baudron and Stern [BS01]. This auctioneer is assumed semi-trusted in the sense that he follows the protocol but registers all internal data and published data. Bids are compared using a specific circuit for this auction which is evaluated by the auctioneer. This solution is private in the sense that as long as the auctioneer does not collude with any bidders then he learns nothing about the bids. Unfortunately it is not verifiably correct. Bidders can not check that the circuit is correctly evaluated. The auctioneer must therefore be trusted. On the other hand bidders prove in zero-knowledge that they know the contents of their bids.

To reduce the trust put into the auctioneer, a second party may be added. In [Cac99] the electronic auction has two semi-trusted servers. The protocol is not fully private since one of the servers gets information about the partial order of bids. The other server and the players remain entirely oblivious of the bids. If the two servers collude then they can determine the bids.

Abe and Suzuki add a trusted authority holding the private key for the auction in [AS02]. This prevents the auctioneer from decrypting the bids. Their protocol is based on mix-servers, and also the trusted party decrypts bid. No formal definition of the security of a mix network has been given. Consequently there are no proofs of security. We refer to [Wik02] for further discussion. The protocol is private in the sense that the auctioneer does not get any information about the bids, but instead it involves a trusted party which always gets all information. Also, the protocol is definitely not zero-knowledge.

Yet another solution involving two parties was proposed by [NPS99]. In addition to the auctioneer, they introduce an authorized auction issuer who guarantees security. The auction is computed by a circuit which is coded by the authorized auction issuer. The evaluation of the circuit is based on *proxy oblivious transfer* and aims to ensure security as long as at least one player is honest. The system has a security flaw such that one of the two auction servers can cheat and consequently to modify bids arbitrarily without it being detected. In [JS02] they present a solution to the problem, which they call *verifiable proxy oblivious transfer*. Using verifiable proxy oblivious transfer, the protocol becomes unmalleable. The protocol is private and conveys no information other than the winning bid. It is secure as long as the auctioneer and the auction issuer do not collude. Several other solutions reducing the power of the auctioneer by adding an additional entity have been suggested, such as [BN00, Kik01].

By spreading the auctioneers' functionality among a set of servers, we reduce the amount of trust we must put in each server. In this way we eliminate the need of an extra trusted party. It has been argued by [NPS99] that this is not a realistic scenario, and using some sort of trust is better. We prefer to focus on security, and therefore look at a distributed setting.

One of the earliest suggestions for a secure electronic auction is [FR95]. This solution uses a set of auction servers to run the auction. Unfortunately the paper only focuses on avoiding disclosure of the bids before the bidding is over, after which all information about the bids is disclosed.

A solution, using a set of auction servers, which is more focused on not disclosing bids is [JJ00]. The main component is a circuit consisting of binary gates. Each gate is evaluated by running a mix and the PET protocol. The auction protocol is private under the condition that mix-networks do not reveal or leak any information, now that we have shown that neither PET nor SEQ disclose information.

Another solution using both a set of auction servers and a circuit based solution is [KO02]. They note that in all circuit solutions, the communication and computational complexity relies mainly on the size of the circuit. Therefore the main focus of the paper is to reduce the number of gates needed. The technique used to evaluate the circuit is similar to that of [JJ00]. Other solutions that distribute the functionality of the auctioneer on a set of auction servers are [HTK99, KHT98, SM00].

Very few of the above mentioned papers have formal proofs of security, which has been our focus when designing the PET auction.

7.2 The PET Auction Setting

The auction system shown in Figure 7.1 involves the three kinds of participants, the auction service, the seller and the bidders. All participants are modeled as Turing Machines and run in time polynomial in the security parameter k (see section 3).

The Auction Service:

The auction service is the entity that runs the auction. It advertises the auction, precomputes necessary values such as distributed keys, collects bids and computes the winner and the winning price.

The system consists of n servers, to which we refer as auction servers or simply as servers. We assume that a majority of the servers are honest and independent. This distributes the trust. The servers communicate with each other via a bulletin board (see section 3.1).

All decisions that need consensus among the servers are resolved by voting. The servers post a message if they accept. When $\geq t$ servers have voted for acceptance, then a decision is accepted.

Seller:

The seller is the party who wants to sell an item, the Thing. For this purpose he contacts an auction service in order to run an electronic auction for the item. We focus on the simple case when one seller has one item to sell, but also discuss the case when the seller has several identical items to sell.

Bidders:

The bidders are the parties interested in buying the Thing. Bidders value the Thing and submit their bids to the auction system. The seller can be one of the bidders.

Each bidder has an identity bID_i. We assume that it is issued by, for example, a central authority,

Bidders communicate, i.e., submit bids, with the auction service via the servers bulletin board. Each message is published together with the identity of the bidder or the auction server.

7.3 The PET Auction Protocol

An electronic auction is executed in three phases: *the setup phase*, *the bidding phase* and *the computing phase.*

The setup phase starts when the auction servers accept an auction assignment. Each auction the servers accept gets a unique auction identification number, $auctionID$. This number consists of the sequence number of the auction and the current date. The auction service publishes the description of Thing together with the $auctionID$ on the bulletin board.

The auction servers jointly compute a discrete-log instance $(p, q, g)_k$ and a random generator h, a public key y, public key-shares $(y_1, \ldots, y_n)$ and private key-shares $(x_1, \ldots, x_n)$ using the bulletin board to communicate among each other. The process of computing these values reveals $(p, q, g)_k$, h, y and $(y_1, \ldots, y_n)$. The protocols to compute these values are such that all observers can verify the values' correctness. For the protocols themselves we refer to [Jar01].

The auction service also determines a value of T_2, which is the length of the bidding time. This value may as well be a fixed parameter for the auction service.

When these values are computed and published the seller submits a price list of acceptable prices. We denote this price list $A = \{a_1, a_2, \ldots, a_m\}$, where $a_i < a_{i+1}$. The length of the price list, m, needs to be such that the probability is small that we get a collision in the winning bid. If the price list is a list of reasonable prices, we believe that in practice, if we use a list which is cn in length, where c is some constant > 1, the probability that we have a collision in the highest bid is small. If the auction service accepts the price-list, it publishes it.

By now all information necessary for the auction is computed and published and the servers start the bidding phase by publishing a $START$ message.

Bidders can now publish their bids. A bid must be on the following form in order to be considered: $B_i = (E(b_i), bID_i,\ auctionID, Sig_i)$. The value b_i is the value the bidder with identity bID_i is willing to pay and $E(b_i)$ is the encryption of b_i under key y. The $auctionID$ is attached to bids to make sure that they are submitted to the correct auction, and to make sure that no old bids are reused. The value Sig_i is bidder i's signature on the encrypted bid, the encrypted identity, and the auction identification number. The public key used to verify the signature is assumed known.

After time T_2 has elapsed, the auction servers publish an END message to mark the end of the bidding. Bids published after the END message are not considered.

This concludes the bidding phase, and is starts the computing phase, which starts with the servers jointly constructing a list of bids they consider correct. This can be done as follows: Each server verifies the signatures on the received bids, removes the incorrect bids. A random server publishes its list of bids it considers accepted. Removal or addition to this list is performed by voting. Call this list $B = B_1, \ldots, B_n$.

In the computational phase the auction servers want to compare the submitted bids with the possible bids, i.e., they want to compare a value a_i with $E(b_j)$. We can write $E(b_j) = (\gamma_j, \delta_j)$ since $E(b_j)$ is an ElGamal encryption. We also have that $E(b_j/a_i) = (\gamma_j/a_i, \delta_j)$, which is the input to one round of SEQ. Each server can, entirely on its own, compute the list of comparisons:

$$\begin{aligned} E(b_1/a_m), \ldots, E(b_n/a_m) \quad &, \quad E(b_1/a_{m-1}), \ldots, E(b_n/a_{m-1}), \ldots \\ \ldots \quad &, \quad E(b_1/a_1), \ldots, E(b_n/a_1). \end{aligned} \tag{7.1}$$

Using this list, the auction service can now compute the outcome of the auction by running the SEQ protocol with the list of comparisons as main input. The idea is to compare each price with each bid, until a match is found. The details how to compute the winner and the winning bid depend on which kind of auction we intend to run.

7.3.1 Highest Price Auction

A highest price auction is an auction where the bidder with the highest bid wins the auction, paying the price he bid.

To run a highest price auction the servers run the SEQ protocol on the list of comparisons. Each comparison is input to the PET protocol which determines whether a bid is equal to a given price a_i or not. By starting with the highest price and comparing all bids with this price the servers can jointly determine if any bidder bid the highest price. If no match is found the servers descend to the second highest price and repeat the procedure. This continues until an equality is found,

Input to the auction service: the security parameter k

- The setup phase:

1. The seller submits an auction request and a description of the Thing
2. The auction service votes for acceptance.

   If the auction service accepts the assignment, it publishes the description of the Thing on the bulletin board together with the $auctionID$.
3. The auction service jointly computes a discrete-log instance $(p, q, g)_k$, a random generator h, a public key y, the length of the auction T_2, and public and private key-shares $y_1, \dots, y_n, x_1, \dots, x_n$.
4. The seller submits a list of acceptable prices $A = \{a_1, a_2, \dots, a_m\}$ to the auction service. These are also published.
5. The auction service votes for starting the bidding phase. If there is consensus they publish a $START$ message.

- The bidding phase:

1. Each bidder submits an encrypted bid $B_i = (E(b_i), bID_i, auctionID, Sig_i)$.
2. After time T_2 the auction service votes for ending the bidding phase. If there is consensus they publish an END message.

- The computing phase:

1. The auction service checks the correctness of bids and builds a list B of accepted bids.
2. The auction service builds a comparison list of encryptions, $E(b_1, /a_m), \dots, E(b_n, /a_m)$, $E(b_1, /a_{m-1}), \dots, E(b_n, /a_{m-1})$, $\dots, E(b_1, /a_1), \dots, E(b_n, /a_1)$.
3. The auction service run the sequential PET protocol, i.e., the SEQ protocol, to compute the winner and the winning bid.

Figure 7.1. The PET auction protocol

after which the procedure is stopped. The value a_i at which we got the equality indicates the highest bid and is the selling price.

7.3.2 Vickery Auction

A Vickery auction, or a second price auction as it is occasionally called, is an auction where the bidder with the highest bid wins the auction, but pays the price of the second highest bid. Using this strategy, it does not pay to speculate in how others value the Thing.

To run a Vickery auction, the servers run the SEQ protocol until they find the second equality. This gives the second highest bid, which is the winning price. By jointly decrypting the identity of the highest bidder, the servers find the identity of the winner.

Ideally in a Vickery auction the bidding price of the winner is not revealed. Unfortunately, using our simple algorithm the value of the highest bid is also revealed as well as the identity of the bidder bidding the second highest price.

7.3.3 Selling Many Items

The problem of how to auction several items is an economic strategic problem and is a wide area of research. We therefore present the following observations, but refrain from any further discussion. The observations are made under the assumption that the seller has r identical Things to sell.

We can easily implement a *highest bid* r*-auction* and a r*:th-price Vickery auction* using the SEQ-search.

The highest bid r-auction simply find the r highest bids and the identities of these bidders, who pay the price they bid. This again reveals only such information we wish to reveal.

The r:th-price Vickery auction is as follows. By running the SEQ-search until $r+1$ equalities have been found we have a list of $r+1$ prices and $r+1$ encrypted identities. We assign price $r+1$ to bidder i. The identity of the bidder submitting the $r+1$:st highest price is never decrypted, and thus never revealed. As in a normal Vickery auction, this protocol reveals the highest bid apart from the information we want it to reveal, i.e., the identities of the winners and their winning prices.

7.3.4 Resolving Ties

Resolving ties in single sealed bid auctions may be a problem. Establishing and publishing the fact that there is a tie already reveals more information than is revealed in the ideal case.

Using the PET auction we are able to resolve ties by ignoring them, possibly by evaluating bids in order of submission. The earlier a bidder submits a bid, the bigger chance he has to win if there is a tie. The auction service and all observers, except the bidder placing the tying bid, are unaware of the fact that there was a tie.

7.4 Analysis of a PET Auction

Below, we discuss the security and take a look at the computational and the time complexity of a PET auction.

7.4.1 The Fairness of a PET Auction

During the bidding phase the bidders submit their bids. The auction service does not start computing until the bidding phase is over, and the bidders can not place any more bids. The bids in themselves are randomly encrypted using ElGamal. Given that all participants are polynomial in the security parameter, they get no information about the bids from the encryptions, assuming that ElGamal is semantically secure.

However, the ElGamal encryption scheme is malleable. This implies that given an ElGamal encryption, a bidder can produce a new random encryption of the same plain-text as the given encryption or a function of it. This allows a bidder to place the same bid as some other bidder (or a function of that bid).

This is not a desirable feature. Depending on the structure of the price-list, this may be a very bad feature. If, for example, each price is a constant factor times the previous, it is easy to bid higher than some given bid. With a carefully designed price-list and by evaluating bids in order of submission we can avoid this kind of attack.

Another solution is to force each player to perform a zero-knowledge proof of knowledge of the blinding factor in the encryption of his or her bid, i.e., proving the "knowledge" of r_i of g^{r_i}, from $E(b_i) = (b_i y^{r_i}, g^{r_i})$. This indirectly asserts that the bidder knows the encrypted bid. This proof is very similar to the DLOG proof of knowledge, with the difference that there, the players also prove that two exponents are the same. Another difference is that the validity of the proof is determined by the auction servers. The proof is presented in Figure 7.2. For proofs of completeness, soundness and zero-knowledge we refer to those for the DLOG protocol.

We also want to point out that it is possible to place bids for prices not in the price list. These bids are considered in the algorithm, but stand only a negligible a chance of being a winning bid, without it actually being so. This is the same chance any bid has of falsely being considered equal to the comparison value. For this probability we refer to the proof of Lemma 6.2, the completeness of PET.

Public input: $(p, q, g)_k$, $b_i y^{r_i}, g^{r_i}$

Each bidder does the following:

1. Selects $s_i \in_R \mathbb{Z}_q$ and computes
$$S_i = g^{s_i}.$$
2. Selects and publishes $a_i \in_R \{0,1\}^k$.
3.
   - Selects $d_i \in \{0,1\}^k$.
   - Computes $v_i = \mathcal{O}(d_i a_1 \ldots a_n S_i g^{r_i} i)$.
   - Computes $w_i = s_i + r_i v_i$.
   - Publishes w_i and d_i.

The validity is checked by the auction servers by checking the following equation:

$$g^{w_i} = S_i (g^{r_i})^{v_i}. \tag{7.2}$$

Figure 7.2. The BID proof of knowledge

7.4.2 The Security of a PET Auction

Each run of PET in SEQ, i.e., each comparison, reveals whether the two messages that are being compared are equal or not. In the case of PET auctions we compare the encrypted prices from the price list, $E(a_i)$ with an encryption of each bid $E(b_i)$. The bids are never compared with each other. Since the servers compare the bids with the (known) prices from the price list in descending order they get some information about the bids, it gives the bids an upper bound. This information is revealed independent of how the auction is computed. Otherwise, the servers learn nothing about the bids. This is formalized in Lemma 6.13 showing zero-knowledge for the SEQ protocol. This is the only place where computation on the bids is performed.

Each bid is only compared with the prices from the price list. If a server by chance knows a bid, it learns nothing from a comparison where the bid is involved. This implies that collaboration between a set of corrupt servers and a bidder does not give any information about the bids of other players.

7.4.3 The Complexity of a PET Auction

The algorithms in a PET auction consist of a constant number of synchronized rounds, where the start of each round requires that all servers terminated the previous round. Part of these rounds contain calls to sub-protocols, having a constant number of synchronized rounds each. To estimate the time complexity of a PET auction, we look at the three phases of the auction separately.

- In the setup phase, only the auction servers perform computation. The most expensive computation is the joint encryption of all m messages, which runs in time $O(nm)$. The other algorithms all run in time $O(n)$.
- The bidding phase involves both the servers and the bidders. The bidders only submit bids whereas the servers perform one voting, running in time $O(n)$.
- In the computing phase only the n auction servers are involved. This phase starts with producing a list of accepted bids. They run the SEQ protocol to find the winner and the winning bid. The worst case of a PET auction is when all bidders bid the lowest possible price, a_1. To find the first equality the servers must perform $M(m-1)$ comparisons. They therefore perform $O(Mm)$ runs of PET. PET runs in time $O(n)$ giving a running time of $O(Mmn)$.

In all we conclude that an execution of a PET auction has a total running time of $O(Mmn)$.

Each run of PET each server is required to perform a constant number of exponentiations. In all, running an auction implemented by SEQ-search, each server performs $O(Mmn)$ exponentiations.

Chapter 8

Conclusions

The main focus of this thesis was to construct provably secure protocols, and to provide proofs of completeness, soundness, and zero-knowledge for these protocols.

The proofs of zero-knowledge turned out to be the difficult part. In theory, the notion of zero-knowledge is simple. All zero-knowledge proofs are based on constructing a simulator that uses public and adversarial input to produce an output with a probability distribution indistinguishable from the probability distribution of the output of a real run of the protocol. But, actually constructing the protocol and the simulator giving the necessary probability distributions is a non-trivial and tiresome task. The main obstacle is showing the indistinguishability of probability distributions. We solved this by defining additional simulators whose outputs define probability distributions lying between the two mentioned above. In this manner, we were able to construct a chain of indistinguishable probability distributions, which gave us the desired result: showing indistinguishability of the two probability distributions we started with.

To be able to construct the proof of soundness and the proof of zero-knowledge for a protocol, we needed to tweak and adjust the protocol to fit the needs of the two different proofs. The main obstacle was in changing the protocol to fit one of the proofs without destroying the other.

We succeeded in constructing proofs for completeness, soundness, and zero-knowledge for plain-text comparisons. We did not introduce any new techniques in protocol design, but instead combined known methods to build new protocols. All of the protocols we present build on manipulating cipher-texts. More specifically, we used the multiplicative homomorphic property of the ElGamal encryption scheme. We also used the trick of blinding information to avoid disclosing information when decrypting messages, in such a way that the relevant piece of information is not destroyed.

The protocols with their corresponding proofs can be used to develop similar protocols with proofs, such as [MZSR03]. Even though the proofs must be constructed from scratch, the proofs presented in this thesis give an idea of how to construct new proofs.

References

[AS02] M. Abe and K. Suzuki. $m + 1$-st price auction using homomorphic encryption. In *Conference on Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 115–224. Springer-Verlag, 2002.

[BN00] D. Boneh and M. Naor. Timed commitments. In *Crypto'2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer-Verlag, 2000.

[BOCG93] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *ACM Symposium on Theory of Computing*, volume 25, pages 52–61, 1993.

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *ACM Symposium on Theory of Computing*, volume 20, pages 1–10, 1988.

[Bon98] D. Boneh. The decision diffie-hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium*, volume 1423, pages 48–63. Springer-Verlag, 1998.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BS01] O. Baudron and J. Stern. Non-interactive private auctions. In *Financial Cryptography – Fifth International Conference,*, volume 2339 of *Lecture Notes in Computer Science*, pages 364–377. Springer-Verlag, 2001.

[BST01] F. Boudot, B. Schoenmakers, and J. Traore. A fair and efficient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, 111(1-2):23–36, 2001.

[Cac99] C. Cachin. Efficient private bidding and auctions with an oblivious third party. In *ACM conference on Computer and communications security*, pages 120–127. ACM Press, 1999.

[Cac01] C. Cachin. Distributing trust on the Internet. In *Conference on dependable systems and networks (DSN-2001)*, 2001.

[Can00] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(1):143–202, 2000.

[CCD88] D. Chaum, C. Crepeau, and I. Damgard. Multiparty unconditionally secure protocols. In *ACM Symposium on Theory of Computing*, volume 20, pages 11–19, 1988.

[CDD$^+$99] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Advances in Cryptology, Proceedings of EUROCRYPT 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 311–326. Springer-Verlag, 1999.

[CFGN96] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In *ACM Symposium on Theory of Computing*, pages 639–648, 1996.

[CGH98] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *In Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–218, 1998.

[Cha83] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology, Proceedings of CRYPTO 1982*, pages 199–203. Plenum Press, New York and London, 1983.

[Cha85] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[Cha88] D. Chaum. Blinding for unanticipated signatures. In *Advances in Cryptology, Proceedings of EUROCRYPT 1987*, volume 304 of *Lecture Notes in Computer Science*, pages 227–233, 1988.

[CP92] D. Chaum and T. Pedersen. Wallet databases with observers. In *Advances in Cryptology, Proceedings of CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105, 1992.

[DDN91] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. *SIAM. J. Computing*, 30(2):391–437, 1991.

[Des88] Y. Desmedt. Society and group oriented cryptography: A new concept. In *Advances in Cryptology, Proceedings of CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127. Springer-Verlag, 1988.

[Des89] Y. Desmedt. Threshold cryptosystems. In *Advances in Cryptology, Proceedings of CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer-Verlag, 1989.

[ElG85] T. ElGamal. Public key cryptosystem and a signature scheme. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

[FH94] M. Franklin and S. Haber. Joint encryption and message-efficient secure computation. In *Advances in Cryptology, Proceedings of CRYPTO 1993*, volume 773, pages 266–277. Springer-Verlag, 1994.

[FR95] M. Franklin and M. Reiter. The design and implementation of a secure auction service. In *Proc. IEEE Symp. on Security and Privacy*, pages 2–14. IEEE Computer Society Press, 1995.

[FS87] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology, Proceedings of CRYPTO 1986*, pages 186–194. Springer-Verlag, 1987.

[GGJR00] J. A. Garay, R. Gennaro, C. Jutla, and T. Rabin. Secure distributed storage and retrieval. *Theoretical Computer Science*, 243(1–2):363–389, 2000.

[GK96] O. Goldreich and H. Krawczyk. On the composition of Zero-Knowledge Proof systems. *SIAM Journal on Computing*, 25(1):169–192, 1996.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Science*, 28(2):270 – 299, 1984.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game, or A completeness theorem for protocols with honest majority. In *ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[Gol01] O. Goldreich. *Foundations of Cryptography.* Cambridge University Press, 2001.

[Gol02] O. Goldreich. Zero-knowledge twenty years after its invention, 2002. ECCC Report TR02-063.

[HTK99] M. Harkavy, J.D. Tygar, and H. Kikuchi. Electronic auctions with private bids. In *3rd USENIX workshop on Electronic Commerce*, pages 61–73, 1999.

[Jar01] S. Jarecki. *Efficient Threshold Cryptosystems*. PhD thesis, Massachusetts Institute of Technology, 2001.

[JJ00] M. Jakobsson and A. Juels. Mix and match: Secure function evaluation via ciphertexts. In *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 162–177, 2000.

[JS02] A. Juels and M. Szydlo. A two-server, sealed-bid auction protocol. In *Proceedings of Financial Cryptography*, 2002.

[KHT98] H. Kikuchi, M. Harkavy, and D. Tygar. Multi-round anonymous auction. In *IEEE Workshop on Dependable and Real-Time E-Commerce Systems*, pages 62–69, 1998.

[Kik01] H. Kikuchi. (m+1)st-price auction protocol. In *Proceedings of Financial Cryptography*, volume 2339 of *Lecture Notes in Computer Science*, pages 351–363. Springer-Verlag, 2001.

[KO02] K. Kurosawa and W. Ogata. Bit-slice auction circuit. In *ESORICS*, volume 2502 of *Lecture Notes in Computer Science*, pages 24–38. Springer-Verlag, 2002.

[McE78] R.J. McEliece. A public-key cryptosystem based on algebraic coding theory, 1978. Technical Report, Jet Prop. Lab.

[MSJ02] P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. In *Advances in Cryptology, Proceedings of CRYPT0 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 385–400. Springer-Verlag, 2002.

[MZSR03] M. Marsh, L. Zhou, F.B. Schneider, and A. Redz. Distributed blinding for elgamal and its applications to disseminating secrets. Paper in preparation, 2003.

[NPS99] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic Commerce*, pages 129–139, 1999.

[Oka92] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *Advances in Cryptology, Proceedings of CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer-Verlag, 1992.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosty classes. In *Advances in Cryptology, Proceedings of EUROCRYPT 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, 1999.

[Ped91] T. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology, Proceedings of EUROCRYPT 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer-Verlag, 1991.

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. In *Communications of the ACM*, volume 21, pages 120–126., 1978.

[Sal90] A. Salomaa. *Public Key Cryptography.* Springer-Verlag, 1990.

[Sch91] C. P. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[SM00] D.X. Song and J.K. Millen. Secure auctions in a publish/subscribe system, 2000.

[Wik02] D. Wikström. The security of a mix-center based on a semantically secure cryptosystem. In *INDOCRYPT*, volume 2551 of *Lecture Notes in Computer Science*, pages 368–381. Springer-Verlag, 2002.

[Yao82] A. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science*, pages 162–167. IEEE, 1982.