

# Robust Password-Protected Secret Sharing

Michel Abdalla, Mario Cornejo, Anca Nitulescu, and David Pointcheval

ENS, CNRS, INRIA, and PSL Research University, Paris, France  
{michel.abdalla, mario.cornejo, anca.nitulescu, david.pointcheval}@ens.fr

**Abstract.** *Password-protected secret sharing* (PPSS) schemes allow a user to publicly share this high-entropy secret across different servers and to later recover it by interacting with some of these servers using only his password without requiring *any* authenticated data. In particular, this secret will remain safe as long as not too many servers get corrupted. However, servers are not always reliable and the communication can be altered. To address this issue, a *robust* PPSS should additionally guarantee that a user can recover his secret as long as enough servers provide correct answers, and these are received without alteration. In this paper, we propose new robust PPSS schemes which are significantly more efficient than the existing ones. We achieve this goal in two steps. First, we introduce a *Robust Threshold Secret Sharing Scheme with respect to Random Failures* that allows us to drop the verifiable property of *Oblivious Pseudorandom Functions*. Then, we use this new construction to introduce two new robust PPSS schemes that are quite efficient.

**Keywords:** Password-Protected Secret Sharing, Robust Threshold Secret Sharing Scheme with respect to Random Failures, Oblivious Pseudorandom Functions

## 1 Introduction

Nowadays, cloud storage is quite popular with zettabytes of data spread all over the world. Even if providers give some backup guarantees, they cannot always prevent compromises, and so the data are subject to leakage, with possibly huge consequences if the data are sensitive (financial, economic, medical, etc). Clearly, the provider can encrypt the data before storing them, but this is not an end-to-end protection for the user: the provider itself has access to the data. For better security, the user should encrypt the data before sending them to the cloud. But this leads to a key management issue: Users have to remember their secret keys!

Humans cannot remember large secret keys, but just low-entropy passwords (and not too many). Such a password is definitely not enough to deterministically derive a symmetric encryption key, since a simple off-line dictionary attack would allow the recovery. On the other hand, there are techniques using passwords that are not vulnerable to such off-line dictionary attacks, like *password authenticated key exchange* (PAKE) [BM92]. For these PAKE protocols, the best attacks require the adversary to be on-line, and to make the exhaustive search by interacting with the honest parties, hence the idea to combine PAKE with secret sharing, in order to achieve the best of the two worlds. This allows the recovery of a high-entropy symmetric key by interacting with several servers while just using a low-entropy password [FK00,Jab01], without relying on any authenticated data, where the best attacks are on-line dictionary attacks.

**Password-Protected Secret Sharing.** A  $(t, n)$ -*password-protected secret sharing* (PPSS) is a protocol that allows a user to reconstruct a high-entropy secret from a single (human-memorable) password, by communicating with at least  $t + 1$  honest servers (among  $n$  possible ones).

This framework formalized in [BJSL11] first defines a secure *initialization* phase where the secret is processed together with the password, and some server information, in order to distribute the secret among  $n$  independent servers. Only public information (to enable the later reconstruction) is eventually stored on each server. We however stress that this public information does not have to

be authentic for the later security. Then, during the *reconstruction* phase, the user can recover his secret by interacting with any subset of  $t + 1$  honest servers using just his password. If the public information has been altered, the knowledge of the password will be enough to detect it.

A PPSS protocol satisfies the following properties: (i) the user can retrieve the data by executing the reconstruction protocol with the same password as the one used in the initialization phase and it is guaranteed to succeed as long as at least  $t + 1$  honest servers are available. (ii) An attacker who controls up to  $t$  servers cannot learn any information about the secret other than doing an on-line dictionary attack with another server. Two additional properties have been defined: *Soundness* and *Robustness*. The first guarantees that even if the adversary compromises all the servers, it cannot make the user reconstruct a secret different from the one originally stored by the user. On the other hand, robustness guarantees the recovery of the secret as long as the user communicates without disruptions with at least  $t + 1$  honest servers.

Additionally, we point out that the adversary can control all the communication network by blocking, delaying, altering, or duplicating any flow. As such, no server is trusted, and no PKI is assumed either, since the only authenticated data we allow is a short password that the user can remember.

**Contributions.** Our PPSS protocol follows the methodology from [JKK14]: it is based on the use of *pseudorandom functions* (PRFs) evaluated on the password to mask the shares of the secret. These evaluations are performed with servers that own the PRF keys, in an oblivious way, hence the so-called *oblivious pseudorandom functions* (OPRFs).

Our first contribution is to point out that, in order to achieve robustness in such a PPSS protocol, one does not need to distinguish between correct and incorrect shares on individual shares. When using oblivious PRFs (OPRFs) to generate the shares, in general, the shares are either correct (honest server without disruption) or random (dishonest server or alteration of the flows). As a consequence, this is enough for the reconstruction technique to be able to select the correct shares, where their number is high enough, under the assumption that the invalid shares are random.

Taking this into account and based on the assumption that at the reconstruction phase the shares are either correct or random (which will be proven for our OPRFs) we build a new efficient  $(t_\ell, t_r, n)$ -Robust Threshold Secret Sharing Scheme *with respect to Random Failures* that guarantees to efficiently identify the correct values (and reconstruct the secret) if at least  $t_r$  shares are correct. However, if at most  $t_\ell$  shares are correct, the protocol *leaks* no information about which shares are correct. It is indeed important that not too few correct shares can be detected as correct as this could result in offline dictionary attacks. For instance, in the case where shares could be individually checked, a dishonest server could mount an offline dictionary attack. With our new primitive, even  $t_\ell$  corrupted servers cannot perform an offline dictionary attack as they would still need to interact with at least one additional server.

We use this  $(t_\ell, t_r, n)$ -Robust Threshold Secret Sharing Scheme with respect to Random Failures to construct a sound and robust PPSS scheme. More precisely, it allows us to design a generic  $(t_\ell, t_r, n)$ -PPSS scheme that ensures the reconstruction of the secret as long as the user communicates with at least  $t_r$  honest servers without communication alteration (robustness), still preventing any leakage of information in presence of up to  $t_\ell$  corrupted servers.

While in the same vein as [JKK14], our technique takes advantage of the *robustness with respect to random failures* of the threshold secret sharing scheme: it eliminates some strong requirements on the OPRF (namely the so-called *verifiability*). This allows us to avoid many zero-knowledge proofs, and therefore to construct much more efficient OPRF protocols.

We propose two OPRF constructions: The first one, in the random oracle model (ROM) [BR93], is based on the CDH assumption; the second one, in the standard model with a CRS, that obviously evaluates the Naor-Reingold PRF [NR97], based on the DDH assumption.

For the former, efficiency is quite similar to the one in [JKK14], since we just avoid a NIZK, that was efficient on the ROM. However, for the latter, we compare very favorably to their oblivious evaluation of the Naor-Reingold PRF: thanks to the relaxation of our OPRF, our protocol simply uses ElGamal encryption [ElG85] in prime order groups with simple zero-knowledge proofs, whereas [JKK14] has to work in composite order groups with Paillier encryption [Pai99] and more complex zero-knowledge proofs. Our communication load is at most half of theirs, while still being asymptotically the same.

**Related Work.** The first formal definition of Password Protected Secret Sharing was introduced by Bagherzandi *et al.* [BJS11]. They proved their scheme secure in the random oracle model assuming an additional PKI. Moreover, if an adversary is able to obtain the keypair of one server, the adversary can perform an offline attack. Later Camenisch *et al.* [CLN12] introduce a protocol of password-authenticated secret sharing that also assumes a PKI and only two servers. Both protocols contradict the requirement to be *password-only*, since they assume additional authenticated data. Whereas this assumption of a safe PKI makes sense during the initialization phase, which can be run in a safe environment, it is not reasonable to make this assumption for the reconstruction phase, which will be executed many times on various weak devices.

Later, Camenisch *et al.* [CLLN14] introduce a  $(t, n)$ -PPSS (called TPASS, for Threshold Password-Authenticated Secret Sharing) in the Universal Composability (UC) framework [Can01] that is password-only during the reconstruction phase. However, in this protocol all servers jointly validate if the password matches or not. Yi *et al.* in [YHCL15] propose a more efficient TPASS than [CLLN14] based on distributing the password, a secret and a digest of the secret. Nevertheless, in the recovering protocol, at least  $t$  servers execute a broadcasting protocol to generate and return the ElGamal encryptions of both the secret and the digest. Then the users verify it matches. Camenisch *et al.* in [CLN15] present a very lightweight protocol with a similar construction to our work, yet with differences. Each server holds a key that is refreshed at regular time intervals that allows them to recover from corruption through a non-interactive key refresh protocol making it unfeasible to perform an offline attack unless all servers are corrupted at the same time. Since this protocol does not rely on robust secret sharing scheme nor zero-knowledge, it is not possible for the protocol to identify which shares are valid. Then, if in the end the validation fails, the protocol must restart with a different set of servers contradicting the requirement of *robustness* and leading to a possible Denial-of-Service (DoS) attack.

Jarecki *et al.* [JKK14] have been the first to design a PPSS scheme that is both *password-only* during the reconstruction phase and *robust*, to avoid too easy DoS attacks. It makes use of a *Verifiable Oblivious Pseudorandom Function* (VOPRF) that assures robustness by providing computation guarantees from the servers: the user actually knows which server has tried to cheat, or which communication links have been altered.

The solution we propose in this paper follows the previous strategies: we use a secret sharing scheme to divide the user’s secret. Each server stores one share masked by the pseudorandom value computed in an oblivious way on the user’s low-entropy password with the server’s PRF key. Because of this, the server cannot know the value of its share at the reconstruction phase, leading only to two options: Either the real share or a random one.

Our  $(t_\ell, t_r, n)$ -robust threshold secret sharing scheme with respect to random failures allows the user to execute the PPSS protocol in a robust way: If the number of correct servers’ answers is above the threshold  $t_r$ , the user can efficiently identify the valid ones. If the number of answers

is below another threshold  $t_\ell$ , no information is leaked. The main difference to [JKK14] is in the way to achieve robustness: We ask a bit more from the secret sharing scheme, but much less from the OPRF, allowing more efficient constructions for the latter, which high improves on the global efficiency.

## 2 Security Model

In order to analyze the security of Password-Protected Secret Sharing (PPSS) protocols, we first provide a formal description of the security model. This is a game-based security definition, in the same vein as [BR94, BR95] for key distribution schemes and [BPR00] for password-authenticated key exchange. It adapts the PPSS definition from [BJS11] and the security model from [JKK14]. We define security in terms of a *key derivation mechanism* or indistinguishability of the actual secret from a random one, as in [JKK14], since our goal is to later use the secret as a symmetric key. In particular, we do not want to rely on a PKI or any authenticated public values, hence our model description is similar to security models for password-authenticated key exchange protocols.

### 2.1 Password-Protected Secret Sharing

We first describe the participants and the two steps of a PPSS protocol.

**Participants and Parameters.** We assume a fixed set of participants involved in the protocol, each of which is either a user or a server. The set of all participants is the union of the nonempty disjoint and finite sets,  $\mathbf{User} \cup \mathbf{Server}$ .

Each user  $U \in \mathbf{User}$  holds two threshold values  $t_\ell$  and  $t_r$ , where  $t_r$  is the number of shares required to *recover* the secret and  $t_\ell$  the maximum number of shares that can be known without *leaking* any information about the secret, as well as some password  $\mathbf{pw}$  chosen independently and uniformly from a dictionary  $\mathcal{D}$  of cardinality  $\#\mathcal{D}$ .

Each server  $S \in \mathbf{Server}$  holds a secret key  $\mathbf{sk}$ , and possibly an associated public key  $\mathbf{pk}$ . However we stress that even if there is a public key  $\mathbf{pk}$ , authenticity cannot be assumed *a priori* during the reconstruction phase since users will just have to remember their passwords and nothing else that would be required to authenticate additional data.

**Initialization.** The goal of the user  $U$  is to generate a key  $K$  so that he later can recover it with the help of  $t_r$  servers among  $n$  available servers, just using his password. He thus runs an initialization protocol with  $n$  servers, using their public keys, his password and some random coins. He ends up with a random key  $K$  and some additional information  $\mathbf{PInfo}$ : nobody else than  $U$  has any information about  $K$ , however  $\mathbf{PInfo}$  can be made public.

**Secret Reconstruction.** While the initialization phase assumes that all the servers are honest, the public keys are authentic, and the data are not modified during the communication, for the reconstruction phase, the adversary controls the network and can forward, alter, delay, replay, or delete any message. The adversary can also provide fake public data: nothing is authenticated anymore!

Anyway, just using his password, the user  $U$  should be able to recover  $K$ , with the help of the servers, in a verifiable/robust way, even if the public keys in  $\mathbf{PInfo}$  are not guaranteed to be correct.

Each participant (either user or server) can run several executions of the protocol, possibly concurrently, we thus denote an instance  $i$  of player  $P$  as  $P^i$ . Each instance may be activated once only: the adversary is given oracle accesses to interact with all the user's and server's instances that are stateful interactive polynomial-time Turing machines.

## 2.2 The adversarial model

During the reconstruction phase, the adversary is given total control of the network: it can forward, alter, delay, replay, or delete any message sent by any player. To model this ability, it is given access to the following oracles:

- $\text{Execute}(U^i, \{S_k^{j_k}\})$ : This query models a passive attack. This makes an instance  $U^i$  to interact with several instances of servers  $\{S_k^{j_k}\}$  as they would do during the reconstruction protocol. The adversary eventually gets back the entire transcript;
- $\text{Send}(P^i, m)$ : This query models an active attack. This sends a message  $m$  to the instance  $P^i$ . This message  $m$  can be a fresh message, or a replay, a forward, etc. A specific message  $\text{Start}_k^j$  to a user’s instance  $U^i$  makes it initiate a communication with the server’s instance  $S_k^j$ .

The security goal is to guarantee the privacy of the secret key  $K$  reconstructed by the user. This is usually modeled by an indistinguishability game, with access to a  $\text{Test}$ -query, where  $b$  is a global secret random bit:

- $\text{Test}(U^i)$ : This query characterizes the indistinguishability of the key  $K$  computed by instance  $U^i$ . If this instance has not yet completed the reconstruction, the answer is  $\text{UNDEFINED}$ ; if the reconstruction failed, the answer is  $\perp$ ; otherwise, the answer is either the real reconstructed value if  $b = 1$  or a random one (always the same for user  $U$ , but independent of the real one) if  $b = 0$ .

The adversary eventually outputs its guess  $b'$  for the bit  $b$ , to show its ability to distinguish real multiple executions of the protocol from ideal executions: one can note that in the random case ( $b = 0$ ), which models the ideal executions, a user  $U$  always terminates with the same key, or fails. This means that the adversary should not be able to make him accept a different key.

In addition to control the network and the communications, the adversary can corrupt servers, and get back their secret keys, due to, e.g., a poorly-administered server, compromise of a host computer, or cryptanalysis. This is modeled by the  $\text{Corrupt}$ -query:

- $\text{Corrupt}(S_k)$ : This outputs the secret key  $\text{sk}_k$  of the server  $S_k$ .

## 2.3 Semantic Security

**Definition.** Once the initialization phase is completed for many users, with random passwords uniformly and independently drawn from a dictionary  $\mathcal{D}$ , the security game models the indistinguishability of the secret keys, a.k.a. *semantic security*, the adversary can ask as many oracle queries ( $\text{Execute}$ ,  $\text{Send}$ ,  $\text{Test}$ , and  $\text{Corrupt}$ ), as it wants, in any order it wants, in order to guess the bit  $b$ : it outputs its guess  $b'$ . We measure the quality of an adversary  $\mathcal{A}$  by its advantage

$$\text{Adv}(\mathcal{A}) = \Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0] = 2 \times \Pr[b' = b] - 1.$$

**Trivial Attacks.** Two kinds of “on-line dictionary attacks” are unavoidable:

- if the adversary guesses the correct password, it will be able to reconstruct the actual secret  $K$  after  $q_c$  corruption queries and  $t_r - q_c$  interactions with honest servers. Even after just  $t_\ell - q_c$  interactions, it may come up with  $t_\ell$  shares, which may leak some information about the actual secret key: it thereafter asks for an  $\text{Execute}$ -query, and tests the instance involved in this session, to distinguish the real case from the random case. Its success probability is however upper-bounded by  $q_s/(t_\ell - q_c) \times 1/\#\mathcal{D}$ , where  $q_s$  is the number of server instances involved during the attack,  $q_c$  the number of  $\text{Corrupt}$ -queries, and  $\#\mathcal{D}$  the size of the password dictionary.

- whereas the initialization phase was assumed to be done with authentic server public keys, for the reconstruction phase, the adversary can send totally fake public keys in  $\text{PlInfo}$  that it generated itself from a randomly chosen password  $\text{pw}$ . It thus also knows the secret keys and can simulate the view of the user by emulating all the servers. If the password guess was correct, the user should successfully terminate, whereas a wrong guess would lead to inconsistent information. Its success probability is therefore upper-bounded by  $q_u/\#\mathcal{D}$ , where  $q_u$  is the number of user instances involved in the attack.

## 2.4 Secure PPSS

As a consequence, we will say a  $(t_r, n)$ -PPSS scheme is  $(t_\ell, \varepsilon, t)$ -secure if for any adversary  $\mathcal{A}$ , running within time  $t$ , asking at most  $q_c$  **Corrupt**-queries and invoking at most  $q_u$  user instances and  $q_s$  server instances,

$$\text{Adv}(\mathcal{A}) \leq \frac{1}{\#\mathcal{D}} \times \left( \frac{q_s}{t_\ell - q_c} + q_u \right) + \varepsilon.$$

In [JKK14], they proposed such a protocol that achieves the optimal  $t_\ell$ -security, for  $t_\ell = t_r$ , but at the cost of verifiable oblivious pseudorandom functions. Our goal is to build much more efficient protocols, possibly lowering the security level:  $t_\ell = 2t_r - n - 2$ .

Before going into more details about our constructions, let us review the required or expected properties for a PPSS, initiated with a password  $\text{pw}$  and secret  $K$  for a user  $U$ . Some are already covered by the security model, some offer additional features:

- **Correctness.** To be viable, a password-protected secret sharing must guarantee that at least  $t_r$  honest servers should allow the user that plays with his password  $\text{pw}$  to recover his secret  $K$ .
- **Soundness.** As already guaranteed by our security model, when a user terminates with a key  $K'$ , this is the correct key  $K$ . More precisely, when playing with the correct password  $\text{pw}$ , the user ends up with  $K' \in \{K, \perp\}$  without any assumption about the communications and the server behaviors: there are no authenticated channels nor any authenticated data.
- **Robustness.** Due our communication model messages can be lost, modified, or even totally faked by the adversary. Of course, one cannot avoid Denial-of-Service (DoS) attacks, since the adversary can simply block any communication. However, an important property, already required by [JKK14], is the so-called *robustness*: even if the adversary alters many messages, as soon as  $t_r$  communications with servers are unmodified the user can *efficiently* recover its secret.

The general issue with robustness is that when the user has interacted with  $n$  servers but only  $t_r$  shares are valid, the cost of try all the  $t_r$  subsets is exponential! In [JKK14], they addressed this issue by making some inner protocols secure against malicious servers, with additional zero-knowledge proofs of honest behavior, but this is at a high communication cost. Our goal is to provide this property at a much lower cost.

## 3 High-Level Description

We review the well-known computational assumptions in the Appendix A and the classical building blocks in the Appendix B. But here, we present a high-level description of the PPSS protocol, to motivate the needs. Our general construction follows the one from [JKK14], with first an initialization phase and then a reconstruction phase.

Each server  $S_k$  owns a key-pair  $(\text{sk}_k, \text{pk}_k)$  that defines a PRF  $F_k$ , with public parameters defined by  $\text{pk}_k$  and a secret key defined by  $\text{sk}_k$ . For a password  $\text{pw} \in \mathcal{D}$ , the user asks for an oblivious evaluation of  $\pi_k = F_k(\text{pw})$  to  $n$  servers, where  $\Pi = (\text{pk}_k)_k$  is the tuple of the public keys of the involved

servers. The secret key  $K$  is then split into shares  $(s_1, \dots, s_n)$  and some extra public information **PlInfo**, specific to the user is derived from it and distributed to all servers. This information allows the user to later recover his secret, in a robust way.

We stress that during this initialization phase,  $(\mathbf{pk}_k)_k$  are all the true public keys, and  $(\pi_k)_k$  are the correct evaluations of the PRFs. However, during the reconstruction phase, the values provided by the servers are sent through an insecure channel and they might be altered by the adversary: the user interacts with at least  $t_r$  servers, that provide him **PlInfo**, and help him to compute each  $\pi_k = F_k(\mathbf{pw})$  in an oblivious way. We assume that the user received the same value **PlInfo** from at least  $t_r$  servers, and then the user keeps the majority value. Using **PlInfo** and enough evaluations  $\pi_k$ , the user can extract enough shares among  $(s_1, \dots, s_n)$  and reconstruct a value  $K$ . He can then verify whether this is the expected secret key.

We can note that there are two crucial tools for this generic construction:

- a pseudorandom function  $F$  that can be evaluated in an oblivious way: the server input is the secret key  $\mathbf{sk}$  and the user input is the password  $\mathbf{pw}$ , and the user only gets the output  $F_{\mathbf{sk}}(\mathbf{pw})$ , but none of the players learn any additional information about the other player’s input;
- a  $(t_\ell, t_r, n)$ -threshold secret sharing scheme that allows to share a secret among  $n$  players so that any subset of  $t_r$  shares allows efficient reconstruction of the secret, while  $t_\ell$  shares do not leak any information.

An additional non-malleable commitment scheme [DIO98] will provide the soundness, by limiting the ability for an adversary to present a modified **PlInfo**.

However, in order to achieve the robustness to the PPSS protocol, we need to make sure that when  $t_r$  communications with the servers are unmodified, the user can reconstruct the secret: either one can detect alterations of the communications during the oblivious evaluations of the PRF, which is the approach followed by [JKK14] with *Verifiable Oblivious PRFs* (VOPRFs), or one can efficiently reconstruct a secret from any set of shares that contains at least  $t_r$  valid shares, which is our approach with *Robust Threshold Secret Sharing Scheme*. Actually, a weaker notion of robustness for the secret sharing scheme is enough since with the oblivious PRF, if the server is honest and the flow unmodified, the user gets the correct share, but if the server cheats or the flow is modified, the user gets a truly random share, hence our notion of *robustness with respect to random failures*.

## 4 A Robust Threshold Secret Sharing Scheme with Respect to Random Failures

Our technique is generic, and so we start from any threshold secret sharing scheme (see Appendix B), with two algorithms **ShareGen** and **Reconstruct** that respectively share a secret and reconstruct it, to which we will add this new robustness feature, that is enough to get a robust PPSS scheme. Let us first give the intuition of the technique, and we then explain how we can make it in practice.

### 4.1 Intuition

The valid shares are denoted  $(s_1, \dots, s_n)$  and the fingerprints of these shares  $(\sigma_1, \dots, \sigma_n)$ . At the same time of the share distribution, the product  $\mathcal{S}$  of all fingerprints modulo an integer  $N$  is published. In order to reconstruct the secret, having received  $m$  candidate shares, one computes its fingerprints  $(\tau_1, \dots, \tau_m)$  and the product of them  $\mathcal{T} = \prod \tau_i$ . The ratio  $\mathcal{T}/\mathcal{S} \bmod N$  will cancel out the fingerprints of all the correct share values leading to the ratio  $\mathcal{T}'/\mathcal{S}' \bmod N$ , where  $\mathcal{S}'$  is the product of the fingerprints of the valid shares that the receiver does not have in the list of candidates and  $\mathcal{T}'$  the product of the fingerprints of the candidates that are invalid. From  $\mathcal{S}'$ , one

could easily check for every candidate, whether it is in this product or not, and therefore identify which candidate is correct or not.

Of course,  $\mathcal{S}'$  has to be computed with good precision to allow the last verification, but not too much in order to avoid individual checks or any unnecessary leakage of information. The computations are thus performed modulo  $N$ , for a well-chosen value.

## 4.2 Description

We now explain how one can extract  $\mathcal{S}'$  when the candidates are either correct or random.

**Initialization.** We assume we have a set of  $n$  initial values  $(s_1, \dots, s_n)$ , and their fingerprints are  $k$ -bit strings  $(\sigma_1, \dots, \sigma_n)$ . We will additionally assume that fingerprints are randomly distributed between  $2^{k-1}$  and  $2^k - 1$  when the inputs are random. We can use  $\sigma_i = F(s_i) = 1 \parallel (s_i \bmod 2^{k-1})$ : if  $s_i$  is a uniformly distributed large bitstring,  $s_i \bmod 2^{k-1}$  is a  $(k-1)$ -long bitstring uniformly distributed, hence  $\sigma_i$  is uniformly distributed between  $2^{k-1}$  and  $2^k - 1$ .

In the following, we will be given a set of  $m$  candidate shares, whose fingerprints are  $(\tau_1, \dots, \tau_m)$ : these candidates are either correct or random, and so are their fingerprints. From this set of candidate shares, if at least  $t_r$  are correct, we want efficiently to identify the correct values (to *recover* the secret in a threshold secret sharing scheme, hence  $t_r$ ). However, if at most  $t_\ell$  are correct, the protocol should not *leak* any information about which candidates are valid and which are not (hence  $t_\ell$ ).

From the initial set  $(\sigma_1, \dots, \sigma_n)$  of size  $n$  and the threshold  $t_r$ , one chooses a prime number  $N$  such that  $2^{2k(n-t_r)+1} < N \leq 2^{2k(n-t_r)+2}$ , computes the product  $\mathcal{S} = \prod_{i=1}^n \sigma_i \bmod N$ , and publishes  $\text{SSInfo} = (\mathcal{S}, N)$ .

**Reconstruction.** Given the  $\text{SSInfo} = (\mathcal{S}, N)$  and fingerprints  $(\tau_1, \dots, \tau_m)$  of the  $m \leq n$  candidates, which are either correct (at least  $t_r$  of them) or random (all the other ones), one computes the ratio  $\gamma = \prod_{i=1}^m \tau_i / \mathcal{S} \bmod N$ , which can be written as  $\gamma = \mathcal{T}' / \mathcal{S}' \bmod N$ , where  $\mathcal{T}'$  is the product of the fingerprints of the invalid candidates and  $\mathcal{S}'$  the product of the fingerprints of the values that are not in the list of the candidates, both over the integers. Then, we know that  $\mathcal{T}' \leq 2^{k(m-t_r)} - 1 \leq 2^{k(n-t_r)} - 1$  and  $\mathcal{S}' \leq 2^{k(n-t_r)} - 1$ .

Using the following result from [FSW03], we can recover both  $\mathcal{S}'$  and  $\mathcal{T}'$  under appropriate conditions.

**Theorem 1.** (Numerical Rational Number Reconstruction) *Let  $z = \frac{x}{y} \bmod N$  such that  $-X \leq x \leq X$  and  $0 < y \leq Y$ . If  $N$  is relatively prime to  $y$  and  $2XY < N$  then the solution is unique and it is possible to recover  $x$  and  $y$  efficiently by using two-dimensional lattice theory.*

Considering  $X = 2^{k(n-t_r)} - 1$  and  $Y = 2^{k(n-t_r)} - 1$ , we indeed have  $2XY \leq 2(2^{k(n-t_r)} - 1)(2^{k(n-t_r)} - 1) < N$  and  $X > 0, Y > 0$ , hence we can efficiently recover  $\mathcal{T}'$  and  $\mathcal{S}'$  from  $\gamma$ . Thereafter, by testing  $\gcd(\mathcal{T}', \tau_i) \stackrel{?}{=} \tau_i$ , one learns whether  $\tau_i$  is among the invalid candidates or not.

We insist that the candidates are either correct or random (and independent). Therefore, it is quite unlikely that a valid candidate already reduced divides this product. To be more precise, according to [CN00] that analyzes the collision-intractability notion introduced by [GHR99], this probability is indeed small: much more than 1000 numbers of 128 bits are required so that one of them divides the least common multiple of the others with probability 1%. Since  $n$  is much lower than 100 in general, over 128-bit fingerprints, this probability is very small.

**Information Leakage.** On the opposite, we would like to evaluate the information leaked by  $\mathcal{S}$  when there are at most  $t_\ell$  valid values:  $\gamma = \mathcal{T}' / \mathcal{S}' \bmod N$ , with  $\mathcal{S}' \geq 2^{(k-1)(n-t_\ell)}$ . If this bound is

much greater than  $N$ , there is still some entropy, and no way to get any additional information about any individual fingerprint, because of the modular reduction.

We need  $2^{(k-1)(n-t_\ell)} \geq N \times 2^k$  to still have  $k$  bits of entropy (a security in  $2^{-k}$ ), which is achieved if  $2^{(k-1)(n-t_\ell)} \geq 2^{2k(n-t_r)+2+k}$  since  $N \leq 2^{2k(n-t_r)+2}$ . This means that with  $(k-1)(n-t_\ell) \geq 2k(n-t_r)+k+2$ , the required inequality holds. One can rewrite it as  $k \times ((n-t_\ell) - 2(n-t_r) - 1) \geq n-t_\ell+2 \geq n+2$ . When  $k$  is bigger than  $n+2$ , this essentially means  $(n-t_\ell) - 2(n-t_r) - 1 = 2t_r - n - t_\ell - 1 \geq 1$ : by choosing  $t_\ell = 2t_r - n - 2$ , we are safe. More concretely, with the above bounds, we have shown that for any set of  $t_\ell$  values (correct or incorrect),  $\mathcal{S}' > N$  and so  $\gamma$  can be written as a ratio modulo  $N$ . Hence no one can distinguish a set of  $t_\ell$  correct values from a set of  $t_\ell$  random values. This is the argument we will use in the proof of our PPSS protocols.

### 4.3 Application to Shamir's Secret Sharing Scheme

One can of course apply this technique to the well-known Shamir's secret sharing scheme, to build a  $(t_\ell, t_r, n)$ -robust threshold secret sharing scheme with respect to random failures for a secret  $s \in \mathbb{Z}_p$ , where  $p \geq 2^{2k}$ , and  $k$  is the bit-length of the fingerprint: we can safely use  $k = 128$  when  $t_r$  is below a few dozens, which allows to share a 128-bit secret, or even a 256 AES secret key.

## 5 Our Password-Protected Secret Sharing Protocols

Thanks to our new primitive of  $(t_\ell, t_r, n)$ -robust threshold secret sharing scheme with respect to random failures, we do not need to use a Verifiable Oblivious PRF, but an Oblivious PRF only, which guarantees correct or random evaluations. With a classical threshold secret sharing scheme, as in [JKK14], this is not possible to avoid the verifiability of the OPRF. This verifiability is at the cost of zero-knowledge proofs of honest behavior of the servers.

We can now describe our general structure of PPSS protocol, using an OPRF as black-box, assuming it either provides the correct evaluation if the flows with an honest server have not been altered, or a random value as soon as the adversary tries to cheat.

We thereafter provide two instantiations, with two appropriate OPRFs, in the same vein as the ones proposed in [JKK14], using similar computational assumptions (see Appendix A):

- the first OPRF relies on the CDH evaluation, similar to the protocol 2HashDH from [JKK14], but without NIZK. It leads to a PPSS construction quite similar to [JKK14], since the random oracle model allowed them to use efficient NIZKs for the verifiability.
- the second OPRF is an oblivious evaluation of the Naor-Reingold PRF [NR97], in the standard model. Then, in the PPSS, the gain of the zero-knowledge proofs by the server is quite significant, but we still need some proofs from the user, to ensure the input is in the correct domain, otherwise there is no guarantee on the PRF property.

### 5.1 General Description

As already presented in the high-level description, our protocols are in two phases: the initialization phase which is assumed to be executed in a safe environment with reliable communications and correct inputs from the servers, and the reconstruction phase during which the password only is considered correct, while all the other inputs can be faked by the adversary.

**Initialization.** We assume that each server  $S_k$  owns a key pair  $(\text{sk}_k, \text{pk}_k)$  that defines a PRF  $F_k$ , with public parameters defined by  $\text{pk}_k$  and a secret key defined by  $\text{sk}_k$ , that admits an OPRF protocol to allow a user with input  $m$  to evaluate  $F_k(m)$  without leaking any information on  $m$  to the server. We additionally use a  $(t_\ell, t_r, n)$ -robust threshold secret sharing scheme with respect to random failures, where we can assume that  $t_r = 2n/3 + 1$  and  $t_\ell = n/3$  (which is compatible with our previous construction), and a commitment scheme  $\text{Commit}$  (see Appendix B). The user  $U$  first chooses a secret password  $\text{pw}$ :

1. the user interacts with  $n$  servers to obliviously evaluate  $\pi_k = F_k(\text{pw})$ , and  $\Pi = (\text{pk}_k)_k$  is the tuple of the public keys of the involved servers;
2. for a random value  $R = K \| r$ , where  $K$  is the random secret key the user wants to share and  $r$  some random coins. The user generates  $(s_1, \dots, s_n, \text{SSInfo}) \leftarrow \text{ShareGen}(R)$ , so that any subset of  $t_r$  shares among  $\{s_1, \dots, s_n\}$  can efficiently reconstruct  $R$ ;
3. then, the user builds  $\sigma_k = \pi_k \oplus s_k$ , for  $k = 1, \dots, n$ , and sets  $\Sigma = (\sigma_k)_k$ ;
4. the user generates  $C = \text{Commit}(\text{pw}, \Pi, \Sigma, \text{SSInfo}, K; r)$ . We denote by  $\text{PInfo} = (\Pi, \Sigma, \text{SSInfo}, C)$  the public information that the user will need later to recover its secret  $K$ ;
5. the user thus gives  $\text{PInfo}$  to all the servers.

We stress that during this initialization phase, all the values of  $\Pi$  are the real public keys and  $(\pi_k)_k$  are the correct evaluations of the PRFs. On the opposite, during the reconstruction phase, all the values in  $\text{PInfo}$  will be provided by the servers, but through the adversary, who might alter them.

**Reconstruction.** For the reconstruction, the user interacts with at least  $t_r$  servers, that provide him  $\text{PInfo} = (\Pi, \Sigma, \text{SSInfo}, C)$ , and help him to compute  $\pi_k = F_k(\text{pw})$  for several values of  $k$ , using  $\text{pk}_k$  from  $\Pi$ . No information is trusted anymore, and so the reconstruction phase perform several verifications:

1. the user first limits the oblivious evaluations of  $\pi_k = F_k(\text{pw})$  to the servers that sent the same majority tuple  $\text{PInfo} = (\Pi, \Sigma, \text{SSInfo}, C)$ . If the number of such servers is less than  $t_r$ , one aborts with  $K \leftarrow \perp$ ;
2. for all these  $\pi_k$  (or similarly, all the  $k$  he kept), the user computes  $s_k = \sigma_k \oplus \pi_k$ , using  $\sigma_k$  from  $\Sigma$  (from  $\text{PInfo}$ );
3. using these  $\{s_k\}$  with at least  $t_r$  correct shares, and  $\text{SSInfo}$  (from  $\text{PInfo}$ ), with robust threshold secret sharing scheme, the user reconstructs the shared secret  $R$  (or aborts with  $K \leftarrow \perp$  if the reconstruction fails);
4. the user parses the secret  $R$  as  $K \| r$ , and checks, from  $\text{PInfo}$ , whether  $C = \text{Commit}(\text{pw}, \Pi, \Sigma, \text{SSInfo}, K; r)$ ;
5. if the verification succeeds,  $K$  is the expected secret key, otherwise the user aborts with  $K \leftarrow \perp$ .

## 5.2 Protocol I: in the Random Oracle Model

Our first instantiation is based on CDH-like assumptions in the random oracle model. The arithmetic is in a finite cyclic group  $\mathbb{G} = \langle g \rangle$  of prime order  $q$ . We need a full-domain hash function  $H_1$  onto  $\mathbb{G}$ , and another hash function  $H_2$  onto  $\{0, 1\}^{\ell_2}$ . Since we already are in the random oracle model for the PRF, we can implement the commitment scheme with a simple hash function  $H_3$  onto  $\{0, 1\}^{\ell_3}$ :  $C = \text{Commit}(\text{pw}, \Pi, \Sigma, \text{SSInfo}, K; r) := H_3(\text{pw}, \Pi, \Sigma, \text{SSInfo}, K, r)$ , which allows a better efficiency.

For a private key  $\text{sk} = x \in \mathbb{Z}_q$ , we consider the pseudorandom function  $F_x(m) = H_2(m, g^x, H_1(m)^x)$ , for any bitstring  $m \in \{0, 1\}^*$ , where the public key is  $\text{pk} = y = g^x$ . In the Appendix C, we prove this is indeed a PRF. In addition, it admits an oblivious evaluation, that does not leak any information, thanks to the three simulators  $\mathcal{S}im$ ,  $\mathcal{S}im_U$  and  $\mathcal{S}im_S$ , as presenter in Figure 1:  $\mathcal{S}im$  simulates an

honest transcript,  $\mathcal{S}im_U$  simulates an honest user interacting with a malicious server, and  $\mathcal{S}im_S$  simulates an honest server with a malicious user. These simulators will be used by our simulator in the following security proof. They generate perfectly indistinguishable views to the adversary, but they require  $\text{CDH}_g(y, \cdot)$  and  $\text{DDH}_g(y, \cdot, \cdot)$  evaluation, and thus oracle access when the secret keys are not known. Since the indistinguishability of the PRF relies on the  $\text{CDH}_g(y, \cdot)$  assumption, the overall security relies on the One-More Gap Diffie-Hellman (OMGDH) assumption (see Appendix A), as shown in the last step of the proof.

**Theorem 2.** *For any adversary  $\mathcal{A}$ , against the Protocol I, that corrupts no more than  $q_c$  servers, involves at most  $q_s$  instances of the servers,  $q_u$  instances of the user, and asks at most  $q_1, q_2, q_3$  queries to  $H_1, H_2, H_3$ , respectively*

$$\text{Adv}(\mathcal{A}) \leq \left( q_u + \frac{3q_s}{n - 3q_c} \right) \times \frac{1}{\#\mathcal{D}} + n \times \text{Succ}^{\text{omgdh}}(q_1, q_s, t, n \cdot q_u + q_2) + (q_3^2 + 2) \cdot 2^{-\ell_3}.$$

**Security Proof.** The complete and detailed proof of the Theorem is given in Appendix C. The rough idea is the following: in the real attack game, we focus on a unique user, against a static adversary (the corrupted servers are known right after the initialization, and before any reconstruction attempt). All the parameters are honestly generated, the simulator knows the secret informations to answers the queries, and two random keys  $K_0$  (random) and  $K_1$  (real), as well as a bit  $b$ , are selected randomly to answer Test-queries. In the final game, we simulate all the answers to the adversary without using a password. A random value will be chosen at the very end of the simulation and used as a password in order to decide if some bad events should have occurred, which will immediately upper-bound the advantage of the adversary.

We first modify the way Execute-queries are answered, using  $\mathcal{S}im$  that perfectly simulates honest transcripts user-servers, and we set user's key to  $K_1$ .

Then, we deal with Send-queries to the honest user, trying to exclude the cases of a fake public information  $\text{PInfo}'$  (sent by the majority of servers): first, we do as before if the commitment  $C'$  in  $\text{PInfo}'$  is different from the expected value  $C$  generated during the initialization, but eventually we set  $K \leftarrow \perp$ . This would just make a difference for the adversary if  $C'$  indeed contains the good password  $\text{pw}$ , which is defined as the event  $\text{PWinC}$ . This event  $\text{PWinC}$  can be evaluated using the list of queries asked to  $H_3$ . Then, a similar argument applies when a wrong  $\text{PInfo}'$  is sent, but with a correct  $C$ , under the binding propriety of the commitment  $H_3$ .

<u>User</u>	<u>Server</u>
$m$	$\text{pk} = y = g^x$ $\text{sk} = x$
$\alpha \xleftarrow{\$} \mathbb{Z}_q^*, A \leftarrow H_1(m)^\alpha$	$\xrightarrow{A}$
If $B = 1$ , then abort	$\xleftarrow{B} \quad B \leftarrow A^x$
$C \leftarrow B^{1/\alpha}, R \leftarrow H_2(m, y, C)$	

$\mathcal{S}im$	$\mathcal{S}im_U$	$\mathcal{S}im_S$
$\alpha \xleftarrow{\$} \mathbb{Z}_q^*$	$A \xleftarrow{\$} \mathbb{G}$	$\xrightarrow{A}$
$A \leftarrow g^\alpha$	$\xrightarrow{A}$	$\xrightarrow{A}$
$\xrightarrow{A}$	$\xleftarrow{B}$	$\xleftarrow{B}$
$\xleftarrow{B} \quad B \leftarrow y^\alpha$	$\neg \text{DDH}_g(y, A, B)$	$B \leftarrow \text{CDH}_g(y, A)$
	$\implies \text{fail}$	

**Fig. 1.** Secure Oblivious Evaluation of the PRF based on CDH

Once we have fixed this, and we trust the public values, we can use  $\mathcal{S}im_U$ , that perfectly simulates a flow  $A$  from the user to a server, and can decide on the honest behavior of the servers. Then  $\mathcal{S}im_U$  accepts with  $K \leftarrow K_1$  in the honest case or aborts with  $K \leftarrow \perp$  otherwise. Hence, we remark that we answer  $\text{Send}$ -queries without calling the  $H_1$  or  $H_2$  oracles, but just using  $K_1$ , and no secret sharing reconstruction is used anymore.

Next step is to replace all the shares in the initialization phase by random and independent values. We know that until the adversary does not get more than  $t_\ell = n/3$  of these shares, it cannot detect whether they are random or correct. We define the event  $\text{PWinF}$  to be the bad event, where the adversary has enough evaluations of the PRF to notice the change. Again, our simulator is able to decide the event  $\text{PWinF}$  by checking whether  $\text{pw}$  has been queried with the right inputs to  $H_2$ , and how many times.

We eventually replace the hash value  $C$  in the initialization phase by a random  $C$ .

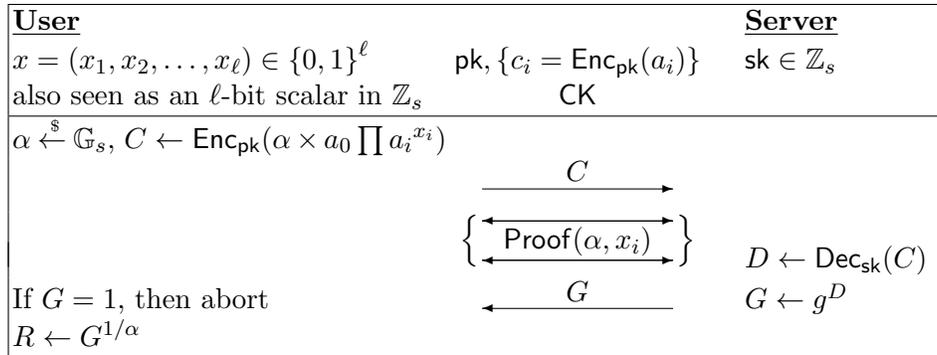
One can note that, in the end, the password  $\text{pw}$  is not used anymore during the simulation, but just to determine whether the events  $\text{PWinC}$  or  $\text{PWinF}$  happened. In addition,  $K_1$  does not appear anymore during the initialization phase, hence one cannot make any difference between  $K_0$  and  $K_1$ :  $\text{Succ}_{\mathcal{A}} = 1/2$  in the last game. As a consequence,  $\text{Adv}(\mathcal{A}) \leq \Pr[\text{PWinC}] + \Pr[\text{PWinF}] + \varepsilon$ , where  $\varepsilon$  comes from the collisions or guess on the random oracles. To evaluate the two events  $\text{PWinC}$  or  $\text{PWinF}$  to happen, we choose a random password  $\text{pw}$  at the very end only:  $\Pr[\text{PWinC}]$  is clearly upper-bounded by  $q_u/\#\mathcal{D}$ , since  $q_u$  is the maximal number of fake commitment attempts containing the right  $\text{pw}$  that can be different from the expected ones;  $\text{PWinF}$  means that the adversary managed to get  $n/3 - q_c$  evaluations of the PRFs under the chosen  $\text{pw}$ , since it can evaluate on its own the values under the  $q_c$  corrupted servers. But unless the adversary gets more evaluations than the number  $q_s$  of queries asked to the servers (which can be proven under the  $\text{OMGDH}$  assumption), the number of *bad* passwords (for which the knows at least  $n/3 - q_c$  evaluations of the PRFs) is less than  $q_s/(n/3 - q_c)$ . So the probability that the chosen  $\text{pw}$  is such a bad password is less than  $q_s/(n/3 - q_c) \times 1/\#\mathcal{D}$ .

### 5.3 Protocol II: in the Standard Model

Our second instantiation makes use of the Naor and Reingold [NR97] pseudorandom function, based on the DDH assumption, in the standard model, with a CRS. We consider the group  $\mathbb{G} = \langle g \rangle$  of prime order  $q$  that is a safe prime:  $q = 2s + 1$ . In the multiplicative group of scalar  $\mathbb{Z}_q^*$ , we consider the cyclic group  $\mathbb{G}_s$  of order  $s$  (this is the group of elements in  $\mathbb{Z}_q^*$  with Jacobi symbol equals to  $+1$ ). In both groups, the DDH assumption can be made.

The PRF key is a tuple  $a = (a_0, a_1, \dots, a_\ell) \xleftarrow{\$} (\mathbb{G}_s \setminus \{1\})^{\ell+1}$ , and  $F_a(x) = g^{a_0} \prod a_i^{x_i}$ , where  $x = (x_1, x_2, \dots, x_\ell) \in \{0, 1\}^\ell$ . This function has been proven to be a PRF under the DDH assumption [NR97] on  $\ell$ -bit inputs. It also admits a simple oblivious evaluation (just the messages  $C$  and  $G$  from Figure 2), using a multiplicatively homomorphic encryption scheme in  $\mathbb{G}_s$ , such as ElGamal for  $(\text{Enc}_{\text{pk}}, \text{Dec}_{\text{sk}})$ , which allows the computation of  $C$  from  $x$ ,  $\alpha$ , and the ciphertexts  $\{c_i\}_i$ . Unfortunately, without additional proofs, this is not secure against malicious users, since it works only for honest inputs  $x \in \{0, 1\}^\ell$ . Hence the more involved protocol presented in Figure 2 that makes use of a zero-knowledge proof of knowledge of  $(x_i)_i \in \{0, 1\}^\ell$  and  $\alpha \in \mathbb{G}_s$ . This can be efficiently done under the sole DDH assumption.

As for the previous OPRF, because of the random blinding factor  $\alpha$ , either  $R = F_a(x)$  or  $R$  is a random element in  $\mathbb{G}$ , unless  $G = 1$ , hence the verification. Therefore, the OPRF protocol guarantees that the value computed by the client is either correct when the server plays honestly or random when the server cheats. The full proof can be found in the Appendix C.



**Fig. 2.** Secure Oblivious Evaluation of the NR-PRF

## 6 Comparisons

We can assume that  $\text{PInfo}$  is stored in the Cloud, it does not need to be sent by each server, then the global communication is linear in  $n$ . More precisely, our first protocol in the random oracle model is quite similar to the one from [JKK14], but with just 2 group elements, since our gain is essentially one NIZK less, and so half the communication cost. Of course, we did not provide any security result in the UC framework [Can01], but our ultimate goal was the same as [JKK14]: an efficient robust password-protected secret sharing scheme, in a BPR-like security model [BPR00]. To this aim, there is no reason to use UC-secure building blocks, but tailored primitives.

Our second protocol in the standard model with a CRS turns out to be much more efficient than theirs: even if it uses the same Naor-Reingold PRF, the oblivious evaluation is much more efficient and relies on the DDH assumption only. Our full construction only makes use of ElGamal and Cramer-Shoup encryption schemes, and no Paillier’s encryption [Pai99] nor Cramer-Shoup signature [CS99] that require both stronger assumptions, such as the strong-RSA assumption and the decisional composite residuosity assumption, and much larger parameters, which lead to huge communication load. The main reason comes from the relaxation on the OPRF: since we do not need verifiability of server’s computations, it does not have to make any zero-knowledge proof, which allows us to use a much more efficient OPRF.

## Acknowledgments

This work was supported in part by the European Research Council under the European Community’s Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – Crypto-Cloud).

## References

- BJSL11. A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. Password-protected secret sharing. In *ACM CCS 11*, pages 433–444. ACM Press, October 2011.
- Bla79. G. R. Blakley. Safeguarding cryptographic keys. *Proceedings of AFIPS 1979 National Computer Conference*, 48:313–317, 1979.
- BM92. S. M. Bellare and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992.
- BNPS03. M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-RSA-inversion problems and the security of Chaum’s blind signature scheme. *Journal of Cryptology*, 16(3):185–215, June 2003.

- BPR00. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000*, LNCS 1807, pages 139–155. Springer, Heidelberg, May 2000.
- BR93. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93*, pages 62–73. ACM Press, November 1993.
- BR94. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO'93*, LNCS 773, pages 232–249. Springer, Heidelberg, August 1994.
- BR95. M. Bellare and P. Rogaway. Provably secure session key distribution: The three party case. In *27th ACM STOC*, pages 57–66. ACM Press, May / June 1995.
- Can01. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
- CFOR12. A. Cevallos, S. Fehr, R. Ostrovsky, and Y. Rabani. Unconditionally-secure robust secret sharing with compact shares. In *EUROCRYPT 2012*, LNCS 7237, pages 195–208. Springer, Heidelberg, April 2012.
- CLLN14. J. Camenisch, A. Lehmann, A. Lysyanskaya, and G. Neven. Memento: How to reconstruct your secrets from a single password in a hostile environment. In *CRYPTO 2014, Part II*, LNCS 8617, pages 256–275. Springer, Heidelberg, August 2014.
- CLN12. J. Camenisch, A. Lysyanskaya, and G. Neven. Practical yet universally composable two-server password-authenticated secret sharing. In *ACM CCS 12*, pages 525–536. ACM Press, October 2012.
- CLN15. J. Camenisch, A. Lehmann, and G. Neven. Optimal distributed password verification. In *ACM CCS 15*, pages 182–194. ACM Press, October 2015.
- CN00. J.-S. Coron and D. Naccache. Security analysis of the Gennaro-Halevi-Rabin signature scheme. In *EUROCRYPT 2000*, LNCS 1807, pages 91–101. Springer, Heidelberg, May 2000.
- CS98. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO'98*, LNCS 1462, pages 13–25. Springer, Heidelberg, August 1998.
- CS99. R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. In *ACM CCS 99*, pages 46–51. ACM Press, November 1999.
- DIO98. G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Non-interactive and non-malleable commitment. In *30th ACM STOC*, pages 141–150. ACM Press, May 1998.
- ELG85. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.
- FIPR05. M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *TCC 2005*, LNCS 3378, pages 303–324. Springer, Heidelberg, February 2005.
- FK00. W. Ford and B. S. Kaliski, Jr. Server-assisted generation of a strong secret from a password. In *Proceedings of the 9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 176–180, Washington, DC, USA, 2000. IEEE Computer Society.
- FSW03. P.-A. Fouque, J. Stern, and J.-G. Wackers. Cryptocomputing with rationals. In *FC 2002*, LNCS 2357, pages 136–146. Springer, Heidelberg, March 2003.
- GGM86. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.
- GHR99. R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *EUROCRYPT'99*, LNCS 1592, pages 123–139. Springer, Heidelberg, May 1999.
- Jab01. D. P. Jablon. Password authentication using multiple servers. In *CT-RSA 2001*, LNCS 2020, pages 344–360. Springer, Heidelberg, April 2001.
- JKK14. S. Jarecki, A. Kiayias, and H. Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In *ASIACRYPT 2014, Part II*, LNCS 8874, pages 233–253. Springer, Heidelberg, December 2014.
- JS13. M. P. Jhanwar and R. Safavi-Naini. Unconditionally-secure robust secret sharing with minimum share size. In *FC 2013*, LNCS 7859, pages 96–110. Springer, Heidelberg, April 2013.
- LP14. A. B. Lewko and V. Pastro. Robust secret sharing schemes against local adversaries. Cryptology ePrint Archive, Report 2014/909, 2014. <http://eprint.iacr.org/2014/909>.
- MS81. R. J. McEliece and D. V. Sarwate. On sharing secrets and reed-solomon codes. *Commun. ACM*, 24(9):583–584, September 1981.
- NR97. M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997.
- Oba11. S. Obana. Almost optimum t-cheater identifiable secret sharing schemes. In *EUROCRYPT 2011*, LNCS 6632, pages 284–302. Springer, Heidelberg, May 2011.
- OP01. T. Okamoto and D. Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In *PKC 2001*, LNCS 1992, pages 104–118. Springer, Heidelberg, February 2001.
- Pai99. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, LNCS 1592, pages 223–238. Springer, Heidelberg, May 1999.

- Sha79. A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.
- TW88. M. Tompa and H. Woll. How to share a secret with cheaters. *Journal of Cryptology*, 1(2):133–138, 1988.
- YHCL15. X. Yi, F. Hao, L. Chen, and J. K. Liu. Practical threshold password-authenticated secret sharing protocol. In *ESORICS 2015, Part I, LNCS 9326*, pages 347–365. Springer, Heidelberg, September 2015.

## A Computational Assumptions

We consider a finite multiplicative cyclic group  $\mathbb{G} = \langle g \rangle$  of prime order  $q$ .

**Computational Diffie-Hellman Assumption (CDH).** The  $\text{CDH}_g$  assumption states that given  $g^x$  and  $g^y$ , where  $x$  and  $y$  were drawn at random from  $\mathbb{Z}_q$ , it is hard to compute  $g^{xy}$ . We denote by  $\text{Succ}^{\text{cdh}}(\mathcal{A})$  the success probability of the adversary  $\mathcal{A}$  in computing  $g^{xy}$ , and more generally,  $\text{Succ}^{\text{cdh}}(t)$  is the best success probability an adversary can get within time  $t$ .

**Decisional Diffie-Hellman Assumption (DDH).** The  $\text{DDH}_g$  assumption states that given one of the two tuples  $(g^x, g^y, g^{xy})$  and  $(g^x, g^y, g^z)$  where  $x, y, z$  are chosen at random and independently from  $\mathbb{Z}_q$ , no efficient algorithm can distinguish between them. We denote by  $\text{Adv}^{\text{ddh}}(\mathcal{A})$  the advantage of the adversary  $\mathcal{A}$  in distinguishing between the two distributions, and more generally,  $\text{Adv}^{\text{ddh}}(t)$  is the best advantage an adversary can get within time  $t$ .

**Gap Diffie-Hellman Assumption (GDH).** The  $\text{GDH}_g$  assumption [OP01] states that the  $\text{CDH}_g$  assumption holds even when the adversary has access to a  $\text{DDH}_g$  oracle that exactly answers for any query  $\text{DDH}_g(g^x, g^y, g^z)$  whether  $z = xy$  or not.  $\text{Succ}^{\text{gdh}}(\mathcal{A})$  and  $\text{Succ}^{\text{gdh}}(t, q_d)$  are defined as above, where  $\mathcal{A}$  can ask up to  $q_d$   $\text{DDH}_g$  oracle queries.

**One-more Gap Diffie-Hellman Assumption (OMGDH).** The  $(n, m)$ -One-more Gap Diffie-Hellman assumption [BNPS03] states that given  $g^x$  where  $x \xleftarrow{\$} \mathbb{Z}_q$ , a list  $(g_1, \dots, g_n) \xleftarrow{\$} \mathbb{G}^n$ , unlimited access to a  $\text{DDH}_g(g^x, \cdot, \cdot)$  oracle, and up to  $m$  queries to a  $\text{CDH}_g(g^x, \cdot)$  oracle, it is hard to output  $m + 1$  valid pairs  $(g_i, g_i^x)$ .

$\text{Succ}^{\text{omgdh}}(n, m, \mathcal{A})$  and  $\text{Succ}^{\text{omgdh}}(n, m, t, q_d)$  are defined as above, where  $\mathcal{A}$  can ask up to  $q_d$   $\text{DDH}_g$  oracle queries.

## B Building Blocks

### B.1 General Definitions

**Threshold Secret Sharing Scheme.** A *threshold secret sharing scheme* allows a user to distribute a secret among different participants preventing a sole party breaking the security or obstructing the reconstruction. This idea was introduced by Shamir [Sha79] and Blakey [Bla79]. A  $(t, n)$ -threshold secret sharing scheme splits a secret  $s$  into  $n$  shares, distributed to  $n$  participants in such a way that any subset of  $t$  ( $0 < t \leq n$ ) participants with valid shares is able to reconstruct the original secret, whereas any subset of less than  $t$  participants leaves the secret completely undetermined.

A  $(t, n)$ -threshold secret sharing scheme is called *perfect* if any subset smaller than  $t$  has no information at all about the secret, in an information-theoretic sense. More precisely, a  $(t, n)$ -threshold secret sharing scheme is defined on a set of  $n$  participants  $P_1, \dots, P_n$ , with algorithms **ShareGen** and **Reconstruct**:

- **ShareGen**( $s, t$ ): on a secret  $s$  and a threshold  $t$ , this algorithm generates  $n$  shares  $(s_1, \dots, s_n)$ , and possible public information **SSInfo**;

- **Reconstruct**( $\{s_i\}, \text{SSInfo}$ ): on a set of  $t$  shares, and the possible additional information  $\text{SSInfo}$ , this algorithm recovers the secret  $s$ .

The correctness guarantees that the **Reconstruct** algorithm recovers the correct initial secret on any set of  $t$  shares. Such a scheme is said secure if any set of less than  $t$  shares cannot reconstruct the secret.

The notion of the threshold secret sharing scheme has been extensively studied, and extensions like verifiability (which is the capability for the participants to verify their shares are correct), robustness, cheater detection, and cheater identification, among others, have been proposed to this basic model [MS81, TW88, Oba11, CFOR12, JS13, LP14].

Verifiable secret sharing schemes actually allow verifiability of individual shares, using the additional  $\text{SSInfo}$  that contains verifiers for every shares. In our proposal we want to have verifiability of shares at a more global level only and avoid individual verifiability because it could allow to a unique corrupted server make an off-line dictionary attack on its own. However, when a subset of valid and invalid shares is given, without verifiability, it is in general quite difficult to extract a subset of  $t$  valid shares and recover the secret. The unique solution is often the exhaustive search among all the subsets of  $t$  shares, which requires an exponential time (in  $n$ ).

**Robust Threshold Secret Sharing Scheme.** Several notions of robustness have been defined in the literature for secret sharing schemes. For our purpose, a secret sharing scheme will be said *robust* if, when a user is given  $m$  shares with at least  $t_r$  valid shares, he can efficiently recover the secret. It will be said *robust with respect to random failures* when the reconstruction is only possible if invalid shares are random, and not fabricated by the adversary, which is enough for our purpose.

In the following, we present a generic technique, to enhance a  $(t, n)$ -threshold secret sharing scheme, that allows to efficiently find the appropriate subset of  $t$  valid shares among a set of candidates, without increasing the size of the shares. More precisely, we will assume that we have a set of  $m$  candidates, with at least  $t$  correct values, whereas the incorrect values are random. To this aim, the additional public information  $\text{SSInfo}$  will contain global information on the shares only, and no information on the individual shares: for the construction we propose in this paper,  $\text{SSInfo}$  is the product of all the fingerprints, modulo a small prime, in order not to leak too much information.

**Oblivious Pseudorandom Functions.** A pseudorandom function [GGM86] (PRF) is actually a keyed-family of functions  $(F_k)_k$ , where the outputs are indistinguishable, for a random key  $k$ , from random elements in the function range. An *oblivious* PRF (OPRF) [FIPR05] is a protocol that allows the sender contribute the key  $k$  and the receiver compute the value of  $F_k(x)$  on any input of  $x$  of the receiver in a way that the sender learns nothing from the protocol.

**Encryption Schemes.** A public-key encryption scheme is a triple  $(\mathcal{K}, \mathcal{E}, \mathcal{D})$  of algorithms. The key generation algorithm  $\mathcal{K}$  takes as input a security parameter and outputs an encryption/decryption key pair  $(\text{ek}, \text{dk})$ . The encryption algorithm  $\mathcal{E}$  takes as input an encryption key  $\text{ek}$  and a message  $m$  and outputs a ciphertext  $c$ . The decryption algorithm  $\mathcal{D}$  takes as input a decryption key  $\text{dk}$  and a ciphertext  $c$  and outputs either the decryption  $m$  of  $c$  or  $\perp$ . The correctness condition required is that for all  $(\text{ek}, \text{dk})$  generated by  $\mathcal{K}$ , and for all messages  $m$ ,  $\mathcal{D}(\text{dk}, \mathcal{E}(\text{ek}, m)) = m$ . Classical security notions for encryption are IND – CPA and IND – CCA, where the adversary tries to distinguish the ciphertext of two messages of its choice, being given just the encryption key, or also access to the decryption oracle, respectively.

**Commitment Schemes.** In a commitment scheme, a *sender* commits on a message  $m$  to a *receiver* without revealing any information, but with the guarantee that at the *opening* time, a

unique message can be revealed. There are two basic properties: the commitment must be *hiding*, which guarantees that no information about  $m$  is leaked during the *commit* phase, and be *binding*, which guarantees that only one message can be revealed during the *opening* phase. Additional classical properties are extractability, equivocability, and non-malleability.

## B.2 Concrete Encryption Schemes

**ElGamal Encryption.** Introduced in 1985, by ElGamal [EIG85], based on the CDH assumption, and achieving IND – CPA security under the DDH, the ElGamal encryption scheme works as follows:

**Key Generation:** Let  $x \in \mathbb{Z}_q$  the decryption key, the associated encryption key is  $y = g^x$ ;

**Encryption:** Given a message  $m \in \mathbb{G}$ , let choose  $r \xleftarrow{\$} \mathbb{Z}_q$ , then compute  $u = g^r$  and  $v = y^r m$ . The ciphertext is  $c = (u, v)$ ;

**Decryption:** Given a ciphertext  $c = (u, v)$ , the message can be decrypted as  $m = v \cdot u^{-x}$ .

More precisely, within time  $t$ :

$$\text{Adv}^{\text{ind-cpa}}(t) \leq 2 \times \text{Adv}^{\text{ddh}}(t).$$

**Cramer-Shoup Encryption.** The Cramer-Shoup encryption scheme [CS98] achieves IND – CCA security under the DDH assumption:

**Key Generation:** Let  $g_1, g_2 \xleftarrow{\$} \mathbb{G}$  and  $x_1, x_2, y_1, y_2, z \xleftarrow{\$} \mathbb{Z}_q$ . Let  $c = g_1^{x_1} g_2^{x_2}$ ,  $d = g_1^{y_1} g_2^{y_2}$ ,  $h = g_1^z$  and a hash function  $H$ , chosen from the family of universal one-way functions. The public key is  $(g_1, g_2, c, d, h, H)$  and the private key is  $(x_1, x_2, y_1, y_2, z)$ ;

**Encryption:** Given a message  $m \in \mathbb{G}$ , let choose  $r \xleftarrow{\$} \mathbb{Z}_q$ , then compute  $u_1 = g_1^r$ ,  $u_2 = g_2^r$ ,  $e = h^r m$ ,  $\alpha = H(u_1, u_2, e)$ , and  $v = c^r d^{r\alpha}$ , the ciphertext is  $c = (u_1, u_2, e, v)$ ;

**Decryption:** Given a ciphertext  $c = (u_1, u_2, e, v)$ , one first computes  $\alpha = H(u_1, u_2, e)$  and checks whether  $u_1^{x_1+y_1\alpha} u_2^{x_2+y_2\alpha} = v$  or not. If this condition does not hold, then it rejects, otherwise it outputs  $m = e/u_1^z$ .

Such an IND – CCA encryption scheme can be used as a perfectly binding commitment scheme. The decryption key allows extractability and the IND – CCA security level makes the commitment scheme non-malleable, but also extractable while still (computationally) hiding.

More precisely, within time  $t$  and after at most  $q_d$  decryption queries:

$$\text{Adv}^{\text{ind-cca}}(t) \leq 2 \times \text{Adv}^{\text{ddh}}(t) + \text{Succ}_H^{2\text{nd}}(t) + 3q_d/q.$$

## C Auxiliary Proofs

### C.1 $F_x$ is a PRF

**Lemma 3.** *The above function  $F_x$  is a PRF under the Computational Diffie-Hellman (CDH) assumption.*

Given an instance  $(g, y = g^x, h)$ , one wants to compute  $h^x = \text{CDH}_g(y, h)$ . Any  $H_1$ -query on a new  $m$  is answered by  $h^z$ , for a random scalar  $z$ , and the tuple  $(m, z)$  is stored in the list  $\Lambda_1$ . For any PRF evaluation on a new  $m$ , one first asks for  $H_1(m)$ , chooses a random value  $r \in \{0, 1\}^\ell$ , answers  $r$  and stores  $(m, z, r)$  in the list  $\Lambda_F$ . For any new  $H_2$ -query  $(m, y, H)$ , one first asks for  $H_1(m)$ , and answers by a random value. A difference happens here from the real case if  $H = \text{CDH}_g(y, H_1(m))$  and  $(m, z, r)$  is in  $\Lambda_F$ , since the answer should be  $r$ , and not a random value. The same problem happens if the  $F$  query is asked later. In both cases, at the end of the game, among all the  $H$  values from the  $H_2$ -queries and the  $(m, z, r) \in \Lambda_F$ , one pair  $(H, z)$  satisfies  $H = \text{CDH}_g(y, H_1(m)) = \text{CDH}_g(y, h^z) = \text{CDH}_g(y, h)^z$ . By choosing it at random, one gets  $\text{CDH}_g(y, h)$  with non-negligible probability.

## C.2 Security Proof of the Protocol I

For the proof we consider an adversary as the one defined in the security model description in Section 2. After the initialization phase, this adversary can ask as many Execute and Send-queries, Test-queries and also Corrupt-queries as it wishes, and has access to the extra random oracles  $H_1$ ,  $H_2$ , and  $H_3$ .

The proof will be performed by a sequence of games, starting from the real indistinguishability game, focusing on a unique user, against a static adversary (the corrupted servers are known right after the initialization, and before any reconstruction attempt). In the final game, the goal to achieve is to simulate all the queries to the adversary without using a password. A random value will be chosen at the very end of the simulation and used as a password in order to decide if some bad event should have occurred, which will immediately upper-bound the advantage of the adversary.

**Game  $G_0$ :** This initial game corresponds to the real attack game, in the random oracle model.

Three oracles are available to the adversary,  $H_1$ ,  $H_2$ , and  $H_3$  and the adversary chooses some servers to be corrupted: the related secret informations are then revealed to the adversary right after the initialization.

First, we emulate the initialization phase, which is honestly performed: we choose one random  $\text{pw}$ ,  $n$  random keys  $(x_k)_k$  for the servers' secret information, which lead to the evaluation of  $(\pi_k)_k$ , together with their public part  $\Pi = (y_k = g^{x_k})_k$ , and one random value corresponding to the secret  $K$ , together with a secret sharing  $(s_1, \dots, s_n, \text{SSInfo})$  of  $R = K || r$ , for a random  $r$ . This last random value  $r$  is used to compute the commitment  $C = H_3(\text{pw}, \Pi, \Sigma, \text{SSInfo}, K, r)$ , where  $\Sigma = (\sigma_k = \pi_k \oplus s_k)_k$ . One also chooses a second random key  $K_0$ , as well as a bit  $b \xleftarrow{\$} \{0, 1\}$ , both used in Test-queries: in a reconstruction execution, if a key  $K_1$  is reconstructed, the Test-query outputs  $K_b$ , if the reconstruction is not completed or failed, the Test-query outputs UNDEFINED or  $\perp$ . For the reconstruction, we simulate all the instances, the user and the servers, in Execute and Send-queries, as the real players would do.

The adversary eventually outputs its guess  $b'$  for the bit  $b$ . The output of the game is the success bit  $S = (b' = b)$ . By definition we have :

$$\text{Succ}_{G_0} = \Pr[S] \qquad \text{Adv}(\mathcal{A}) = 2 \times \text{Succ}_{G_0} - 1$$

**Game  $G_1$ :** We do not modify the initialization, and first deal with Execute-queries, by replacing the user and the servers by the simulator  $\text{Sim}$  that perfectly simulates honest transcripts  $(A, B)$ , and user's key is set to  $K_1$ . The change being just syntactic:  $\text{Succ}_{G_0} = \text{Succ}_{G_1}$ .

**Game  $G_2$ :** We now deal with Send-queries to the user, and namely when the adversary fakes the public information  $\text{PInfo}$  sent to the user: if the majority of at least  $t_r$  tuples  $\text{PInfo}' = (\Pi', \Sigma', \text{SSInfo}', C')$  contains a commitment  $C'$  different from the expected commitment  $C$ , we make the user play as usual, but eventually set  $K \leftarrow \perp$ .

This makes a difference only if in the end this commitment would have been accepted by the user with respect to his password  $\text{pw}$ . Since we use a hash function  $H_3$  modeled as a random oracle,  $C'$  must have been obtained with a query containing  $\text{pw}$ , or the probability to be valid is  $1/2^{\ell_3}$ : We thus define the event  $\text{PWinC}$  to be true if  $C' \neq C$  but  $C'$  is the result of a query of  $H_3$  on a tuple that contains  $\text{pw}$ . And at the end, after the answer  $b'$ , if  $\text{PWinC}$  is set, one sets the output bit  $S$  at random instead of  $(b' = b)$ . In this game, we reduce the success probability of the adversary, but only when  $\text{PWinC}$  happens:  $\text{Succ}_{G_1} \leq \text{Succ}_{G_2} + 1/2^{\ell_3} + \Pr[\text{PWinC}]/2$ . This event  $\text{PWinC}$  can be evaluated by looking at each of the queries asked to  $H_3$  and then checking whether it contains  $\text{pw}$  or not.

**Game  $\mathbf{G}_3$ :** We continue in the same vein for fake public information  $\text{PInfo}'$  (but correct  $C$ ) sent to the user: if the majority of at least  $t_r$  tuples  $\text{PInfo}' = (\Pi', \Sigma', \text{SSInfo}', C)$  contains public information different from the expected ones (the  $\text{PInfo}$  generated during the honest initialization phase), we make the user play as usual, but eventually set  $K \leftarrow \perp$ . Since the hash value  $C$  is unchanged, the input  $(\text{pw}, \Pi', \Sigma', \text{SSInfo}')$  must be unchanged, unless one finds a collision for  $H_3$ :  $\text{Succ}_{\mathbf{G}_2} \leq \text{Succ}_{\mathbf{G}_3} + q_3^2/2^{\ell_3+1}$ .

**Game  $\mathbf{G}_4$ :** We continue with the simulation of the user, but when the majority  $\text{PInfo}$  is the expected one, which guarantees the use of the correct public keys, and thus the knowledge of the associated secret keys. We now use  $\text{Sim}_U$ , that perfectly simulates a flow  $A$  from the user to a server, and can decide on the honest behavior of the server thanks to the server's secret key  $x_k$  to evaluate  $\text{DDH}_g(y_k, \cdot, \cdot)$ . If the behaviors of at least  $t_r$  of the servers are correct, the user accepts with  $K \leftarrow K_1$ , otherwise the user aborts with  $K \leftarrow \perp$ . Since the OPRF protocol uses the random blinding factor  $\alpha$ , either  $C = H_1(\text{pw})^{x_k}$  or  $C$  is a random element in  $\mathbb{G}$ , unless  $B = 1$ , hence the verification.

In the previous game, the robust threshold secret sharing scheme with respect to random failures guaranteed the recovery of the secret in exactly the same cases as here:  $\text{Succ}_{\mathbf{G}_3} = \text{Succ}_{\mathbf{G}_4}$ . We remark that during this game, the  $\text{Send}$ -queries are answered without calling the  $H_1$  oracle, neither  $H_2$  oracle is used for the reconstruction of  $K$ . Instead, after the  $\text{DDH}_g(y, \cdot, \cdot)$  check (using the secret key  $x$ ), the secret  $K$  is directly set to  $K_1$  (or to  $\perp$  if too many failures).

**Game  $\mathbf{G}_5$ :** Since the secret sharing reconstruction is not used anymore, we can thus replace all the shares  $(s_1, \dots, s_n)$  by random and independent values and generate  $\text{SSInfo}$  accordingly in the initialization phase. We know that until the adversary does not get more than  $t_\ell = n/3$  of these shares, it cannot detect whether they are random or correct: let us define the event  $\text{PWinF}$  to be true if more than  $n/3 - q_c$  queries have been asked to the  $H_2$  oracle for the un-corrupted key  $y_k$  on  $\text{pw}$  with the correct  $\text{CDH}$  value, since the adversary can evaluate on its own the values under the  $q_c$  corrupted servers. And at the end, after the answer  $b'$ , if  $\text{PWinF}$  is set, one sets  $S$  at random. As in Game  $\mathbf{G}_2$ , we have the upper-bound:  $\text{Succ}_{\mathbf{G}_4} \leq \text{Succ}_{\mathbf{G}_5} + \Pr[\text{PWinF}]/2$ . Using the servers' secret keys  $(x_k)_k$  (to test  $\text{DDH}_g(y_k, \cdot, \cdot)$  validity), the simulator can check whether  $\text{pw}$  has been queried with the right inputs to  $H_2$  to learn some  $\pi_k$ , and how many times, to set the event  $\text{PWinF}$ .

**Game  $\mathbf{G}_6$ :** Instead of choosing the shares at random, one generates  $\Sigma = (\sigma_k)_k$  and  $\text{SSInfo}$  at random, without computing the  $\pi_k$ 's:  $\text{Succ}_{\mathbf{G}_5} = \text{Succ}_{\mathbf{G}_6}$ .

**Game  $\mathbf{G}_7$ :** We now deal with  $\text{Send}$ -queries to the servers, and replace them by the simulator  $\text{Sim}_S$  to provide answers, using the server's secret key  $x_k$  to evaluate  $\text{CDH}_g(y_k, \cdot)$ :  $\text{Succ}_{\mathbf{G}_6} = \text{Succ}_{\mathbf{G}_7}$ .

**Game  $\mathbf{G}_8$ :** We now replace the hash value  $C$  in the initialization phase by a random  $C$ . This is indistinguishable because of the random oracle property:  $\text{Succ}_{\mathbf{G}_7} = \text{Succ}_{\mathbf{G}_8}$ .

In this last game, one can note that the password  $\text{pw}$  is not used anymore during the simulation, but just to determine whether the events  $\text{PWinC}$  or  $\text{PWinF}$  happened to define the game output  $S$ . In addition,  $K_1$  does not appear any more during the initialization phase (it was just used for the secret sharing, while the shares have been replaced by random shares, and in the commitment, while it has been replaced by a random hash), hence one cannot make any difference between  $K_0$  and  $K_1$ :  $\text{Succ}_{\mathbf{G}_8} = 1/2$ . As a consequence,

$$\text{Adv}(\mathcal{A}) \leq \Pr[\text{PWinC}] + \Pr[\text{PWinF}] + (q_3^2 + 2) \cdot 2^{-\ell_3}.$$

We thus now have to evaluate the probabilities of the two events  $\text{PWinC}$  or  $\text{PWinF}$  to happen, which can be done by choosing a random password  $\text{pw}$  at the very end only (since it is not used anymore during the initialization phase, nor in the reconstruction): About  $\Pr[\text{PWinC}]$ , it is clearly

upper-bounded by  $q_u/\#\mathcal{D}$ , since  $q_u$  is the maximal number of fake commitment attempts containing the right  $\text{pw}$  that can be different from the expected ones; On the other hand, PWinF means that the adversary managed to get  $n/3 - q_c$  evaluations of the PRFs under the chosen  $\text{pw}$ . But unless the adversary gets more evaluations than the number  $q_s$  of queries asked to the servers, the number of *bad* passwords (for which he knows at least  $n/3 - q_c$  evaluations of the PRFs) is less than  $q_s/(n/3 - q_c)$ . So the probability that the chosen  $\text{pw}$  is such a bad password is less than  $q_s/(n/3 - q_c) \times 1/\#\mathcal{D}$ .

The following lemma leads to  $\Pr[\text{PWinF}] \leq q_s/(n/3 - q_c) \times 1/\#\mathcal{D} + n \times \text{Succ}^{\text{omgdh}}(q_1, q_s, t, n \cdot q_u + q_2)$ , which concludes the proof of the theorem.

**Lemma 4.** *Unless one can break a  $(q_1, q_s)$  – OMGDH with  $(n \cdot q_u + q_2)$  queries to the DDH-oracle, no adversary, that involves  $q_s$  instances of the servers,  $q_u$  instances of the user, and asks  $q_1$  queries to  $H_1$  and  $q_2$  queries to  $H_2$ , can get more evaluations of the PRF than the number  $q_s$  of queries asked to the servers.*

If we denote  $q_1$  the number of queries to  $H_1$ , we can also denote  $(h_1, \dots, h_{q_1})$  the list of the answers, which are random group elements. Let us be given a random instance  $(g, y = g^x, h_1, \dots, h_{q_1})$  of the  $(q_1, q_s)$  – OMGDH problem (see Appendix A), then our simulator uses  $y^* = y$  for a randomly chosen server  $k^*$ , and  $y_k = g^{x_k}$  for random scalars  $x_k$ , for the other servers. Getting one-more evaluation of the PRF (under non-corrupted keys) than the number of queries to the (non-corrupted) servers means that this must be the case for at least one of the non-corrupted servers: we hope the  $k^*$ -server to be one of them. Since it is chosen at random, this is a correct guess with probability greater than  $1/n$ .

For the simulation of the  $q_s$  queries  $A$  to the honest servers, for the  $k^*$ -server, the simulator makes one  $\text{CDH}_g(y^*, \cdot)$ -query, while for the others the secret key  $x_k$  is known. For the (at most  $n \times q_u$ ) transcripts  $(A, B)$  obtained by the honest user with the adversary, the simulator makes one  $\text{DDH}_g(y^*, \cdot, \cdot)$ -query when the adversary plays the role of the  $k^*$ -server, but can use  $x_k$  otherwise. Getting one more evaluation of  $F_{k^*}$  than the number  $q$  of queries to the  $k^*$ -server means that for at least  $q + 1$  queries  $(\text{pw}_i, y^*, H)$  to the random oracle  $H_2$ ,  $H = \text{CDH}_g(y^*, H_1(\text{pw}_i))$ . Since  $H_1(\text{pw}_i)$  has been answered by one of the  $h_j$ , one gets  $q + 1$  correct values  $\text{CDH}_g(y^*, h_j)$ , that can be detected using the  $\text{DDH}_g(y^*, \cdot, \cdot)$  oracle on all the  $q_2$  inputs to  $H_2$ . We can of course upper-bound  $q$  by  $q_s$ , hence the lemma.

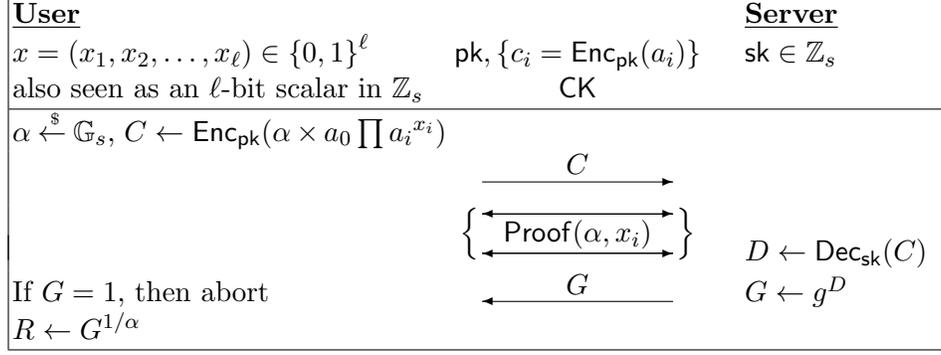
### C.3 Security Proof of the Protocol II

**Theorem 5.** *For any adversary  $\mathcal{A}$ , against the Protocol II using both ElGamal and Cramer-Shoup encryption schemes, that corrupts no more than  $q_c$  servers, involves at most  $q_s$  instances of the servers, and  $q_u$  instances of the user*

$$\begin{aligned} \text{Adv}(\mathcal{A}) \leq & \left( q_u + \frac{3q_s}{n - 3q_c} \right) \times \frac{1}{\#\mathcal{D}} \\ & + ((n - q_c)\ell + 4) \times \text{Adv}^{\text{ddh}}(t + q_s t_{\text{exp}}) + 3 \times \text{Succ}_{\mathcal{H}}^{2\text{nd}}(t) + 6q_u/q, \end{aligned}$$

where  $\ell$  is the size of the password and  $t_{\text{exp}}$  the time for an exponentiation.

The proof will be performed by a sequence of games, as in the ROM model, focusing on a unique user, against a static adversary (the corrupted servers are known from the beginning). A change from the general description of the PPSS protocol in this particular case consists a more efficient way to compute the commitment  $C = \text{Commit}(\text{pw}, \Pi, \Sigma, \text{SSInfo}, K; r)$ , by first a fingerprint  $H = \mathcal{H}(\Pi, \Sigma, \text{SSInfo}, K)$  with a second-preimage-resistant hash function  $\mathcal{H}$ , and then



**Fig. 3.** Secure Oblivious Evaluation of the NR-PRF

$C = \text{Enc}(\text{pw}, H; r)$ , with an IND – CCA encryption scheme. This improves the efficiency, as the information  $(\Pi, \Sigma, \text{SSInfo}, K)$  may be long. More precisely, we use the Cramer-Shoup encryption scheme, denoted  $(\text{CS.Enc}, \text{CS.Dec})$ , for the extractable commitment. We will use the simulators presented in Figure 4, where the  $c_i$ 's have been replaced by random ciphertexts, which is indistinguishable under the IND – CPA security level of the ElGamal encryption scheme, denoted  $(\text{EG.Enc}, \text{EG.Dec})$ .  $\text{Sim}_U$  knows the encrypted value  $D$ , and can thus check the answer. In addition, using **Proof**, a zero-knowledge proof of knowledge of  $(x_i)_i \in \{0, 1\}^\ell$ ,  $\alpha \in \mathbb{G}_s$ , and additional random coins such that  $C$  is correct, we show that this enhanced protocol can check the correctness of the answers from the server. Indeed, from the extractor of **Proof**,  $\text{Sim}_S$  can extract  $(x_i)_i \in \{0, 1\}^\ell$  to ask the PRF oracle, that answers either correctly or at random, as well as  $\alpha \in \mathbb{G}_s$ , to send a blinded answer to the client.

More precisely, using ElGamal encryption, we have (using component-wise multiplication)

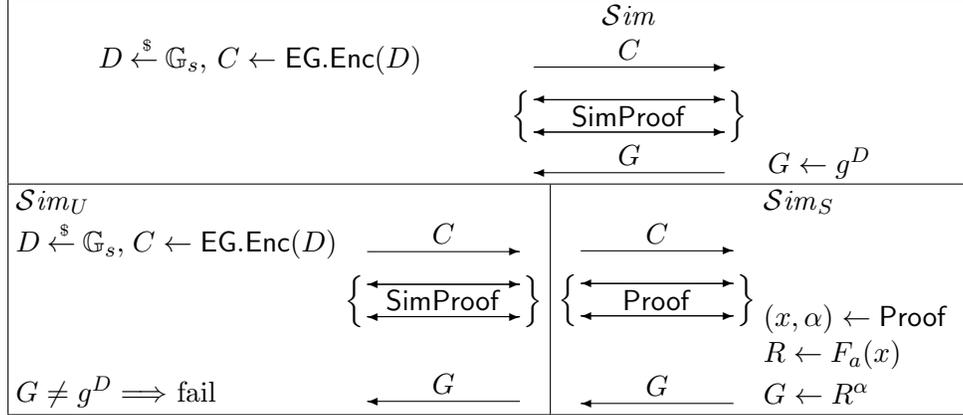
$$C = (g^r, h^r \alpha) \times c_0 \prod c_i^{x_i} = (g^r, h^r) \times c_0 \prod c_i^{x_i} \times (1, \alpha),$$

and one has to prove its knowledge of  $(x_i)_i \in \{0, 1\}^\ell$ ,  $\alpha \in \mathbb{G}_s$ , and  $r \in \mathbb{Z}_s$  that satisfy this relation. To get a straightline extraction, one can use a Cramer-Shoup encryption of  $\alpha$  and  $u^x$ , for a generator  $u \in \mathbb{G}_s$  (assuming the use of a password small enough to allow discrete logarithm computation), where the latter can be seen as  $u^x = \prod (u^{2^i})^{x_i}$ . Otherwise, one can encrypt  $u^{x_i}$  for each index  $i$ .

One should note that we only have to do proofs in  $\mathbb{G}_s$ , which are classical Schnorr-like proofs.

**Game  $\mathbf{G}_0$ :** This initial game corresponds to the real attack game, in the standard model. As in the proof for the ROM PPSS scheme, we emulate the initialization phase, which is honestly performed: we choose one random  $\text{pw}$ ,  $n$  random keys  $(\text{sk}_k)_k$  for the ElGamal encryption scheme and the PRF keys  $a_k = (a_{k,0}, \dots, a_{k,\ell})_k$  that represent the servers' secret information. We generate their corresponding public part  $\Pi = (\text{pk}_k, (c_i = \text{Enc}_{\text{pk}_k}(a_{k,i}))_k)$ , and one random value corresponding to the secret  $K$ , together with a secret sharing  $(s_1, \dots, s_n, \text{SSInfo})$  of  $R = K \parallel r$ , for a random  $r$ . These shares are masked using the values  $(\pi_k)_k$ , obtained as PRF evaluations of  $\text{pw}$  under all the secret strings  $a_k$  of the servers. We then set  $\Sigma = (\sigma_k = \pi_k \oplus s_k)_k$ . The same random value  $r$  in  $R = K \parallel r$  is used to compute the commitment  $C = \text{CS.Enc}(\text{pw}, H; r)$ , where  $H = \mathcal{H}(\Pi, \Sigma, \text{SSInfo}, K)$ .

One also chooses a second random key  $K_0$ , as well as a bit  $b \xleftarrow{\$} \{0, 1\}$ , both used in **Test**-queries: if a key  $K_1$  is reconstructed, the **Test**-query outputs  $K_b$ , if the reconstruction is not completed or failed, the **Test**-query outputs UNDEFINED or  $\perp$ . For the reconstruction, we simulate all the instances, the user and the servers, in **Execute** and **Send**-queries, as the real players would do.



$\mathcal{S}im_S$  extracts  $x$  from **Proof** to build  $R$ , the expected PRF value (either real or random value). The ciphertexts  $c_i$  have been replaced by random encryptions.

**Fig. 4.** Simulators for the OPRF based on CDH

The adversary eventually outputs its guess  $b'$  for the bit  $b$ . The output of the game is the success bit  $S = (b' = b)$ . By definition we have :

$$\text{Succ}_{\mathbf{G}_0} = \Pr[S] \qquad \text{Adv}(\mathcal{A}) = 2 \times \text{Succ}_{\mathbf{G}_0} - 1$$

**Game  $\mathbf{G}_1$ :** We first deal with **Execute**-queries, without modifying the initialization. We replace the user and the servers in the reconstruction protocol by the simulator  $\mathcal{S}im$  from figure 4. This perfectly simulates honest transcripts  $(C, \text{Proof}, G)$ , and user's key is set to  $K_1$ . The change for the values  $C$  and  $G$  is just syntactic, the two values are equivalent to the real ones:

$$\text{Succ}_{\mathbf{G}_0}(\mathcal{A}) = \text{Succ}_{\mathbf{G}_1}(\mathcal{A}).$$

**Game  $\mathbf{G}_2$ :** We consider an adversary that fakes the public information **PlInfo** in **Send**-queries to the user: if the majority of at least  $t_r$  tuples  $\text{PlInfo}' = (\Pi', \Sigma', \text{SSInfo}', C')$  contains a commitment  $C'$  different from the expected commitment  $C$ , we make the user play as usual, but eventually set  $K \leftarrow \perp$ .

The reconstruction protocol guarantees that if the majority tuple  $\text{PlInfo} = (\Pi, \Sigma, \text{SSInfo}, C)$  does not contain the expected commitment  $C$ , the user aborts with  $K \leftarrow \perp$ . This makes a difference only if in the end this commitment would have been accepted by the user with respect to his password. Since we use a perfectly binding commitment (an encryption scheme), the ciphertext  $C'$  must contain the correct pw: We thus define the event **PWinC** to be true if  $C' \neq C$  but contains pw.

We simulate the game by checking if **PWinC** is set at the end, after receiving the answer  $b'$ . If so, one sets the output bit  $S$  at random instead of  $(b' = b)$ . In this game, we reduce the success probability of the adversary, but only when **PWinC** happens:

$$\text{Succ}_{\mathbf{G}_1} \leq \text{Succ}_{\mathbf{G}_2} + \Pr[\text{PWinC}]/2$$

This event **PWinC** can be evaluated by decrypting the commitment  $C = \text{CS.Enc}(\text{pw}, H; r)$  using the decryption key, and then checking whether it contains pw or not.

**Game  $\mathbf{G}_3$ :** We continue in the same vein for fake public information  $\text{PlInfo}'$  (but correct  $C$ ) sent to the user: if the majority of at least  $t_r$  tuples  $\text{PlInfo}' = (\Pi', \Sigma', \text{SSInfo}', C)$  contains public

information different from the expected ones (the  $\text{PInfo}$  generated during the honest initialization phase), we make the user play as usual, but eventually set  $K \leftarrow \perp$ .

The OPRF protocol guarantees that the value computed by the client is either correct when the server plays honestly or random when the server cheats and the robustness with respect to random failures of the threshold secret sharing scheme guarantees the recovery of the secret in the honest case. When the value  $C$  is unchanged, the value  $H' = \mathcal{H}(\Pi', \Sigma', \text{SSInfo}', K')$  must be the same as the initial commitment input  $H = \mathcal{H}(\Pi, \Sigma, \text{SSInfo}, K)$  in order to be accepted by the user, since this is an encryption scheme, with a unique decryption. If in the end of the previous the simulator was accepting the key  $K'$ , this means that we have a second pre-image  $(\Pi', \Sigma', \text{SSInfo}', K')$  of the initial  $H = \mathcal{H}(\Pi, \Sigma, \text{SSInfo}, K)$ . As a consequence, this simulation is perfectly indistinguishable from the previous one unless one finds a second pre-image to  $H = \mathcal{H}(\Pi, \Sigma, \text{SSInfo}, K)$  (where  $t$  is essentially the running time of  $\mathcal{A}$ ):

$$\text{Succ}_{\mathbf{G}_2}(\mathcal{A}) \leq \text{Succ}_{\mathbf{G}_3}(\mathcal{A}) + \text{Succ}_{\mathcal{H}}^{2\text{nd}}(t).$$

**Game  $\mathbf{G}_4$ :** We are still dealing with  $\text{Send}$ -queries to the user, but we consider the case of the event  $\neg\text{PWinC}$ . We now use  $\text{Sim}_U$ , that perfectly simulates a flow  $C$  from the user to a server, and can decide on the honest behavior of the server by choosing itself the value  $D$  (the decryption of  $C$ ), and using it to check the correctness of servers' answers.

If the behaviors of at least  $t_r$  of the servers are correct, the user accepts with  $K \leftarrow K_1$ , otherwise the user aborts with  $K \leftarrow \perp$ .

In the previous game, the robust threshold secret sharing scheme with respect to random failures guaranteed the recovery of the secret in exactly the same cases as here:

$$\text{Succ}_{\mathbf{G}_3} = \text{Succ}_{\mathbf{G}_4}.$$

We remark that during this game, the  $\text{Send}$ -queries are answered without computing the PRF for the reconstruction of  $K$ . Instead, after  $G = g^D$  check, the secret  $K$  is directly set to  $K_1$  (or to  $\perp$  if too many failures).

**Game  $\mathbf{G}_5$ :** We now deal with  $\text{Send}$ -queries to the servers, and replace them by the simulator  $\text{Sim}_S$  to provide answers, getting  $x$  and the blinding factor  $\alpha$  from the extractor of the proof: in order to set the appropriate output to  $R$  (that is  $F_a(x)$ ), the server can simply answer  $G \leftarrow R^\alpha$ . The simulation is perfect:

$$\text{Succ}_{\mathbf{G}_4} = \text{Succ}_{\mathbf{G}_5}.$$

**Game  $\mathbf{G}_6$ :** In the servers' simulation, the value  $R$  is now chosen at random for new  $x$  (for the uncorrupted servers). This corresponds to replace  $F_a$  by a truly random function when calling to the PRF oracle. Under the pseudo-randomness of the Naor-Reingold PRF:

$$\text{Succ}_{\mathbf{G}_5} \leq \text{Succ}_{\mathbf{G}_6} + (n - q_c)\ell \times \text{Adv}^{\text{ddh}}(t + q_s t_{\text{exp}}),$$

where  $t_{\text{exp}}$  is the additional time for exponentiations in the reduction of the PRF.

**Game  $\mathbf{G}_7$ :** Instead of choosing the  $\pi_k$  at random, one generates  $\Sigma = (\sigma_k)_k$  and  $\text{SSInfo}$  at random. This leads to random and independent shares  $(s_1, \dots, s_n)$ . We know that until the adversary does not get more than  $t_\ell = n/3$  of these shares, it cannot detect whether they are independent or redundant (as should be a secret sharing): let us define the event  $\text{PWinF}'$ , a little bit different from the previous proof, to be true if more than  $n/3 - q_c$  queries have been asked to the servers on  $\text{pw}$ , since the adversary can evaluate on its own the values under the  $q_c$  corrupted servers. Then, from these values it could remark inconsistencies. At the end, after the answer  $b'$ , if  $\text{PWinF}'$  is set, one sets  $S$  at random. Since the PRF's are replaced by truly random functions, these

queries do not reveal anything on the other values of the functions, we have the upper-bound:  $\text{Succ}_{\mathbf{G}_6} \leq \text{Succ}_{\mathbf{G}_7} + \Pr[\text{PWinF}']/2$ . Thanks to the extractability of  $x$  from the proof, we are able to check whether  $\text{pw}$  has been used, and how many times in order to set the event  $\text{PWinF}'$ .

**Game  $\mathbf{G}_8$ :** We now replace the commitment  $C$  in the initialization phase by a dummy commitment to 0. This is indistinguishable under the indistinguishability of the encryption scheme (Cramer-Shoup encryption), but the decryption key is required to evaluate  $\text{PWinC}$ :

$$\text{Succ}_{\mathbf{G}_7} \leq \text{Succ}_{\mathbf{G}_8} + \text{Adv}_{\text{CS}}^{\text{ind-cca}}(t).$$

**Game  $\mathbf{G}_9$ :** The key  $K_1$  does not appear any more in the simulation of the secret sharing, as the values  $\text{PInfo}$  have been replaced by random and independent values instead of shares and the commitment  $C$  is currently computed for 0. Then, we can replace  $K_1$  by  $K_0$  in the reconstruction phase, which makes the real and random cases indistinguishable:

$$\text{Succ}_{\mathbf{G}_8}(\mathcal{A}) = \text{Succ}_{\mathbf{G}_9}(\mathcal{A}) = 1/2.$$

In this final game, the password does not appear any more in the initialization of  $\text{PInfo}$ , and the simulator does not make use of it either, except to abort if  $\text{PWinC}$  or  $\text{PWinF}$  happen. But these events can be evaluated at the very end only, by choosing a random password  $\text{pw}$  when the adversary outputs its guess  $b'$ :

- $\Pr[\text{PWinC}]$  is clearly upper-bounded by  $q_u/\#\mathcal{D}$ , since  $q_u$  is the maximal number of fake commitment attempts that could be different from the expected one but with  $\text{pw}$ ;
- $\Pr[\text{PWinF}']$  is clearly upper-bounded by  $q_s/(n/3 - q_c) \times 1/\#\mathcal{D}$ , since  $q_s/(n/3 - q_c)$  is the maximal number of passwords for which the adversary asked for  $n/3 - q_c$  OPRF evaluations.

In conclusion, we have:

$$\begin{aligned} \text{Adv}(\mathcal{A}) &\leq \left( q_u + \frac{3q_s}{n - 3q_c} \right) \times \frac{1}{\#\mathcal{D}} \\ &\quad + (n - q_c)\ell \times \text{Adv}^{\text{ddh}}(t) + 2 \times \text{Adv}_{\text{CS}}^{\text{ind-cca}}(t) + \text{Succ}_{\mathcal{H}}^{2\text{nd}}(t). \end{aligned}$$

Since we use the Cramer-Shoup encryption scheme, this leads to the bound of the Theorem.