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 its 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 propose a generic technique to build a *Robust Gap Threshold Secret Sharing Scheme* (RGTSSS) from any threshold secret sharing scheme. In the PPSS construction, this allows us to drop the verifiable property of *Oblivious Pseudorandom Functions* (OPRF). Then, we use this new approach to design two new robust PPSS schemes that are quite efficient, from two OPRFs. They are proven in the random oracle model, just because our RGTSSS construction requires random non-malleable fingerprints. This is easily guaranteed when the hash function is modeled as a random oracle.

Keywords: Password-Protected Secret Sharing, Robust Gap Threshold Secret Sharing Scheme, 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 offline dictionary attack would allow the recovery. On the other hand, there are techniques using passwords that are not vulnerable to such offline dictionary attacks, like *password authenticated key exchange* (PAKE) [BM92]. For these PAKE protocols, the best attacks require the adversary to be online, 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 online 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. However, in [BJSL11] they prove their scheme secure in the random oracle model assuming an additional PKI. 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.

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 online 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 main contribution is the efficient realization of the robustness, that consists in a single check at the very end of the protocol, during the secret reconstruction.

We point out that, in order to achieve robustness in a PPSS protocol, one does not need to distinguish between correct and incorrect shares at each individual evaluation with a server as in the verifying setting presented in [JKK14].

Actually, we propose a new efficient method to convert any Secret Sharing Scheme in a (t_ℓ, t_r, n)-*Robust Gap Threshold Secret Sharing Scheme* (RGTSSS) 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_ℓ shares are correct, the protocol *leaks* no information about which shares are correct.

More precisely, we are using an encoding to generate fingerprints of the shares. The assumption that invalid shares encode into random fingerprints is enough for the reconstruction technique to be able to select the correct shares, when their number is high enough. This can be achieved using a hash function modeled as a random oracle [BR93].

Such a (t_ℓ, t_r, n)-RGTSSS 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 and reconstruct the secret. If the number of answers is below another threshold t_ℓ, no information about the secret is leaked. 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 easily mount an offline dictionary attack. With our new primitive, even t_ℓ corrupted servers cannot perform an offline dictionary attack as they would still need to interact with at least one additional server. 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 highly improves on the global efficiency.

While similar to [JKKX16] in terms of server interaction efficiency, our technique takes advantage of the RGTSSS to optimize the secret reconstruction. The scheme proposed by [JKKX16] has one significant drawback: the client is supposed to specify the exact set of servers involved in the secret recovery from the beginning, which may lead to frequent failures as the servers may misbehave. Moreover, in case of such a

failure, the user is unable to detect the cheating servers. To overcome this drawback when a large number of servers are involved in the protocol, our approach makes use of the *robustness* feature, to ensure the recovery of the secret and the detection of dishonest servers.

We propose two efficient OPRF constructions: The first one is based on the One-More Gap Diffie-Hellman assumption and its efficiency is quite similar to the one in [JKKX16]. Secondly, we introduce a new oblivious evaluation of the Naor-Reingold PRF [NR97], based on the sole DDH assumption.

For this new construction, we compare very favorably to other oblivious evaluations of the Naor-Reingold PRF: our protocol simply uses ElGamal encryption [ElG85] in prime order groups with simple zero-knowledge proofs, whereas for example the scheme in [JKK14] has to work in composite order groups with Paillier encryption [Pai99] and more complex zero-knowledge proofs.

By combining these building bricks, we eventually reach efficient PPSS schemes that satisfy *Soundness* and *Robustness* properties. The two proposed solutions are eventually proven in the *Random Oracle Model* (ROM) [BR93], as our RGTSSS construction requires random non-malleable fingerprints. This can be achieved by using a hash function that is modeled as a random oracle.

Related Work. 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]. This concept was later generalized by using two thresholds, a *upper* and a *lower* one to set the size of the sets to reconstruct and to preserve privacy respectively. In Shamir's secret sharing scheme, the privacy threshold is defined as t and the reconstruction threshold as $t+1$. When this gap is higher, then the secret sharing scheme is called *ramp* scheme [BM84,BSV93,MJ96,MS81]. Ramp schemes to achieve a robust secret sharing scheme have been extensively studied, we refer the reader to [Che15] and [BPRW15].

The first formal definition of Password Protected Secret Sharing was introduced by Bagherzandi *et al.* [BJSL11]. 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 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 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 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. Recently, the work [JKKX16] improves the performance of this password-only PPSS on the

cost of dropping the robustness property. Their protocol is relaxing the verifiable property of the OPRF, giving up the ability to discard incorrect computations during interactions with servers. This can be a good alternative for a small number n of servers, the only setting that allows checking in a reasonable time different subsets of servers until finding a non-corrupted one.

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.

2 Security Model

In order to analyze the security of 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 [BJSL11] 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 PAKE.

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, $\mathtt{User} \cup \mathtt{Server}$.

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

Each server $S \in \mathtt{Server}$ holds a secret key sk, and possibly an associated public key pk. However we stress that even if there is a public key 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 PInfo: nobody else than U has any information about K, however 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 some public keys in 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:

- $\mathsf{Execute}(U^i, \{S_k^{j_k}\})$: 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;
- $\mathsf{Send}(P^i, m)$: 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 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 Test-query, where b is a global secret random bit:

- $\mathsf{Test}(U^i)$: 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 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 $\mathsf{Corrupt}$-query:

- $\mathsf{Corrupt}(S_k)$: This outputs the secret key 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 indistinguishably of the secret keys, a.k.a. *semantic security*, the adversary can ask as many oracle queries ($\mathsf{Execute}$, Send, Test, and $\mathsf{Corrupt}$), 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

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

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_ℓ shares, which may leak some information about the actual secret key: it thereafter asks for an $\mathsf{Execute}$-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 $\mathsf{Corrupt}$-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 PInfo that it generated itself from a randomly chosen password 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_ℓ, ε, t)-secure if for any adversary $\mathcal{A}$, running within time t, asking at most $q_c < t_\ell$ $\mathsf{Corrupt}$-queries and invoking at most q_u user instances and q_s server instances,

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

In [JKK14], they proposed such a protocol that achieves the optimal t_ℓ-security, for $t_\ell + 1 = t_r$, but at the cost of verifiable oblivious pseudorandom functions. Our goal is to build much more efficient protocols, possibly lowering the security level: $t_\ell = 3t_r - 2n - 1$.

Before going into more details about our constructions, let us review the required or expected properties for a PPSS, initiated with a password 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 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 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 to 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 trying 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 and the classical building blocks in the Appendices A and B respectively. 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 $(\mathsf{sk}_k, \mathsf{pk}_k)$ that defines a PRF F_k, with public parameters defined by pk_k and a secret key defined by sk_k. For a password $\mathsf{pw} \in \mathcal{D}$, the user asks for an oblivious evaluation of $\pi_k = F_k(\mathsf{pw})$ to n servers, where $\Pi = (\mathsf{pk}_k)_k$ is the tuple of the public keys of the involved servers. The secret key K is then split into shares $(s_1, \ldots, s_n)$ and some extra public information PInfo, 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, $(\mathsf{pk}_k)_k$ 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 PInfo, and help him to compute each $\pi_k = F_k(\mathsf{pw})$ in an oblivious way. We assume that the user received the same value PInfo from at least t_r servers, and then the user keeps the majority value. Using PInfo and enough evaluations π_k, the user can extract enough shares among $(s_1, \ldots, s_n)$ and reconstruct a value K. He can then verify whether this is the expected secret key,

from the majority PInfo which is however not considered authentic. 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 sk and the user input is the password pw, and the user only gets the output $F_{\mathsf{sk}}(\mathsf{pw})$, but none of the players learn any additional information about the other player's input;
- a (t_ℓ, 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_ℓ 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 PInfo, whereas it controls all the communications.

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 Gap Threshold Secret Sharing Scheme*.

4 A Robust Gap Threshold Secret Sharing Scheme

Our technique is generic, and so we start from any threshold secret sharing scheme, with two algorithms ShareGen and Reconstruct that respectively share a secret and reconstruct it. One can for example use the classical Shamir's secret sharing scheme [Sha79] to which we will add this new robustness feature, at the cost of having a threshold gap secret sharing scheme that is enough to get a robust PPSS scheme (for details about secret sharing schemes see the Appendix D.

Let us first give the intuition of the technique, and we then explain how we can realize it in practice.

4.1 Intuition

The valid shares are denoted $(s_1, \ldots, s_n)$ and the fingerprints of these shares $(\sigma_1, \ldots, \sigma_n)$. 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 $(\tau_1, \ldots, \tau_m)$ 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 detect the valid shares when the fingerprints are either correct or random.

Initialization. We assume we have a set of n initial values $(s_1, \ldots, s_n)$, and their k-bit string fingerprints $(\sigma_1, \ldots, \sigma_n)$. As fingerprint function we use a hash function $F : \{0,1\}^* \to \{0,1\}^k$ modeled as a random oracle.

In the following, we will be given a set of m candidate shares, whose fingerprints are $(\tau_1, \ldots, \tau_m)$: these fingerprints are either correct (the same as in the list $(\sigma_1, \ldots, \sigma_n)$ or random for incorrect candidate

shares). From this set of candidate shares, if at least t_r are correct, we want to efficiently identify the correct values (to *recover* the secret in a threshold secret sharing scheme, hence the r-subscript in t_r). However, if at most t_ℓ are correct, the protocol should not *leak* any information about which candidates are valid and which are not (hence the ℓ-subscript in t_ℓ).

From the initial set $(\sigma_1, \ldots, \sigma_n)$ 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 $\mathsf{SSInfo} = (\mathcal{S}, N)$.

Reconstruction. Given the $\mathsf{SSInfo} = (\mathcal{S}, N)$ and fingerprints $(\tau_1, \ldots, \tau_m)$ 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 $\mathcal{T}' < 2^{k(m-t_r)} \leq 2^{k(n-t_r)}$ and $\mathcal{S}' < 2^{k(n-t_r)}$.

Our experimental results (for details, see the Appendix D) show that on one hand, approximately one half of the cases $\mathcal{T}'$ and $\mathcal{S}'$ are coprime. On the other, both values share some small factors. We denote $\mathcal{T}''/\mathcal{S}''$ as the irreducible fraction where all the small common factors of $\mathcal{T}'/\mathcal{S}'$ were canceled out.

Using the following result from [FSW03], we can recover the $\mathcal{T}''/\mathcal{S}''$ of $\gamma = \mathcal{T}'/\mathcal{S}' = \mathcal{T}/\mathcal{S}$, with $\mathcal{T}'' \leq \mathcal{T}' < 2^{k(n-t_r)}$ and $\mathcal{S}'' \leq \mathcal{S}' < 2^{k(n-t_r)}$, 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 γ.

Now, if τ_i is the fingerprint of a valid share, it should be canceled out in $\mathcal{T}'$, but there might still be some small factors in common between τ_i and $\mathcal{T}''$. On the other hand, if τ_i is the fingerprint of a random invalid share, it should not be completely canceled out in $\mathcal{T}'$. However, there is still a chance that some small factors have been canceled out, leading to $\mathcal{T}''$ in the irreducible form.

Hence, our decision algorithm is the following one: we denote t_i the bit size of $|\gcd(\mathcal{T}'', \tau_i)|$; if $t_i \geq k/2$, this is an invalid share, otherwise this is a valid share. In Figure 1, we present experimental results that validate this decision algorithm for $k = 128$-bit. It is possible to see clearly that for a valid τ_i, t_i is a small number (half of them equal to 1) and for an invalid τ_i, t_i is a big number (44% of them is equal to 2^k). We have computed 2^{21} times the value of $\gcd(\mathcal{T}'', \tau_i)$ and in case of Figure 1a, the highest bit size of t_i is 35 (much less than 64). On the other hand, in Figure 1b the least value is 96 (much more than 64). Both with probability 1 over 2^{21}. A more fine analysis with different sizes of the fingerprint and the number of shares can be found in the Appendix D.

Information Leakage. On the opposite, we would like to evaluate the information leaked by $\mathcal{S}$ when there are at most t_ℓ valid values. More precisely, given $\mathcal{S}$, is it possible to distinguish t_ℓ valid values from t_ℓ random values?

For a t_r-threshold secret sharing scheme, the entropy of the tuple $(\sigma_1, \ldots, \sigma_n)$ is $k(t_r - 1)$. Since $\mathcal{S}$ reveals the product modulo N, with $N < 2^{2k(n-t_r)+2}$, the remaining entropy on the shares is at least $k(t_r - 1) - 2k(n - t_r) - 2 = k(3t_r - 2n - 1) - 2$. If this is greater than kt_ℓ, no one can distinguish t_ℓ random values from t_ℓ correct values for the shares: we thus need $k(3t_r - 2n - 1) - 2 \geq kt_\ell$. When $k > 2$, this essentially means $t_\ell \leq 3t_r - 2n - 1$: by choosing $t_\ell = 3t_r - 2n - 1$, we are safe. For example, one can take $t_r = \lceil 3n/4 \rceil$ and $t_\ell = \lfloor n/4 \rfloor$.

5 Our Password-Protected Secret Sharing Protocols

Thanks to our new (t_ℓ, t_r, n)-RGTSSS, we do not need to use a VOPRF. 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.

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

- the first OPRF relies on the CDH evaluation, similar to the protocol 2HashDH from [JKK14], but without NIZKs. It leads to a PPSS construction quite similar to [JKKX16].
- the second OPRF is an oblivious evaluation of the Naor-Reingold PRF [NR97]. 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 $(\mathsf{sk}_k, \mathsf{pk}_k)$ that defines a PRF F_k, with public parameters defined by pk_k and a secret key defined by 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_ℓ, t_r, n)-robust gap threshold secret sharing scheme, where we can assume that $t_r = 3n/4$ and $t_\ell = n/4 - 1$ (which is compatible with our previous construction), and a commitment scheme Commit (see the Appendix B). The user U first chooses a secret password pw:

1. the user interacts with n servers to obliviously evaluate $\pi_k = F_k(\mathsf{pw})$, and $\Pi = (\mathsf{pk}_k)_k$ 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 $(s_1, \ldots, s_n, \mathsf{SSInfo}) \leftarrow \mathsf{ShareGen}(R)$, so that any subset of t_r shares among $\{s_1, \ldots, s_n\}$ can efficiently reconstruct R;
3. then, the user builds $\sigma_k = \pi_k \oplus s_k$, for $k = 1, \ldots, n$, and sets $\Sigma = (\sigma_k)_k$;
4. the user generates $C = \mathsf{Commit}(\mathsf{pw}, \Pi, \Sigma, \mathsf{SSInfo}, K; r)$. We denote by $\mathsf{PInfo} = (\Pi, \Sigma, \mathsf{SSInfo}, C)$ the public information that the user will need later to recover its secret K;
5. the user thus gives PInfo to all the servers.

We stress that during this initialization phase, all the values of Π 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 PInfo will be provided by the servers, but through the adversary, who might alter them.

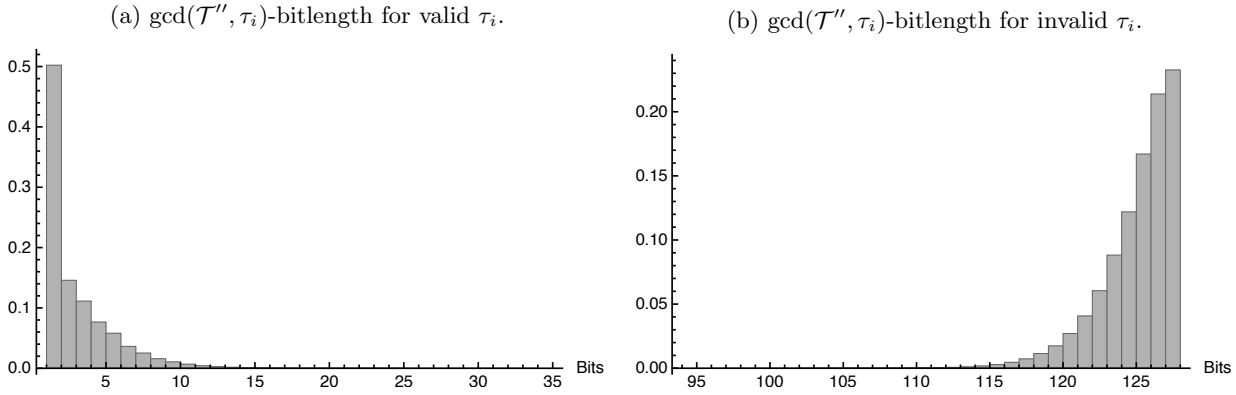


Fig. 1: Length in bits of gcd($\mathcal{T}''$, τ_i) for a fingerprint of size 128-bits and 32 shares

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

1. the user first limits the oblivious evaluations of $\pi_k = F_k(\mathsf{pw})$ to the servers that sent the same majority tuple $\mathsf{PInfo} = (\Pi, \Sigma, \mathsf{SSInfo}, C)$. If the number of such servers is less than t_r, one aborts with $K \leftarrow \perp$;
2. for all these π_k (or similarly, all the k he kept), the user computes $s_k = \sigma_k \oplus \pi_k$, using σ_k from Σ (from PInfo);
3. using these $\{s_k\}$ with at least t_r correct shares, and SSInfo (from PInfo), with RGTSSS, 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 PInfo, whether $C = \mathsf{Commit}(\mathsf{pw}, \Pi, \Sigma, \mathsf{SSInfo}, K; r)$;
5. if the verification succeeds, K is the expected secret key, otherwise the user aborts with $K \leftarrow \perp$.

5.2 Protocol I: One-More-Gap-Diffie-Hellman-based PRF

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 $\{0,1\}^{\ell_3}$: $C = \mathsf{Commit}(\mathsf{pw}, \Pi, \Sigma, \mathsf{SSInfo}, K; r) := H_3(\mathsf{pw}, \Pi, \Sigma, \mathsf{SSInfo}, K, r)$, which allows a better efficiency.

For a private key $\mathsf{sk} = x \in \mathbb{Z}_q$, 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 $\mathsf{pk} = y = g^x$. 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 presented in Figure 2: $\mathcal{S}im$ 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 full security proof. They generate perfectly indistinguishable views to the adversary, but they require $\mathsf{CDH}_g(y, \cdot)$ and $\mathsf{DDH}_g(y, \cdot, \cdot)$ evaluation, and thus oracle access when the secret keys are not known. Since the indistinguishability of the PRF relies on the $\mathsf{CDH}_g(y, \cdot)$ assumption, the overall security relies on the One-More Gap Diffie-Hellman (OMGDH) assumption (see the 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*

$$\mathsf{Adv}(\mathcal{A}) \leq \left(q_u + \frac{4q_s}{n - 4q_c}\right) \times \frac{1}{\#\mathcal{D}} + n \times \mathsf{Succ}^{\mathsf{omgdh}}(q_1, q_s, t, n \cdot q_u + q_2) + (q_3^2 + 2) \cdot 2^{-\ell_3}.$$

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

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

Fig. 2: Secure Oblivious Evaluation of the PRF based on OMGDH

Security Proof. The complete and detailed proof of the Theorem is given in the 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 PInfo' (sent by the majority of servers): first, we do as before if the commitment C' in 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 pw, which is defined as the event PWinC. This event PWinC can be evaluated using the list of queries asked to H_3. Then, a similar argument applies when a wrong PInfo' is sent, but with a correct C, under the binding propriety of the commitment H_3.

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 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 $t_\ell = n/4$ of these shares, it cannot detect whether they are random or correct. We define the event 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 PWinF by checking whether 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 pw is not used anymore during the simulation, but just to determine whether the events PWinC or 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: $\mathsf{Succ}_{\mathcal{A}} = 1/2$ in the last game. As a consequence, $\mathsf{Adv}(\mathcal{A}) \leq \Pr[\mathsf{PWinC}] + \Pr[\mathsf{PWinF}] + \varepsilon$, where ε comes from the collisions or guess on the random oracles. To evaluate the two events PWinC or PWinF to happen, we choose a random password pw at the very end only: $\Pr[\mathsf{PWinC}]$ is clearly upper-bounded by $q_u/\#\mathcal{D}$, since q_u is the maximal number of fake commitment attempts containing the right pw that can be different from the expected ones; PWinF means that the adversary managed to get $n/4 - q_c$ evaluations of the PRFs under the chosen 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 OMGDH assumption), the number of *bad* passwords (for which the knows at least $n/4 - q_c$ evaluations of the PRFs) is less than $q_s/(n/4 - q_c)$. So the probability that the chosen pw is such a bad password is less than $q_s/(n/4 - q_c) \times 1/\#\mathcal{D}$.

5.3 Protocol II: DDH-based PRF

Our second instantiation makes use of the Naor and Reingold [NR97] pseudorandom function. 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.

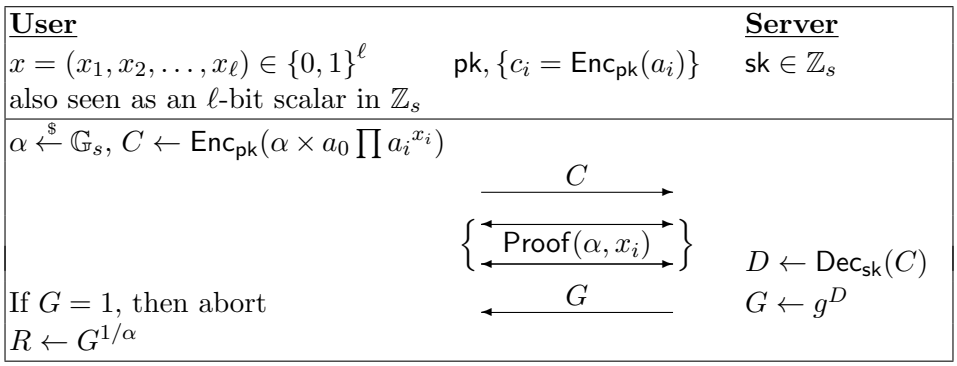


Fig. 3: Secure Oblivious Evaluation of the NR-PRF

The PRF key is a tuple $a = (a_0, a_1, \ldots, a_\ell) \xleftarrow{\$} (\mathbb{G}_s \backslash \{1\})^{\ell+1}$, and $F_a(x) = g^{a_0 \prod {a_i}^{x_i}}$, where $x = (x_1, x_2, \ldots, x_\ell) \in \{0,1\}^\ell$. This function has been proven to be a PRF under the DDH assumption [NR97] on ℓ-bit inputs. It also admits a simple oblivious evaluation (just the messages C and G from Figure 3), using a multiplicatively homomorphic encryption scheme in $\mathbb{G}_s$, such as ElGamal for $(\mathsf{Enc}_{\mathsf{pk}}, \mathsf{Dec}_{\mathsf{sk}})$, which allows the computation of C from x, α, 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 3 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. Whereas our oblivious evaluation of the PRF is in the standard model, overall, the PPSS protocol based on this OPRF is in the random oracle model as it makes use of the RGTSSS. As a consequence, one could replace the interactive ZK proofs by NIZK proofs "à la Schnorr". This would reduce the number of flows to only 2. The full proof can be found in the Appendix C.

6 Comparisons

We can assume that 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 is quite similar to the one from [JKKX16]. 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 algebraic OPRF structure is more efficient than the one in [FIPR05], since their construction makes use of Oblivious Transfers (OT) and expensive public-key operations. In the online setting, this kind of protocols are almost infeasible, as the number of desired OTs is not known in advance while our zero-knowledge proofs are much simpler to use. Given the work of Ishai *et al.* [IKNP03], a better efficiency can be achieved, considering each OT evaluation at the cost of a private-key operation. In our case, the main cost in communication is that of a single zero-knowledge proof.

Our second protocol, based on this oblivious evaluation and with an additionally CRS turns out to be much more efficient than the one from [JKK14]. 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

We are grateful to Stanislaw Jarecki for his valuable comments on this work. 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 – CryptoCloud).

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.

BM84. G. R. Blakley and C. Meadows. Security of ramp schemes. In *CRYPTO'84*, *LNCS* 196, pages 242–268. Springer, Heidelberg, August 1984.

BM92. S. M. Bellovin 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.

BPRW15. A. Bishop, V. Pastro, R. Rajaraman, and D. Wichs. Essentially optimal robust secret sharing with maximal corruptions. Cryptology ePrint Archive, Report 2015/1032, 2015. `http://eprint.iacr.org/2015/1032`.

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.

BSV93. C. Blundo, A. D. Santis, and U. Vaccaro. Efficient sharing of many secrets. In *Proceedings of the 10th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '93, pages 692–703, London, UK, UK, 1993. Springer-Verlag.

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.

Che15. M. Cheraghchi. Nearly optimal robust secret sharing. Cryptology ePrint Archive, Report 2015/951, 2015. `http://eprint.iacr.org/2015/951`.

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.

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.

IKNP03. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, *LNCS* 2729, pages 145–161. Springer, Heidelberg, August 2003.

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.

JKKX16. S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. Highly-Efficient and Composable Password-Protected Secret Sharing. Cryptology ePrint Archive, Report 2016/144, 2016. `http://eprint.iacr.org/`.

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`.

MJ96. K. Martin and W.-A. Jackson. "a combinatorial interpretation of ramp schemes". *Australasian Journal of Combinatorics*, 14:51–60, 1996.

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 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 $\mathsf{Succ}^{\mathsf{cdh}}(\mathcal{A})$ the success probability of the adversary $\mathcal{A}$ in computing g^{xy}, and more generally, $\mathsf{Succ}^{\mathsf{cdh}}(t)$ is the best success probability an adversary can get within time t.

Decisional Diffie-Hellman Assumption (DDH). The 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 $\mathsf{Adv}^{\mathsf{ddh}}(\mathcal{A})$ the advantage of the adversary $\mathcal{A}$ in distinguishing between the two distributions, and more generally, $\mathsf{Adv}^{\mathsf{ddh}}(t)$ is the best advantage an adversary can get within time t.

Gap Diffie-Hellman Assumption (GDH). The GDH_g assumption [OP01] states that the CDH_g assumption holds even when the adversary has access to a DDH_g oracle that exactly answers for any query $\mathsf{DDH}_g(g^x, g^y, g^z)$ whether $z = xy$ or not. $\mathsf{Succ}^{\mathsf{gdh}}(\mathcal{A})$ and $\mathsf{Succ}^{\mathsf{gdh}}(t, q_d)$ are defined as above, where $\mathcal{A}$ can ask up to q_d 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 $(g_1, \cdots, g_n) \xleftarrow{\$} \mathbb{G}^n$, unlimited access to a $\mathsf{DDH}_g(g^x, \cdot, \cdot)$ oracle, and up to m queries to a $\mathsf{CDH}_g(g^x, \cdot)$ oracle, it is hard to output $m+1$ valid pairs (g_i, g_i^x).

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

B Building Blocks

B.1 General Definitions

Threshold Secret Sharing Scheme. 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 $P_1, \ldots, P_n$, with algorithms $\mathsf{ShareGen}$ and $\mathsf{Reconstruct}$:

- $\mathsf{ShareGen}(s, t)$: on a secret s and a threshold t, this algorithm generates n shares $(s_1, \ldots, s_n)$, and possible public information SSInfo;
- $\mathsf{Reconstruct}(\{s_i\}, \mathsf{SSInfo})$: on a set of t shares, and the possible additional information SSInfo, this algorithm recovers the secret s.

The correctness guarantees that the $\mathsf{Reconstruct}$ 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 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 SSInfo will contain global information on the shares only, and no information on the individual shares: for the construction we propose in this paper, 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 $(\mathsf{ek}, \mathsf{dk})$. The encryption algorithm $\mathcal{E}$ takes as input an encryption key ek and a message m and outputs a ciphertext c. The decryption algorithm $\mathcal{D}$ takes as input a decryption key dk and a ciphertext c and outputs either the decryption m of c or $\perp$. The correctness condition required is that for all $(\mathsf{ek}, \mathsf{dk})$ generated by $\mathcal{K}$, and for all messages m, $\mathcal{D}(\mathsf{dk}, \mathcal{E}(\mathsf{ek}, m)) = m$. Classical security notions for encryption are $\mathsf{IND}-\mathsf{CPA}$ and $\mathsf{IND}-\mathsf{CCA}$, 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 [ElG85], based on the CDH assumption, and achieving $\mathsf{IND}-\mathsf{CPA}$ 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:

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

Cramer-Shoup Encryption. The Cramer-Shoup encryption scheme [CS98] achieves $\mathsf{IND}-\mathsf{CCA}$ 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 $\mathsf{IND}-\mathsf{CCA}$ encryption scheme can be used as a perfectly binding commitment scheme. The decryption key allows extractability and the $\mathsf{IND}-\mathsf{CCA}$ 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:

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

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 $h^x = \mathsf{CDH}_g(y, h)$. 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 Λ_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 Λ_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 $H = \mathsf{CDH}_g(y, H_1(m))$ and (m, z, r) is in Λ_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 $H = \mathsf{CDH}_g(y, H_1(m)) = \mathsf{CDH}_g(y, h^z) = \mathsf{CDH}_g(y, h)^z$. By choosing it at random, one gets $\mathsf{CDH}_g(y, h)$ 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 $\mathbf{G}_0$: 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 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 $(s_1, \ldots, s_n, \mathsf{SSInfo})$ of $R = K \| r$, for a random r. This last random value r is used to compute the commitment $C = H_3(\mathsf{pw}, \Pi, \Sigma, \mathsf{SSInfo}, K, r)$, 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 :

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

Game $\mathbf{G}_1$: We do not modify the initialization, and first deal with Execute-queries, by replacing the user and the servers by the simulator $\mathcal{S}im$ that perfectly simulates honest transcripts (A, B), and user's key is set to K_1. The change being just syntactic: $\mathsf{Succ}_{\mathbf{G}_0} = \mathsf{Succ}_{\mathbf{G}_1}$.

Game $\mathbf{G}_2$: We now deal with Send-queries to the user, and namely when the adversary fakes the public information PInfo sent to the user: if the majority of at least t_r tuples $\mathsf{PInfo}' = (\Pi', \Sigma', \mathsf{SSInfo}', C')$

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 pw. Since we use a hash function H_3 modeled as a random oracle, C' must have been obtained with a query containing pw, or the probability to be valid is $1/2^{\ell_3}$: We thus define the event PWinC to be true if $C' \neq C$ but C' is the result of a query of H_3 on a tuple that contains pw. And at the end, after the answer b', if 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 PWinC happens: $\mathsf{Succ}_{\mathbf{G}_1} \leq \mathsf{Succ}_{\mathbf{G}_2} + 1/2^{\ell_3} + \Pr[\mathsf{PWinC}]/2$. This event PWinC can be evaluated by looking at each of the queries asked to H_3 and then checking whether it contains pw or not.

Game $\mathbf{G}_3$: We continue in the same vein for fake public information PInfo' (but correct C) sent to the user: if the majority of at least t_r tuples $\mathsf{PInfo}' = (\Pi', \Sigma', \mathsf{SSInfo}', C)$ contains public information different from the expected ones (the 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 $(\mathsf{pw}, \Pi', \Sigma', \mathsf{SSInfo}')$ must be unchanged, unless one finds a collision for H_3:

$$\mathsf{Succ}_{\mathbf{G}_2} \leq \mathsf{Succ}_{\mathbf{G}_3} + q_3^2/2^{\ell_3+1}.$$

Game $\mathbf{G}_4$: We continue with the simulation of the user, but when the majority 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 $\mathcal{S}im_U$, that perfectly simulates a flow A from the user to a server, and can decide on the honest behavior of the server thanks to the server' secret key x_k to evaluate $\mathsf{DDH}_g(y_k, \cdot, \cdot)$. If the behaviours of at least t_r of the servers are correct, thanks to the RGTSSS scheme, 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 α, either $C = H_1(\mathsf{pw})^{x_k}$ or C is a random element in $\mathbb{G}$, unless $B = 1$, hence the verification. Applying the hash function H_2 to obtain the final value R assures us that the shares are either correct, or random, hence the encoding into random non-malleable fingerprints in the RGTSSS process.
In the previous game, RGTSSS guaranteed the recovery of the secret in exactly the same cases as here: $\mathsf{Succ}_{\mathbf{G}_3} = \mathsf{Succ}_{\mathbf{G}_4}$. We remark that during this game, the Send-queries are answered without calling the H_1 oracle, neither H_2 oracle is used for the reconstruction of K. Instead, after the $\mathsf{DDH}_g(y, \cdot, \cdot)$ 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 $(s_1, \ldots, s_n)$ by random and independent values and generate SSInfo accordingly in the initialization phase. We know that until the adversary does not get more than $t_\ell = n/4$ of these shares, it cannot detect whether they are random or correct: let us define the event PWinF to be true if more than $n/4 - q_c$ queries have been asked to the H_2 oracle for the un-corrupted key y_k on pw with the correct 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 PWinF is set, one sets S at random. As in Game $\mathbf{G}_2$, we have the upper-bound: $\mathsf{Succ}_{\mathbf{G}_4} \leq \mathsf{Succ}_{\mathbf{G}_5} + \Pr[\mathsf{PWinF}]/2$. Using the servers' secret keys $(x_k)_k$ (to test $\mathsf{DDH}_g(y_k, \cdot, \cdot)$ validity), the simulator can check whether pw has been queried with the right inputs to H_2 to learn some π_k, and how many times, to set the event PWinF.

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

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

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: $\mathsf{Succ}_{\mathbf{G}_7} = \mathsf{Succ}_{\mathbf{G}_8}$.

In this last game, one can note that the password pw is not used anymore during the simulation, but just to determine whether the events PWinC or 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: $\mathsf{Succ}_{\mathbf{G}_8} = 1/2$. As a consequence,

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

We thus now have to evaluate the probabilities of the two events PWinC or PWinF to happen, which can be done by choosing a random password pw at the very end only (since it is not used anymore during the initialization phase, nor in the reconstruction): About $\Pr[\mathsf{PWinC}]$, it is clearly upper-bounded by $q_u/\#\mathcal{D}$, since q_u is the maximal number of fake commitment attempts containing the right pw that can be different from the expected ones; On the other hand, PWinF means that the adversary managed to get $n/4 - q_c$ evaluations of the PRFs under the chosen 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 $n/4 - q_c$ evaluations of the PRFs) is less than $q_s/(n/4 - q_c)$. So the probability that the chosen pw is such a bad password is less than $q_s/(n/4 - q_c) \times 1/\#\mathcal{D}$.

The following lemma leads to $\Pr[\mathsf{PWinF}] \leq q_s/(n/4 - q_c) \times 1/\#\mathcal{D} + n \times \mathsf{Succ}^{\mathsf{omgdh}}(q_1, q_s, t, n \cdot q_u + q_2)$, which concludes the proof of the theorem.

Lemma 4. *Unless one can break a $(q_1, q_s) - \mathsf{OMGDH}$ 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 $(g, y = g^x, h_1, \cdots, h_{q_1})$ of the $(q_1, q_s) - \mathsf{OMGDH}$ 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 $\mathsf{CDH}_g(y^*, \cdot)$-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 $\mathsf{DDH}_g(y^*, \cdot, \cdot)$-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 (pw_i, y^*, H) to the random oracle H_2, $H = \mathsf{CDH}_g(y^*, H_1(\mathsf{pw}_i))$. Since $H_1(\mathsf{pw}_i)$ has been answered by one of the h_j, one gets $q+1$ correct values $\mathsf{CDH}_g(y^*, h_j)$, that can be detected using the $\mathsf{DDH}_g(y^*, \cdot, \cdot)$ 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*

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

where ℓ is the size of the password and t_{exp} the time for an exponentiation.

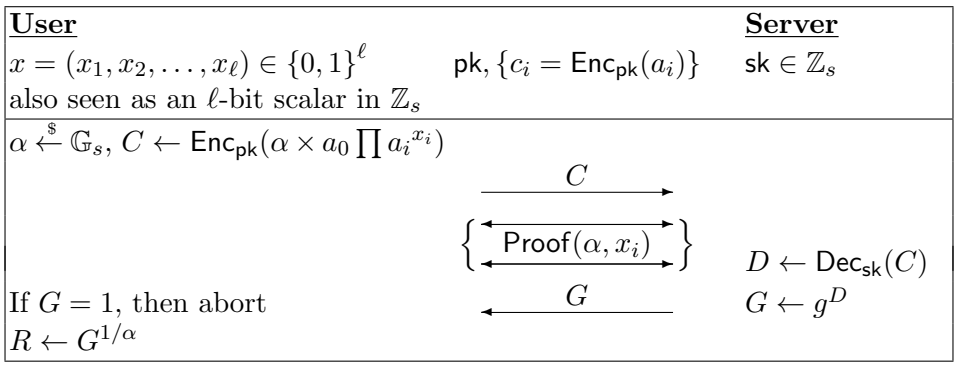


Fig. 4: Secure Oblivious Evaluation of the NR-PRF

The proof will be performed by a sequence of games, as for the previous protocol, 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 $C = \mathsf{Commit}(\mathsf{pw}, \Pi, \Sigma, \mathsf{SSInfo}, K; r)$, by first a fingerprint $H = \mathcal{H}(\Pi, \Sigma, \mathsf{SSInfo}, K)$ with a second-preimage-resistant hash function $\mathcal{H}$, and then $C = \mathsf{Enc}(\mathsf{pw}, H; r)$, with an $\mathsf{IND-CCA}$ encryption scheme. This improves the efficiency, as the information $(\Pi, \Sigma, \mathsf{SSInfo}, K)$ may be long. More precisely, we use the Cramer-Shoup encryption scheme, denoted $(\mathsf{CS.Enc}, \mathsf{CS.Dec})$, for the extractable commitment. We will use the simulators presented in Figure 5, where the c_i's have been replaced by random ciphertexts, which is indistinguishable under the $\mathsf{IND-CPA}$ security level of the ElGamal encryption scheme, denoted $(\mathsf{EG.Enc}, \mathsf{EG.Dec})$. $\mathcal{S}im_U$ 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, $\mathcal{S}im_S$ 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)

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

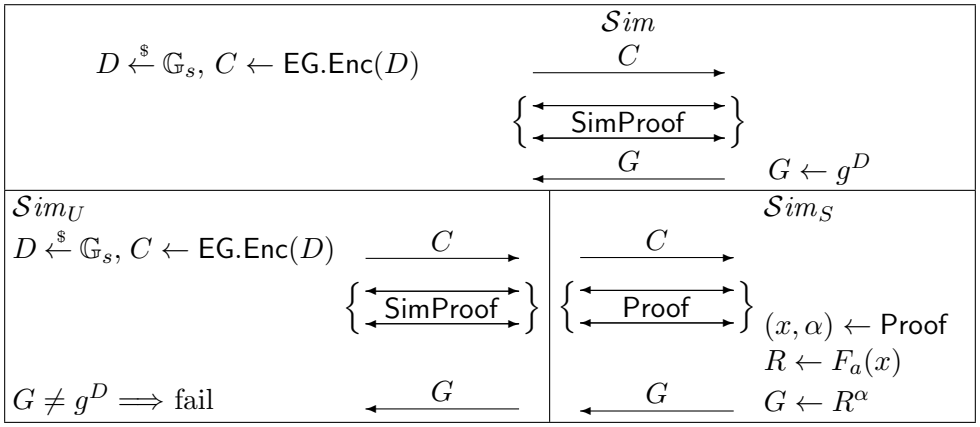
and one has to prove its knowledge of $(x_i)_i \in \{0,1\}^\ell$, $\alpha \in \mathbb{G}_s$, and $r \in \mathbb{Z}_s$ that satisfy this relation. To get a straightline extraction, one can use a Cramer-Shoup encryption of α and u^x, for a generator $u \in \mathbb{G}_s$ (assuming the use of a password 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. As in the proof for the PPSS Protocol I, we emulate the initialization phase, which is honestly performed: we choose one random pw, n random keys $(\mathsf{sk}_k)_k$ for the ElGamal encryption scheme and the PRF keys $a_k = (a_{k,0}, \ldots a_{k,\ell})_k$ that represent the servers' secret information. We generate their corresponding public part $\Pi = (\mathsf{pk}_k, (c_i = \mathsf{Enc}_{\mathsf{pk}_k}(a_{k,i}))_k)$, and one random value corresponding to the secret K, together with a secret sharing $(s_1, \ldots, s_n, \mathsf{SSInfo})$ of $R = K\|r$, for a random r. These shares are masked using the values $(\pi_k)_k$, obtained as PRF evaluations of 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 $R = K\|r$ is used to compute the commitment $C = \mathsf{CS.Enc}(\mathsf{pw}, H; r)$, where $H = \mathcal{H}(\Pi, \Sigma, \mathsf{SSInfo}, K)$.

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,



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

Fig. 5: Simulators for the OPRF based on CDH

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 :

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

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 5. This perfectly simulates honest transcripts (C, 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:

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

Game $\mathbf{G}_2$: We consider an adversary that fakes the public information PInfo in Send-queries to the user: if the majority of at least t_r tuples $\mathsf{PInfo}' = (\Pi', \Sigma', \mathsf{SSInfo}', C')$ 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 $\mathsf{PInfo} = (\Pi, \Sigma, \mathsf{SSInfo}, C)$ 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:

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

This event PWinC can be evaluated by decrypting the commitment $C = \mathsf{CS.Enc}(\mathsf{pw}, H; r)$ 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 PInfo' (but correct C) sent to the user: if the majority of at least t_r tuples $\mathsf{PInfo}' = (\Pi', \Sigma', \mathsf{SSInfo}', C)$ contains public information different from the expected ones (the PInfo generated during the honest initialization phase), we make the user play as usual, but eventually set $K \leftarrow \perp$.

The RGTSSS protocol generates either correct fingerprints when computed from shares obtained with the help of honest servers or random independent fingerprints when the servers cheat and the shares obtained by the client are not the expected ones. The reconstruction using RGTSSS guarantees the recovery of the secret in the honest case. When the value C is unchanged, the value $H' = \mathcal{H}(\Pi', \Sigma', \mathsf{SSInfo}', K')$ must be the same as the initial commitment input $H = \mathcal{H}(\Pi, \Sigma, \mathsf{SSInfo}, K)$ 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 $(\Pi', \Sigma', \mathsf{SSInfo}', K')$ of the initial $H = \mathcal{H}(\Pi, \Sigma, \mathsf{SSInfo}, K)$. As a consequence, this simulation is perfectly indistinguishable from the previous one unless one finds a second pre-image to $H = \mathcal{H}(\Pi, \Sigma, \mathsf{SSInfo}, K)$ (where t is essentially the running time of $\mathcal{A}$):

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

Game $\mathbf{G}_4$: We are still dealing with Send-queries to the user, but we consider the case of the event $\neg\mathsf{PWinC}$. We now use $\mathcal{S}im_U$, 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 RGTSSS guaranteed the recovery of the secret in exactly the same cases as here:

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

We remark that during this game, the 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 Send-queries to the servers, and replace them by the simulator $\mathcal{S}im_S$ to provide answers, getting x and the blinding factor α 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:

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

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:

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

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

Game $\mathbf{G}_7$: Instead of choosing the π_k at random, one generates $\Sigma = (\sigma_k)_k$ and SSInfo at random. This leads to random and independent shares $(s_1, \ldots, s_n)$. We know that until the adversary does not get more than $t_\ell = n/4$ of these shares, it cannot detect whether they are independent or redundant (as should be a secret sharing): let us define the event PWinF', a little bit different from the previous proof, to be true if more than $n/4 - q_c$ queries have been asked to the servers on 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 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: $\mathsf{Succ}_{\mathbf{G}_6} \leq \mathsf{Succ}_{\mathbf{G}_7} + \Pr[\mathsf{PWinF}']/2$. Thanks to the extractability of x from the proof, we are able to check whether pw has been used, and how many times in order to set the event 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 PWinC:

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

.

Game $\mathbf{G}_9$: The key K_1 does not appear any more in the simulation of the secret sharing, as the values 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:

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

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

- $\Pr[\mathsf{PWinC}]$ 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 pw;
- $\Pr[\mathsf{PWinF}']$ is clearly upper-bounded by $q_s/(n/4 - q_c) \times 1/\#\mathcal{D}$, since $q_s/(n/4 - q_c)$ is the maximal number of passwords for which the adversary asked for $n/4 - q_c$ OPRF evaluations.

In conclusion, we have:

$$\begin{aligned}\mathsf{Adv}(\mathcal{A}) \leq &\left(q_u + \frac{4q_s}{n - 4q_c}\right) \times \frac{1}{\#\mathcal{D}} \\ &+ (n - q_c)\ell \times \mathsf{Adv}^{\mathsf{ddh}}(t) + 2 \times \mathsf{Adv}^{\mathsf{ind-cca}}_{\mathsf{CS}}(t) + \mathsf{Succ}^{\mathsf{2nd}}_{\mathcal{H}}(t).\end{aligned}$$

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

D Experimental Results

We have implemented the decision algorithm to validate the idea of taking the $|\gcd(\mathcal{T}'', \tau_i)| \geq k/2$. We have tested 2^{21} random shares for $k = \{64, 96, 128, 256\}$-bit fingerprints and for $n = \{32, 44, 60, 92\}$ shares to evaluate the distribution of large prime numbers in different settings. In Table 1 we present the probabilities for the best cases, which are: (i) $\gcd(\mathcal{T}'', \tau_i) = 1$ (coprime) when τ_i is correct, meaning that all common factors are canceled out and (ii) $|\gcd(\mathcal{T}'', \tau_i)| = k$ (no factors were canceled out) when τ_i is incorrect. We can remark these probabilities are between 40% and 50%.

Table 1: Probabilities of the best cases (i.e., probability that $\gcd(\mathcal{T}'', \tau_i) = 1$ when τ_i is correct and probability that $|\gcd(\mathcal{T}'', \tau_i)| = k$ when τ_i is random)

n	k							
	64		96		128		256	
	τ_i correct	τ_i random	τ_i correct	τ_i random	τ_i correct	τ_i random	τ_i correct	τ_i random
32	49,24%	45,12%	49,62%	49,68%	49,68%	44,70%	50,05%	44,55%
44	75,75%	43,78%	47,93%	43,62%	47,66%	43,20%	47,10%	43,88%
60	45,51%	42,41%	45,68%	42,24%	42,75%	45,92%	45,92%	41,95%
92	43,05%	40,93%	42,76%	41,15%	43,08%	40,77%	43,34%	40,58%

On the other hand, in Table 2 we present our worst-cases of the decision algorithm. One can see that for $k = 64$ the algorithm fails, with both too large $\gcd(\mathcal{T}'', \tau_i)$ when τ_i is correct and too small $\gcd(\mathcal{T}'', \tau_i)$ when τ_i is random, leading to false positive and false negative decisions. This is due to the too small fingerprints. Indeed, increasing k to 96, the probability of false positive/negative decisions is drastically reduced: the worst cases are far from $k/2$, even for $n = 92$. No bad decisions are taken among the millions of tests.

Table 2: Bit size t_i of $\gcd(\mathcal{T}'', \tau_i)$ for different sizes k of the fingerprint and numbers n of shares

n	k							
	64		96		128		256	
	τ_i correct	τ_i random	τ_i correct	τ_i random	τ_i correct	τ_i random	τ_i correct	τ_i random
32	**33**	**29**	35	65	35	96	34	223
44	**41**	**26**	35	60	40	92	36	221
60	**39**	**27**	38	61	40	94	39	220
92	**44**	**25**	43	53	53	84	43	217

In the Figure 6 we present the distribution of $t_i = |\gcd(\mathcal{T}'', \tau_i)|$ for $k = 96$ and $n = 32$, when τ_i a valid fingerprint. It is possible to see that there is a high probability of t_i be equals to 1 ($\mathcal{T}''$ and τ_i coprimes) and the probability is reducing while the bit size is increasing. Our experiments suggest that, for our setting, the probability of bit-lengths is bounded by $2^{-x/2}$. In our case ($k = 96$ and $n = 32$), the probability of deciding a τ_i as valid while it is not (false positive) looks approximately 2^{-28}.

Fig. 6: Distribution of the bit-length of $\gcd(\mathcal{T}'', \tau_i)$ for correct τ_i, when $k = 96$ and $n = 32$