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. Our contributions are two-fold: First, we propose a generic technique to build a *Robust Gap Threshold Secret Sharing Scheme* (RGTSSS) from some threshold secret sharing schemes. 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, which is provided by an ideal hash function.

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

I. Askoxylakis et al. (Eds.): ESORICS 2016, Part II, LNCS 9879, pp. 61–79, 2016.
DOI: 10.1007/978-3-319-45741-3_4

are not vulnerable to such offline dictionary attacks, like *password authenticated key exchange* (PAKE) [7]. 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 [18,22], 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 [2] 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 [2] 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, and provides consistent but fake public information, it cannot make the user reconstruct and accept 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.

We stress 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 [23]: 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, in an oblivious way, with servers that own the PRF keys, hence the so-called *oblivious pseudorandom functions* (OPRFs).

Our main contribution is the efficient realization of the robustness in only one round of communication with each server, possibly in a concurrent way. We also avoid any complex zero-knowledge proof. This comes from the fact that we do not need to distinguish between correct and incorrect shares at each individual evaluation with a server as in [23]. Compared to the later solution with ZK proofs given in [24], our scheme needs only a single global check at the very end, during the secret reconstruction, which significantly reduces the communication costs.

Actually, we propose a new efficient method to convert some Secret Sharing Schemes into (t_ℓ, t_r, n)-*Robust Gap Threshold Secret Sharing Schemes* (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 $t_\ell - 1$ shares are correct, the protocol *leaks* no information about which shares are correct. Our construction is more general and with similar efficiency than using error-correcting code such as Reed-Solomon [27]. Such a (t_ℓ, t_r, n)-RGTSSS allows constructing a sound and robust PPSS scheme: 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 strictly 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 $t_\ell - 1$ 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 [23] 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 [24] in terms of server interaction efficiency for the PRF evaluation, our technique takes advantage of the RGTSSS to optimize the secret reconstruction. The scheme proposed by [24] 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 of the secret sharing scheme 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 [24]. Secondly, we introduce a new oblivious evaluation of the Naor-Reingold PRF [25], 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 [17] in prime order groups with simple zero-knowledge proofs, whereas for example the scheme in [23] has to work in composite order groups with Paillier encryption [26] 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) [4], 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 [4].

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 [28] and Blakey [9]. 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. Ramp schemes to achieve a robust secret sharing scheme have been extensively studied, we refer the reader to [8,14]. While this is well-known that the Shamir secret sharing scheme can be made robust using Reed-Solomon error correcting codes, our approach is more general with similar efficiency.

The first formal definition of *Password Protected Secret Sharing* was introduced by Bagherzandi *et al.* [2]. 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.* [12] 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.* [10] introduce a (t, n)-PPSS (called TPASS, for Threshold Password-Authenticated Secret Sharing) in the Universal Composability (UC) framework [13] 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.* [29] 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.* [11] present a very lightweight protocol with a similar construction to our work, yet with differences. 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.* [23] 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 [24] 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.

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 [5,6] for key distribution schemes and [3] for password-authenticated key exchange. It adapts the PPSS definition from [2] and the security model from [23]. We define security in terms of a *key derivation mechanism* or indistinguishability of the actual secret from a random one, as in [23], 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

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 is the number of shares that start *leaking* some 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 information in PInfo is 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 is thus 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 gets the entire transcript;
- $\mathsf{Send}(P^i, m)$: This query models an active attack. This sends a message m to the instance P^i. 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. 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 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 (Execute, Send, Test, and 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 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 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 $q_c < t_\ell$ 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 [23], they proposed such a protocol that achieves the optimal t_ℓ-security, for $t_\ell = t_r$, but at the cost of verifiable oblivious pseudorandom functions. Our goal is to build much more efficient protocols, possibly with a larger gap between t_ℓ and t_r.

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 guaranteed by our security model, when a user terminates with a key K', this is the correct key K in almost all the cases, unless the adversary guesses the password. More precisely, when playing with the correct password pw, the user should end up with $K' \in \{K, \perp\}$:

$$\Pr[K' \notin \{\perp, K\}] \leq \frac{1}{\#\mathcal{D}} \times \left(\frac{q_s}{t_\ell - q_c} + q_u\right) + \varepsilon.$$

Robustness. While one cannot avoid Denial-of-Service (DoS) attacks, since the adversary can simply block any communication, an important property, already required by [23], 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 [23], 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 full version [1]. Our general construction follows the one from [23], with first an initialization phase and then a reconstruction phase.

Each server S_i owns a key-pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ that defines a PRF F_i, with public parameters defined by pk_i and a secret key defined by sk_i. For a password $\mathsf{pw} \in \mathcal{D}$, the user asks for an oblivious evaluation of $\pi_i = F_i(\mathsf{pw})$ to n servers, where $\Pi = (\mathsf{pk}_i)_i$ 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, $(\mathsf{pk}_i)_i$ are all the true public keys, and $(\pi_i)_i$ 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 $\pi_i = F_i(\mathsf{pw})$ 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_i, 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 $t_\ell - 1$ shares do not leak any information.

An additional non-malleable commitment scheme [16] 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 [23] 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 can generically apply to most threshold secret sharing schemes, with two algorithms ShareGen and Reconstruct that respectively share a secret into n parts and reconstruct it from t_r shares (while no information leaks from $t_r - 1$ shares, which look independent random elements). One can for example use the classical Shamir's secret sharing scheme [28] 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 full version [1]).

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 $t_\ell - 1$ 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, the number of shares that start *leaking* information).

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 $\mathcal{S}' < 2^{k(n-t_r)}$.

Unfortunately, using the following result from [19], we can only recover the irreducible fraction $\mathcal{T}''/\mathcal{S}''$ of γ, where all the small common factors of $\mathcal{T}'/\mathcal{S}'$ were canceled out, 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 $\mathcal{T}''$ (we assume that the size of the common part is less than half of the size of τ_i). 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 (we assume that less than half of it cancels). 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 Fig. 1, we present experimental results that validate this decision algorithm for 128-bit fingerprints. It clearly shows 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 large 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 Fig. 1a, the highest bit size of t_i is 35 (much less than 64). On the other hand, in Fig. 1b the least value is 96 (much more than 64). A more fine analysis can be found in the full version [1].

Information Leakage. On the opposite, we would like to evaluate the information leaked by $\mathcal{S}$ when there are at most $t_\ell - 1$ valid values. More precisely, given $\mathcal{S}$, is it possible to distinguish $t_\ell - 1$ valid values for the shares from $t_\ell - 1$ random values? We focus on a t_r-threshold secret sharing scheme, for a k-bit secret and k-bit shares. Then, the entropy of the tuple $(\sigma_1, \ldots, \sigma_n)$ is $k(t_r - 1)$. Since $\mathcal{S}$ reveals the product of the k-bit fingerprints 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 $k(t_\ell - 1)$, no one can distinguish $t_\ell - 1$ random values from $t_\ell - 1$ correct values for the shares:

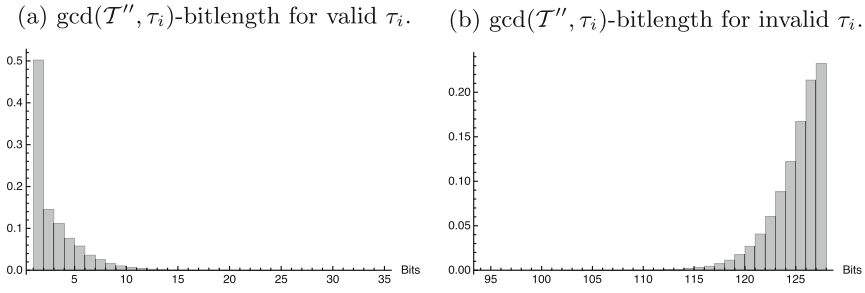

Fig. 1. Length in bits of $\gcd(\mathcal{T}'', \tau_i)$ for a fingerprint of size 128-bits and 32 shares

we thus need $k(3t_r - 2n - 1) - 2 \geq k(t_\ell - 1)$. When $k > 2$, this essentially means $t_\ell \leq 3t_r - 2n$: by choosing $t_\ell = 3t_r - 2n$, we are safe. For example, one can take $t_r = \lceil 3n/4 \rceil$ and $t_\ell = \lfloor n/4 \rfloor$. And the same argument, with $2k$-bit secret and shares but still k-bit fingerprints, leads to $t_r = \lceil 2n/3 \rceil$ and $t_\ell = \lfloor n/3 \rfloor$, which makes sense for a 256-bit secret key and 128-bit fingerprints.

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, as in [23], which is at the cost of complex zero-knowledge proofs. 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 [23], using similar computational assumptions (see the full version [1]):

- the first OPRF relies on the CDH evaluation, similar to the protocol 2HashDH, but without NIZKs. The PPSS construction is then quite similar to [24].
- the second OPRF is an oblivious evaluation of the Naor-Reingold PRF [25]. Then, in the PPSS, the gain of the zero-knowledge proofs by the server is quite significant.

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 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_i owns a key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ that defines a PRF F_i, with public parameters defined by pk_i and a secret key defined by sk_i, that admits an OPRF protocol to allow a user with input m to evaluate $F_i(m)$ without leaking any information on m to the server.

We additionally use a (t_ℓ, t_r, n)-robust gap threshold secret sharing scheme and a non-malleable commitment scheme (see the full version [1]). Since we already are in the random-oracle model for the PRF, we can implement the commitment scheme with a simple second-preimage-resistant hash function H_{Com}, which allows a better efficiency. The user U first chooses a secret password pw:

1. the user interacts with n servers to obliviously evaluate $\pi_i = F_i(\mathsf{pw})$, and $\Pi = (\mathsf{pk}_i)_i$ 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 reconstruct and r some random coins for the commitment. 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 recover R;
3. then, the user builds $\sigma_i = \pi_i \oplus s_i$, for $i = 1, \ldots, n$, and sets $\Sigma = (\sigma_i)_i$;

4. the user generates $\mathsf{Com} = H_{\mathsf{Com}}(\mathsf{pw}, \Pi, \Sigma, \mathsf{SSInfo}, K; r)$. We denote by $\mathsf{PInfo} = (\Pi, \Sigma, \mathsf{SSInfo}, \mathsf{Com})$ the public information that the user will need later to recover his 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 $(\pi_i)_i$ 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.

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

1. the user first limits the oblivious evaluations of $\pi_i = F_i(\mathsf{pw})$ to the servers that sent the same majority tuple $\mathsf{PInfo} = (\Pi, \Sigma, \mathsf{SSInfo}, \mathsf{Com})$. If the number of such servers is less than t_r, one aborts with $K \leftarrow \perp$;
2. for all these π_i (or similarly, all the i he kept), the user computes $s_i = \sigma_i \oplus \pi_i$, using σ_i from Σ (from PInfo);
3. using these $\{s_i\}$ 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 $\mathsf{Com} = H_{\mathsf{Com}}(\mathsf{pw}, \Pi, \Sigma, \mathsf{SSInfo}, K; r)$;
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 $\{0,1\}^{\ell_2}$. The commitment scheme uses a simple hash function $H_{\mathsf{Com}} = H_3$ onto $\{0,1\}^{\ell_3}$.

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 full version [1], we prove this is indeed a PRF, as already shown in [23].

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 Fig. 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 full version [1]) as shown in the last step of the proof.

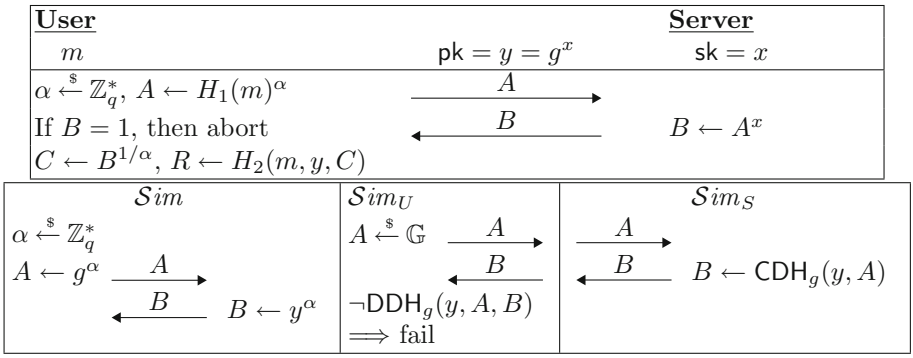

Fig. 2. Secure oblivious evaluation of the PRF based on OMGDH

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*

$$\mathsf{Adv}(\mathcal{A}) \leq \left(q_u + \frac{4q_s}{n - 4q_c}\right) \times \frac{1}{\#\mathcal{D}} + \varepsilon.$$

where $\varepsilon = n \times \mathsf{Succ}^{\mathsf{omgdh}}(q_1, q_s, t, n \cdot q_u + q_2) + (q_3^2 + 2) \cdot 2^{-\ell_3}/4$.

Security Proof. The complete and detailed proof of the Theorem is given in the full version [1]. 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 $\mathsf{Execute}$-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 Com' 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 Com' 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 Com, 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 Com in the initialization phase by a random Com.

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 guesses in 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 [25] 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

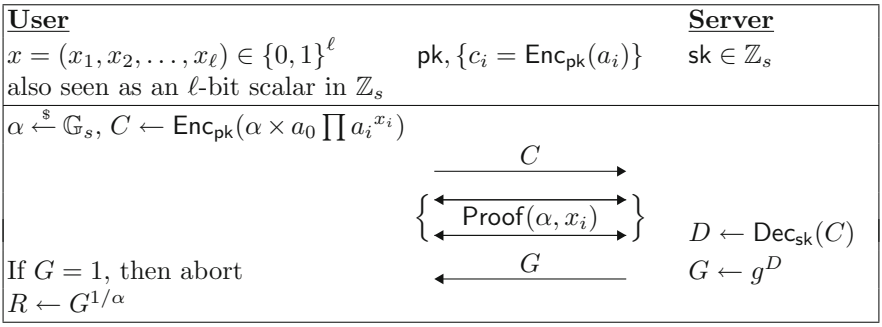

Fig. 3. Secure oblivious evaluation of the NR-PRF

$\mathbb{G}_s$ of order s (this is the group of elements in $\mathbb{Z}_q^*$ with Jacobi symbol equals to $+1$). In both groups, the DDH assumption can be made.

The PRF key is a tuple $a = (a_0, a_1, \ldots, a_\ell) \xleftarrow{\$} (\mathbb{G}_s \backslash \{1\})^{\ell+1}$, 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 [25] on ℓ-bit inputs. It also admits a simple oblivious evaluation (just the messages C and G from Fig. 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 Fig. 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 of our protocol II (including the DDH-based OPRF) can be found in the full version [1].

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 [24]. Of course, we did not provide any security result in the UC framework [13], but our ultimate goal was the same as [23]: an efficient robust password-protected secret sharing scheme, in a BPR-like security model [3]. 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 [20], 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.* [21], 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 [23]. 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 [26] nor Cramer-Shoup signature [15] 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

1. Abdalla, M., Cornejo, M., Nitulescu, A., Pointcheval, D.: Robust password-protected secret sharing. Cryptology ePrint Archive, Report 2016/123 (2016). http://eprint.iacr.org/2016/123
2. Bagherzandi, A., Jarecki, S., Saxena, N., Lu, Y.: Password-protected secret sharing. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM CCS 2011, pp. 433–444. ACM Press, October 2011
3. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
4. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Ashby, V. (ed.) ACM CCS 1993, pp. 62–73. ACM Press, November 1993
5. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994)
6. Bellare, M., Rogaway, P.: Provably secure session key distribution: the three party case. In: 27th ACM STOC, pp. 57–66. ACM Press, May/June 1995
7. Bellovin, S.M., Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy, pp. 72–84. IEEE Computer Society Press, May 1992
8. Bishop, A., Pastro, V., Rajaraman, R., Wichs, D.: Essentially optimal robust secret sharing with maximal corruptions. Cryptology ePrint Archive, Report 2015/1032 (2015). http://eprint.iacr.org/2015/1032

9. Blakley, G.R.: Safeguarding cryptographic keys. In: Proceedings of AFIPS 1979 National Computer Conference, vol. 48, pp. 313–317 (1979)
10. Camenisch, J., Lehmann, A., Lysyanskaya, A., Neven, G.: Memento: how to reconstruct your secrets from a single password in a hostile environment. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 256–275. Springer, Heidelberg (2014)
11. Camenisch, J., Lehmann, A., Neven, G.: Optimal distributed password verification. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015, pp. 182–194. ACM Press, October 2015
12. Camenisch, J., Lysyanskaya, A., Neven, G.: Practical yet universally composable two-server password-authenticated secret sharing. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012, pp. 525–536. ACM Press, October 2012
13. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press, October 2001
14. Cheraghchi, M.: Nearly optimal robust secret sharing. Cryptology ePrint Archive, Report 2015/951 (2015). http://eprint.iacr.org/2015/951
15. Cramer, R., Shoup, V.: Signature schemes based on the strong RSA assumption. In: ACM CCS 1999, pp. 46–51. ACM Press, November 1999
16. Di Crescenzo, G., Ishai, Y., Ostrovsky, R.: Non-interactive and non-malleable commitment. In: 30th ACM STOC, pp. 141–150. ACM Press, May 1998
17. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory **31**, 469–472 (1985)
18. Ford, W., Kaliski Jr., B.S.: 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, pp. 176–180. IEEE Computer Society, Washington, DC (2000)
19. Fouque, P.-A., Stern, J., Wackers, J.-G.: CryptoComputing with rationals. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 136–146. Springer, Heidelberg (2003)
20. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Heidelberg (2005)
21. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
22. Jablon, D.P.: Password authentication using multiple servers. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 344–360. Springer, Heidelberg (2001)
23. Jarecki, S., Kiayias, A., Krawczyk, H.: Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 233–253. Springer, Heidelberg (2014)
24. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: Highly-Efficient and Composable Password-Protected Secret Sharing. Cryptology ePrint Archive, Report 2016/144 (2016). http://eprint.iacr.org/
25. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. In: 38th FOCS, pp. 458–467. IEEE Computer Society Press, October 1997
26. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)

27. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. J. Soc. Ind. Appl. Math. **8**(2), 300–304 (1960)
28. Shamir, A.: How to share a secret. Commun. Assoc. Comput. Mach. **22**(11), 612–613 (1979)
29. Yi, X., Hao, F., Chen, L., Liu, J.K.: Practical threshold password-authenticated secret sharing protocol. In: Pernul, G., Ryan, P.Y.A., Weippl, E. (eds.) ESORICS. LNCS, vol. 9326, pp. 347–365. Springer, Heidelberg (2015). doi:10.1007/978-3-319-24174-6_18