Highly-Efficient and Composable Password-Protected Secret Sharing (Or: How to Protect Your Bitcoin Wallet Online)

Stanislaw Jarecki*, Aggelos Kiayias†, Hugo Krawczyk‡ and Jiayu Xu*
*U. California Irvine, USA, Email: {stasio@ics,jiayu@}.uci.edu
†University of Athens, Greece, Email: aggelos@di.uoa.gr
‡IBM Research, NY, USA, Email:hugo@ee.technion.ac.il

***Abstract*—PPSS is a central primitive introduced by Bagherzandi et al. [2] which allows a user to store a secret among n servers such that the user can later reconstruct the secret with the sole possession of a single password by contacting $t+1$ ($t<n$) servers. At the same time, an attacker breaking into t of these servers - and controlling all communication channels - learns nothing about the secret (or the password). Thus, PPSS schemes are ideal for on-line storing of valuable secrets when retrieval solely relies on a memorizable password.**

We show the most efficient Password-Protected Secret Sharing (PPSS) to date (and its implied Threshold-PAKE scheme), which is optimal in round communication as in Jarecki et al. [11] but which improves computation and communication complexity over that scheme requiring a *single* per-server exponentiation for the client and a single exponentiation for the server. As with the schemes from [11] and Camenisch et al. [4] we do not require secure channels or PKI other than in the initialization stage.

We prove the security of our PPSS scheme in the Universally Composable (UC) model. For this we present a UC definition of PPSS that relaxes the UC formalism of [4] in a way that enables more efficient PPSS schemes (by dispensing with the need to extract the user's password in the simulation) and present a UC-based definition of Oblivious PRF (OPRF) that is more general than the (Verifiable) OPRF definition from [11] and is also crucial for enabling our performance optimization.

1. Introduction

Two well-known facts about passwords: They are the dominant form of authentication in the online world and they are routinely leaked, particularly via offline dictionary attacks performed on password-related information stolen from Internet servers (e.g., [16]). Yet another fact is that passwords are protecting increasingly valuable information from financial to personal to high-value bitcoin wallets. A natural approach for strengthening such protection would be to distribute the information over a set of servers using a threshold secret sharing system, but then the question is how a user would authenticate to these servers. Reusing the same password in all servers only increases the password (and the stored information) vulnerability while memorizing a different strong password for each server is unrealistic.

PPSS. To address this problem, Bagherzandi et al. [2] introduced the notion of *Password-Protected Secret Sharing (PPSS)* in which a user distributes its information (a secret) among n servers such that the compromise of any t of them leaks no information on the secret (and password) while contacting $t+1$ servers with the right password reconstructs the secret. A PPSS is implied by any Threshold Password Authenticated Key Exchange (T-PAKE) [15] (and vice versa, as shown in [11]). Before [2] the most efficient T-PAKE protocol by MacKenzie et al. [15] required several rounds of communication and transmission of $O(n^2)$ group elements, while [2] showed a PPSS protocol where a user exchanges 3 constant-sized messages (4 in the typical case that a user initiates the communication) with each server and performs 8 (multi) exponentiations per server. In addition, the PPSS solution of [2] assumes PKI-authenticated channels between a user and each server during both initialization and reconstruction. Later, Camenisch et al. [4] presented a PPSS protocol (called T-PASS, for *Threshold Password-Authenticated Secret Sharing*, in their work) that does not require PKI authentication during the reconstruction phase but costs 14 client exponentiations per server and 7 server-side exponentiations for each server; it also requires 10 messages between a user and each server in secret reconstruction. Jarecki et al. [11] provided the most efficient PPSS solution to date, also without PKI authentication in the reconstruction, with a reconstruction procedure that takes a single round (two messages) between a user and each server and only costs two multi-exponentiations per server for the client and roughly

two multi-exponentiations per each participating server.

Jarecki et al. [11] used an indistinguishability-based definition of PPSS security adopted from [2] to the password-only setting (i.e. no PKI assumption), while [4] provided a Universally Composable (UC) definition of PPSS (a.k.a. T-PASS) functionality. The essence of these definitions is that the only attack available to the adversary is the inevitable one, namely, an online dictionary attack where validating a single password guess requires either $t+1$ instances of the servers or the user to interact with the man-in-the-middle adversary.

1.1. Our contributions.

Highly efficient performance. Our starting point is the PPSS scheme of Jarecki et al. [11], the most efficient PPSS solution to date, which we modify and improve in several ways. First, we reduce the computational cost of the protocol to *single exponentiation per server* for the client, and to a *a single exponentiation* for each server, which together with an additional exponentiation which the client reuse for all servers brings the total client cost to $t + 2$ exponentiations.[1] At the same time, we preserve the already optimal round complexity of the scheme but further lower communication by eliminating three group elements sent from server to client in the scheme of [11] (the communication from client to server is already minimal in [11] requiring the transmission of a single group element). These savings are obtained by eliminating the per-server public parameters in the solution from [11] and by forgoing the zero-knowledge proofs between each server and client required by their scheme. As with previous efficient PPSS solutions, our protocol is analyzed in the random oracle model.

Non-reliance on PKI and secure channels. A main accomplishment of the PPSS schemes of [4] and [11] is the non-reliance on secure channels or public-key infrastructure during the reconstruction phase. This is a major benefit since PPSS assumes a user that only knows its user id and password, and does not carry auxiliary devices with authenticated information. Moreover, the increasing vulnerabilities of certificate-based authentication translates into weaknesses in schemes that rely on such authentication. Fortunately, we achieve our optimal performance while still dispensing with the need of secure channels or PKI (except for a trusted initialization protocol needed in all PPSS schemes).

1. We remark that the client cost, while smaller by roughly a factor of two than the client cost in [11], is *not* minimal: Our client uses $O(t)$ exponentiations, while the client in the otherwise much more communication and computation heavy T-PAKE scheme of MacKenzie et al. [15] uses $O(1)$ exponentiations. Moreover, our client costs can be further reduced, as we show in a forthcoming follow-up work: See also footnote 3 in Section 5.

UC security. In addition to the significant performance improvement relative to [11], we also improve on their security analysis. That is, we provide two proofs of security for our PPSS protocol. First, we show that our solution satisfies the indistinguishability-based definition of PPSS security used in [2], [11]. However, we also provide a proof that the same scheme satisfies a UC formalization of PPSS. This formalization is in itself a significant contribution of our work. Indeed, while a UC definition of PPSS appeared in the work of Camenisch et al. [4], our formulation significantly relaxes this functionality in a way that enables the proof of our much more efficient scheme. To obtain this relaxed UC functionality (and the UC proof of our scheme) we utilize a *ticketing mechanism*, used e.g. by [11] in their formalism of V-OPRF (see below), which allows us to dispense with the need to *extract* the user's input (in this case, the user's password) during an execution of a UC PPSS scheme, something that requires heavier cryptographic mechanisms and results in higher performance costs, as with the scheme of [4]. The ticketing mechanism ensures that in order to test a single password guess, the attacker must impersonate the user to $t+1$ servers or impersonate $t+1$ servers to the client, which is optimal in terms of security against guessing attacks and constitutes the very essence of the PPSS security notion.

OPRF. A central ingredient in the scheme of [11] that we preserve in our solution is the notion of a *Oblivious PRF* [9], [13]. Roughly speaking, an OPRF is a protocol between two parties, one holding a key k for a PRF f and one holding an input x, where no party learns anything except for the input holder that learns $f_k(x)$. This notion has been shown to be useful in many different contexts [7], [9], [13], [14] but defining it so that it can be implemented inexpensively is non-trivial. For example, using an MPC-type definition would require a costly implementation to achieve concurrent security (as needed here) and would require secure channels (undesired here). To resolve this problem, [11] introduced a UC-based OPRF definition that allowed them to build a PPSS scheme with concurrent security and without secure channels for reconstruction. Moreover, their use of a ticketing mechanism in their OPRF definition (which is also similar in spirit to e.g. blind signature definitions, cf. [1]) allowed them to obtain very efficient instantiations by avoiding extractable proofs of knowledge or similar costly mechanisms (this is the ticketing mechanism that, as said before, we have borrowed for our own UC formulation of PPSS). At the same time, in order for the OPRF to fit their PPSS scheme, [11] strengthen the security notion of OPRF adding a *verifiability* property that allows users

to detect dishonest behaviors of the PPSS servers during reconstruction. Unfortunately, this additional property introduces the need to use zero-knowledge proofs in the implementation of their OPRF, costing one multi-exponentiation for the client and server in each client-server interaction and leads to an increase in the amount of communication as well.

Here we resolve this problem by relaxing the Verifiable OPRF (V-OPRF) notion of [11] into a plain UC OPRF functionality that does not provide verifiability and therefore enables an optimal implementation without zero-knowledge proofs at all (it also has the potential of better fitting other OPRF applications). By showing that this weaker notion of OPRF suffices for realizing our PPSS, we obtain significant performance benefits. On the other hand, forgoing the verifiability property weakens the *robustness* of our PPSS solution, namely, the ability to discard incorrect computations during reconstruction. Yet, we can enjoy the best of the two worlds: We can run the highly efficient protocol without zero-knowledge proofs which we prove secure here, and only resort to the ZK proofs in case the reconstruction fails. Thus, in the normal case of a non-adversarial run the cost of zero-knowledge is saved. Finally, we remark that in real-world applications we expect n to be a small number, in which case checking different subsets of $t+1$ servers until finding a non-corrupted subset is a practical approach that completely dispenses with zero-knowledge proofs.

Applications. As mentioned above, PPSS has obvious applications for protecting data whose security depends on a user-memorable password. In particular, this data can take the form of credentials and keys that when reconstructed enable strong authentication by the user. In particular, this allows the implementation of T-PAKE schemes that bootstrap password authentication while protecting the password against server compromise for as long as no more than a threshold of servers is compromised. Yet another application of PPSS is for multi-factor authentication protocols (e.g., [17]) where a user's device acts as a "server" in a T-PAKE scheme. Improving the efficiency of PPSS (and the resultant T-PAKE) is important for all the above cases and particularly so for enabling the use of a (weak) device as a PPSS server. In a follow-up work we will formalize UC functionality for T-PAKE as the threshold extension of UC PAKE model of [6], and we will show, extending the result from [11], that the combination of UC-PPSS and UC-KE (key exchange) results in a UC-secure T-PAKE scheme.

Recent Related Work. We know of two further recent examples of using efficient ROM-based implementation of an OPRF protocol very similar to either the V-OPRF used in [11] or to the OPRF we use here, in order to implement some variant of a T-PAKE functionality: A Pythia Service of [8] and the Distributed Password Verification scheme of [5]. The first scheme uses a close variant of the V-OPRF used in [11] for threshold verification of password authentication, but these OPRF's, and hence the resulting T-PAKE, are not computationally minimal like because they use zero-knowledge proofs (as in [11]) and bilinear maps. Moreover, their proposal assumes the PKI setting. The second scheme uses essentially the same low-cost OPRF scheme as the one we analyze here, but it also assumes the PKI setting. Indeed, the T-PAKE schemes of [5], [8] are customized for an application where the role of the client is played by a *gateway server*, rather than the end-user, who connects to a threshold of trustee servers to verify the user's password. In such application the user is assumed to send its password to the gateway server over a PKI-authenticated link, and the gateway server can be assumed to have authenticated connection to the trustee servers. By contrast, the reconstruction phase in our PPSS (and the resulting T-PAKE), while perfectly applicable to such setting as well, can also be executed by the end user without any further trust assumptions.[2]

2. The OPRF Functionality $\mathcal{F}_{\text{OPRF}}$

Notation. We use the following notational conventions:

- If x is a string, then $x[L]$ denotes its left half.
- If n is an integer, then $[n]$ denotes set $\{1,...,n\}$.
- $\mathbf{0}$ is the vector of n 0's, where n is a consistent defined elsewhere; 0^l is the l-length string of all zeroes (we simplify it to 0 if there is no possibility of confusion).
- If D is a set, then $|D|$ denotes its cardinality.
- ":=" denotes the computation of a deterministic function, while "$\leftarrow$" denotes the computation of a randomized algorithm.
- If D is a set, then $x \leftarrow_{\text{R}} D$ denotes "picking x uniformly at random from D";
- If $\mathbf{G}$ is a game and E is an event, then $\Pr[E||\mathbf{G}]$ denotes the probability that E occurs in $\mathbf{G}$.

The Functionality. Please refer to Section 1 for an introduction of the notion of OPRF and its applicability to PPSS schemes. Here we introduce our UC functionality, $\mathcal{F}_{\text{OPRF}}$, presented in Figure 1. It is derived from the $\mathcal{F}_{\text{VOPRF}}$ functionality of [11] by stripping off the "verifiability" properties of the latter. Specifically,

2. Apart of this fundamental difference in trust assumptions, the T-PAKE schemes of [8] and [5] have additional functional differences from our PPSS scheme. In particular, the T-PAKE scheme of [5] is proactive, unlike ours, but uses additive secret-sharing, i.e. it only works for $t = n-1$ while our scheme works for any threshold $t<n$.

$\mathcal{F}_{\text{VOPRF}}$ allows a user to check consistency between different runs of the OPRF; namely, that each time that the function is run with the same sender on the same input, the same answer is received (otherwise the user rejects). This requires senders to have public keys and requires the OPRF implementation to involve zero-knowledge proofs. By omitting the verifiability condition we simplify the OPRF definition, implementation and applicability.

The functionality $\mathcal{F}_{\text{OPRF}}$ involves users, senders and an (ideal-world) adversary denoted $U, S, \mathcal{A}^*$, respectively. We denote by l the security parameter which determines the output size of the PRF. OPRF evaluation is triggered by an (EVAL, sid, S, x) command from user U requesting the computation of the PRF of server S on input x. SNDRCOMPLETE and RCVCOMPLETE denote the completion of S's and U's computation, respectively. An (EVAL, sid, S, x) operation from a user U with sender S is completed by a (RCVCOMPLETE, sid, U, S, S^*) where S^* is the identity of a server that is specified by $\mathcal{A}^*$ and may or may not be equal to the sender S in the EVAL command. In the case $S = S^*$ we have a computation with the intended sender S while the second case, $S \neq S^*$, corresponds to the attacker channeling the request to a different sender, possibly a corrupted one. In fact, we allow the adversary to specify a value S^* which may not even be a server identity (in such case S^* will be interpreted as a pointer to a function table). Thus, there is no guarantee of correctness of the evaluation request and, moreover, two requests with same S and x can be answered differently. This is where our OPRF formalism $\mathcal{F}_{\text{OPRF}}$ differs fundamentally from the definition of $\mathcal{F}_{\text{VOPRF}}$ in [11] which ensures correct and user-verifiable OPRF computation. This relaxation simplifies the OPRF functionality by dispensing with two essential elements in $\mathcal{F}_{\text{VOPRF}}$, namely, the "public parameters" π associated with each server (and used by the user to check correct evaluation) and the requirement that even corrupted senders must commit to computing an arbitrary but deterministic function (represented in $\mathcal{F}_{\text{VOPRF}}$ by the circuit M).

While we allow $\mathcal{A}^*$ to route a user's request to the wrong sender we do make sure that $\mathcal{A}^*$ cannot forge computations by honest senders. As in [11] this is enforced via a ticketing mechanism that ensures that for any honest server S, the number of user-completed OPRF evaluations (i.e., RCVCOMPLETE activations) with S is no more than the number of SNDRCOMPLETE activations of S. Specifically, each sender S is associated with a ticket value $\mathsf{tx}(S)$. Each time that a sender S completes his interaction with a user, $\mathsf{tx}(S)$ increases by 1; each time a user, either honest or corrupt, completes an interaction that is associated to S, $\mathsf{tx}(S)$ decreases by 1 (provided that it was not zero). This ticketing approach dispenses with the need to extract users' inputs when building a simulator for proving the security of a given realization of the functionality. This simplification (which is shared with [11]), together with the relaxation of the verifiability property as discussed above, allows for the very simple and efficient OPRF realization presented in the next section.

The functionality $\mathcal{F}_{\text{OPRF}}$ maintains a table T used to record the results of the PRF evaluation by different senders on user-requested inputs. Specifically, entry $T(S, x)$ is defined as the sender's S PRF evaluated on input x. The table's entries are initially undefined and are filled with random values by $\mathcal{F}_{\text{OPRF}}$ upon RCVCOMPLETE activations. We note that $T(S, x)$ is chosen at random even in case that server S is corrupted or it corresponds to a function table of the adversary. As a result, any realization of $\mathcal{F}_{\text{OPRF}}$ needs to ensure that evaluations with corrupted senders result in (pseudo) random outputs and even respect the ticketing mechanism. Note that the values $T(S, x)$ are communicated to the requesting user but the inputs x remain fully hidden as they are never communicated to any party, including $\mathcal{A}^*$. Finally, note that $\mathcal{F}_{\text{OPRF}}$ provides the adversary with direct access to all function tables by allowing the issuance of EVAL requests directly to $\mathcal{F}_{\text{OPRF}}$.

3. Realization of $\mathcal{F}_{\text{OPRF}}$

The 2HashDH scheme. In Figure 2, we present an efficient realization of $\mathcal{F}_{\text{OPRF}}$ in the random oracle model, based on the 2HashDH-NIZK construction of $\mathcal{F}_{\text{VOPRF}}$ in [11] from which we eliminate the zero-knowledge proofs and the corresponding "public keys" of servers.

This construction relies on a cyclic group of prime order m. Let g be a generator of the group. The private key k is chosen at random from $\mathbb{Z}_m$. Each user U maintains a table T_U which consists of tuples of the form (S, x, r, f). Let $\mathcal{Z}$ be the environment. The construction uses two hash functions, H_1 and H_2.

The PRF is defined using blinded exponentiation, i.e., as $f_k(x) = H_2(x, H_1(x)^k)$. For each value x the user U wants to evaluate, U picks a random element r in $\mathbb{Z}_m$, which remains the same among different senders. When U wants to compute $f_k(x)$ where k is the private key of a specific sender S, he sends $a = H_1(x)^r$ to S; S sends back $b = a^k = H_1(x)^{rk}$ to U, and U outputs $f = H_2(x, b^{1/r}) = H_2(x, H_1(x)^k)$.

We describe the protocol in detail in Fig. 2. The protocol is described to in the $\mathcal{F}_{\text{AUTH}}$-hybrid world something that facilitates authenticated communication between participants (see [3] for more details on this

For each sender S, a table $T(S,\cdot)$ is initialized to empty.

- Upon receiving (EVAL, sid, S, x) from user U (resp. $\mathcal{A}^*$), record $\langle U, S, x\rangle$ (resp. $\langle \mathcal{A}^*, S, x\rangle$) and send (EVAL, sid, U, S) (resp. $(\text{EVAL}, sid, \mathcal{A}^*, S)$) to $\mathcal{A}^*$.
- Upon receiving $(\text{SNDRCOMPLETE}, sid, S)$ from $\mathcal{A}^*$ increment $\mathsf{tx}(S)$ (or set to 1 if previously undefined) and send $(\text{SNDRCOMPLETE}, sid)$ to S or to $\mathcal{A}^*$ if S is not a sender's identity.
- Upon receiving $(\text{RCVCOMPLETE}, sid, P, S, S^*)$ from $\mathcal{A}^*$, recover record $\langle P, S, x\rangle$; abort if such triple does not exist or S is honest and $S^* \neq S$ or $\mathsf{tx}(S^*)$ is 0 or undefined. Otherwise decrement $\mathsf{tx}(S^*)$ and:
  - If $T(S^*, x)$ is defined, then send $(\text{EVAL}, sid, T(S^*, x))$ to P.
  - Otherwise pick ρ at random from $\{0,1\}^l$, set $T(S^*, x) := \rho$ and send (EVAL, sid, ρ) to P.

Figure 1. Functionality $\mathcal{F}_{\text{OPRF}}$.

- Upon receiving (EVAL, sid, S, x) from $\mathcal{Z}$, U scans its table T_U and does the following:
  - If there exists r and f such that $(S, x, r, f) \in T_U$, then U outputs (EVAL, f) to $\mathcal{Z}$.
  - Otherwise if there exists S', r and f such that $S' \neq S$ and $(S', x, r, f) \in T_U$, then U sends $(\text{SEND}, (sid, n), U, S, a := H_1(x)^r)$ where n is a random nonce. to $\mathcal{F}_{\text{AUTH}}$.
  - Otherwise U picks $r \leftarrow_{\text{R}} \mathbb{Z}_m$ and sends $a := H_1(x)^r$ to S; adds $(S, x, r, *)$ to T_U.
- Upon receiving $(\text{SENT}, (sid, n), U, S, a)$ from $\mathcal{F}_{\text{AUTH}}$, S sends $(\text{SEND}, (sid, n+1), S, U, b := a^k)$ to $\mathcal{F}_{\text{AUTH}}$ where k is the key of S and outputs $(\text{SNDRCOMPLETE}, sid)$ to $\mathcal{Z}$.
- Upon receiving $(\text{SENT}, (sid, n+1), S, U, b)$ from $\mathcal{F}_{\text{AUTH}}$, U finds $(S, x, r, *)$ in T_U and changes it to (S, x, r, f) and outputs (EVAL, sid, $f := H_2(x, b^{1/r})$) to $\mathcal{Z}$.

Figure 2. Protocol 2HashDH in the $\mathcal{F}_{\text{AUTH}}$-hybrid world.

functionality). Note that this by itself does not guarantee the authenticity of the response to an (EVAL, sid, S, x) activation of the protocol; indeed, in case S is corrupted there is no way to ensure that the value ρ obtained at the end of the protocol originates from a well-defined PRF owned by S (for instance S could be playing man-in-the-middle between users and two honest senders S_1, S_2 and thus S could be relaying responses from S_1 or S_2's depending on the identity of the callee U).

Security analysis. We prove the security of the 2HashDH under the (N, Q) one-more Diffie-Hellman (DH) assumption, which states that for any polynomial-time adversary $\mathcal{A}$,

$$\Pr_{k \leftarrow_{\text{R}} \mathbb{Z}_m, g_i \leftarrow_{\text{R}} \langle g\rangle}[\mathcal{A}^{(\cdot)^k, \mathsf{DDH}(\cdot)}(g, g^k, g_1, ..., g_n) = S]$$

is negligible, where $S = \{(g_{j_s}, g_{j_s}^k) | s = 1, ..., Q+1\}$, Q is the number of $\mathcal{A}$'s queries to the $(\cdot)^k$ oracle, and $j_s \in [N]$ for $s \in [Q+1]$.

In other words, suppose $\mathcal{A}$ is equipped with a "kth power" oracle and a DDH oracle, and THE number of queries to the former is limited by Q. $\mathcal{A}$ is given N random elements in $\langle g\rangle$. Since $\mathcal{A}$ is allowed to query the $(\cdot)^k$ oracle Q times, it is able to compute the kth power of any Q of the N elements. The assumption postulates that the probability that $\mathcal{A}$ computes the kth power of any $Q+1$ of the N elements (i.e. computes the kth power of "one more" element) is negligible.

Theorem 1. *Suppose the (N, Q)-one more DH assumption holds for $\langle g\rangle$, where Q is the number of functionality executions (i.e. the total amount of messages with* EVAL*sent by U) and $N = Q + q_1$ where q_1 is the number of $H_1(\cdot)$ queries. Then protocol 2Hash-DH UC-realizes the $\mathcal{F}_{\text{OPRF}}$ functionality.*

More precisely, for any adversary against the protocol in Section 2, there is a simulator SIM *that produces a view in the simulated world that no environment can distinguish with advantage better than $q_S \cdot \epsilon_{\mathsf{omdh},\langle g\rangle}(N, Q) + N^2/m$, where q_S is the number of senders, $\epsilon_{\mathsf{omdh},\langle g\rangle}(N, Q)$ is the probability of violating the (N, Q)-one more DH assumption and $m = |\langle g\rangle|$.*

1) Pick and record N random elements $r_1, ..., r_N \in \mathbb{Z}_m$, and set $g_1 := g^{r_1}, ..., g_N := g^{r_N}$. Set counter $J := 1$.
2) Every time when there is a fresh query x to $H_1(\cdot)$, answer it with g_J and record (x, r_J, g_J). Set $J := J+1$.
3) Upon receiving (EVAL, sid, U, S) from $\mathcal{F}_{\text{OPRF}}$, send g_J to $\mathcal{A}$ as U's message to S (suitably formatted to appear originating from the $\mathcal{F}_{\text{AUTH}}$ adversary interface) and record $\langle U, S, r_J, g_J \rangle$. Set $J := J + 1$.
4) Upon receiving a from $\mathcal{A}$ as some user's U message to a sender S
   - If there is a pair $\langle S, k \rangle$ stored in this step, then send a^k as the response of S for user U to $\mathcal{A}$.
   - Otherwise pick $k \leftarrow_{\text{R}} \mathbb{Z}_m$, record $\langle S, k \rangle$ and send a^k as the response of S for user U to $\mathcal{A}$.

   In either case, send (SNDRCOMPLETE, sid, S) to $\mathcal{F}_{\text{OPRF}}$.
5) Upon receiving b from $\mathcal{A}$ as some sender's S' message to a user U, recover r_j and g_j corresponding to U (as stored in step 4) and do the following:
   - If a pair $\langle S, k \rangle$ was recorded in step 4 with $b = g_j^k$, send (RCVCOMPLETE, sid, U, S', S) to $\mathcal{F}_{\text{OPRF}}$.
   - Otherwise if there exists p such that y_p is defined and $b^{1/r_j} = y_p$, then send (SNDRCOMPLETE, sid, p) and (RCVCOMPLETE, sid, U, S', p) to $\mathcal{F}_{\text{OPRF}}$.
   - Otherwise, choose a unique label p, set $y_p := b^{1/r_j}$, record (p, y_p), and send (SNDRCOMPLETE, sid, p) and (RCVCOMPLETE, sid, U, S', p) to $\mathcal{F}_{\text{OPRF}}$.
6) Every time when there is a fresh query (x, u) to $H_2(\cdot, \cdot)$,
   - If there exists a triple (x, r_i, g_i) stored in step 2, then
     - if there exists a pair $\langle S, k \rangle$ recorded in step 4 such that $u = g_i^k$, set $S' = S$.
     - if there exists a pair (p, y_p) recorded in step 5 such that $u = y_p^{r_i}$, set $S' = p$.

     then (i) send (EVAL, sid, p', x) to $\mathcal{F}_{\text{OPRF}}$ for a unique label p', (ii) if $S' = p$, then send (SNDRCOMPLETE, sid, S') (iii) finally, send (RCVCOMPLETE, sid, $\mathcal{A}^*$, p', S'). If $\mathcal{F}_{\text{OPRF}}$ ignores this message then abort and output FAIL. Otherwise after receiving $\mathcal{F}_{\text{OPRF}}$'s response (EVAL, sid, ρ), set $H_2(x, u) := \rho$.
   - Otherwise set $H_2(x, u)$ to be a random string in $\{0, 1\}^l$.

Figure 3. The Simulator SIM for the 2HashDH Protocol.

Proof. For any adversary $\mathcal{A}$, we construct a simulator SIM as in Figure 3.

We now argue that the SIM above generates a view to an arbitrary $\mathcal{Z}$ which is indistinguishable from the view in the real world. Without loss of generality, suppose $\mathcal{A}$ is a "dummy" adversary who merely pass through all its computation to $\mathcal{Z}$.

Consider case 1 where $\mathcal{Z}$ is in the simulated world and is able to control U, S and $\mathcal{A}$ using the following interfaces:

- U: $\mathcal{Z}$ sends (EVAL, sid, S, x) to U (and U transmits this message to $\mathcal{F}_{\text{OPRF}}$); U outputs ρ to $\mathcal{Z}$.
- S: S sends (SNDRCOMPLETE, sid) to $\mathcal{Z}$ when instructed by $\mathcal{F}_{\text{OPRF}}$.
- $\mathcal{A}$: $\mathcal{Z}$ sends a to $\mathcal{A}$ when $\mathcal{A}$ acts as a user (and $\mathcal{A}$ gives this message to SIM who acts as a sender), and sends b to $\mathcal{A}$ when $\mathcal{A}$ acts as a sender (and $\mathcal{A}$ transmits this message to SIM who acts as a user). $\mathcal{A}$ sends whatever it receives from SIM to $\mathcal{Z}$.

The view of $\mathcal{Z}$ in this case is as follows:

- (SNDRCOMPLETE, sid) from S,
- (EVAL, sid, ρ) from U,
- g_j from $\mathcal{A}$ when $\mathcal{A}$ acts as a sender,
- a^k from $\mathcal{A}$ when $\mathcal{A}$ acts as a user, and
- responses to $H_1(\cdot)$ and $H_2(\cdot, \cdot)$ queries.

Now consider case 2 where $\mathcal{Z}$ is in the real world and is able to control U, S and $\mathcal{A}$ using the same interfaces as in case 1. The view of $\mathcal{Z}$ in this case is as follows:

- (SNDRCOMPLETE, sid) from S,
- (EVAL, sid, $f = H_2(x, H_1(x)^k)$) from U,
- $H_1(x)^r$ from $\mathcal{A}$ when $\mathcal{A}$ acts as a sender,
- a^k from $\mathcal{A}$ when $\mathcal{A}$ acts as a user, and
- responses to $H_1(\cdot)$ and $H_2(\cdot, \cdot)$ queries.

We now compare the views generated in the two cases:

- (SNDRCOMPLETE, sid): In case 1, $\mathcal{A}$ receives (SNDRCOMPLETE, sid) in step 4: after SIM receives a from $\mathcal{A}$, it sends a^k to $\mathcal{A}$ and (SNDRCOMPLETE, sid, S) to $\mathcal{F}_{\text{OPRF}}$, and $\mathcal{F}_{\text{OPRF}}$ sends (SNDRCOMPLETE, sid) to S (who in turn transmits this message to $\mathcal{Z}$). In case 2, after S receives a from $\mathcal{A}$, it sends a^k to $\mathcal{A}$ and (SNDRCOMPLETE, sid) to $\mathcal{Z}$. From $\mathcal{Z}$'s point of view, the two cases are exactly the same.

- (EVAL, sid, ρ) and (EVAL, sid, $f = H_2(x, H_1(x)^k)$): In case 1, U outputs (EVAL, sid, ρ) in step 6. The only way to distinguish between ρ and f is to query the $H_2(\cdot,\cdot)$ oracle. However, once $(x, u = H_1(x)^k)$ is queried, SIM is able to detect this: If k is the key of an honest sender, then $u = g_i^k$ holds, which is the first condition of the first case in step 6. If k is the key of a sender generated by SIM, it is supposed to be $y_p = b^{1/r_j} = g_j^{k/r_j} = (g^{r_j})^{k/r_j} = g^k$, hence $u = g_i^k = (g^{r_i})^k = (g^k)^{r_i} = y_p^{r_i}$ holds, which is the second condition of the first case in step 6. After that, SIM can get the proper ρ value by interacting with $\mathcal{F}_{\text{OPRF}}$, i.e. these two values will be the same (provided that FAIL does not occur).
- g_j and $H_1(x)^r$: In case 1, $\mathcal{A}$ receives g_j in step 4, which is a random element in $\langle g \rangle$. In case 2, $\mathcal{A}$ receives $H_1(x)^r$ where r is a random value in $\mathbb{Z}_m$ chosen by U and not revealed to $\mathcal{A}$. Therefore, from $\mathcal{Z}$'s point of view, $H_1(x)^r$ is random, and cannot be distinguished from the random g_j.
- a^k: In case 1, $\mathcal{A}$ sends a to SIM and receives a^k from SIM, where k is a random value. In case 2, $\mathcal{A}$ sends a to S and receives a^k from S, where k is the sender's key, which is randomly chosen. The two cases are the same.
- answers to $H_1(\cdot)$ and $H_2(\cdot,\cdot)$ queries: In case 1, the answers are random from $\mathcal{Z}$'s point of view. Therefore, $\mathcal{Z}$ cannot distinguish them from answers in case 2.

We conclude that if FAIL does not occur, $\mathcal{Z}$ is not able to distinguish between case 1 (the simulated world) and case 2 (the real world). Now we upper bound $Pr[\text{FAIL}]$. Event FAIL occurs in the first case of step 6, where SIM sends a (RCVCOMPLETE, sid, $\mathcal{A}^*$, p', S') message to $\mathcal{F}_{\text{OPRF}}$ but it is ignored. This can only happen when the first condition holds and $\mathsf{tx}(S') = 0$. For any $\tilde{S}$, let $\text{FAIL}(\tilde{S})$ be the particular event that the (RCVCOMPLETE, sid, $\mathcal{A}^*$, p', $\tilde{S}$) message is ignored. Note that the quantification over $\tilde{S}$ is over all the honest servers' identities. We can see that $\Pr[\text{FAIL}] \leq \sum_{\tilde{S}} \Pr[\text{FAIL}(\tilde{S})]$, where the number of terms in the summation is bounded by q_S, the number of senders.

We now upper bound $\Pr[\text{FAIL}(\tilde{S})]$ by reducing $\text{FAIL}(\tilde{S})$ to the one-more DH problem. Using the reduction $\mathcal{R}_{\tilde{S}}$ in Figure 4, we can see the following two facts:

- Every time the $(\cdot)^{\tilde{k}}$ oracle is used (in step 4), a $(\text{SNDRCOMPLETE}, sid, \tilde{S})$ message is sent to $\mathcal{F}_{\text{OPRF}}$, so $\mathsf{tx}(\tilde{S})$ increases.
- Every time $\mathsf{tx}(\tilde{S})$ decreases (in the first case of step 5 or the first condition of the first case of step 6), a pair of the form $(z, z^{\tilde{k}})$ is recorded, where z is either $H_1(x)$ or the message sent to $\mathcal{A}$ in step 3; in either case, z is in $g_1, ..., g_N$.

Therefore, if $\text{FAIL}(\tilde{S})$ occurs, the number of pairs recorded is more than the number of DDH oracle queries by one (assuming that there is no collision in $g_1, ..., g_N$), so the one more DH-assumption is violated. That is, $\Pr[\text{FAIL}(\tilde{S})|\text{There is no collision in } g_1, ..., g_N] = \epsilon_{\mathsf{omdh},\langle g \rangle}(N, Q)$. So

$$\Pr[\text{FAIL}|\text{There is no collision in } g_1, ..., g_N] \leq q_S \cdot \epsilon_{\mathsf{omdh},\langle g \rangle}(N, Q)$$

On the other hand, the probability that there is collision in $g_1, ..., g_N$ is upper bounded by N^2/m. Thus, we have

$$\Pr[\text{FAIL}] \leq q_S \cdot \epsilon_{\mathsf{omdh},\langle g \rangle}(N, Q) + N^2/m.$$

That concludes our proof. □

Remark. Although in this work $\mathcal{F}_{\text{OPRF}}$ is used to construct a PPSS scheme, we mention here that OPRF is of interest in many other areas, such as keyword search [9] and secure function evaluation of set intersection [10], [13]. Therefore, our protocol 2HashDH may have further applications.

4. PPSS: Security Definitions

Password-Protected Secret Sharing (PPSS) was defined in [2] as (password-protected) secret-sharing of an *arbitrary message* and it was re-defined in [11] as protecting a *random key*. We adopt this latter notion here because it is more flexible: a secret key can be used not only for encryption, but also for signatures, key exchange protocols, and so on.

Definition 1. *PPSS scheme* (ParGen, SKeyGen, INIT, REC) *involves user U and n servers $S_1, ..., S_n$:*

ParGen*: Given security parameter ℓ, generates public parameters crs.*

SKeyGen*: S_i, given crs, generates its private state σ_i.*

INIT *is a protocol between U and $S_1, ..., S_n$ s.t.*

- *U runs algorithm $\mathsf{U}_{\mathsf{Init}}$, which given crs and a password* pw*, outputs a private key $K \in \{0,1\}^\ell$.*
- *S_i runs algorithm $\mathsf{S}_{\mathsf{Init}}$, which given crs and σ_i, outputs user-specific information ω_i.*

REC *is a protocol between by U and $S_1, ..., S_n$ s.t.*

- *U runs algorithm $\mathsf{U}_{\mathsf{Rec}}$, which given crs and* pw*, outputs $K' \in \{0,1\}^\ell \cup \{\perp\}$.*
- *S_i runs algorithm $\mathsf{S}_{\mathsf{Rec}}$ on input crs, σ_i, ω_i.*

Suppose $(Q, g, y = g^{\tilde{k}}, g_1, ..., g_N)$ is an instance of the one-more DH problem. We run the simulator, with the following revisions:

- In step 1, only set $J := 1$ and omit all other processes.
- In step 2 and step 3, use the challenges $g_1, ..., g_N$ instead of random elements in $\langle g \rangle$ to answer $H_1(\cdot)$ queries (in step 2) and prepare the messages sent to $\mathcal{A}$ (in step 3). Besides, since r_j is unknown, record (x, g_j) instead of (x, r_j, g_j) (in step 2), and $\langle U, g_j \rangle$ instead of $\langle U, r_j, g_j \rangle$ (in step 3).
- In step 4, if $\tilde{a}$ acts as a message to $\tilde{S}$, then use the $(\cdot)^{\tilde{k}}$ oracle to compute $\tilde{b} := \tilde{a}^{\tilde{k}}$. Record $(\tilde{a}, \tilde{b})$ instead of $\langle \tilde{S}, \tilde{k} \rangle$.
- In step 5, do the following instead: Upon receiving b from $\mathcal{A}$ as some sender's S' message to a user U, recover the corresponding g_j (stored in step 3), record (g_j, b), and do the following:
  - If DDH$(g_j, b, \tilde{a}, \tilde{b})$ then send (RCVCOMPLETE, sid, U, S', $\tilde{S}$) to $\mathcal{F}_{\text{OPRF}}$.
  - Otherwise if there exists a pair $\langle S, k \rangle$ stored in step 4 such that $b = a^k$, then send (RCVCOMPLETE, sid, U, S', S) to $\mathcal{F}_{\text{OPRF}}$.
  - Otherwise if there exists p such that a_p and b_p are defined and DDH(g_j, b, a_p, b_p), then send (SNDRCOMPLETE, sid, p) and (RCVCOMPLETE, sid, U, S', p) to $\mathcal{F}_{\text{OPRF}}$.
  - Otherwise choose a unique value p, set $a_p := g_j, b_p := b$, and send (SNDRCOMPLETE, sid, p) and (RCVCOMPLETE, sid, U, S', p) to $\mathcal{F}_{\text{OPRF}}$.
- In the first case of step 6, use the following to determine S' and proceed as before.
  - If DDH$(g_i, u, \tilde{a}, \tilde{b})$, then $S' = \tilde{S}$.
  - If there exists a pair $\langle S, k \rangle$ stored in step 4 such that $u = g_i^k$ then $S' = S$.
  - if there exists p such that DDH(g_i, u, a_p, b_p) then $S' = p$.

Figure 4. The Reduction $\mathcal{R}_{\tilde{S}}$ to the One-More DH Problem.

4.1. The Game-Based Definition

Here we recall the indistinguishability (or game-based) PPSS definition from [11], simplified to the case where adversary corrupts the maximal threshold t of servers, while in Section 4.2 below we upgrade this definition to the UC setting.

Correctness. If user U has interacts with uncorrupted servers then it must reconstruct the same key that was generated in the initialization process; that is, for any ℓ, any $crs \leftarrow \mathsf{ParGen}(1^\ell)$, any $(\sigma_1, ..., \sigma_n) \leftarrow \mathsf{SKeyGen}(crs)$, any $\mathsf{pw} \in \mathsf{D}$, any $K \leftarrow \text{INIT}(crs, \mathsf{pw})$, and any $K' \leftarrow \text{REC}(crs, \mathsf{pw})$, we have $K' = K$.

Security. Assume that an adversary $\mathcal{A}$ corrupts subset B of up to t out of the n servers and acts as the "man-in-the-middle" in q instances of PPSS reconstruction protocol. Intuitively, the security of PPSS means that the key K should remain pseudorandom to $\mathcal{A}$ except for probability $q/|\mathsf{D}|$. To model this formally, the security game picks bit b and at the end of the INIT protocol let $\mathcal{A}$ see key $K^{(b)}$ which is defined as the real key K output by U for $b = 1$, or a random ℓ-bit string for $b = 0$. To model $\mathcal{A}^*$'s interaction in REC instances we let $\mathcal{A}^*$ interact with oracles $\mathsf{S}_{\mathsf{Rec}}$ and $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$, where $\mathsf{S}_{\mathsf{Rec}}$ runs S_i's REC protocol on input $(crs, \sigma_i, \omega_i)$ for $i \notin B$, while $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ runs $\mathsf{U}_{\mathsf{Rec}}$ on input (crs, pw) but when $\mathsf{U}_{\mathsf{Rec}}$ outputs its local output K' or $\perp$, $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ processes this output as follows: If $b = 1$ then it forwards K' or $\perp$ it unchanged to $\mathcal{A}^*$, but if $b = 0$ then it forwards $K^{(0)}$ instead of K' if $\mathsf{U}_{\mathsf{Rec}}$'s output is $K' \neq \perp$, and forwards $\perp$ otherwise.

Definition 2. *A PPSS scheme with parameters (t, n) is (q_U, q_S, ϵ)-secure if for any polynomial-time algorithm $\mathcal{A}$ and any $B \in \{1, ..., n\}$ s.t. $|B| = t$, we have*

$$\mathbf{Adv}^{\mathbf{ppss}}_{\mathcal{A}} \leq (q_U + q_S)/|D| + \epsilon,$$

where $\mathbf{Adv}^{\mathbf{ppss}}_{\mathcal{A}} = |p_1 - p_0|$ for $p_b = \Pr[b' = 1]$ where b' is $\mathcal{A}$'s output in a game below defined for $b = 0, 1$:

(1) Set $\mathsf{pw} \leftarrow_R \mathsf{D}$, $crs \leftarrow \mathsf{ParGen}(1^\ell)$ and $\sigma_i \leftarrow \mathsf{SKeyGen}(crs)$ for $i \in [n]$. Send crs to $\mathcal{A}$.

(2) Let $\mathsf{U}_{\mathsf{Init}}(crs, \mathsf{pw})$ interact with $\mathsf{S}_{\mathsf{Init}}(crs, \sigma_i)$ for all i, where $\mathcal{A}$ has full control over each S_i for $i \in B$ (and can modify their protocols), while the channels between U and each S_i for $i \notin B$ are authenticated. Let $K^{(1)}$ be $\mathsf{U}_{\mathsf{Init}}$'s output in the above interaction, and let $K^{(0)} \leftarrow_R \{0, 1\}^\ell$. Send $K^{(b)}$ to $\mathcal{A}$.

(3) $\mathcal{A}$ outputs b' after q_U interactions with $\mathsf{U}^{\diamond}_{\mathsf{Rec}}(crs, \mathsf{pw}, b, K^{(b)})$ and q_S interactions with $\mathsf{S}_{\mathsf{Rec}}$.

Soundness. As argued in [11], the above security notion also implies a soundness property, i.e. that an efficient

adversary controlling up to t servers S_i is unable to make U output key K' in protocol REC s.t. $K' \neq K$ where K was output by U in protocol INIT. If there exists $\mathcal{A}$ who successfully attack soundness then $\mathcal{A}'$ can run $\mathcal{A}$ and output 0 if U's outputs K in Init and K' in Rec are the same, and 1 otherwise: If $\mathcal{A}$ breaks PPSS soundness then $\mathcal{A}'$ breaks PPSS security.

4.2. The UC Definition

We define a universally composable (UC) notion of PPSS as a secure realization of an ideal PPSS functionality $\mathcal{F}_{\text{PPSS}}$ presented in Figure 5. The PPSS functionality we define is weaker than the PPPS functionality of [4] (called T-PASS in that paper), since it obviates the need for extracting malicious clients' inputs at the time the PPSS reconstruction protocol takes place. In order to avoid requiring such on-line input extraction we use a ticketing mechanism which is similar to the one we use in the UC definition of OPRF in Section 2. We believe that any protocol that realizes the T-PASS functionality of [4] should also realize our functionality $\mathcal{F}_{\text{PPSS}}$ of Figure 5, however it can also be implemented by a much more lightweight protocol that most likely do not realize the T-PASS functionality of [4]. Functionality $\mathcal{F}_{\text{PPSS}}$ we propose has three interfaces, Initialization, Reconstruction, and PasswordTest, on which we elaborate below.

The *Initialization* command INIT represents a user with a unique username, represented by sid, at some network entity U initializing a PPSS instance with a set of n servers $\mathcal{SI} = \{S_1, \ldots, S_n\}$ on input a password pw. The servers in $\mathcal{SI}$ become activated for this instance provided that the ideal-world adversary $\mathcal{A}^*$ agrees by sending command SINIT, and if all servers in $\mathcal{SI}$ are activated the instance generates a random secret K output by U if adversary agrees by sending command UINIT. (In the real protocol this corresponds to adversary forwarding protocol messages correctly.) If set $\mathcal{SI}$ contains $t+1$ corrupted servers then $\mathcal{A}^*$ receives (pw, K), which corresponds to U creating a password-protected secret-sharing among n players of whom more than t are corrupted, at which point a (t, n)-threshold secret-sharing no longer protects its secrets.

The *Reconstruction* command REC represents a user at a potentially different network entity U' attempting to recover the secret initialized above using password pw', which might or might not equal to pw above. The reconstruction operation is directed to some set of $t+1$ servers $\mathcal{SR}$, which might or might not overlap with set $\mathcal{SI}$ above. It is important to emphasize that the user maintains no state between the initialization and the reconstruction operations except for memorizing password pw (and username sid), although a failure to remember pw correctly is also allowed, and it is modeled by setting $\mathsf{pw}' \neq \mathsf{pw}$. In particular, the user might connect to a different set of servers in the initialization and in the reconstruction. Hence, for example, if a user falls prey to a phishing attack, she could execute the reconstruction protocol with servers $\mathcal{SR}$ s.t. $\mathcal{SR} \cap \mathcal{SI} = \emptyset$ and all servers in $\mathcal{SR}$ are corrupted. However, by the rules of the $\mathcal{F}_{\text{PPSS}}$ functionality which we explain below, the worst thing that can happen in this case is the inevitable on-line guessing attack: The adversary can execute the reconstruction protocol on behalf of the corrupt servers $\mathcal{SR}$ for some chosen password pw^* and secret key K^*, and in the case $\mathsf{pw}^* = \mathsf{pw}$ it would cause U to reconstruct K^* instead of K (or $\perp$).

SREC and UREC commands control the view of the reconstruction protocol by the servers and the user in a similar way as in the INIT above, but with some significant differences: First, U's sessions with any corrupt server in $\mathcal{SI}$ can be "routed" by $\mathcal{A}^*$ to any other server, hence in UREC command $\mathcal{A}^*$ specifies a set $\mathcal{SC}$ of servers who effectively participate in this reconstruction, with the only constrain that $\mathcal{SC}$ must contain all uncorrupted servers in $\mathcal{SI}$. Secondly, user's completion can result in two different outcomes (in addition to failure which $\mathcal{A}^*$ can except in the case when $\mathsf{pw}' = \mathsf{pw}$ and $\mathcal{SR}$ contains only uncorrupted servers in $\mathcal{SI}$): The default case is that the reconstruction works and U' outputs key K created in the initialization, which happens when $\mathsf{pw}' = \mathsf{pw}$, i.e. U' ran on the correct password, $\mathcal{SC} \subseteq \mathcal{SI}$, i.e. U' connected to servers participating in the initialization, and there were either no corrupt servers in the set $\mathcal{SR}$ with which U' attempted the reconstruction, or, if some of those servers are corrupt $\mathcal{A}^*$ still allowed the protocol to succeed by setting the flag variable to 1. Another case is when U' connected only to corrupt servers (and $\mathcal{A}^*$ does not route these connections to uncorrupted servers, hence we require not only that $\mathcal{SR} \subseteq \mathsf{CorrSrv}$ but also that $\mathcal{SC} \subseteq \mathsf{CorrSrv}$, which is stronger because $\mathcal{SR} \setminus \mathsf{CorrSrv} \subseteq \mathcal{SC}$), because such reconstruction session offers $\mathcal{A}^*$ an ability to perform an *on-line guessing attack on the user*, i.e. $\mathcal{A}^*$ can specify its password guess pw^* and, in case $\mathsf{pw}^* = \mathsf{pw}$, cause U' to output any value K^* specified by $\mathcal{A}^*$. Indeed, a PPSS scheme which, like ours, does not assume any other source of user's security apart of the password, and in particular does *not* assume PKI for security, cannot offer stronger protection in the case the user executes the protocol with an adversary who guesses her password.

Finally, the *Test Password* command TESTPWD lets adversary $\mathcal{A}^*$ perform an *on-line guessing attack on the servers*, i.e. specify a password guess pw^* and a

Initialize $\mathsf{tested}(\mathsf{pw})$ to $\varnothing$ and $\mathsf{tx}(S)$ to 0 for all S.
Initialization:

- Upon receiving $(\text{INIT}, sid, \mathcal{SI}, \mathsf{pw})$ for $|\mathcal{SI}|{=}n$ from U, record $\langle \text{INIT}, sid, \mathcal{SI}, \mathsf{pw}\rangle$ and send $(\text{INIT}, U, sid, \mathcal{SI})$ to $\mathcal{A}^*$. (Ignore other INIT commands.) Pick $K{\leftarrow}\{0,1\}^l$, and if $|\mathcal{SI} \cap \mathsf{CorrSrv}| \geq t+1$ then send (K, pw) to $\mathcal{A}^*$.
- Upon receiving (SINIT, sid, S) from $\mathcal{A}^*$, if record $\langle \text{INIT}, U, sid, \mathcal{SI}, \mathsf{pw}\rangle$ exists and $S \in \mathcal{SI}$ then mark S as ACTIVE and send (SINIT, sid) to S.
- Upon receiving (UINIT, sid) from $\mathcal{A}^*$, if record $\langle \text{INIT}, U, sid, \mathcal{SI}, \mathsf{pw}\rangle$ exists and all servers in $\mathcal{SI}$ are marked ACTIVE then add K to $\langle \text{INIT}, U, sid, \mathcal{SI}, \mathsf{pw}\rangle$ and send (UINIT, sid, K) to U.

Reconstruction:

- Upon receiving $(\text{REC}, sid, ssid, \mathcal{SR}, \mathsf{pw}')$ for $|\mathcal{SR}|{=}t{+}1$ from U', retrieve record $\langle \text{INIT}, U, sid, \mathcal{SI}, \mathsf{pw}, K\rangle$, record $\langle \text{REC}, U', sid, ssid, \mathcal{SI}, \mathcal{SR}, \mathsf{pw}, \mathsf{pw}'\rangle$ and send $(\text{REC}, U', sid, ssid, \mathcal{SR})$ to $\mathcal{A}^*$. Ignore future REC commands involving the same $ssid$.
- Upon receiving $(\text{SREC}, sid, ssid, S)$ from $\mathcal{A}^*$, if S is marked ACTIVE then increment $\mathsf{tx}(S)$ and send $(\text{SREC}, sid, ssid)$ to S.
- Upon receiving $(\text{UREC}, sid, ssid, \mathcal{SC}, \mathsf{flag}, \mathsf{pw}^*, K^*)$ for $|\mathcal{SC}|{=}t{+}1$ from $\mathcal{A}^*$, if a record $\langle \text{REC}, U', sid, ssid, \mathcal{SI}, \mathcal{SR}, \mathsf{pw}, \mathsf{pw}', K\rangle$ exists s.t. $\mathcal{SR} \setminus \mathsf{CorrSrv} \subseteq \mathcal{SC}$ and $\mathsf{tx}(S){>}0$ for all S in $\mathcal{SC}$ then decrement $\mathsf{tx}(S)$ for all such S and send $(\text{UREC}, sid, ssid, \text{RES})$ to U' s.t.:
  - $\text{RES} := K$ if $(\mathsf{pw}' = \mathsf{pw}) \wedge (\mathcal{SC} \subseteq \mathcal{SI}) \wedge [(\mathsf{flag} = 1) \vee (\mathcal{SR} \cap \mathsf{CorrSrv} = \varnothing)]$;
  - $\text{RES} := K^*$ if $(\mathsf{pw}' = \mathsf{pw}^*) \wedge (\mathcal{SC} \subseteq \mathsf{CorrSrv}) \wedge (\mathsf{flag} = 2)$;
  - $\text{RES} :=$ FAIL otherwise.

Password Test:

- Upon receiving $(\text{TESTPWD}, sid, S_i, \mathsf{pw}^*)$ from $\mathcal{A}^*$, if $\mathsf{tx}(S_i) > 0$ then set $\mathsf{tested}(\mathsf{pw}^*) := \mathsf{tested}(\mathsf{pw}^*) \cup \{S_i\}$ and $\mathsf{tx}(S_i) := \mathsf{tx}(S_i){-}1$, retrieve $\langle \text{INIT}, U, \mathcal{SI}, \mathsf{pw}, K\rangle$, and if $|\mathcal{SI} \cap (\mathsf{tested}(\mathsf{pw}^*) \cup \mathsf{CorrSrv})| \geq t{+}1$, then return K to $\mathcal{A}^*$ if $\mathsf{pw}^* = \mathsf{pw}$, else return FAIL.

Figure 5. Ideal (t, n)-Threshold PPSS Functionality $\mathcal{F}_{\text{PPSS}}$.

set $\mathcal{S}$ of at least $t + 1$ servers in $\mathcal{SI}$, and learn key K if $\mathsf{pw}^* = \mathsf{pw}$. However, $\mathcal{F}_{\text{PPSS}}$ allows such guessing attack to proceed only if $\mathcal{A}^*$ engages the servers in $\mathcal{S}$ in reconstruction protocol instances for username sid, as represented by SREC commands. Every time such command is issued for some server S, functionality $\mathcal{F}_{\text{PPSS}}$ increments a *ticket counter* $\mathsf{tx}(S)$, and the adversary can make a password guess only $\mathsf{tx}(S) > 0$ for all $S \in \mathcal{S}$, and the counters of serves in $\mathcal{S}$ are decremented as result of such password-testing attempt. Since a PPSS server cannot tell if a reconstruction protocol instance is originated by an honest user or by the adversary, any such reconstruction session can be used *either* for completion of honest user reconstruction instances or for instances executed by the adversary. However, the ticket-counting mechanism of $\mathcal{F}_{\text{PPSS}}$ enforces that any PPSS instance completed by $t + 1$ can be "used up" either for a single instance of the honest user reconstructing her secret or for a single instance of an adversary who attempts the reconstruction on a guessed password pw^*.

5. Proposed PPSS Scheme

In Figure 6 we present a PPSS scheme π_{PPSS} which is secure in the random oracle model assuming a non-malleable commitment scheme. Protocol π_{PPSS} uses the functionality $\mathcal{F}_{\text{OPRF}}$ as part of the construction and it assumes that communication proceeds over authenticated channels, modeled by functionality $\mathcal{F}_{\text{AUTH}}$. Figure 7 shows a concrete implementation of this PPSS scheme π_{PPSS} using the 2HashDH OPRF from Figure 2 and a hash-based commitment secure in the random oracle model. The resulting protocol takes one interaction round in reconstruction and two in initialization, and in

Parameters: Security parameter ℓ, threshold parameters $t, n \in \mathbb{N}, t \leq n$, field $\mathbb{F} := GF(2^\ell)$, instance of commitment scheme COM, hash function H with range $\{0,1\}^{2\ell}$.

INIT for user U:

1) On input $(\text{INIT}, sid, \{S_1, ..., S_n\}, \mathsf{pw})$, pick $s \in_R \mathbb{F}$ and parse $H(s)$ as $[r||K]$.
2) Generate $(s_1, ..., s_n)$ as a (t, n) Shamir's secret-sharing of s over $\mathbb{F}$ and set $\boldsymbol{s} := (s_1, ..., s_n)$.
3) Send $(\text{EVAL}, (sid, i), S_i, \mathsf{pw})$ to $\mathcal{F}_{\text{OPRF}}$ and wait for response $(\text{EVAL}, (sid, i), \rho_i)$, for all $i \in [n]$.
4) Compute $e_i := s_i \oplus \rho_i$ for $i \in [n]$, set $\boldsymbol{e} := (e_1, ..., e_n)$, and compute $C := \mathsf{COM}((\mathsf{pw}, \boldsymbol{e}, \boldsymbol{s}); r)$.
5) Set $\omega := (\boldsymbol{e}, C)$ and send $(\text{SEND}, (sid, i, 0), S_i, \omega)$ to $\mathcal{F}_{\text{AUTH}}$ for $i \in [n]$.
6) If $\mathcal{F}_{\text{AUTH}}$ returns $(\text{SENT}, (sid, i, 1), S_i, \text{ACK})$ for all $i \in [n]$, output (UINIT, sid, K).

INIT for server S:

1) On message $(\text{SNDRCOMPLETE}, (sid, i))$ from $\mathcal{F}_{\text{OPRF}}$ for some i, create a record $\langle sid, i \rangle$.
2) On message $(\text{SENT}, (sid, i, 0), U, \omega)$ from $\mathcal{F}_{\text{AUTH}}$ for some existing record $\langle sid, i \rangle$, append ω to it, send $(\text{SEND}, (sid, i, 1), U, \text{ACK})$ to $\mathcal{F}_{\text{AUTH}}$, and output (SINIT, sid).

REC for user U:

1) On input $(\text{REC}, sid, ssid, \mathcal{SR}, \mathsf{pw}')$, send $(\text{EVAL}, (sid, ssid, j), S'_j, \mathsf{pw}')$ to $\mathcal{F}_{\text{OPRF}}$ for all $S'_j \in \mathcal{SR}$.
2) If $\mathcal{F}_{\text{OPRF}}$ returns $(\text{EVAL}, (sid, ssid, j), \sigma_j)$ and $\mathcal{F}_{\text{AUTH}}$ returns $(\text{SENT}, (sid, ssid, j, 1), S'_j, (i_j, \omega_j))$ for all $j \in [t+1]$, then if $i_{j_1} = i_{j_2}$ or if $\omega_{j_1} \neq \omega_{j_2}$ for any $j_1 \neq j_2$ then output $(\text{UREC}, sid, ssid, \text{FAIL})$ and stop. Otherwise set $\rho'_{i_j} := \sigma_j$ for all $j \in [t+1]$, and set $I := \{i_j \mid j \in [t+1]\}$.
3) Parse any ω_j as $(\boldsymbol{e}', C')$, parse $\boldsymbol{e}'$ as $(e'_1, ..., e'_n)$, and set $s'_i := e'_i \oplus \rho'_i$ for all $i \in I$.
4) Recover s' and the shares s'_i for $i \notin I$ by interpolating points (i, s'_i) for $i \in I$.
5) Set $\boldsymbol{s'} := (s'_1, ..., s'_n)$, parse $H(s')$ as $[r'||K']$, and output $(\text{UREC}, sid, ssid, \text{RES})$ for $\text{RES} = K'$ if $C' = \mathsf{COM}((\mathsf{pw}', \boldsymbol{e'}, \boldsymbol{s'}); r')$ and $\text{RES} = \text{FAIL}$ otherwise.

REC for server S:

1) Given message $(\text{SNDRCOMPLETE}, (sid, ssid, j))$ from $\mathcal{F}_{\text{OPRF}}$, if S holds record $\langle sid, i, \omega \rangle$ then send $(\text{SEND}, (sid, ssid, j, 1), U, (i, \omega))$ to $\mathcal{F}_{\text{AUTH}}$ and output $(\text{SREC}, sid, ssid)$.

Figure 6. Protocol π_{PPSS} which realizes UC functionality $\mathcal{F}_{\text{PPSS}}$ in the $(\mathcal{F}_{\text{OPRF}}, \mathcal{F}_{\text{AUTH}})$-hybrid world.

both phases each server makes 1 exponentiation while the user makes $t + 2$.[3]

We note that the assumption that all communication proceeds over authenticated channels is a useful simplification for security analysis. However, while some form of authenticated channels is necessary in initialization, e.g. implemented by TLS authenticated under servers' PKI certificates, in reconstruction such PKI-based authentication is a form of security "hedging", but it's not strictly necessary. As our security analysis shows, each reconstruction instance executed with malicious servers gives the adversary the ability to make one online password guess, exactly the same as in any password-only PAKE scheme. But if the user forms an authenticated connection to at least one uncorrupted server, e.g. using PKI certificates, then our UC security model shows that the adversary then loses this online password guessing ability.

Note on Robustness. The ideal PPSS functionality $\mathcal{F}_{\text{PPSS}}$ we specify in Section 4 does not yield a robust PPSS scheme in a generic way, e.g. because functionality $\mathcal{F}_{\text{PPSS}}$ requests that the user specifies the exact set of $t + 1$ servers and in the case of failure does not let the user know which of the servers in the requested set have failed to perform correctly. (In that aspect our PPSS functionality is similar to the one proposed in [4].) Indeed, we believe that it would be useful to generalize

3. In a forthcoming work we show that client-side costs can be reduced further by "de-blinding" the OPRF responses of all servers in one multi-exponentiation. In this alternative scheme the secret-sharing is performed on the level of OPRF *key*, and this shared-key OPRF variant is conceptualized as a *Threshold OPRF*.

Parameters: Security parameter ℓ, threshold parameters $t, n \in \mathbb{N}, t \leq n$, field $\mathbb{F} = GF(2^\ell)$, cyclic group of prime order m with generator g; hash functions H_1, H_2, H_3 with ranges $\langle g \rangle, \{0,1\}^\ell, \{0,1\}^{2\ell}$.

Communication Setting: Communication between $U, S_1, ..., S_n$ goes over *authenticated channels*, e.g. TLS.

INIT for user U on input $(sid, \{S_1, ..., S_n\}, \mathsf{pw})$:

1) Choose $s \in_R \mathbb{F}$ and generate shares $(s_1, \ldots, s_n)$ as a (t, n)-secret-sharing of s over $\mathbb{F}$.
2) Pick $\alpha \in_R \mathbb{Z}_m$, compute $a = (H_1(\mathsf{pw}))^\alpha$, and send $((sid, i), a)$ to S_i for $i \in [n]$.
3) Given $b_i (= a^{k_i})$ from each S_i, set $\rho_i = H_2(\mathsf{pw}, b_i^{1/\alpha})$ and $e_i = s_i \oplus \rho_i$ for all $i \in [n]$.
4) Set $\boldsymbol{e} := (e_1, \ldots, e_n)$, $\boldsymbol{s} := (s_1, ..., s_n)$, $[r||K] := H_3(0, s)$, $C = H_3(1, \mathsf{pw}, \boldsymbol{e}, \boldsymbol{s}, r)$, and $\omega := (\boldsymbol{e}, c)$.
5) Send $((sid, i, 0), \omega)$ to S_i for all $i \in [n]$, and given $((sid, i, 1), \text{ACK})$ for all $i \in [n]$ output K.

INIT for server S_i:

Pick $k_i \in_R \mathbb{Z}_m$, on message (sid, a_i) from U, abort if sid not unique, otherwise reply with $b_i = (a_i)^{k_i}$. On message $((sid, i, 0), \omega)$ from U, store (sid, i, ω_i, k_i) and send $((sid, i, 1), \text{ACK})$ to U.

REC for user U on input $(sid, ssid, \{S'_1, ..., S'_{t+1}\}, \mathsf{pw})$:

1) Pick $\alpha \in_R \mathbb{Z}_m$, compute $a = (H_1(\mathsf{pw}))^\alpha$, and send $((sid, ssid, j), a)$ to S_j for $j \in [t+1]$.
2) Given $((sid, ssid, j), (b_j, i_j, \omega_j))$ from each S_j, set $\sigma_j = H_2(\mathsf{pw}, b_j^{1/\alpha})$ for $j \in [t+1]$. Abort if $i_{j_1} = i_{j_2}$ or $\omega_{j_1} \neq \omega_{j_2}$ for any $j_1 \neq j_2$. Otherwise set $\rho_{i_j} := \sigma_j$ for all $j \in [t+1]$ and $I := \{i_j \mid j \in [t+1]\}$.
3) Parse ω_1 as $(\boldsymbol{e}, C)$ and $\boldsymbol{e}$ as $(e_1, ..., e_n)$. Set $s_i := e_i \oplus \rho_i$ for each $i \in I$.
4) Recover s and the shares s_i for $i \notin I$ by interpolating points (i, s_i) for $i \in I$. Set $\boldsymbol{s} := (s_1, ..., s_n)$.
5) Parse $H_3(0, s)$ as $[r||K]$, and output K if $C = H_3(1, \mathsf{pw}, \boldsymbol{e}, \boldsymbol{s}, r)$ and FAIL otherwise.

REC for server S_i:

Given $((sid, ssid, j), a)$ from U, abort if $a \notin \langle g \rangle$ or S_i does not hold a record (sid, i, ω, k_i) with the matching sid. Otherwise compute $b := a^{k_i}$ and send $((sid, ssid, j, 1), (b, i, \omega))$ back to U.

Figure 7. Concrete Instantiation of PPSS Protocol π_{PPSS} of Figure 6, based on 2HashDH OPRF

the PPSS functionality we present by (1) letting the user direct the reconstruction protocol instance to a server set $\mathcal{SR}$ containing more than $t+1$ servers, instead of fixing $|\mathcal{SR}|$ to $t+1$ as in $\mathcal{F}_{\text{PPSS}}$ of Figure 5; and (2) reporting on the identity of servers in $\mathcal{SR}$ which failed to respond consistently with others in the reconstruction attempt. However, although we do not show that formally, we believe that protocol π_{PPSS} which we present here can be easily extended to provide both properties: The user could initialize the protocol with more than $t+1$ servers, and if in step 2 of REC the user gets more than $t+1$ replies but the reconstruction fails on the first subset of $t+1$ replies, the user could either search for the $(t+1)$-element subset for which reconstruction succeeds, or it could request that each responding server S_j provides a ZK proof that its response b_j is correct under public S_j's public verification parameter $\pi_j = g^{k_j}$, as in the Verifiable OPRF used in the PPSS scheme of [11]. A vector $\boldsymbol{\pi}$ of these public parameters would have to be committed along $(\mathsf{pw}, \boldsymbol{e}, \boldsymbol{s})$ in C, but to reduce communication C would be a commitment to $(\mathsf{pw}, \boldsymbol{e}, \boldsymbol{s}, h)$ where $h = H(\boldsymbol{\pi})$, hence only the hash value h would have to be transmitted along with $(\boldsymbol{e}, C)$ by each S_j in the default case of no active faults.

5.1. Game-Based Security of Scheme π_{PPSS}

We prove that protocol π_{PPSS} in Figure 6 satisfies the game-based PPSS security notion (see Section 4.1).

Theorem 2. *If* COM *is ϵ_B-binding, ϵ_H-hiding and ϵ_{NM}-non-malleable commitment scheme and H is a random oracle then the PPSS scheme π_{PPSS} of Figure 6 is (q_U, q_S, ϵ)-secure for $\epsilon = \epsilon_B + 2\epsilon_H + q_U\epsilon_{NM} + (q_H + q_U)/2^\ell$, where q_H is the bound on the number of adversary's queries to H.*

Proof. We first describe the security experiment. Of the four processes of the execution of the scheme, ParGen and SKeyGen are trivial, so we omit them here. We assume that t among n servers are corrupt. Let B denote the set of these corrupt servers.

In Init, the adversary $\mathcal{A}$ is assumed not to interfere in the computation of $f_g(\mathsf{pw})$ where $g \notin B$. However, for $b \in B$, $\mathcal{A}$ can send a pointer q, which may or may not be b, so that U receives $f_q(\mathsf{pw})$. Let Q be the set of such q's (so Q is a t-element set of pointers).

In Rec, adversary $\mathcal{A}$ interacts with two oracles, $\mathsf{U}_{\mathsf{Rec}}$ and $\mathsf{S}_{\mathsf{Rec}}$, which run the code of U and $S_1, ..., S_n$ in Rec, respectively, which creates the following view for $\mathcal{A}$:

When interacting with $\mathsf{U}_{\mathsf{Rec}}$, $\mathcal{A}$'s view is a function of string ω which U will use in this instance, and two sets of pointers P and P':

- $\mathcal{A}$ sends a set of $t+1$ ω'_j values to U in step 1. Note that U continues execution only if all these values are the same; otherwise U aborts. This is equivalent to the scenario where $\mathcal{A}$ specifies ω' and a $(t+1)$-element sequence P in $\{1,...,n\}$ and sends an $\omega'_P = \omega'$ for $p \in P$ to U. Let $\omega' = (\boldsymbol{e'}, C') = ((e'_1, ..., e'_n), C')$.
- When U sends (EVAL, sid, S_p, pw) to $\mathcal{F}$, A can send $(\text{UINIT}, sid, U, p')$ to $\mathcal{F}$ in step 2, where p' is a pointer which may or may not equal to p. Since each $p \in P$ corresponds to a p', all p''s form a $(t+1)$-element sequence of pointers P' (not necessarily in $\{1,...,n\}$). That is, $\mathcal{A}$ makes $\rho'_p = f_{p'}(\mathsf{pw})$. Since UINIT, $sid = (S_1, ..., S_n)$ and U remain the same during the whole session, we can omit these and assume that A inputs P'.

When interacting with $\mathsf{S}_{\mathsf{Rec}}$, $\mathcal{A}$'s view is formed by $\mathcal{A}$'s ability to send (EVAL, sid, S_i, x) for any $i \in \{1,...,n\}$ and any $x \in D$ to $\mathcal{F}_{\text{PPSS}}$ in step 2 (that is, $\mathcal{A}$ computes $f_i(x)$). Here the output of $\mathsf{S}_{\mathsf{Rec}}$ is decided by $\mathcal{A}$'s inputs i and x.

Recall also that the security experiment depends on bit b: If $b = 0$, the output of $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ is a random string (provided that $\mathsf{U}_{\mathsf{Rec}}$ does not output $\perp$), while if $b = 1$, the output of $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ is the output of $\mathsf{U}_{\mathsf{Rec}}$. Below we only describe the case where $b = 1$, because for the $b = 0$ case the security game is a simple modification of the $b = 1$ game.

Below we define a series of security games starting with game $\mathbf{G}_0$ which is identical to the security game the adversary $\mathcal{A}$ interacts with in the case $b = 1$. In each game we define the following events:

- COMCOR: On some interaction with $\mathsf{U}_{\mathsf{Rec}}$,
$$C' = \mathsf{COM}((\mathsf{pw}, \boldsymbol{e'}, \boldsymbol{s'}); r'),$$
where $\boldsymbol{s'}$ and r' are values recovered by $\mathsf{U}_{\mathsf{Rec}}$ in the games.
- E_H: On some interaction with $\mathsf{U}_{\mathsf{Rec}}$ or $\mathsf{S}_{\mathsf{Rec}}$, $\mathcal{A}$ queries s to the random oracle $H(\cdot)$.
- E_S: On some interaction with $\mathsf{S}_{\mathsf{Rec}}$, $\mathcal{A}$'s input i is the pointer of some honest server, and $x = \mathsf{pw}$. (That is, $\mathcal{A}$ computes $f_i(\mathsf{pw})$.)
- E_U: On some interaction with $\mathsf{U}_{\mathsf{Rec}}$,
$$(C' \neq C \vee P' \neq Q|P) \wedge \text{COMCOR}.$$
(In the security experiment, this means that $\mathcal{A}$ sends something other than ω in step 1 of Rec, or changes some pointers on which f of pw is computed as in Init, but U's check passes.)

Let $E = E_H \vee E_S \vee E_U$.

Let $\mathbf{G}_0$ be the security experiment. Let $\mathbf{G}_1$ be a modification of $\mathbf{G}_0$, where e_i and s_i ($i \in \{1,...,n\}$) are generated as follows: first choose all e_i's at random. For $b \in B$, set $s_b := e_b \oplus f_q(\mathsf{pw})$, where q is the pointer in Q which corresponds to b. Then compute s_g ($g \notin B$) by polynomial interpolation using s and s_b ($b \in B$). Finally set $f_g(\mathsf{pw}) := e_g \oplus s_g$. By the one-time pad argument and by the fact that adversary controls up to t values $f_i(\mathsf{pw})$ it follows that $\mathbf{G}_1$ is identical to $\mathbf{G}_0$.

Initialization:
1. Pick $\mathsf{pw} \leftarrow_{\text{R}} D$.
2. Pick $s \leftarrow_{\text{R}} \mathbb{F}$.
3. Parse $H(s)$ as $[r||K]$.
4. Generate $(s_1, ..., s_n)$ as a (t, n) Shamir's secret-sharing of s over $\mathbb{F}$. Set $\boldsymbol{s} := (s_1, ..., s_n)$.
5. $\mathcal{A}$ inputs a t-element set of pointers Q. Compute $\rho_g := f_g(\mathsf{pw})$ ($g \notin B$) and $\rho_b := f_q(\mathsf{pw})$ ($b \in B$), where q is the pointer in Q which corresponds to i and each $f_i(\cdot)$ is a random function onto $\{0,1\}^\ell$
6. Set $e_i := s_i \oplus \rho_i$ ($i \in [n]$) and $\boldsymbol{e} := (e_1, ..., e_n)$.
7. Compute $C := \mathsf{COM}((\mathsf{pw}, \boldsymbol{e}, \boldsymbol{s}); r)$.
8. Send $\omega := (\boldsymbol{e}, C)$ to $\mathcal{A}$.

Reconstruction (repeated q_U times)
1. $\mathcal{A}$ inputs $\omega' = (\boldsymbol{e'}, C')$, a sequence P in $[n]$ and a sequence P' such that $|P| = |P'| = t + 1$.
2. For each $p \in P$, compute $\rho'_p := f_{p'}(\mathsf{pw})$ where p' is a pointer in P' corresponding to p.
3. Compute $s'_p := e'_p \oplus \rho'_p$.
4. Recover s' using the $t+1$ shares s'_p ($p \in P$).
5. Parse $H(s')$ as $[r'||K']$.
6. Compute s'_q ($q \in \{1,...,n\} - P$) by polynomial interpolation using the values s' and s'_p ($p \in P$). Set $\boldsymbol{s'} := (s'_1, ..., s'_n)$.
7. If $C' = \mathsf{COM}((\mathsf{pw}, \boldsymbol{e'}, \boldsymbol{s'}); r')$, then output K'. Otherwise reject (i.e. output $\perp$).

Figure 8. The Security Experiment $\mathbf{G}_0$.

Let $\mathbf{G}'_1$ be a modification of $\mathbf{G}_1$, where in step 5 of Rec, $[r'||K']$ is defined as $[r||K]$ if $s' = s$ and $H(s')$ otherwise. Since $H(s) = [r||K]$ according to step 3 of Init, $\mathbf{G}'_1$ is simply a change of notion of $\mathbf{G}_1$, and they are unconditionally identical.

Let $\mathbf{G}_2$ be a modification of $\mathbf{G}'_1$, where in step 3 of Init, $[r||K]$ is picked as a random string, and the game challenger aborts if $\mathcal{A}$ queries s on $H(\cdot)$.

We can see that if E_H does not occur, $\mathbf{G}_2$ and $\mathbf{G}'_1$ are identical. Therefore, we have

$$\Pr[E_H||\mathbf{G}_2] = \Pr[E_H||\mathbf{G}'_1].$$

Let $\mathbf{G}_3$ be a modification of $\mathbf{G}_2$, which outputs K generated in Init if $\omega' = \omega$ and $P' = Q|P$, and rejects

otherwise. However, the game challenger still computes all the values computed in $\mathbf{G}_2$. Note that

$$\neg E_U = (\omega' = \omega \wedge P' = Q|P) \vee \neg\textsc{ComCor}.$$

Moreover,

- If $\omega' = \omega \wedge P' = Q|P$, then both $\mathbf{G}_2$ and $\mathbf{G}_3$ output K;
- If $\neg(\omega' = \omega \wedge P' = Q|P) \wedge \neg\textsc{ComCor}$, then both $\mathbf{G}_2$ and $\mathbf{G}_3$ reject.

Thus, $\mathbf{G}_3$ is identical to $\mathbf{G}_2$ assuming E_U does not occur. Therefore, we have

$$\Pr[E_U||\mathbf{G}_3] = \Pr[E_U||\mathbf{G}_2].$$

Let $\mathbf{G}_4$ be exactly the same with $\mathbf{G}_3$, except that the game challenger outputs a random string if $\mathcal{A}$ queries $f_g(\mathsf{pw})$ where g is the pointer of an honest server (i.e. E_S occurs). Obviously, if E_S does not occur, $\mathbf{G}_4$ and $\mathbf{G}_3$ are identical. Therefore, we have

$$\Pr[E_S||\mathbf{G}_4] = \Pr[E_S||\mathbf{G}_3].$$

Note that $\mathbf{G}_4$'s output K is independently random of everything else. Therefore, we have

$$\Pr[b' = b||\mathbf{G}_4] = 1/2.$$

Furthermore, $\mathbf{G}_0$ and $\mathbf{G}_4$ are identical if E does not happen. Besides $\mathcal{A}$'s input and output, the event E also involves pw (in E_S) and s (in E_H and E_U). We can see that $\mathcal{A}$'s view of pw does not change from $\mathbf{G}_0$ to $\mathbf{G}_4$; $\mathcal{A}$'s view of s does not change from $\mathbf{G}_0$ to $\mathbf{G}_3$, and does not change from $\mathbf{G}_3$ to $\mathbf{G}_4$ assuming E_S does not occur. Thus, we have

$$\begin{aligned}\Pr[E_S||\mathbf{G}_0] &= \Pr[E_S||\mathbf{G}_4],\\ \Pr[E_H \wedge \neg E_S||\mathbf{G}_0] &= \Pr[E_H \wedge \neg E_S||\mathbf{G}_4],\\ \Pr[E_U \wedge \neg E_S||\mathbf{G}_0] &= \Pr[E_U \wedge \neg E_S||\mathbf{G}_4].\end{aligned}$$

This implies

$$\begin{aligned}\Pr[E||\mathbf{G}_0] &= \Pr[E||\mathbf{G}_4],\\ \Pr[b' = b \wedge \neg E||\mathbf{G}_0] &= \Pr[b' = b \wedge \neg E||\mathbf{G}_4].\end{aligned}$$

Now we upper bound $\Pr[E_U||\mathbf{G}_4]$. We break E_U into three sub-events:

- E_{U1}: $C' \neq C \wedge \textsc{ComCor}$.
  We will argue that
  $$\Pr[E_{U1} \wedge \neg E_S||\mathbf{G}_4] \leq q_U(1/|D| + \epsilon_{NM}).$$
  Note that E_{U1} may occur at $\mathcal{A}$'s jth query to $\mathsf{U}_{\mathsf{Rec}}$ for any $j = 1, ..., q_U$. Let E^j_{U1} be the event that E_{U1} happens at $\mathcal{A}$'s jth query to $\mathsf{U}_{\mathsf{Rec}}$. Obviously $\Pr[E_{U1} \wedge \neg E_S||\mathbf{G}_4] \leq \sum_{j=1}^{q_U} \Pr[E^j_{U1} \wedge \neg E_S||\mathbf{G}_4]$.
  We will argue that
  $$\Pr[E^j_{U1} \wedge \neg E_S||\mathbf{G}_4] \leq 1/|D| + \epsilon_{NM}$$
  for any $j = 1, ..., q_U$ by constructing a reduction to COM's non-malleability property. The reduction picks e_i's and $\rho_p(x)$'s for $p \in B$ at random, receives C and sends $(\boldsymbol{e}, C)$ to $\mathcal{A}$. At $\mathcal{A}$'s first $j-1$ queries to $\mathsf{U}_{\mathsf{Rec}}$, the reduction simply processes as the game challenger in Rec. At $\mathcal{A}$'s jth query to $\mathsf{U}_{\mathsf{Rec}}$, the reduction receives pw, $\boldsymbol{e}$, $\boldsymbol{s}$ and r, then computes and outputs the corresponding $\boldsymbol{e'}$, $\boldsymbol{s'}$ and r' using $\rho'_p(\mathsf{pw})$ for $p \in B$.
- E_{U2}: $C' = C \wedge (\boldsymbol{e'}, \boldsymbol{s'}) \neq (\boldsymbol{e}, \boldsymbol{s}) \wedge \textsc{ComCor}$.
  We will argue that
  $$\Pr[E_{U2} \wedge \neg E_S||\mathbf{G}_4] \leq \epsilon_B$$
  by constructing a reduction to COM's binding property . In Init, the reduction proceeds as the game challenger. In Rec, it computes $\boldsymbol{s'}$ and r' accordingly, and outputs $(C, (\mathsf{pw}, \boldsymbol{e}), (\mathsf{pw}, \boldsymbol{e'}), r, r')$ once $C' = C = \mathsf{COM}((\mathsf{pw}, \boldsymbol{e'}, \boldsymbol{s'}); r')$ holds.
- E_{U3}: $C' = C \wedge (\boldsymbol{e'}, \boldsymbol{s'}) = (\boldsymbol{e}, \boldsymbol{s}) \wedge P' \neq Q|P \wedge \textsc{ComCor}$. Since $P' \neq Q|P$, there is at least one element $p' \in P'$ and one element $p \in Q|P$ that are different. However, we have $s'_p = s_p$, $s'_{p'} = e'_{p'} \oplus f_{p'}(\mathsf{pw})$, $s_p = e_p \oplus f_q(\mathsf{pw})$, and $e'_p = e_p$. Therefore, $f_{p'}(\mathsf{pw}) = f_p(\mathsf{pw})$. Since $f_{p'}(\mathsf{pw})$ and $f_p(\mathsf{pw})$ are both random strings in $\{0,1\}^\ell$, the probability that this occurs in one query is $1/2^\ell$, which implies that $\Pr[E_{U3}||\mathbf{G}_4] \leq q_U/2^\ell$.

In sum, we get

$$\Pr[E_U||\mathbf{G}_4] \leq \epsilon_B + q_U(1/|D| + \epsilon_{NM}) + 1/2^\ell.$$

Let $\mathbf{G}_5$ be a modification of $\mathbf{G}_4$, s.t. C is set as $\mathsf{COM}((0, \boldsymbol{e}, \boldsymbol{0}); r)$ instead of $\mathsf{COM}((\mathsf{pw}, \boldsymbol{e}, \boldsymbol{s}); r)$. Since s and pw are independently random from everything else in $\mathbf{G}_5$, we have

$$\begin{aligned}\Pr[E_H||\mathbf{G}_5] &\leq q_H/2^\ell,\\ \Pr[E_S||\mathbf{G}_5] &\leq q_S/|D|.\end{aligned}$$

Furthermore, an easy reduction shows that

$$\begin{aligned}\Pr[E_H||\mathbf{G}_4] &\leq \Pr[E_H||\mathbf{G}_5] + \epsilon_H,\\ \Pr[E_S||\mathbf{G}_4] &\leq \Pr[E_S||\mathbf{G}_5] + \epsilon_H.\end{aligned}$$

Therefore,

$$\begin{aligned}\Pr[E||\mathbf{G}_4] &= Pr[E_U \vee E_H \vee E_S||\mathbf{G}_4]\\ &\leq \Pr[E_U||\mathbf{G}_4] + \Pr[E_H||\mathbf{G}_4] + \Pr[E_S||\mathbf{G}_4]\\ &= (q_U + q_S)/|D| + \epsilon.\end{aligned}$$

Combining all the results we get above, we conclude

$$\begin{aligned}\Pr[b'{=}b||\mathbf{G}_0] &\leq \Pr[E||\mathbf{G}_0] + \Pr[b'{=}b \wedge \neg E||\mathbf{G}_0]\\ &= \Pr[E||\mathbf{G}_4] + \Pr[b' = b \wedge \neg E||\mathbf{G}_4]\\ &= \Pr[E||\mathbf{G}_4] + \Pr[b' = b||\mathbf{G}_4]\Pr[\neg E||\mathbf{G}_4]\\ &= \Pr[E||\mathbf{G}_4] + 1/2(1 - \Pr[E||\mathbf{G}_4])\\ &= 1/2 + 1/2\Pr[E||\mathbf{G}_4]\\ &= 1/2[1 + (q_U + q_S)/|D| + \epsilon],\end{aligned}$$

which implies the theorem. □

Set $\mathsf{tx}(S) := 0$, $T(S,x) := \perp$, $\mathsf{tested}(\mathsf{pw}) := \varnothing$, $\mathsf{pointer}(i) := \perp$, and $\mathsf{pointer}(ssid, j) := \perp$, for all values $S, x, \mathsf{pw}, i, ssid, j$. Generate $(\mathsf{COM}, \mathsf{td})$ using equivocable commitment generator $\mathsf{tdCKG}(1^\ell)$.

1) On $(\text{INIT}, sid, U, \mathcal{SI})$ from $\mathcal{F}_{\text{PPSS}}$, for all $i \in [n]$ set $S_i := \mathcal{SI}[i]$ and send $(\text{EVAL}, (sid, i), U, S_i)$ to $\mathcal{A}$. If $\mathcal{F}_{\text{PPSS}}$ in addition sends (pw, K), record it.

2) On $(\text{SNDRCOMPLETE}, (sid, i), S)$ from $\mathcal{A}$, set $\mathsf{tx}(S)$++, record (i, S).

3) On $(\text{RCVCOMPLETE}, (sid, i), U, S_i, S_i^*)$ from $\mathcal{A}$, ignore this message if S_i is honest and $S_i^* \neq S_i$, or $\mathsf{tx}(S_i^*) = 0$. Otherwise set $\mathsf{tx}(S_i^*)$−− and $\mathsf{pointer}(i) := S_i^*$. If $\mathsf{pointer}(i) \neq \perp$ for all $i \in [n]$ then also do:

   If $\mathcal{F}_{\text{PPSS}}$ sent (K, pw) in step 1 then for all $i \in [n]$ set $\rho_i := T(\mathsf{pointer}(i), \mathsf{pw})$ (assign it to a random value in $\mathbb{F}$ if undefined), pick $\boldsymbol{s}$ by evaluating a random t-degree polynomial over $\mathbb{F}$, set $\boldsymbol{e} := \boldsymbol{s} \oplus \boldsymbol{\rho}$, pick $r \leftarrow \{0,1\}^\ell$, define $H(s) := [r||K]$, set $C := \mathsf{COM}((\mathsf{pw}, \boldsymbol{e}, \boldsymbol{s}); r)$. Otherwise, i.e. if (K, pw) wasn't sent in step 1, pick $\boldsymbol{e} \leftarrow_{\text{R}} \mathbb{F}^n$ and set $(C, \mathsf{td}_C) \leftarrow \mathsf{tdCom}(\mathsf{td})$. In either case, set $\omega := (\boldsymbol{e}, C)$ and send $(\text{SEND}, (sid, i, 0), U, \mathcal{SI}[i], \omega)$ to $\mathcal{A}$ for all $i \in [n]$.

4) On $(\text{SENT}, (sid, i, 0), U, S, \omega)$ from $\mathcal{A}$, ignore this message if (i, S) hasn't been recorded in step 2. Otherwise send $(\text{SEND}, (sid, i), S, U, \text{ACK})$ to $\mathcal{A}$ and (SINIT, sid, S) to $\mathcal{F}_{\text{PPSS}}$.

5) On $(\text{SENT}, (sid, i, 1), S, U, \text{ACK})$ from $\mathcal{A}$, ignore message if ω wasn't created in step 3 or if $S \neq \mathcal{SI}[i]$. Otherwise mark i as ack'ed. If all $i \in [n]$ are ack'ed send (UINIT, sid) to $\mathcal{F}_{\text{PPSS}}$.

6) On $(\text{REC}, sid, ssid, U', \mathcal{SR})$ from $\mathcal{F}_{\text{PPSS}}$, send $(\text{EVAL}, (sid, ssid, j), U, S_j)$ to $\mathcal{A}$ for all $S_j \in \mathcal{SR}$.

7) On $(\text{SNDRCOMPLETE}, (sid, ssid, \cdot), S)$ from $\mathcal{A}$, set $\mathsf{tx}(S)$++, send $(\text{SREC}, sid, ssid, S)$ to $\mathcal{F}_{\text{PPSS}}$.

8) On $(\text{RCVCOMPLETE}, (sid, ssid, j), U, S_j, S_j^*)$ *and* $(\text{SENT}, (sid, ssid, j), S_j, U, (i_j, \omega_j))$ from $\mathcal{A}$, ignore this message if S_j doesn't match S_j in the EVAL query which SIM sent for this $(ssid, j)$ in step 6, or if S_j is honest and $S_j^* \neq S_j$, or if $\mathsf{tx}(S_j^*) = 0$. Otherwise set $\mathsf{tx}(S_j^*)$−− and set $\mathsf{pointer}(ssid, j) := S_j^*$.

   If $\mathsf{pointer}(ssid, j) \neq \perp$ for all $j \in [t+1]$ then set $\mathcal{SC}$ as the set of server-name pointers $\mathsf{pointer}(ssid, j)$ for $j \in [t+1]$ and send $(\text{UREC}, sid, ssid, \mathcal{SC}, \mathsf{flag}, \mathsf{pw}^*, K^*)$ to $\mathcal{F}_{\text{PPSS}}$ for $(\mathsf{flag}, \mathsf{pw}^*, K^*)$ set as follows:

   a) If $i_{j_1} = i_{j_2}$ or $\omega_{j_1} \neq \omega_{j_2}$ for any $j_1 \neq j_2$ then set $(\mathsf{flag}, \mathsf{pw}^*, K^*) := (0, \perp, \perp)$.

   b) If $\omega_j = \omega$ and $\mathsf{pointer}(ssid, j) = \mathsf{pointer}(i_j)$ for all $j \in [t+1]$ where ω is the value created by SIM in step 3 then set $(\mathsf{flag}, \mathsf{pw}^*, K^*) := (1, \perp, \perp)$.

   c) Define X as a set of values x s.t. $T(\mathsf{pointer}(ssid, j), x) \neq \perp$ for all $j \in [t+1]$. For each $x \in X$ do the following subprocedure:

      Set $\rho'_{i_j} := T(\mathsf{pointer}(ssid, j), x)$ for all $j \in [t+1]$, and set $I := \{i_j \mid j \in [t+1]\}$, and parse any ω_j as $(\boldsymbol{e}', C')$, set $s'_i := e'_i \oplus \rho'_i$ for all $i \in I$, interpolate $(s', \{s'_i\}_{i \notin I})$ from $\{(i, s'_i)\}_{i \in I}$, set $[r'||K'] := H(s')$. Finally, if $C' = \mathsf{COM}((\mathsf{pw}', \boldsymbol{e}', \boldsymbol{s}'); r')$ then set $(\mathsf{flag}, \mathsf{pw}^*, K^*) := (2, x, K')$ (and break the loop).

   If above loop processes all $x \in X$ without breaking, set $(\mathsf{flag}, \mathsf{pw}^*, K^*) := (0, \perp, \perp)$.

9) On (EVAL, sid, S, x) from corrupt U^* (or $\mathcal{A}$) and $(\text{RCVCOMPLETE}, sid, U^*, S, S^*)$ from $\mathcal{A}$, ignore this message if S is honest and $S^* \neq S$ or if $\mathsf{tx}(S^*) = 0$. Otherwise set $\mathsf{tx}(S^*)$−−, add S^* to $\mathsf{tested}(x)$, send $(\text{TESTPWD}, sid, S^*, x)$ to $\mathcal{F}_{\text{PPSS}}$, and given reply RES do the following:

   Case 1: If RES = FAIL then send $(\text{EVAL}, T(S^*, x))$ to $\mathcal{A}$. (If $T(S^*, x)$ is undefined pick $T(S^*, x) \leftarrow_{\text{R}} \mathbb{F}$.)

   Case 2: If RES $= K$ then set $\rho_i := T(S_i, \mathsf{pw})$ and $I := I \cup \{i\}$ for all $i \in [n]$ s.t. $\mathsf{pointer}(i) = S_i$ and $S_i \in \mathsf{tested}(\mathsf{pw})$. Set $s_i = e_i \oplus \rho_i$ for all $i \in I$. Interpolate $(s, \{s_i\}_{i \notin I})$ from $\{(i, s_i)\}_{i \in I}$, set $r \leftarrow \mathsf{Equiv}(\mathsf{td}_C, (\mathsf{pw}, \boldsymbol{e}, \boldsymbol{s}))$, $H(s) := [r||K]$, and set $T(S_i, x) := s_i \oplus e_i$ for al $i \notin I$.

Figure 9. Simulator SIM for showing that protocol π_{PPSS} realizes functionality $\mathcal{F}_{\text{PPSS}}$.

5.2. UC Security of the Proposed PPSS Scheme

Protocol π_{PPSS} in Figure 6 satisfies not only the game-based PPSS security definition but it also realizes the ideal PPSS functionality proposed in Section 4.2:

Theorem 3. *If* COM *is an equivocable and non-malleable commitment scheme and* H *is a random oracle then the PPSS scheme* π_{PPSS} *of Figure 6 UC-realizes PPSS functionality* $\mathcal{F}_{\mathrm{PPSS}}$ *in the* $(\mathcal{F}_{\mathrm{OPRF}}, \mathcal{F}_{\mathrm{AUTH}})$*-hybrid world.*

The full proof of theorem 3 below will appear in the full version of this paper, and here we will only sketch this security argument. Here we only exhibit a construction of a simulator SIM, presented in Figure 9, which we use to show that protocol π_{PPSS} realizes functionality $\mathcal{F}_{\mathrm{PPSS}}$ given $\mathcal{F}_{\mathrm{OPRF}}$, $\mathcal{F}_{\mathrm{AUTH}}$, an equivocable commitment scheme COM and a random oracle model for hash function H. We note that the requirement on the commitment scheme are stronger in theorem 3 than in theorem 2, and include an additional property of equivocability. In a nutshell, a commitment scheme is *equivocable* if the standard commitment procedures are amended by a triple of algorithms $(\mathsf{tdCKG}, \mathsf{tdCom}, \mathsf{Equiv})$, where $\mathsf{tdCKG}(1^\ell)$ creates an instance COM of the commitment scheme together with an *equivocation trapdoor* td, $\mathsf{tdCom}(\mathsf{td})$ creates a fake commitment C together with C-specific equivocation trapdoor td_C s.t. for any m procedure $\mathsf{Equiv}(\mathsf{td}_c, m)$ generates string r s.t. r is a decommitment of C to m. Simulator SIM in Figure 9 uses algorithm tdCKG to create an instance of the commitment scheme together with an equivocation trapdoor, and it uses this trapdoor to create C in the $\omega = (\boldsymbol{e}, C)$ public parameters created in the INIT procedure as a *fake commitment*. The reason SIM needs to create commitment C in this way is because SIM does not know the real password pw and secret key K which $\mathcal{F}_{\mathrm{PPSS}}$ created on behalf of the honest user in this PPSS instance, but when the adversary $\mathcal{A}$ playing the role of a corrupt user U^* performs the PPSS reconstruction with $t+1$ honest servers (see the last step in Figure 9) on some password guess x, SIM forwards this x to $\mathcal{F}_{\mathrm{PPSS}}$ via the TESTPWD interface and if it turns out that $\mathcal{A}$'s guess is correct, i.e. $x = \mathsf{pw}$, then SIM has to "open" the committed public parameter ω into a valid reconstruction of K on password pw. Since pw is part of the information committed in C, this is where SIM uses the ability to equivocate the commitment by opening it to the correct value.

Acknowledgments. Stanislaw Jarecki is supported by NSF CICI award, #1547435. Aggelos Kiayias is supported by ERC project CODAMODA, #259152. Hugo Krawczyk is supported by ONR Contract N00014-14-C-0113. This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and NSF grant #CNS-1523467.

References

[1] M. Abe, M. Okhubo. A Framework for Universally Composable Non-committing Blind Signatures. In *Advances in Cryptology - ASIACRYPT 2009*, pp. 435–450, 2009.

[2] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. Password-protected secret sharing. In *ACM Conference on Computer and Communications Security*, pp. 433-444, 2011.

[3] R. Canetti, Universally Composable Security: A New Paradigm for Cryptographic Protocols. In Foundations of Computer Science, FOCS 2001, pp. 136-145.

[4] 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, pp. 256–275.

[5] J. Camenisch, A. Lehmann, and G. Neven. Optimal Distributed Password Verification, in Computer and Communications Security, CCS'2015, pp. 182–194.

[6] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally Composable Password-Based Key Exchange, in Advances in Cryptology – EUROCRYPT'2005, pp. 404–421.

[7] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for Boolean queries. In Crypto'2013. https://eprint.iacr.org/2013/169

[8] A. Everspaugh, R. Chatterjee, S. Scott, A. Juels, and T. Ristenpart. The Pythia PRF Service. In USENIX Security Symposium'2015. https://eprint.iacr.org/2015/644

[9] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In Theory of Cryptography Conference, TCC'2005.

[10] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In Theory of Cryptography Conference, TCC'2008, pp. 155–175.

[11] S. Jarecki, A. Kiayias, and H. Krawczyk. Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model. In Asiacrypt'2014, pp. 233–253. https://eprint.iacr.org/2014/650

[12] S. Jarecki, H. Krawczyk, M. Shirvanian and N. Saxena, Device-Enhanced Password Protocols with Optimal Online-Offline Protection. https://eprint.iacr.org/2015/1099

[13] S. Jarecki, X. Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In Theory Of Cryptography Conference, TCC'2009, pp. 577–594.

[14] S. Jarecki and X. Liu. Fast secure computation of set intersection. In Security and Cryptography for Networks Conference, SCN'2010, pp.418–435.

[15] P. D. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. In *J. Cryptology*, 19(1):27–66, 2006.

[16] New York Times. Russian Hackers Amass Over a Billion Internet Passwords, 08-06-2014. Available at: http://goo.gl/aXzqj8.

[17] M. Shirvanian, S. Jarecki, N. Saxena, and N. Nathan. Two-factor authentication resilient to server compromise using mix-bandwidth devices. In The Network and Distributed System Security Symposium, NDSS'2014.