TOPPSS: Cost-Minimal Password-Protected Secret Sharing Based on Threshold OPRF

Stanisław Jarecki[1(✉)], Aggelos Kiayias[2], Hugo Krawczyk[3], and Jiayu Xu[1]

[1] University of California, Irvine, USA
stasio@ics.uci.edu, jiayux@uci.edu
[2] University of Edinburgh, Edinburgh, UK
aggelos@kiayias.com
[3] IBM Research, New York City, USA
hugo@ee.technion.ac.il

Abstract. We present TOPPSS, the most efficient Password-Protected Secret Sharing (PPSS) scheme to date. A (t, n)-threshold PPSS, introduced by Bagherzandi et al. [4], allows a user to share a secret among n servers so that the secret can later be reconstructed by the user from any subset of $t + 1$ servers with the sole knowledge of a password. It is guaranteed that any coalition of up to t corrupt servers learns nothing about the secret (or the password). In addition to providing strong protection to secrets stored online, PPSS schemes give rise to efficient Threshold PAKE (T-PAKE) protocols that armor single-server password authentication against the inherent vulnerability to offline dictionary attacks in case of server compromise.

TOPPSS is *password-only*, i.e. it does not rely on public keys in reconstruction, and enjoys remarkable efficiency: A single communication round, a single exponentiation per server and just two exponentiations per client regardless of the number of servers. TOPPSS satisfies threshold security under the (Gap) One-More Diffie-Hellman (OMDH) assumption in the random-oracle model as in prior efficient realizations of PPSS/T-PAKE [18,19]. Moreover, we show that TOPPSS realizes the *Universally Composable* PPSS notion of [19] under a generalization of OMDH, the *Threshold* One-More Diffie-Hellman (T-OMDH) assumption. We show that the T-OMDH and OMDH assumptions are both hard in the generic group model.

The key technical tool we introduce is a universally composable *Threshold Oblivious PRF* which is of independent interest and applicability.

1 Introduction

Passwords have well-known weaknesses as authentication tokens, foremost because of their vulnerability to offline dictionary attacks in case of the

S. Jarecki—Supported by NSF grant CNS-1547435.

A. Kiayias—Supported by ERC project CODAMODA and H2020 Project Panoramix, #653497.

H. Krawczyk—Supported by ONR Contract N00014-14-C-0113.

D. Gollmann et al. (Eds.): ACNS 2017, LNCS 10355, pp. 39–58, 2017.
DOI: 10.1007/978-3-319-61204-1_3

all-too-common leakage of the database of password hashes stored by the authentication server (see e.g., [1]). Worse still, most people re-use their passwords across multiple services, hence a break-in into one service effectively breaks the security of others. Yet, because of their convenience, passwords are a dominant form of authentication, and the amount and value of information protected using passwords keeps growing. Defenses such as the use of secondary authentication factors (e.g., a PIN generated by a personal device or a USB dongle) increase protection against on-line attacks but not against offline attacks upon server compromise. Techniques such as Password Authenticated Key Exchange (PAKE) [6,8] improve on today's de-facto standard of "password over TLS" authentication by eliminating the reliance on a Public Key Infrastructure (PKI), but they do not help against offline attacks after server compromise.

T-PAKE and PPSS. To address the threat of offline dictionary attacks on the server, Mackenzie et al. [26] introduced (t, n)-Threshold PAKE (T-PAKE), which replaces a single authentication server with a group of n servers and leaks no information on passwords even if up to t servers are corrupted. Bagherzandi et al. [4] proposed a related notion of Password-Protected Secret Sharing (PPSS) which simplifies the notion of T-PAKE by reducing the goal of key exchange between user and servers to that of the user retrieving a single secret previously shared with the servers. Specifically, a (t, n)-PPSS scheme, as formulated in the PKI-free setting by [18], allows a user to share a random secret s among n servers under the protection of her password pw s.t. (1) a reconstruction protocol involving at least $t+1$ honest servers recovers s if the user inputs the (correct) password pw; (2) the compromise of up to t servers leaks no information about either s or pw; (3) an adversary who corrupts $t' \leq t$ servers and has q_U interactions with the user and q_S interactions with the uncorrupted servers can test at most $\frac{q_S}{t-t'+1} + q_U$ passwords. (In the PKI setting one can set $q_U = 0$.)

The PPSS notion is useful in the design of efficient T-PAKE's because of the low-overhead generic PPSS-to-TPAKE compiler [4,18]. It is also an important primitive in its own right, allowing for online storage of sensitive information like keys, credentials, or personal records, with availability and privacy protection. The only token needed for retrieving stored information is a *single* password, and both information and password remain private if no more than t servers are compromised (and if the adversary does not guess or learn the password).

In this paper we present TOPPSS, the most efficient PPSS scheme to date – and using the PPSS-to-TPAKE compiler of [18] also the most efficient T-PAKE – with a hard-to-beat complexity as detailed below. Our work builds on the works of Jarecki et al. [18,19] who constructed PPSS protocols based on *Oblivious Pseudorandom Functions (OPRF)*, formulated as a universally composable (UC) functionality. The works of [18,19] define UC OPRF differently, but each instantiates its OPRF notion using the *blinded Diffie-Hellman* technique, following Ford and Kaliski [15], under the so-called (Gap) *One-More Diffie-Hellman* (OMDH) assumption [5,22] in the Random Oracle Model (ROM). Using one OPRF construction, [18] showed a PPSS whose reconstruction phase takes a single round between a user and $t+1$ servers, with 2 (multi)exponentiations per

server and $2t+3$ for the user. The PPSS of [19] uses a simplified OPRF scheme secure under the same assumptions, with 1 exponentiation per server and $t+2$ for the user. In addition to improving on [18] in efficiency, the latter scheme satisfies a stronger PPSS notion formulated as a UC functionality, which we adopt here.

Our Contributions. We present TOPPSS, a simple PPSS protocol with remarkable and hard to beat performance. The reconstruction procedure requires *just one exponentiation per server and a total of two exponentiations for the user* (independent of the number of servers), plus $O(t)$ modular multiplications by each party. Communication is also optimal: The user sends a single group element to a subset of $t+1$ servers and gets one group element from each server. Furthermore, we show that this "minimal cost" (and PKI-free) PPSS satisfies the strong UC notion of PPSS from [19]. This contribution is based on the observation that a more efficient PPSS can result from replacing the OPRF used in the protocols of [18,19] with its *threshold* (or multi-party) counterpart which we define as *Threshold OPRF (T-OPRF)*. We provide a UC definition of T-OPRF as a functionality that allows a group of servers to secret-share a key k for PRF f with a shared PRF evaluation protocol which lets the user compute $f_k(x)$ on her input x, s.t. both x and k are secret if no more than t of n servers are corrupted. T-OPRF is an input-oblivious strengthening of Distributed PRF (DPRF) of Naor et al. [27], hence in particular T-OPRF can replace DPRF in all its applications, e.g. for corruption-resilient Key Distribution Center, and long-term information protection (see [27]).

Using this strong notion of T-OPRF security we show a compiler which transforms UC T-OPRF into UC PPSS at negligible additional cost (in ROM). In particular, TOPPSS is obtained by designing a T-OPRF protocol, denoted 2HashTDH, with the efficiency parameters stated above. This T-OPRF protocol is essentially a "threshold exponentiation" protocol, where each server computes m^{k_i} on input m where k_i is the server's secret-share of the PRF key k. We prove that TOPPSS realizes UC T-OPRF under the following assumptions in ROM. Let $t' \leq t$ denote the number of parties actually controlled by an attacker. First, our results imply that in the so-called *full corruption* case, i.e. if $t' = t$, the same (Gap) OMDH assumption used in [18,19] implies that the attacker must query one uncorrupted party per each input on which the attacker wants to obtain the function value. Since this is the case when the attacker controls the full threshold t of servers it is also the case for any $t' < t$. In the application to PPSS this means that the attacker can test up to $q_S + q_U$ passwords, which matches the $\frac{q_S}{t-t'+1} + q_U$ bound for $t' = t$. Since many existing works on T-PAKE, e.g. [2,9,14,23,26,31], implicitly assume the $t' = t$ case by defining security using the simplified $q_S + q_U$ bound on the number of passwords the adversary can test, we call this level of security a *standard threshold security* for T-PAKE/PPSS.

Secondly, for the general case of $t' \leq t$, we show that TOPPSS achieves the stronger $\frac{q_S}{t-t'+1} + q_U$ bound assuming a generalization of the OMDH assumption which we call (Gap) *Threshold One-More Diffie-Hellman (T-OMDH)*. As a sanity check for the T-OMDH assumption we show that the T-OMDH problem is

hard in the generic group model. Since OMDH is a special case of T-OMDH, to the best of our knowledge this is also the first generic group analysis of OMDH. The stricter bound implies that an adversary controlling $t' \leq t$ servers must contact $t - t' + 1$ uncorrupted servers for each input on which it wants to compute the function, which coincides with the *standard threshold security* notion when $t' = t$, but it is stronger for $t' < t$. For example, it means that the default network adversary who does corrupt any party but runs q sessions with each server, can test up to $qn/(t+1)$ passwords, whereas the *standard threshold security* would in this case upper-bound the number of tested passwords only by qn.

As a point of comparison in the full version of this paper [20] we consider a generic compiler from *any* OPRF to T-OPRF. This compiler performs multi-party computation of the server code in the underlying OPRF protocol, but in the case of the OPRF of [19] such MPC protocol has the same low computational cost as the customized T-OPRF protocol 2HashTDH discussed above, i.e. 1 exponentiation per server and 2 for the user, with the only drawback of adding an additional communication round to enforce an agreement between the servers on the client's input to the MPC protocol. On the other hand, since the security depends only on the base OPRF, the resultant two-round T-OPRF protocol achieves the $\frac{q_S}{t-t'+1} + q_U$ bound based solely on OMDH for all $t' \leq t$.

Other Applications. Oblivious PRFs have found multiple applications which can also enjoy the benefits of a threshold version, particularly given the remarkable efficiency of our schemes. Examples of such applications include search on encrypted data [13,17], set intersection [22], and multiple-server DE-PAKE (device enhanced PAKE) [21].

Related Work. The first (t, n)-Threshold PAKE (T-PAKE) by Mackenzie et al. [26] required ROM in the security analysis and relied on PKI, namely, it assumed that the client can validate the public keys of the servers *during the reconstruction phase*.[1] Gennaro and Raimondo [14] dispensed with ROM and PKI (in authentication) but increased protocol costs. Abdalla et al. [2] showed a PKI-free T-PAKE in ROM with fewer communication rounds than T-PAKE of [26] but the client establishes a key with only one designated *gateway* server. Yi et al. [31] showed a similar round-reduction without ROM. The case of $n = 2$ servers, known as 2-PAKE, received special attention starting with Brainard et al. [9,29] on 2-PAKE in ROM and PKI, and several works [7,23–25] addressed the non-PKI and no-ROM case. Still, each of these T-PAKE schemes requires server-to-server communication. If communication is mediated by the client then the lowest round complexity is 3 for $n > 2$ [2] and 2 for $n = 2$ [7,25].

Bagherzandi et al. [4] introduced the notion of Password-Protected Secret Sharing (PPSS) with the goal of simplifying T-PAKE protocols. Specifically,

[1] When we say that PPSS/T-PAKE assumes PKI we mean that it relies on it for the security of the reconstruction/authentication phase. By contrast, the *initialization phase* of any PPSS/T-PAKE solution must assume some trust infrastructure, e.g. PKI, or otherwise each party could be initializing the scheme with an impostor.

they showed a PPSS protocol in ROM assuming PKI, with 2 rounds, constant-sized messages, and $8(t+1)$ (multi) exponentiations per client, and a low-cost PKI-model compiler from PPSS to T-PAKE. Camenisch et al. [10] constructed another PPSS scheme, called T-PASS, for *Threshold Password-Authenticated Secret Sharing*, without assuming PKI but with $14n$ exponentiations for the client, 7 exponentiations per server, and 5 rounds of communication.

Jarecki et al. [18,19] showed significantly faster PPSS protocols, also without assuming PKI (in reconstruction): The PPSS of [18] takes a single round (two messages) between a user and each server, and uses 2 (multi) exponentiations per server and $2t+3$ (multi) exponentiations for the client, secure under (Gap) OMDH in ROM. (They also show a 4-message non-ROM PPSS with $O(n \cdot |\mathsf{pw}|)$ exponentiations using Paillier encryption.) The PPSS of [19] improves upon this with a single-round PPSS with 1 exponentiation per server and $t+2$ exponentiations for the client, also under OMDH in ROM. In related works, [11] showed a single-round *proactive* PPSS in the PKI setting for the case of $t = n$, and [3] showed general methods for ensuring robustness in PPSS reconstruction, and a non-ROM PPSS using $O(|\mathsf{pw}|)$ exponentiations in a prime-order group.

Another important aspect of these PPSS solutions is the type of security notion they achieve. Both the PKI-model PPSS notion of [4] and the PKI-free PPSS notion of [18] were indistinguishability-based, while [10,19] provided Universally Composable (UC) definitions of the PPSS functionality. The essence of the UC PPSS definition of [19], which we adopt here, is that the only attack the adversary can stage is the inevitable one, namely, an online dictionary attack where validating a single password guess requires interaction with either $t+1$ instances of the servers or with the user. The UC definitions have further advantages for a password-based notion like PPSS, e.g. they imply security in the presence of non-uniformly distributed passwords, correlated passwords used for different services, and password mistyping.

Organization. In Sect. 2 we define the fundamental tool TOPPSS relies on, namely T-OPRF, as a UC functionality. In Sect. 3 we show a single-round, 1exp/server + 2exp/client realization of T-OPRF, protocol 2HashTDH, secure in ROM under the Threshold OMDH assumption we introduce in that section. In Sect. 4 we show a low-cost compiler from T-OPRF to PPSS, which we exemplify in Sect. 5 with a concrete instantiation using 2HashTDH.

2 Universally Composable Threshold OPRF

Notation. We use ":=" for deterministic assignment, "$\leftarrow$" for randomized assignment, and "$\leftarrow_R$" for uniform sampling from some set.

The T-OPRF Functionality. In the introduction we gave an informal overview of the notion of Threshold Oblivious PRF (T-OPRF) and its applicability, e.g. to PPSS schemes. Here we provide a formal definition of this notion as a secure realization of the UC functionality $\mathcal{F}_{\text{TOPRF}}$ shown in Fig. 1 which

generalizes the single-server (non-verifiable) OPRF functionality of [19] to the multi-party setting. In the $\mathcal{F}_{\text{TOPRF}}$ setting, the PRF key is effectively controlled by a collection of n servers and it remains secret as long as no more than a threshold t of these servers are corrupted. Such (t, n)-threshold "collective control" over a functionality can be realized as we show in our 2HashTDH realization in Sect. 3. We chose to base the T-OPRF notion on the *non-verifiable OPRF* notion of [19] rather than the *verifiable OPRF* notion of [18] because the former was shown to have a more efficient realization under the same assumptions, and because this form of OPRF suffices in the key application of interest to us, namely, Password-Protected Secret-Sharing.

Assume $\mathsf{tx}(p, S)$ and $T(p, x)$ are undefined for all p, x, S.

Initialization

- On message $(\text{INIT}, sid, \mathcal{SI})$ from S, ignore it if $|\mathcal{SI}| \neq n$ or S is active. Otherwise mark S as "active" and if no record $\langle sid, [...] \rangle$ exists, pick any previously unused label p and record $\langle sid, \mathcal{SI}, p \rangle$. Send $(\text{INIT}, sid, S, \mathcal{SI}, p)$ to $\mathcal{A}^*$.
- On message $(\text{INIT}, sid, \mathcal{A}^*, p)$ from $\mathcal{A}^*$, check that p is a label that has not been used before, record $\langle \mathcal{A}^*, p \rangle$ and return $(\text{INIT}, sid, \mathcal{A}^*, p)$ to $\mathcal{A}^*$.
- On message $(\text{INITCOMPLETE}, sid, S)$ from $\mathcal{A}^*$, retrieve tuple $\langle sid, \mathcal{SI}, p \rangle$. Ignore the message if there is no such tuple or $S \notin \mathcal{SI}$ or not all servers in $\mathcal{SI}$ are active. Otherwise, send $(\text{INITCOMPLETE}, sid)$ to S and mark S as "initialized."

Evaluation

- On message $(\text{EVAL}, sid, ssid, \mathcal{SE}, x)$ from $P \in \{U, \mathcal{A}^*\}$, retrieve $\langle sid, \mathcal{SI}, p \rangle$ if $P = U$ or $\langle \mathcal{A}^*, p \rangle$ if $P = \mathcal{A}^*$. Ignore this message if there is no such tuple, if $|\mathcal{SE}| \neq t{+}1$, or if tuple $\langle ssid, P, \cdot, \cdot, \cdot \rangle$ already exists. Otherwise record $\langle ssid, P, p, \mathcal{SE}, x \rangle$ and send $(\text{EVAL}, sid, ssid, P, \mathcal{SE})$ to $\mathcal{A}^*$.
- On message $(\text{SNDRCOMPLETE}, sid, ssid, S)$ from $\mathcal{A}^*$, retrieve tuple $\langle sid, \mathcal{SI}, p \rangle$. Ignore this message if there is no such tuple, or if $S \notin \mathcal{SI}$, or if S is not initialized. Otherwise, set $\mathsf{tx}(p, S)$++ (or set it to 1 if $\mathsf{tx}(p, S)$ is undefined), and send $(\text{SNDRCOMPLETE}, sid, ssid)$ to S.
- On message $(\text{RCVCOMPLETE}, sid, ssid, P, p^*)$ from $\mathcal{A}^*$, retrieve $\langle ssid, P, p, \mathcal{SE}, x \rangle$. Ignore this message if there is no such tuple, or if any of the following conditions fails: (i) if $p^* = p$ then $|\{S \in \mathcal{SI} \mid \mathsf{tx}(p, S) > 0\}| > t$, (ii) if all servers in $\mathcal{SE}$ are honest then $p^* = p$. Otherwise, if $p^* = p$ then set $\mathsf{tx}(p, S)$-- for any $t + 1$ distinct $S \in \mathcal{SI}$ s.t. $\mathsf{tx}(p, S) > 0$, and if $T(p^*, x)$ is undefined then pick $\rho \leftarrow_{\text{R}} \{0,1\}^{\ell}$ and set $T(p^*, x) := \rho$. Finally, send $(\text{EVAL}, sid, ssid, T(p^*, x))$ to P.

Fig. 1. Functionality $\mathcal{F}_{\text{TOPRF}}$ with parameters t, n.

The T-OPRF functionality of Fig. 1 has two stages, Initialization and Evaluation. The functionality enforces that the outputs of any such function are uniformly disributed, similarly to the single-server OPRF notion of [19], even in the case that the adversary controls the private key and/or its sharing among the

n servers. In more detail, in the initialization stage, a set of n servers, denoted $\mathcal{SI}$, are activated at the discretion of the adversary. The stage is complete when all servers become active. Note that the set may include adversarial servers, yet the functionality guarantees that all servers identified in $\mathcal{SI}$ become active by the end of the initialization stage. The initialization also specifies a parameter p used to identify a table $T(p, \cdot)$ of random values that defines the proper PRF values computed by the user when interacting with any subset of $t + 1$ *honest* servers from the set $\mathcal{SI}$. Additional parameters p^*, and corresponding tables $T(p^*, \cdot)$, can be specified by the adversary to represent rogue tables with values computed by the user in interaction with corrupted servers (see more on this below). The parameter p is also used to identify a counter $\mathsf{tx}(p, S)$ for each $S \in \mathcal{SI}$ as specified below.

In the evaluation stage, users connect to an arbitrary set of servers $\mathcal{SE}$ chosen by the adversary and which may arbitrarily overlap with $\mathcal{SI}$ (representing the fact that the user has no memory of who the servers in $\mathcal{SI}$ are). When, at the discretion of the adversary, a server $S \in \mathcal{SI}$ completes its interaction, the functionality increases the counter $\mathsf{tx}(p, S)$. Eventually, the adversary can trigger a response to the user which will be drawn from one of the tables maintained by the functionality. Recall that in addition to the proper table $T(p, \cdot)$ the adversary can register additional function tables $T(p^*, \cdot)$ and may connect an evaluation request from a user to any such table of its choice.

The security guarantees provided by the T-OPRF functionality are the following: (1) it enforces the use of the proper function table p whenever the set of servers $\mathcal{SE}$ selected for an evaluation are all honest; (2) it "charges" $t + 1$ server tickets for accessing the proper table p by decrementing (non-zero) ticket counters $\mathsf{tx}(p, S)$ for an arbitrary set of $t + 1$ servers in $\mathcal{SI}$; and (3) all tables T (the proper table p as well as any additional ones set by the adversary with $p^* \neq p$) are filled with random entries that are chosen on demand as the functionality responds back to the user. These guarantees ensure that at least $t+1-t'$ honest servers from $\mathcal{SI}$ need to be contacted for the proper function to be evaluated once. To see why this is the case observe that $t + 1$ tickets are "spent" (decremented) during evaluation which correspond to at least $t + 1 - t'$ tickets from honest ticketing counters. This implies that $t+1$ servers from $\mathcal{SI}$ have registered a SndrComplete message as this is the only event that triggers a counter increment. In the real world this corresponds to the event that a server has completed its interaction with a user that attempts to perform an evaluation.

It is important to highlight that the functionality does not necessarily decrement the ticketing counters of the servers identified in the chosen evaluation set $\mathcal{SE}$; rather, it decrements an arbitrary set of $t+1$ non-zero counters for servers in $\mathcal{SI}$. This reflects the fact that the functionality does not provide any guarantee about the identities of the responding servers. For instance, this means that we allow for an implementation of T-OPRF where an honest user U attempts to connect to a set of servers $\mathcal{SE}_1$ that are corrupted and its message is rerouted by the adversary so that, unbeknownst to U, an honest set of servers servers $\mathcal{SE}_2$ becomes the responder set.

Another important point regarding the T-OPRF functionality is that while it guarantees correct OPRF evaluation in case the user completes an undisturbed interaction with $t+1$ honest servers in $\mathcal{SI}$, the ideal world adversary may also maintain an arbitrary collection of random tables and connect a user to them, if desired, as long as the responder set is not composed of honest servers only. For instance, the adversary can assign to a subset of corrupted servers $\mathcal{SE}_1$ a certain function table, while it can assign a different function table to a different subset of corrupted servers $\mathcal{SE}_2$. While the two function tables will be independent, they are not under the control of the ideal world adversary completely: their contents will be populated by the ideal functionality with random values independently of each other. In practice this means that we allow for an implementation where two successive evaluation requests for the same x value result in a different (but still random) value to be produced, depending on which set of servers the user connects to. We stress that the secrecy of the input x is always preserved irrespectively of the subset of servers the user communicates with. At the same time, observe that the randomness requirement imposed for adversarial tables restricts our ability to implement the functionality to the random oracle setting.

3 Threshold OPRF Protocol from OMDH and T-OMDH

Here we present our Threshold Oblivious PRF protocol, called 2HashTDH, that instantiates the $\mathcal{F}_{\text{TOPRF}}$ functionality defined in Sect. 2. Thus, 2HashTDH provides a secure T-OPRF for use in general applications and, in particular, as the basis for our PPSS scheme, TOPPSS, presented in Sect. 4. The 2HashTDH scheme is formally defined as a realization of $\mathcal{F}_{\text{TOPRF}}$ in Fig. 3. In a nutshell, it is a threshold version of the 2HashDH OPRF from [19], recalled in Fig. 2. The underlying PRF, $f_k(x) = H_2(x, (H_1(x))^k)$, remains unchanged, but the key k is shared using Shamir secret-sharing across n servers, where server S_i stores the key share k_i. The initialization of such secret-sharing can be done via a Distributed Key Generation (DKG) for discrete-log-based systems, e.g. [16], and in Fig. 2 we assume it is done with a UC functionality $\mathcal{F}_{\text{DKG}}$ which we discuss further below. For evaluation, given any subset $\mathcal{SE}$ of $t+1$ servers, the user U sends to each of them the *same* message $a = (H'(x))^r$ for random r, exactly as

PRF Definition

G: a group of prime order m; H, H': hash functions with resp. ranges $\{0,1\}^\ell$ and G. PRF f from $\{0,1\}^*$ to $\{0,1\}^\ell$: For a key $k \leftarrow_{\text{R}} \mathbb{Z}_m$, define $f_k(x) = H(x, (H'(x))^k)$.

Oblivious Computation of PRF $f_k(x)$ between user U and server S

1. On input x, U chooses $r \leftarrow_{\text{R}} \mathbb{Z}_m$; sends $a = (H'(x))^r$ to S.
2. S verifies that the received a is in group G and if so it responds with $b = a^k$.
3. U outputs $f_k(x) = H(x, b^{1/r})$.

Fig. 2. The 2HashDH OPRF [19]

in the single-server OPRF protocol 2HashDH. If each server S_i in $\mathcal{SE}$ returned $b_i = a^{k_i}$ then U could reconstruct the value a^k using standard Lagrange interpolation in the exponent, i.e. $a^k = \prod_{i \in \mathcal{SE}} b_i^{\lambda_i}$ with the Lagrange coefficients λ_i computed using the indexes of servers in $\mathcal{SE}$. After computing a^k, the value of $f_k(x)$ is computed by U by deblinding a^k exactly as in the case of protocol 2HashDH. Note that this takes a single exponentiation for each server and two exponentiations for the user (to compute a and to deblind a^k) plus one multi-exponentiation by U to compute the Lagrange interpolation on the b_i values. We optimize this function evaluation by having each server S_i compute $b_i = a^{\lambda_i \cdot k_i}$, which costs one exponentiation and $O(t)$ multiplications and divisions in $\mathbb{Z}_m$ to compute λ_i. (Note that S_i must know set $\mathcal{SE}$ to compute λ_i.) This way U can compute a^k using only t multiplications instead of a multi-exponentiation, and the total costs are 1 exps for each S_i and 2 exps for U.

Protocol 2HashTDH can be also be seen as a simplification of a protocol which results from a generic transformation of any OPRF to T-TOPRF using multi-party secure computation of the server code, and then applying this transformation to the 2HashDH OPRF of [19]. The server in 2HashDH computes a^k on input a, and the MPC protocol for it is exactly the *threshold exponentiation* protocol described above, except that this generic OPRF to T-OPRF transformation must assure that the servers perform the MPC subprotocol on the same input a, and this involves an additional round of server-to-server interaction, which the 2HashTDH protocol avoids. We refer to the full version of this paper [20] for the specification of this general OPRF to T-OPRF compiler.

Roadmap. In Sect. 3.1 we show protocol 2HashTDH and explain the assumptions taken in its specification. In Sect. 3.2 we introduce the T-OMDH assumption, a generalization of OMDH, and we show that it is equivalent to OMDH in several cases, including the *full corruption* case $t' = t$ discussed in the introduction. In Sect. 3.3 we show that protocol 2HashTDH realizes the Threshold OPRF functionality $\mathcal{F}_{\text{TOPRF}}$ under the T-OMDH assumption in ROM for any threshold parameters (t, n) and any number $t' < t$ of corrupted servers. It follows that protocol 2HashTDH achieves the *standard threshold security* property, which corresponds to the full corruption case, under just OMDH in ROM. Note that the non-threshold OPRF 2HashDH of [19] also relies on OMDH.

3.1 T-OPRF Protocol Based on T-OMDH Assumption

The 2HashTDH T-OPRF protocol is shown in Fig. 3, relying on realizations of functionalities $\mathcal{F}_{\text{DKG}}, \mathcal{F}_{\text{AUTH}}$ and $\mathcal{F}_{\text{SEC}}$, which model, respectively, the distributed key generation, authenticated channel, and secure channel. Assuming these functionalities, the 2HashTDH protocol realizes the UC T-OPRF functionality defined in Sect. 2, under the T-OMDH assumption in ROM. As we argue in Sect. 3.2, this implies security under OMDH in ROM in several cases, including the *full corruption* case, where the adversary corrupts $t' = t$ servers, and the *additive sharing* case, where $t = n - 1$. Functionalities $\mathcal{F}_{\text{DKG}}, \mathcal{F}_{\text{AUTH}}, \mathcal{F}_{\text{SEC}}$ all have well-known efficient realizations in ROM under the Diffie-Hellman assumption which is implied by OMDH, and hence also by T-OMDH.

Let $\mathcal{F}_{\text{DKG}}, \mathcal{F}_{\text{AUTH}}$ and $\mathcal{F}_{\text{SEC}}$ be, respectively, the distributed key generation, authenticated channel, and secure channel functionalities; Let G be a cyclic group of prime order m; Let H_1, H_2 be hash functions with resp. ranges G and $\{0,1\}^{\ell}$.

Initialization

S1: On input $(\text{INIT}, sid, \mathcal{SI}{=}(S_1, \ldots, S_n))$, server S forwards this input to $\mathcal{F}_{\text{DKG}}$.
S2: On message $(\text{INITCOMPLETE}, sid, y, k_i)$ from $\mathcal{F}_{\text{DKG}}$, server S records $(sid, \mathcal{SI}, y, i, k_i)$, marks itself active, and outputs $(\text{INITCOMPLETE}, sid)$.

Evaluation

U1: On input $(\text{EVAL}, sid, ssid, \mathcal{SE}, x)$, U picks $r \leftarrow_{\text{R}} \mathbb{Z}_m$, computes $a := H_1(x)^r$, and sends $(\text{SEND}, (sid, ssid, 1), S, (\mathcal{SE}, a))$ to $\mathcal{F}_{\text{AUTH}}$ for all $S \in \mathcal{SE}$.
S1: On message $(\text{SENT}, (sid, ssid, 1), U, (\mathcal{SE}, a))$ from $\mathcal{F}_{\text{AUTH}}$, server S, provided it is active, computes $b_i := a^{\lambda_i \cdot k_i}$ where λ_i is a Lagrange interpolation coefficient for index i and index set $\mathcal{SE}$, sends $(\text{SEND}, (sid, ssid, 2), U, b_i)$ to $\mathcal{F}_{\text{AUTH}}$, and outputs $(\text{SNDRCOMPLETE}, sid, ssid)$.
U2: When U receives $(\text{SENT}, (sid, ssid, 2), S_i, b_i)$ from $\mathcal{F}_{\text{AUTH}}$ for all $S_i \in \mathcal{SE}$, it outputs $(\text{EVAL}, sid, ssid, H_2(x, (\prod_{S_i \in \mathcal{SE}} b_i)^{1/r}))$.

Fig. 3. Protocol 2HashTDH realizing $\mathcal{F}_{\text{TOPRF}}$ assuming $\mathcal{F}_{\text{DKG}}, \mathcal{F}_{\text{SEC}}, \mathcal{F}_{\text{AUTH}}$.

Note on Authentic and Secret Channels. In Fig. 3 protocol 2HashTDH is presented in the $(\mathcal{F}_{\text{AUTH}}, \mathcal{F}_{\text{SEC}}, \mathcal{F}_{\text{DKG}})$-hybrid world, i.e., assuming that there are both authenticated and secure (i.e. authenticated and secret) channels between protocol participants. We refer to [12] for the UC models of authenticated and secret channels, but simply speaking, what the authenticated and secure channel functionalities model is that if party P_1 sends message m to party P_2 using $\mathcal{F}_{\text{AUTH}}$ command $(\text{SEND}, sid, P_2, m)$, then P_2 will be able to authenticate m as originated from P_1, i.e. if P_2 receives command $(\text{SENT}, sid, P_1, m')$, it is guaranteed that $m' = m$, and if P_1 sends m to P_2 using $\mathcal{F}_{\text{SEC}}$ command $(\text{SEND}, sid, P_2, m)$, then P_2 can verify authenticity of P_1's message as above, but in addition m will be hidden to the adversary unless P_2 is corrupted.

We note that using ideal functionalities such $\mathcal{F}_{\text{AUTH}}, \mathcal{F}_{\text{SEC}}$ in the hybrid world, does not determine their implementation when the UC protocol is deployed in the real world. This is because they only describe how the adversarial model against the protocol is envisioned. For instance, $\mathcal{F}_{\text{AUTH}}$ *may* be realized using a PKI involving all connected participants, or it may be simply substituted by unauthenticated TCP/IP communication in case it is deemed that modifying message contents is not a relevant threat in the protocol deployment. Indeed, this will also be the case in our setting since we allow the (adversarial) environment to choose the servers that a user connects in the evaluation stage of the protocol in a way that is independent from the initialization servers; in this way, any man-in-the-middle scenario can be simulated by the adversary without

violating the $\mathcal{F}_{\text{AUTH}}$ constraints. Similarly, $\mathcal{F}_{\text{SEC}}$ *may* be implemented by TLS, but may also be achieved in other ways, e.g., physically transferring private state between the parties engaged in the protocol.

A second important point is that if a user U initializes a T-OPRF instance with a server set $\mathcal{SI} = \{S_1, ..., S_n\}$ such that some subset B of $\mathcal{SI}$ is made of corrupt entities (which models both the fact that some $\mathcal{SI}$ members are corrupt and the fact that U can execute T-OPRF initialization on an incorrect set of servers), then in this case command (SEND, sid, S_i, m) for $S_i \in B$ will leak m to the adversary, and if U receives (SENT, sid, S_i, m) from $\mathcal{F}_{\text{AUTH}}$ for $S_i \in B$, we can assume that the adversary supplies message m. In other words, the $\mathcal{F}_{\text{AUTH}}$ and $\mathcal{F}_{\text{SEC}}$ channels implement authenticated and/or secret point-to-point message delivery only if they are executed for a proper and non-corrupt server. We note that we assume a secret channel $\mathcal{F}_{\text{SEC}}$ in addition to an authenticated channel $\mathcal{F}_{\text{AUTH}}$ solely to simplify the description of T-OPRF initialization. Indeed, the former can be built from the latter [12], e.g. by having each server S_i first send its encryption public key to U using the authenticated channel.

Note on Distributed Key Generation. Protocol 2HashTDH assumes that servers in $\mathcal{SI}$ establish a secret-sharing $(k_1, ..., k_n)$ of a random key k over authenticated channels via a Distributed Key Generation (DKG) functionality $\mathcal{F}_{\text{DKG}}$, shown in Fig. 4. The DKG sub-protocol for discrete-log based cryptosystems can be efficiently realized without user's involvement [16,30], but if the call to initialize a TOPRF instance is executed by an *honest user* U then the DKG subprotocol can be even simpler, because U can generate sharing $(k_1, ..., k_n)$ of k and then distribute the shares among the servers in $\mathcal{SI}$. Note that since our realizations of $\mathcal{F}_{\text{TOPRF}}$ pertains only to the *static* adversarial model, where the identity of corrupt parties is determined at the outset, we would not explicitly require that the parties erase the information used in the initialization, but any implementation should erase such information. In our specification of protocol 2HashTDH we rely on the $\mathcal{F}_{\text{DKG}}$ functionality to abstract from any specific DKG implementation, e.g. whether it is done by the server or by an honest user.

3.2 Threshold OMDH Assumption

Notation. If n is an integer, then $[n] = \{1, ..., n\}$. If D is a set, then $|D|$ is its cardinality. We use bold font to denote vectors, e.g. $\boldsymbol{a} = [a_1, ..., a_n]$. If $\boldsymbol{a}$ and $\boldsymbol{b}$ are two vectors of the same dimension, then $\boldsymbol{a} \odot \boldsymbol{b}$ is their Hadamard (component-wise) product. If $|\boldsymbol{a}| = n$ and J is a sequence in $[n]$ then $\boldsymbol{a}_J$ denotes the components of $\boldsymbol{a}$ with indices in J, i.e. $[a_{i_1}, ..., a_{i_k}]^T$ if $J = (i_1, ..., i_k)$.

Let $\mathcal{I}_w$ be the set of w-element subsets of $[n]$, i.e. $\mathcal{I}_w = \{I \subseteq [n]\ s.t.\ |I| = w\}$. Let $W(\boldsymbol{a})$ be the hamming weight of $\boldsymbol{a}$. Let $\mathcal{V}_w$ be the set of n-bit binary vectors $\boldsymbol{q}$ s.t. $W(\boldsymbol{q}) = w$, i.e. $\mathcal{V}_w = \{\boldsymbol{v} \in \{0,1\}^n\ s.t.\ v_i = 1 \text{ iff } i \in \mathcal{I}_w\}$. For $\boldsymbol{q} = [q_1, \ldots, q_n]$ define $C_w(\boldsymbol{q})$ as the maximum integer m for which there exist $\boldsymbol{v}_1, ..., \boldsymbol{v}_m \in \mathcal{V}_w$ (not necessarily distinct) s.t. $\boldsymbol{v}_1 + ... + \boldsymbol{v}_m \leq \boldsymbol{q}$. In other words, $C_w(\boldsymbol{q})$ is the

Let g generate a cyclic group G of prime order m; Let t, n be integers s.t. $t < n$.

- On message $(\text{INIT}, sid, \mathcal{SI})$ from S, ignore it if $|\mathcal{SI}| \neq n$ or S is active or $S \notin \mathcal{SI}$. Otherwise, if no record $\langle sid, [...] \rangle$ exists, let Corrupted be the subset of $\mathcal{SI}$ that is corrupted and set $t' = |\mathsf{Corrupted}|$. If $t' \leq t$ then pick $a_0, a_1, \ldots, a_{t-t'} \leftarrow_{\text{R}} \mathbb{Z}_m$, record $\langle sid, \mathcal{SI}, a_0, a_1, \ldots, a_{t-t'} \rangle$ else record $\langle sid, \mathcal{SI} \rangle$. Irrespectively, mark S as "active", and send $(\text{INIT}, sid, P, \mathcal{SI})$ to $\mathcal{A}^*$.
- On message $(\text{INIT}, sid, \mathcal{SI}, s)$ from a corrupted $S \in \mathcal{SI}$, if the record $\langle sid, \mathcal{SI}, [...] \rangle$ exists, record $\langle \mathcal{A}^*, S, s \rangle$, mark S as active and send $(\text{INIT}, sid, \mathcal{A}^*, S)$ to $\mathcal{A}^*$.
- On message $(\text{INITCOMPLETE}, sid, S_i)$ from $\mathcal{A}^*$, retrieve tuple $\langle sid, \mathcal{SI}, a_0, a_1, \ldots, a_{t-t'} \rangle$. Ignore the message, if there is no such tuple, or if $S_i \notin \mathcal{SI}$ or not all servers in $\mathcal{SI}$ are active. Otherwise, send $(\text{INITCOMPLETE}, sid, g^{a_0}, i, s_i)$ to S_i and $(\text{INITCOMPLETE}, sid, S_i, g^{a_0})$ to $\mathcal{A}^*$, where $s_i = p(i)$ and $p(x)$ is a polynomial whose first $t - t' + 1$ coefficients match $a_0, a_1, \ldots, a_{t-t'}$ and $p(j) = s_j$ for each j such that $S_j \in \mathsf{Corrupted}$ and $\langle \mathcal{A}^*, S_j, s_j \rangle$ has been previously recorded.

Fig. 4. Distributed key generation functionality $\mathcal{F}_{\text{DKG}}$ [30].

maximum number of times one can subtract elements in $\mathcal{V}_w$ from $\boldsymbol{q}$ s.t. the result remains $\geq \boldsymbol{0}$. For example if and $\boldsymbol{q} = [3, 3, 4]$ then $C_2(\boldsymbol{q}) = 4$ because $\boldsymbol{q} = 2 \times [1, 0, 1] + [1, 1, 0] + 2 \times [0, 1, 1]$.

T-OMDH Intuition. Let $\langle g \rangle$ be a cyclic group of prime order $m > n$. The T-OMDH assumption considers the setting where a random exponent $k \in \mathbb{Z}_m$ is secret-shared using a random t-degree polynomial $p(\cdot)$, and the n trustees holding shares $k_1{=}p(1), \ldots, k_n{=}p(n)$ implement a "threshold exponentiation" protocol which computes a^k for any given $a \in \langle g \rangle$ and $k = p(0)$. Let $\mathsf{TOMDH}_p(\cdot, \cdot)$ be an oracle which on input $(i, a) \in [n] \times \langle g \rangle$ outputs $a^{p(i)}$. The standard way to implement threshold exponentiation is to choose a set $I \in \mathcal{I}_{t+1}$, compute $b_i = \mathsf{TOMDH}_p(i, a) = a^{k_i}$ for each i in I and derive a^k as $\prod_{i \in I} b_i^{\lambda_i}$ using Lagrange interpolation coefficients λ_i s.t. $k = \sum_{i \in I} \lambda_i \cdot k_i$. The T-OMDH assumption states that querying oracle $\mathsf{TOMDH}_p(\cdot, \cdot)$ on at least $t + 1$ different points $i \in [n]$ is *necessary* to compute $a^{p(0)}$ for a given random challenge a. More generally, T-OMDH considers an experiment where the attacker $\mathcal{A}$ receives a challenge set $R = \{g_1, ..., g_N\}$ of random elements in $\langle g \rangle$ and is given access to the $\mathsf{TOMDH}_p(\cdot, \cdot)$ oracle for random t-degree polynomial $p(\cdot)$. T-OMDH assumption states that $\mathcal{A}$ can compute g_j^k for $k = p(0)$ for no more than $C_{t+1}(q_1, \ldots, q_n)$ elements $g_j \in R$, where q_i is the number of $\mathcal{A}$'s queries to $\mathsf{TOMDH}_p(i, \cdot)$.

The above intuition and Definition 1 below correspond to the setting where the attacker does not control any of the trustees holding shares of p, hence it needs $t + 1$ queries to $\mathsf{TOMDH}_p(\cdot, \cdot)$ to compute $a^{p(0)}$ for each random challenge a. Later we extend this definition to the case where $\mathcal{A}$ controls a subset of trustees.

Definition 1. *The (t, n, N, Q)-Threshold One-More Diffie Hellman (T-OMDH) assumption holds in group $\langle g \rangle$ of prime order m if the probability of any polynomial-time adversary $\mathcal{A}$ winning the following game is negligible. $\mathcal{A}$ receives challenge set $R = \{g_1, \ldots, g_N\}$ where $g_i \leftarrow_R \langle g \rangle$ for $i \in [N]$, and is given access to an oracle $\mathsf{TOMDH}_p(\cdot, \cdot)$ for a random t-degree polynomial $p(\cdot)$ over $\mathbb{Z}_m$. $\mathcal{A}$ wins if it outputs g_j^k where $k = p(0)$ for $Q + 1$ different elements g_j in R, and if $C_{t+1}(q_1, \ldots, q_n) \leq Q$ where q_i is the number of $\mathcal{A}$'s queries to $\mathsf{TOMDH}_p(i, \cdot)$.*

Note that the (N, Q)-OMDH assumption [5,22] is the (t, n, N, Q)-T-OMDH assumption for $t = 0$ and any $n \geq 1$, because then $p(\cdot)$ is a constant polynomial and $C_1(\boldsymbol{q}) = W(\boldsymbol{q})$, i.e. the total number of $\mathcal{A}$'s $\mathsf{TOMDH}_p(\cdot, \cdot)$ queries.

T-OMDH: The General Case. In its general form, the T-OMDH assumption corresponds to computing g_j^k if some subset of $t' \leq t$ trustees holding shares $k_i = p(i)$ is corrupt, and hence the adversary can not only learn these shares but can also set them at will.

Definition 2. *The (t', t, n, N, Q)-T-OMDH assumption holds in group $\langle g \rangle$ of prime order m if for any $\mathsf{B} \subseteq [n]$ s.t. $|\mathsf{B}| = t' \leq t$, the probability of any polynomial-time adversary $\mathcal{A}$ winning the following game is negligible. On input a challenge set $R = \{g_1, \ldots, g_N\}$ where $g_i \leftarrow_R \langle g \rangle$ for $i \in [N]$, adversary $\mathcal{A}$ specifies a set of t' values $\{\alpha_j\}_{j \in \mathsf{B}}$ in $\mathbb{Z}_m$. A random t-degree polynomial $p(\cdot)$ over $\mathbb{Z}_m$ is then chosen subject to the constraint that $p(j) = \alpha_j$ for $j \in \mathsf{B}$, and the adversary $\mathcal{A}$ is given access to oracle $\mathsf{TOMDH}_p(\cdot, \cdot)$. We say that $\mathcal{A}$ wins if it outputs g_j^k where $k = p(0)$ for $Q + 1$ different elements g_j in R, and if $C_{t-t'+1}(q_1, \ldots, q_n) \leq Q$ where q_i for $i \notin \mathsf{B}$ is the number of $\mathcal{A}$'s queries to $\mathsf{TOMDH}_p(i, \cdot)$, and $q_i = 0$ for $i \in \mathsf{B}$.*

Note that (t', t, n, N, Q)-T-OMDH is identical to (t, n, N, Q)-T-OMDH for $t' = 0$.

Gap T-OMDH. In order to prove the security of T-OPRF, we need to extend the T-OMDH assumption stated in Definition 2 to its "gap" form, i.e. suppose $\langle g \rangle$ is a gap group where $\mathcal{A}$ is in addition given access to the DDH oracle in $\langle g \rangle$.

Definition 3. *The* Gap *(t', t, n, N, Q)-T-OMDH assumption is the T-OMDH assumption of Definition 2 except that $\mathcal{A}$ is also given access to the* DDH *oracle in group $\langle g \rangle$, which on input (a, b, c, d) outputs 1 if $\log_a b = \log_c d$ and 0 otherwise.*

In the full version of this paper [20] we show that the (Gap) (t, t', n, N, Q)-T-OMDH assumption holds in the generic group model for any (t', t, n). Specifically, the advantage of a T-OMDH adversary restricted to r generic group operations is upper-bounded by $O(Qr^2/m)$, assuming $r \geq Q \geq N$. This is larger by factor Q from the $O(r^2/m)$ upper-bounds on generic group attacks against many static problems related to discrete logarithm [28], and this weakening is caused by the presence of up to Q-degree polynomials of the "target" secret $k = p(0)$ in the representation of the group elements which the adversary can compute given access to $\mathsf{TOMDH}_p(\cdot, \cdot)$ using the query pattern $\boldsymbol{q} = [q_1, ..., q_n]$ s.t. $C_{t-t'+1}(\boldsymbol{q}) \leq Q$. Since (Q, N)-OMDH is identical to (t', t, n, N, Q)-T-OMDH for $(t', t) = (0, 0)$

and any n, the same upper-bound applies applies to OMDH, and to the best of our knowledge this is the first generic model security hardness argument for the OMDH (or Gap OMDH) assumption.

T-OMDH = OMDH in Full Corruption and Additive Sharing Cases. The T-OMDH and OMDH assumptions are equivalent in two important cases, namely the *full corruption* case of $t' = t$, for any (t, n), and in the *additive sharing* case of $t = n - 1$, for any t'. We refer to the full version of this paper [20] for (easy) proofs of above equivalences. Note also that whereas the question whether the T-OMDH and OMDH assumptions are equivalent for any $t' < t$ and $t + 1 < n$ remains open, in the full version [20] we also show the same generic group hardness bound for both problems.

3.3 Security Analysis of 2HashTDH

Protocol 2HashTDH protocol of Fig. 3 is secure under the T-OMDH assumption. As a corollary of the fact that in the full corruption case of $t' = t$ faults the T-OMDH and OMDH assumptions are equivalent, Theorem 1 implies that protocol 2HashTDH is secure under OMDH in ROM in the full corruption case of $t' = t$. The proof of Theorem 1 appears in the full version of this paper [20].

Theorem 1. *Protocol 2HashTDH realizes functionality $\mathcal{F}_{\text{TOPRF}}$ with parameters t, n in the $(\mathcal{F}_{\text{AUTH}}, \mathcal{F}_{\text{SEC}}, \mathcal{F}_{\text{DKG}})$-hybrid model, assuming static corruptions, hash functions $H_1(\cdot)$ and $H_2(\cdot, \cdot)$ modelled as Random Oracles, and the Gap (t', t, n, N, Q)-T-OMDH on group $\langle g \rangle$, where Q is the number of* Eval *messages sent by any user, $N = Q + q_1$ where q_1 is the number of $H_1(\cdot)$ queries the adversary makes, and $t' < t$ is the number of corrupted servers in $\mathcal{SI}$.*

Specifically, for any efficient adversary $\mathcal{A}$ against protocol 2HashTDH, there is a simulator SIM *s.t. no efficient environment $\mathcal{Z}$ can distinguish the view of $\mathcal{A}$ interacting with the real 2HashTDH protocol and the the view of* SIM *interacting with the ideal functionality $\mathcal{F}_{\text{TOPRF}}$, with advantage better than $q_T \cdot \epsilon(N, Q) + N^2/m$, where q_T is the number of TOPRF instances, $\epsilon(N, Q)$ is the bound on the probability that any algorithm of the same cost violates the Gap (t', t, n, N, Q)-T-OMDH assumption, and $m = |\langle g \rangle|$.*

4 TOPPSS: A PPSS Scheme Based on T-OPRF

In Fig. 5 we show a compiler which converts a T-OPRF scheme which realizes the UC T-OPRF notion of Sect. 2 into a PPSS scheme, called TOPPSS, which realizes UC PPSS functionality of [19]. The terminology of the UC setting might obscure the amazing practicality of this construction, so in Sect. 5 we show a concrete implementation of this scheme with the $\mathcal{F}_{\text{TOPRF}}$ functionality implemented using the T-OPRF instantiation 2HashTDH from Sect. 3.

TOPPSS Overview. To explain the mechanics of TOPPSS based on the T-OPRF functionality, it is instructive to compare it to the OPRF-based PPSS

Let $\mathcal{F}_{\text{AUTH}}$ and $\mathcal{F}_{\text{TOPRF}}$ be, respectively, the authenticated channel and the T-OPRF functionality; Let $H(\cdot)$ be a hash function with range $\{0,1\}^{\ell}$.

INIT for user U:

1. On input $(\text{INIT}, sid, \mathcal{SI}, \mathsf{pw})$, send $(\text{SEND}, (sid, 0), S, \mathcal{SI})$ to $\mathcal{F}_{\text{AUTH}}$ for all S in $\mathcal{SI}$.
2. On $(\text{SENT}, (sid, 1), S, \text{DONE})$ from $\mathcal{F}_{\text{AUTH}}$ for all $S \in \mathcal{SI}$, send $(\text{EVAL}, sid, 0, \mathcal{SE}, \mathsf{pw})$ to $\mathcal{F}_{\text{TOPRF}}$ for any $\mathcal{SE} \subseteq \mathcal{SI}$ such that $|\mathcal{SE}| = t + 1$.
3. On $\mathcal{F}_{\text{TOPRF}}$'s response $(\text{EVAL}, sid, 0, v)$, parse $H(v)$ as $[C|K]$ and send $(\text{SEND}, (sid, 2), S, C)$ to $\mathcal{F}_{\text{AUTH}}$ for every $S \in \mathcal{SI}$.
4. On $(\text{SENT}, (sid, 3), S, \text{ACK})$ for all $S \in \mathcal{SI}$ from $\mathcal{F}_{\text{AUTH}}$, output (UINIT, sid, K).

INIT for server S:

1. On $(\text{SENT}, (sid, 0), U, \mathcal{SI})$ from $\mathcal{F}_{\text{AUTH}}$, send $(\text{INIT}, sid, \mathcal{SI})$ to $\mathcal{F}_{\text{TOPRF}}$.
2. On $(\text{INITCOMPLETE}, sid)$ from $\mathcal{F}_{\text{TOPRF}}$, send $(\text{SEND}, (sid, 1), U, \text{DONE})$ to $\mathcal{F}_{\text{AUTH}}$.
3. On $(\text{SENT}, (sid, 2), U, C)$ from $\mathcal{F}_{\text{AUTH}}$, record (sid, C), send $(\text{SEND}, (sid, 3), U, \text{ACK})$ 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 $(\text{EVAL}, sid, [1|ssid], \mathcal{SR}, \mathsf{pw}')$ to $\mathcal{F}_{\text{TOPRF}}$.
2. On $\mathcal{F}_{\text{TOPRF}}$'s response $(\text{EVAL}, sid, [1|ssid], v')$ and $(\text{SENT}, (sid, ssid, 1), S, C')$ from $\mathcal{F}_{\text{AUTH}}$ for all $S \in \mathcal{SR}$, if each message contains C' s.t. $[C'|K'] = H(v')$, then set $\text{RES} := K'$, otherwise set $\text{RES} := \text{FAIL}$. Output $(\text{UREC}, sid, ssid, \text{RES})$.

REC for server S:

1. On $(\text{SNDRCOMPLETE}, sid, [1|ssid])$ from $\mathcal{F}_{\text{TOPRF}}$, if S holds record (sid, C), then send $(\text{SEND}, (sid, ssid, 1), U, C)$ to $\mathcal{F}_{\text{AUTH}}$ and output $(\text{SREC}, sid, ssid)$.

Fig. 5. The TOPPSS protocol

scheme of [19]. In that scheme each server holds its own *independently random* key k_i for an OPRF f. At initialization, the secret to be protected is processed with a (t, n) secret sharing scheme and each share is stored at one of n servers, where server S_i stores the i-th share encrypted under $f_{k_i}(\mathsf{pw})$. At reconstruction, the user receives the encrypted shares from $t+1$ servers which it decrypts using the values $f_{k_i}(\mathsf{pw})$ that it learns by running the OPRF on pw with each of these servers. By contrast, in our TOPPSS scheme, which is T-OPRF-based, the (random) secret to be protected is defined as a single PRF value $v = f_k(\mathsf{pw})$ where k is a key secret-shared as part of a T-OPRF scheme. This provides a significant performance gain by reducing the number of exponentiations performed by the user from $t+2$ to just 2. In the scheme of [19] implemented with 2HashDH, the user computes the OPRF sub-protocol with each server independently, which

involves one blinding operation re-used across all servers, but requires one de-blinding operation *per server* for a total of $t + 2$ exponentiations. By contrast, in the T-OPRF protocol 2HashTDH of Sect. 3 the user performs a single blinding and de-blinding, hence just 2 exponentiations, regardless of the number of servers and threshold t.

Note that the T-OPRF functionality allows the user to evaluate function $f_k(\cdot)$ on the user's password pw, without leaking any information about pw, but it does not let the user verify whether the function is computed correctly. Indeed, following the rules of functionality $\mathcal{F}_{\text{TOPRF}}$, either corrupt servers or a man-in-the-middle adversary could make the user compute $f_k(\mathsf{pw})$ on key k of their choice. If the dictionary D from which the user draws her password is small, the adversary can potentially pick k s.t. function $f_k(\cdot)$ behaves on domain D in some ways the adversary can exploit (e.g., reducing the number of possible outputs). However, since $\mathcal{F}_{\text{TOPRF}}$ assures that $f_k(\cdot)$ behaves like a random function for all k's, even for k's chosen by the adversary, it suffices to include a commitment to the master secret $v = f_k(\mathsf{pw})$ in the information that the servers send to the user, so that the user can verify its correctness. The adversary can still pick k but if $f_k(\cdot)$ is pseudorandom for all k then the adversary cannot change either k or v without guessing pw. Note that the randomness for verifying this commitment must be derived from the committed plaintext $f_k(\mathsf{pw})$ itself as this is the only value the user can retrieve using its only input pw. Although this mechanism requires the commitment scheme to be deterministic, the hiding property of the commitment is still satisfied thanks to the pseudorandomness of the committed plaintext $v = f_k(\mathsf{pw})$ (and assuming no more than t corruptions).

Since our realizations of $\mathcal{F}_{\text{TOPRF}}$, protocol 2HashTDH, requires the Random Oracle Model (ROM) for hash functions in the security analysis, we implement this commitment simply with another hash function modeled as a random oracle. Finally, since the user needs to verify the master-secret v as well as to derive a key K from it, we implement both operation using a single hash function call, i.e. we set $[C|K]$ to $H(v)$ where H hashes onto strings of length 2ℓ.

The proof of the following theorem is in the full version of this paper [20].

Theorem 2. *The TOPPSS scheme of Fig. 5 UC-realizes the PPSS functionality $\mathcal{F}_{\text{PPSS}}$ assuming access to the T-OPRF functionality $\mathcal{F}_{\text{TOPRF}}$ and to the authenticated message delivery functionality $\mathcal{F}_{\text{AUTH}}$, and assuming that hash function H is a random oracle.*

5 Concrete Instantiation of TOPPSS Using 2HashTDH

For concreteness we show an instantiation of TOPPSS with the T-OPRF functionality realized by protocol 2HashTDH from Fig. 3 in Sect. 3. In this figure we realize the $\mathcal{F}_{\text{DKG}}$ subprotocol assuming an honest user U, because in the context of a PPSS protocol, we only care about security for PPSS instances which were initialized with an honest user. Hence we simply have U create the sharing of the T-OPRF key and distributing it among the servers in $\mathcal{SI}$ (see a note on DKG in

Parameters: Security parameter ℓ, threshold parameters t, n, cyclic group $\langle g \rangle$ of prime order m, hash functions H_1, H_2, H_3 with ranges $\langle g \rangle$, $\{0,1\}^{2\ell}$ and $\{0,1\}^{2\ell}$.
Communication Setting: Communication between U and S_i's in INIT goes over secure channels, e.g. TLS, communication in REC not necessarily (see text).

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

1. Pick $k \leftarrow_R \mathbb{Z}_m$, generate $(k_1, \ldots, k_n)$ as a (t, n)-Shamir's secret sharing of k over $\mathbb{Z}_m$, and send $(sid, \mathcal{SI}, i, k_i)$ to each $S_i \in \mathcal{SI}$.
2. After receiving ACK's from all servers in $\mathcal{SI}$, compute $v := H_2(\mathsf{pw}, H_1(\mathsf{pw})^k)$, parse $H_3(v)$ as $[C|K]$, send (sid, C) to all $S_i \in \mathcal{SI}$ and output K.

INIT for server S_i:

1. On $(sid, \mathcal{SI}, i, k_i)$ from U, abort if sid not unique or $S_i \notin \mathcal{SI}$, otherwise record $(sid, \mathcal{SI}, i, k_i)$ and send (sid, ACK) to U.
2. On (sid, C) from U, append C to tuple $(sid, \mathcal{SI}, i, k_i)$.

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

1. Pick $r \leftarrow_R \mathbb{Z}_m$, set $a := H_1(x)^r$, send $((sid, ssid), (\mathcal{SR}, a))$ to all $S'_i \in \mathcal{SR}$.
2. After receiving message $((sid, ssid), b_i, C)$ from all servers in $\mathcal{SR}$, abort if not all C's are the same. Otherwise compute $b := \prod_{S'_i \in \mathcal{SR}} b_i$ and $v := H_2(x, b^{1/r})$.
3. Parse $H_3(v)$ as $[C'|K']$. If $C' = C$ then output K', otherwise output FAIL.

REC for server $S_i \in \mathcal{SR}$:

1. On message $((sid, ssid), (\mathcal{SR}, a))$ from U, recover stored tuple $(sid, \mathcal{SI}, i, k_i, C)$. (Abort if there is no such tuple or if $a \notin \langle g \rangle$.) Compute $b_i := a^{\lambda_i \cdot k_i}$ where λ_i is an interpolation coefficient for index i and the subset of $t+1$ indices defined by $\mathcal{SR} \subseteq \mathcal{SI}$, and send $((sid, ssid), b_i, C)$ to U.

Fig. 6. Concrete instantiation of TOPPSS based on 2HashTDH T-OPRF.

Sect. 3.1). Note that if we implement $\mathcal{F}_{\text{DKG}}$ in this user-centric way then we do not have to execute T-OPRF evaluation for U to compute $v = f_k(\mathsf{pw})$ as part of the initialization: User U can just compute $v = f_k(\mathsf{pw})$ locally because U picked the TOPRF key k (Fig. 6).

On the Role of Secure Channels. The communication in such instantiation of TOPPSS must go over secure channels in the *initialization phase*, which in practice could be implemented using e.g. TLS.[2] In the *reconstruction phase*, the communication does not have to go over secure channels, because TOPPSS is

[2] Note that if the $\mathcal{F}_{\text{DKG}}$ was instantiated with the distributed key generation then authenticated channels would suffice for the communication between the user and the servers because the TOPRF evaluation protocol does not need secure channels. However, the standard realization of $\mathcal{F}_{\text{DKG}}$ [30] would require secure channels between the servers.

secure in the password-only, i.e. PKI-free, model. However using TLS would offer a security benefit against the network adversary as a *hedge* against any server-spoofing attacks due to which the user might be tricked to run the PPSS reconstruction with the wrong set of servers. To see the benefit of running a PPSS protocol over TLS channels, denote the set of server identities which U inputs in the reconstruction as $\mathcal{SR}$. In the case of running PPSS reconstruction over TLS these can be equated with the public keys the user would use in the TLS sessions with the $t+1$ servers in the reconstruction. Consider the following two cases, and refer to the specification of the UC PPSS functionality $\mathcal{F}_{\text{PPSS}}$ of [19], which we include in the full version of this paper [20].

Case I: Every server S' in set $\mathcal{SR}$ is either incorrect (i.e. $S' \notin \mathcal{SI}$) and w.l.o.g. represents a malicious entity, or it is correct (i.e. $S' \in \mathcal{SI}$) but it is corrupted. In this case, according to $\mathcal{F}_{\text{PPSS}}$ specifications (see line 3b of the reconstruction phase), the adversary can perform one on-line password guess on such session. In other words, if the user runs reconstruction with incorrect/corrupt servers, the security is as in a (password-only) PAKE, i.e. the adversary can attempt to authenticate to such user using a password guess pw^*, and test if $\mathsf{pw}^* = \mathsf{pw}$.

Case II: There are *some* servers S' in set $\mathcal{SR}$ which are both correct (i.e. $S' \in \mathcal{SI}$) and uncorrupted. In this case, according to $\mathcal{F}_{\text{PPSS}}$ specifications (lines 3a and 3b of $\mathcal{F}_{\text{PPSS}}$), the adversary cannot learn anything from such instance, and can only either let it execute (line 3a) in which case U reconstructs the (correct!) secret K, or interfere with the protocol (line 3c) and make U output FAIL.
In short, if PPSS reconstruction is executed over insecure channels then the man-in-the-middle adversary could make every reconstruction instance fall into Case I. By contrast, executing it over TLS forces the reconstruction instances to fall into Case II, unless the adversary tricks U to execute the reconstruction for the set of servers $\mathcal{SR}$ which includes *only* corrupt entities, in which case such reconstruction instance (and only such instance) falls back into Case I.

Note on sid/ssid Monikers. As we explain above, it is not essential for security of reconstruction that the user remembers the servers in the initialization set $\mathcal{SI}$. It might also be helpful to clarify the potential security implications of sid/ssid monikers which we assume are inputs in the initialization and the reconstruction phase. String *sid* (which stands for "session ID" in the AKE and UC terminology) in the context of a PPSS scheme can be equated with a "user ID", because it is a string which servers in $\mathcal{SI}$ will use to disambiguate between multiple PPSS instances which they can potentially service. It is therefore sensible to require that U remembers this user ID string *sid* in addition to her password pw. On the other hand, string *ssid* could be a nonce, or some application-determined identifier of a unique PPSS reconstruction session.

References

1. Russian hackers amass over a billion internet passwords. New York Times, 08 June 2014. http://goo.gl/aXzqj8

2. Abdalla, M., Chevassut, O., Fouque, P.-A., Pointcheval, D.: A simple threshold authenticated key exchange from short secrets. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 566–584. Springer, Heidelberg (2005). doi:10.1007/11593447_31
3. Abdalla, M., Cornejo, M., Nitulescu, A., Pointcheval, D.: Robust password-protected secret sharing. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016. LNCS, vol. 9879, pp. 61–79. Springer, Cham (2016). doi:10.1007/978-3-319-45741-3_4
4. Bagherzandi, A., Jarecki, S., Saxena, N., Lu, Y.: Password-protected secret sharing. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, pp. 433–444. ACM (2011)
5. Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M., et al.: The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. J. Cryptol. **16**(3), 185–215 (2003)
6. 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). doi:10.1007/3-540-45539-6_11
7. Blazy, O., Chevalier, C., Vergnaud, D.: Mitigating server breaches in password-based authentication: secure and efficient solutions. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 3–18. Springer, Cham (2016). doi:10.1007/978-3-319-29485-8_1
8. Boyko, V., MacKenzie, P., Patel, S.: Provably secure password-authenticated key exchange using Diffie-Hellman. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer, Heidelberg (2000). doi:10.1007/3-540-45539-6_12
9. Brainard, J., Juels, A., Kaliski, B., Szydlo, M.: Nightingale: a new two-server approach for authentication with short secrets. In: 12th USENIX Security Symposium, pp. 201–213. IEEE Computer Society (2003)
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. LNCS, vol. 8617, pp. 256–275. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44381-1_15
11. Camenisch, J., Lehmann, A., Neven, G.: Optimal distributed password verification. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 182–194. ACM (2015)
12. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, 2001, pp. 136–145. IEEE (2001)
13. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.-C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 353–373. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40041-4_20
14. Raimondo, M., Gennaro, R.: Provably secure threshold password-authenticated key exchange. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 507–523. Springer, Heidelberg (2003). doi:10.1007/3-540-39200-9_32
15. Ford, W., Kaliski, B.S.: Server-assisted generation of a strong secret from a password. In: Proeedings of the IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises 2000, (WET ICE 2000), pp. 176–180. IEEE (2000)
16. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. J. Cryptol. **20**(1), 51–83 (2007)

17. Jarecki, S., Jutla, C., Krawczyk, H., Rosu, M., Steiner, M.: Outsourced symmetric private information retrieval. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pp. 875–888. ACM (2013)
18. 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. LNCS, vol. 8874, pp. 233–253. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45608-8_13
19. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: Highly-efficient and composable password-protected secret sharing (or: how to protect your bitcoin wallet online). In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 276–291. IEEE (2016)
20. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: Threshold oblivious PRF and minimal-cost password-protected secret sharing. Cryptology ePrint Archive (2017). http://eprint.iacr.org/2017/[TBD]
21. Jarecki, S., Krawczyk, H., Shirvanian, M., Saxena, N.: Device-enhanced password protocols with optimal online-offline protection. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, pp. 177–188. ACM (2016)
22. Jarecki, S., Liu, X.: Fast secure computation of set intersection. In: Garay, J.A., Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 418–435. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15317-4_26
23. Katz, J., MacKenzie, P., Taban, G., Gligor, V.: Two-server password-only authenticated key exchange. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 1–16. Springer, Heidelberg (2005). doi:10.1007/11496137_1
24. Kiefer, F., Manulis, M.: Distributed smooth projective hashing and its application to two-server password authenticated key exchange. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 199–216. Springer, Cham (2014). doi:10.1007/978-3-319-07536-5_13
25. Kiefer, F., Manulis, M.: Universally composable two-server PAKE. In: Bishop, M., Nascimento, A. (eds.) ISC 2016. LNCS, vol. 9866, pp. 147–166. Springer, Cham (2016). doi:10.1007/978-3-319-45871-7_10
26. MacKenzie, P., Shrimpton, T., Jakobsson, M.: Threshold password-authenticated key exchange. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 385–400. Springer, Heidelberg (2002). doi:10.1007/3-540-45708-9_25
27. Naor, M., Pinkas, B., Reingold, O.: Distributed pseudo-random functions and KDCs. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 327–346. Springer, Heidelberg (1999). doi:10.1007/3-540-48910-X_23
28. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997). doi:10.1007/3-540-69053-0_18
29. Szydlo, M., Kaliski, B.: Proofs for two-server password authentication. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 227–244. Springer, Heidelberg (2005). doi:10.1007/978-3-540-30574-3_16
30. Wikström, D.: Universally composable DKG with linear number of exponentiations. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 263–277. Springer, Heidelberg (2005). doi:10.1007/978-3-540-30598-9_19
31. 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 2015. LNCS, vol. 9326, pp. 347–365. Springer, Cham (2015). doi:10.1007/978-3-319-24174-6_18