TOPPSS: Cost-minimal Password-Protected Secret Sharing based on Threshold OPRF

(Full version of a paper which appeared in ACNS'17 [20])

Stanisław Jarecki[1] *, Aggelos Kiayias[2] **, Hugo Krawczyk[3] ***, and Jiayu Xu[1]

[1] University of California, Irvine
[2] University of Edinburgh
[3] IBM Research

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 all-too-common leakage of the database of password hashes stored by the authentication

* Supported by NSF grant CNS-1547435.
** Supported by ERC project CODAMODA and H2020 Project Panoramix, #653497.
*** Supported by ONR Contract N00014-14-C-0113.

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-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 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*.[4] 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, 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

[4] 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.

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 Section 2 we define the fundamental tool TOPPSS relies on, namely T-OPRF, as a UC functionality. In Section 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 Section 4 we show a low-cost compiler from T-OPRF to PPSS, which we exemplify in Section 5 with a concrete instantiation using 2HashTDH. Finally, in Section 6 we include a generic compiler from any OPRF to T-OPRF, which yields a PPSS scheme similar to the one of Section 5 except for requiring two communication rounds instead of one.

2 Universally Composable Threshold OPRF

Notation. We use ℓ to denote the security parameter; If a, b are strings, then $[a|b]$ is their concatenation; If n is a positive integer, then $[n] = \{1, ..., n\}$; If D is a set, then $|D|$ is its cardinality; ":=" denotes a deterministic assignment; "$\leftarrow$" denotes a randomized assignment; "$\leftarrow_{\mathrm{R}}$" denotes 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.

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

Initialization

- On message (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 (INIT, $sid, S, \mathcal{SI}, p$) to $\mathcal{A}^*$.
- On message (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 (INIT, $sid, \mathcal{A}^*, p$) to $\mathcal{A}^*$.
- On message (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 (INITCOMPLETE, sid) to S and mark S as "initialized."

Evaluation

- On message (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 (EVAL, $sid, ssid, P, \mathcal{SE}$) to $\mathcal{A}^*$.
- On message (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 (SNDRCOMPLETE, $sid, ssid$) to S.
- On message (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 $\rho \leftarrow_{\mathrm{R}} \{0,1\}^{\ell}$ and set $T(p^*, x) := \rho$. Finally, send (EVAL, $sid, ssid, T(p^*, x)$) to P.

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

to PPSS schemes. Here we provide a formal definition of this notion, namely as a secure realization of a UC functionality $\mathcal{F}_{\text{TOPRF}}$ shown in Figure 1. The $\mathcal{F}_{\text{TOPRF}}$ functionality defined here is a generalization of 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 using secret-sharing as we show in our 2HashTDH realization in Section 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 weaker form of OPRF suffices in the key application of interest to us, namely, Password-Protected Secret-Sharing.

The T-OPRF functionality of Figure 1 has two stages, Initialization and Evaluation. The T-ORPF functionality enforces that the values of any such function remain random to the adversary, similarly as the single-server OPRF notion of [19], even in the case the adversary controls the private key and/or its sharing among the n servers (but is not privy to the value the T-OPRF is evaluated on). In more details, 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.

During this stage a ticket counter associated with the function controlled by the set of servers is initialized and so is an empty table implementing the random function shared by the $\mathcal{SI}$ servers.

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" (decre-

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 $k \leftarrow_R \mathbb{Z}_m$, 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 $r \leftarrow_R \mathbb{Z}_m$; 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]

mented) 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 connects to. We stress that this does not jeopardize the privacy of the value x in any way. 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 Section 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 Section 4. The 2HashTDH scheme is formally defined as a realization of $\mathcal{F}_{\text{TOPRF}}$ in Figure 3. In a nutshell, it is a threshold version of the 2HashDH OPRF from [19], recalled in Figure 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 Figure 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 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. For completeness, we include the specification of this general OPRF to T-OPRF compiler in Section 6.

Roadmap. In Section 3.1 we show protocol 2HashTDH and explain the assumptions taken in its specification. In Section 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 Section 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 Figure 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

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 $\mathcal{F}_{\text{DKG}}, \mathcal{F}_{\text{SEC}}, \mathcal{F}_{\text{AUTH}}$.

these functionalities, the 2HashTDH protocol realizes the UC T-OPRF functionality defined in Section 2, under the T-OMDH assumption in ROM. As we argue in Section 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.

Note on Authentic and Secret Channels. In Figure 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 $(\text{SEND}, sid, S_i, m)$ for $S_i \in B$ will leak m to the adversary, and if U receives $(\text{SENT}, sid, S_i, m)$ 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 Figure 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

Additional Notation. We use bold font to denote vectors, e.g. $\boldsymbol{a} = [a_1, ..., a_n]^T$. 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. $\mathcal{I}_w = \{I \subseteq [n] \text{ s.t. } |I| = w\}$. Let $W(\boldsymbol{a})$ be the hamming weight of $\boldsymbol{a}$. Let $\mathcal{V}_w$ be the set of n-bit binary vectors $\boldsymbol{q}$

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 $\mathsf{Corrupted}$ 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].

s.t. $W(\boldsymbol{q}) = w$, i.e. $\mathcal{V}_w = \{\boldsymbol{v} \in \{0,1\}^n \text{ s.t. } v_i = 1 \text{ iff } i \in \mathcal{I}_w\}$. For $\boldsymbol{q} = [q_1, \ldots, q_n]^T$ 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 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. $k = \sum_{i \in I} \lambda_i k_i$. 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. 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 $C_{t+1}(q_1,\ldots,q_n)\le Q$ 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 $n\ge 1$, 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 $t'\le t$ 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. $|\mathsf{B}|=t'\le t$, 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 $C_{t-t'+1}(q_1,\ldots,q_n)\le Q$ 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 Theorem 7 in Section A 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 $r\ge Q\ge N$. 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'. The following two theorems relate the non-gap versions of T-OMDH and OMDH, but equivalent statements hold for the gap versions of these assumptions as well.

Theorem 1. *(t', t, n, N, Q)-T-OMDH is equivalent to (N, Q)-OMDH for $t' = t$.*

Proof. If $t' = t$, the bound $C_{t-t'+1}(\boldsymbol{q})$ on Q simplifies to $\sum_{i \notin \mathsf{B}} q_i$, i.e. the bound on the number of g_j's for which $\mathcal{A}$ can compute $g_j^{p(0)}$ is the total number of $\mathcal{A}$'s queries to non-corrupted trustees.

Let $\mathcal{A}$ be an adversary that breaks the (t, t, n, N, Q)-T-OMDH assumption making total $Q = \sum_{i \notin \mathsf{B}} q_i$ queries, for some t-element set $\mathsf{B} = \{\alpha_1, ..., \alpha_t\}$ and an assignment $F : \mathsf{B} \to \mathbb{Z}_m$ of shares of corrupt trustees. Note that k, B, F define a unique t-degree polynomial $p(\cdot)$ s.t. $p(0) = k$ and $p(\alpha) = F(\alpha)$ for all $\alpha \in \mathsf{B}$. For any $i \in [n] \setminus \mathsf{B}$, let $\lambda_{i,0}, \ldots, \lambda_{i,t}$ be the Lagrange coefficients s.t. $p(i) = \lambda_{i,0} p(0) + \sum_{j=1}^{t} \lambda_{i,j} p(\alpha_j)$.

Consider reduction $\mathcal{R}$ which breaks (N, Q)-OMDH using $\mathcal{A}$ as follows: Given the challenge set $C = \{g_1, \ldots, g_N\}$ and Q accesses to oracle $(\cdot)^k$ for $k \leftarrow_{\mathrm{R}} \mathbb{Z}_m$, reduction $\mathcal{R}$ passes C to $\mathcal{A}$, and given $F : \mathsf{B} \to \mathbb{Z}_m$ and any (i, a) query of $\mathcal{A}$ to $\mathsf{TOMDH}_p(\cdot,\cdot)$, $\mathcal{R}$ queries $(\cdot)^k$ on a to get $b = a^k$ and sends $b' = b^{\lambda_{i,0}} \cdot a^{\lambda_{i,1} F(\alpha_1) + \ldots + \lambda_{i,t} F(\alpha_t)}$ to $\mathcal{A}$. In this way $\mathcal{R}$ consistently answers $\mathcal{A}$'s queries to $\mathsf{TOMDH}_p(\cdot,\cdot)$ for the unique t-degree polynomial $p(\cdot)$ s.t. $p(0) = k$ and $p(\alpha) = F(\alpha)$ for $\alpha \in \mathsf{B}$. Hence in particular if $\mathcal{A}$ wins, i.e. its output $\boldsymbol{V}$ includes $g_j^{p(0)}$ for at least $Q+1$ of g_j's, if $\mathcal{R}$ copies $\mathcal{A}$'s output then $\mathcal{R}$ will break its (N, Q)-OMDH challenge.

Theorem 2. *(t', t, n, N, Q)-T-OMDH is equivalent to (N, Q)-OMDH for $n = t + 1$.*

Proof. If $n = t + 1$ then shares $k_i = p(i)$ for $i \in [n] \setminus \mathsf{B}$ are uniformly random in $\mathbb{Z}_m$ and $p(0) = \sum_{i=1}^{n} \lambda_i p(i)$ for known constants λ_i. Note also that $C_{n-t'}(\boldsymbol{q}) = \min_{i \notin \mathsf{B}} q_i$, i.e. the bound on the number of g_j's for which $\mathcal{A}$ can compute $g_j^{p(0)}$ is the minimal number of queries the adversary makes to an uncorrupted trustee.

Let $\mathcal{A}$ be an adversary that breaks the (t', t, n, N, Q)-T-OMDH for $n = t+1$, making q_i queries to uncorrupted trustee $i \notin \mathsf{B}$ s.t. $\min_{i \notin \mathsf{B}} q_i \leq Q$, for some t'-element subset $\mathsf{B} \subseteq [n]$ and an assignment $F : \mathsf{B} \to \mathbb{Z}_m$ of shares of corrupt trustees. Consider reduction $\mathcal{R}$ which breaks (N, Q)-OMDH using $\mathcal{A}$ as follows: $\mathcal{R}$ passes C to $\mathcal{A}$, guesses the index $i^* \leftarrow_{\mathrm{R}} ([n] \setminus \mathsf{B})$ of the trustee whom $\mathcal{A}$ will query the least, picks shares $k_i \in \mathbb{Z}_m$ for $i \in [n] \setminus (\mathsf{B} \cup \{i^*\})$, sets $k_i = F(i)$

for $i \in \mathsf{B}$, and replies to each $\mathsf{TOMDH}_p(\cdot,\cdot)$ query (i,a) for $i \neq i^*$ with a^{k_i}, while given each query (i^*,a) reduction $\mathcal{R}$ queries oracle $(\cdot)^k$ to compute $b = a^k$ and replies to $\mathcal{A}$ with $(b \cdot a^{\sum_{i \neq i^*} \lambda_i k_i})^{1/\lambda_i}$. In this way $\mathcal{R}$ consistently answers $\mathcal{A}$'s queries to $\mathsf{TOMDH}_p(\cdot,\cdot)$ for the random $(n-1)$-degree polynomial p s.t. $p(0) = k$ and $p(i) = F(i)$ for $i \in \mathsf{B}$. If $\mathcal{R}$ guesses index i^* correctly then it makes at most $\min_i q_i = Q$ queries to $(\cdot)^k$, and if $\mathcal{A}$ computes $g_j^{p(0)} = g_j^k$ for at least $Q+1$ of g_j's then so does $\mathcal{R}$. Since $\mathcal{R}$ guesses i^* correctly with probability at least $1/n$, $\mathcal{R}$'s advantage against (N,Q)-OMDH is ϵ/n where ϵ is $\mathcal{A}$'s advantage against (t',t,n,N,Q)-T-OMDH for $n = t+1$.

T-OMDH vs. OMDH for General Threshold Parameters. It is less clear how to relate the T-OMDH and OMDH problems for any t', t, i.e. if $t' < t$, and $t < n-1$. To simplify the notation take $t' = 0$ and $n = 2(t+1)$. Adversary $\mathcal{A}$ breaks the T-OMDH assumption if, for example, it computes $(g_j)^k$ on $2s+1$ challenges after making s queries to $\mathsf{TOMDH}_p(i,\cdot)$ for each $i \in [n]$ where $k = p(0)$ and $k_i = p(i)$ for $i \in [n]$. (Note that if $\boldsymbol{q} = [s,\ldots,s]$ and $n = 2(t+1)$ then $C_{t+1}(\boldsymbol{q}) = 2s$, hence computing $(g_j)^k$ on $2s+1$ challenges breaks the assumption.) It is not clear how an efficient reduction $\mathcal{R}$ can break the OMDH problem given access to $\mathcal{A}$, because reduction $\mathcal{R}$ would seemingly have to satisfy the following constraints: (1) $\mathcal{R}$ would have to make only $2s$ queries to $(\cdot)^k$, but it would have to service s queries to $\mathsf{TOMDH}_p(i,\cdot)$ for each i, i.e. ns queries to $\mathsf{TOMDH}_p(\cdot,\cdot)$ in total, and $n = 2(t+1) \geq 4$; (2) $\mathcal{R}$ would presumably need to equate secret k in its OMDH challenge with value $p(0)$ in the T-OMDH challenge; (3) $\mathcal{R}$ would presumably need to answer each $\mathsf{TOMDH}_p(i,\cdot)$ query consistently, i.e. $\mathcal{R}$ has to reply to $\mathsf{TOMDH}_p(i,a)$ with a^{k_i} for some fixed vector of exponents $\boldsymbol{k} = [k_1,\ldots,k_n]^T$, because otherwise $\mathcal{A}$ can distinguish interaction with $\mathcal{R}$ from the real security game by checking if $(\mathsf{TOMDH}_p(i,a))^r = \mathsf{TOMDH}_p(i,a^r)$. Since a^r and a are independent group elements for $r \leftarrow_{\mathrm{R}} \mathbb{Z}_m$, it is not clear how $\mathcal{R}$ could detect $\mathcal{A}$'s queries which are designed to test if $\mathcal{R}$ responds to $\mathcal{A}$' TOMDH_p queries $(i,\cdot)$ with consistent answers; (4) Finally, values $(k_1,\ldots,k_n) = (p(1),\ldots,p(n))$ would need to satisfy linear constraints imposed by the polynomial of degree $t < n-1$, because $\mathcal{A}$ could test that $\mathcal{R}$'s responses to TOMDH_p queries satisfy these constraints, similarly as described above. Conditions (2-4) can be met e.g. if $\mathcal{R}$ picks $k_i = p(i)$ at random for $i = 1,\ldots,t$, and sets $p(i)$ for $i > t$ as a linear function of $k = p(0)$ and these first t values of $p(\cdot)$. But then it is not clear how $\mathcal{R}$ could reply to any $\mathsf{TOMDH}_p(i,\cdot)$ query for $i > t$ without querying $(\cdot)^k$, thus making $(n-t)s = (t+2)s > 2s$ queries to $(\cdot)^k$, violating condition (1).

3.3 Security Analysis of 2HashTDH

Protocol 2HashTDH protocol of Figure 3 is secure under the T-OMDH assumption. As a corollary of Theorem 1 from Section 3.2, Theorem 3 implies that protocol 2HashTDH is secure under OMDH in ROM in the full corruption case of $t' = t$. The proof of Theorem 3 appears in Appendix B.

Theorem 3. *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 Figure 5 we show a compiler which converts a T-OPRF scheme which realizes the UC T-OPRF notion of Section 2 into a PPSS scheme, called TOPPSS, which realizes UC PPSS functionality of [19].(We include this PPSS functionality for reference in Appendix D.) The terminology of the UC setting might obscure the amazing practicality of this construction, so in Section 5 we show a concrete implementation of this scheme with the $\mathcal{F}_{\text{TOPRF}}$ functionality implemented using the T-OPRF instantiation 2HashTDH from Section 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 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 Section 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,

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

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 Appendix C.

Theorem 4. *The TOPPSS scheme of Figure 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 Figure 3 in Section 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 section 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.

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.[5] In the *reconstruction phase*, the communication does not have to go over secure channels, because TOPPSS is 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

[5] 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.

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 $\{0,1\}^{2\ell}$, respectively.
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.

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 Figure 10 in Appendix D.

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.

6 Generic T-OPRF Construction from any OPRF

One can use generic Multi-Party Computation to convert *any* OPRF scheme into a Threshold OPRF protocol. The following is a blueprint for a T-OPRF with threshold parameters (t, n) given an OPRF scheme, a Message Authentication Code (MAC) scheme, and a generic MPC protocol:

(I) T-OPRF Initialization. The initialization runs a (t, n)-threshold MPC for the the U-S initialization protocol of the OPRF, where S's output state k is replaced by the secret-sharing $(k_1, ..., k_n)$ of k where each S_i receives k_i. In addition, each pair of servers (S_i, S_j) establishes a shared MAC key K_{ij}.

(II) T-OPRF Evaluation. The user's T-OPRF evaluation algorithm is as in the underlying OPRF, except that U broadcasts each message to all servers S_i. However, the server's evaluation algorithm is replaced by the following protocol. Let r_i be the randomness S_i chooses in its first protocol message. Then in each protocol round p the servers do the following: (1) $S_1, \ldots, S_n$ agree on the message $a^{(p)}$ which U sent in this protocol round as follows: For every i and j, server S_i sends a MAC on this message using key K_{ij} to S_j. S_j aborts if any server does not send a valid MAC on $a^{(p)}$ to S_j. (2) $S_1, \ldots, S_n$ run a (t, n)-threshold MPC protocol for computing S's response in p-th round of the OPRF protocol, given the public input U's messages $a^{(1)}, ..., a^{(p)}$ and server S's local input k, r. The

local input of S_i in this MPC is (k_i, r_i) where $(k_1, ..., k_n)$ is the secret-sharing of k and $r = r_1 \oplus ... \oplus r_n$. The MPC protocol computes S's response in the p-th round of the OPRF protocol, and this output is received by U.

When applying this transformation to the OPRF from Figure 2 where the only operation by the server is to raise the value a sent by the client to the power of k, we get a T-OPRF protocol where each server S_i first verifies the MAC's on value a from all other servers and then computes an exponentiation a^{k_i} where $k_1, \ldots, k_n$ is a secret sharing of k. Hence, this is the same protocol as 2HashTDH *except for the added MAC-verification round*. While a round of MAC broadcast would be computationally inexpensive, requiring an extra round of interaction would make this protocol less practical. However, while less efficient than 2HashTDH, the security of such generically constructed T-OPRF can be shown based on the same assumption needed for the base OPRF, namely, Gap-OMDH. Note that the PPSS scheme can be obtained from this T-OPRF, at the same cost and under the same assumptions, using the T-OPRF-to-PPSS compiler of Section 4.

References

1. Russian hackers amass over a billion internet passwords. *New York Times*, 08-06-2014. `http://goo.gl/aXzqj8`.
2. M. Abdalla, O. Chevassut, P.-A. Fouque, and D. Pointcheval. A simple threshold authenticated key exchange from short secrets. In *Advances in Cryptology – ASIACRYPT 2005*, pages 566–584. Springer, 2005.
3. M. Abdalla, M. Cornejo, A. Nitulescu, and D. Pointcheval. Robust password-protected secret sharing. In *Computer Security – ESORICS 2016*, pages 61–79. Springer, 2016.
4. A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. Password-protected secret sharing. In *ACM Conference on Computer and Communications Security CCS 2011*, pages 433–444. ACM, 2011.
5. M. Bellare, C. Namprempre, D. Pointcheval, M. Semanko, et al. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, 2003.
6. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology – EUROCRYPT 2000*, pages 139–155. Springer, 2000.
7. O. Blazy, C. Chevalier, and D. Vergnaud. Mitigating server breaches in password-based authentication: Secure and efficient solutions. In *Topics in Cryptology – CT-RSA 2016*, pages 3–18. Springer, 2016.
8. V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *Advances in Cryptology – EUROCRYPT 2000*, pages 156–171. Springer, 2000.
9. J. Brainard, A. Juels, B. Kaliski, and M. Szydlo. Nightingale: A new two-server approach for authentication with short secrets. In *USENIX Security Symposium - SECURITY 2003*, pages 201–213. IEEE, 2003.
10. J. Camenisch, A. Lehmann, A. Lysyanskaya, and G. Neven. Memento: How to reconstruct your secrets from a single password in a hostile environment. In *Advances in Cryptology – CRYPTO 2014*, pages 256–275. Springer, 2014.

11. J. Camenisch, A. Lehmann, and G. Neven. Optimal distributed password verification. In *ACM Conference on Computer and Communications Security - CCS 2015*, pages 182–194. ACM, 2015.
12. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE Symposium on Foundations of Computer Science - FOCS 2001*, pages 136–145. IEEE, 2001.
13. D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology – CRYPTO 2013*, pages 353–373. Springer, 2013.
14. M. Di Raimondo and R. Gennaro. Provably secure threshold password-authenticated key exchange. In *Advances in Cryptology – EUROCRYPT 2003*, pages 507–523. Springer, 2003.
15. W. Ford and B. S. Kaliski Jr. Server-assisted generation of a strong secret from a password. In *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises – WET ICE 2000*, pages 176–180. IEEE, 2000.
16. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.
17. S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Outsourced symmetric private information retrieval. In *ACM Conference on Computer and Communications Security - CCS 2013*, pages 875–888. ACM, 2013.
18. S. Jarecki, A. Kiayias, and H. Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In *Advances in Cryptology – ASIACRYPT 2014*, pages 233–253. Springer, 2014.
19. S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. Highly-efficient and composable password-protected secret sharing (Or: how to protect your bitcoin wallet online). In *IEEE European Symposium on Security and Privacy – EuroS&P 2016*, pages 276–291. IEEE, 2016.
20. S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF. In *Applied Cryptology and Network Security – ACNS 2017*, pages 39–58. Springer, 2017.
21. S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena. Device-enhanced password protocols with optimal online-offline protection. In *ACM Asia Conference on Computer and Communications Security – AsiaCCS 2016*, pages 177–188. ACM, 2016.
22. S. Jarecki and X. Liu. Fast secure computation of set intersection. In *Security and Cryptography for Networks – SCN 2010*, pages 418–435. Springer, 2010.
23. J. Katz, P. Mackenzie, G. Taban, and V. Gligor. Two-server password-only authenticated key exchange. In *Applied Cryptology and Network Security – ACNS 2005*, pages 1–16. Springer, 2005.
24. F. Kiefer and M. Manulis. Distributed smooth projective hashing and its application to two-server password authenticated key exchange. In *Applied Cryptology and Network Security – ACNS 2014*, pages 199–216. Springer, 2014.
25. F. Kiefer and M. Manulis. Universally composable two-server PAKE. In *Informational Security – ISC 2016*, pages 147–166. Springer, 2016.
26. P. D. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. In *Advances in Cryptology – CRYPTO 2002*, pages 385–400. Springer, 2002.

27. M. Naor, B. Pinkas, and O. Reingold. Distributed pseudo-random functions and KDCs. In *Advances in Cryptology – EUROCRYPT 1999*, pages 327–346. Springer, 1999.
28. V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology – EUROCRYPT 1997*, pages 256–266. Springer, 1997.
29. M. Szydlo and B. S. Kaliski. Proofs for two-server password authentication. In *Topics in Cryptology – CT-RSA 2005*, pages 227–244. Springer, 2005.
30. D. Wikström. Universally composable DKG with linear number of exponentiations. In *Security and Cryptography for Networks – SCN 2004*, pages 263–277. Springer, 2004.
31. X. Yi, F. Hao, L. Chen, and J. K. Liu. Practical threshold password-authenticated secret sharing protocol. In *Computer Security – ESORICS 2015*, pages 347–365. Springer, 2015.

A Threshold-OMDH Assumption in Generic Groups

We show the hardness of the (Gap) T-OMDH problem defined in Section 3.2 in the generic group model. First in Theorem 6 we argue the $t' = 0$ case of the non-gap version of this problem, then in Theorem 7 we extend it to general t', and in Theorem 8 we explain how the proof is adapted to the case of a gap group. To the best of our knowledge, this is also the first analysis of generic group hardness of the (gap) OMDH assumption.

Equivalence of (N, Q)-T-OMDH and $(Q{+}1, Q)$-T-OMDH. Note that the point which Bellare et al. [5] made about the One-More RSA assumption, namely that the (N, Q)-One-More problem for any $N > Q$ is equivalent to the (N, Q)-One-More problem for $N = Q + 1$, holds for the (Gap) T-OMDH problem as well, and it is a simple observation in this case because we specify the T-OMDH assumptions only in the context of a group of a known prime order. This implies in particular that in Theorems 6, 7, and 8 below it suffices to consider the case $N = Q + 1$. The theorem below is stated (and argued) for the non-gap version of the T-OMDH assumption, but the corresponding fact holds also in the case of the gap version of this assumption.

Theorem 5. *(t', t, n, N, Q)-T-OMDH is equivalent to $(t', t, n, Q{+}1, Q)$-T-OMDH.*

Proof. Given attacker $\mathcal{A}$ against (t', t, n, N, Q)-T-OMDH, reduction $\mathcal{R}$ attacks $(t', t, n, Q + 1, Q)$-T-OMDH as follows: On a challenge vector $[g_1, ..., g_{Q+1}]$, $\mathcal{R}$ picks N sequences of random linear coefficients $(\beta[i, 1], ..., \beta[i, Q + 1])$ in $\mathbb{Z}_m$, for $i = 1, ..., N$, sets $g'_i = g_1^{\beta[i,1]} \cdot ... \cdot g_{Q+1}^{\beta[i,Q+1]}$, and sends $[g'_1, ..., g'_N]$ as a challenge set to $\mathcal{A}$. $\mathcal{R}$ passes assignment F of shares of corrupted trustees as $\mathcal{A}$ chooses them, and answers its TOMDH oracle query of $\mathcal{A}$ by making the same query itself. Finally, if $\mathcal{A}$ outputs some $(Q{+}1)$-element subset $J \subset [n]$ and $v_j = (g'_j)^{p(0)}$ for each $j \in J$, then the $(Q{+}1)$-by-$(Q{+}1)$ matrix M s.t. the k-th row of M is $(\beta[j_k, 1], ..., \beta[j_k, Q{+}1])$, satisfies that $[g'_{j_1}, ..., g'_{j_{Q+1}}] = [g_1, ..., g_{Q+1}] \cdot M^T$, where matrix multiplication stands for exponentiation, i.e. $g'_{j_k} = g_1^{M[k,1]} \cdot ... \cdot g_{Q+1}^{M[k,Q+1]}$. Since β's are random in $\mathbb{Z}_m$ and g_j's are random in $\langle g \rangle$, the probability that M is

non-invertible is negligible (see [5]), in which case $[g_1, ..., g_{Q+1}] = [g'_{j_1}, ..., g'_{j_{Q+1}}] \cdot (M^{-1})^T$, and if $\mathcal{R}$ outputs $[z_1, ..., z_{Q+1}] = [v_{j_1}, ..., v_{j_{Q+1}}] \cdot (M^{-1})^T$ then $z_i = (g_i)^{p(0)}$ for each i.

Before moving on to the proof of the main theorem, we briefly review the definition and some basic properties of the Hadamard product of two vectors. Recall that the Hadamard product of $\boldsymbol{a} = [a_1, \ldots, a_n]^T$ and $\boldsymbol{b} = [b_1, \ldots, b_n]^T$ is defined as

$$\boldsymbol{a} \odot \boldsymbol{b} = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \odot \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ \vdots \\ a_n b_n \end{bmatrix} = \begin{bmatrix} b_1 & & \\ & \ddots & \\ & & b_n \end{bmatrix} \boldsymbol{a}.$$

From the rule above, we can see the following four properties:

(i) $\boldsymbol{a} \odot \left(\sum_{j=1}^{m} \boldsymbol{b}_j\right) = \sum_{j=1}^{m} (\boldsymbol{a} \odot \boldsymbol{b}_j)$,

(ii) $\boldsymbol{a} \odot (x\boldsymbol{b}) = x(\boldsymbol{a} \odot \boldsymbol{b})$ where x is a scalar,

(iii) If $\boldsymbol{k} \odot \boldsymbol{a} = \boldsymbol{0}$ and none of $\boldsymbol{k}$'s entries is 0, then $\boldsymbol{a} = \boldsymbol{0}$,

And by (i) and (iii) we also get:

(iv) If $\sum_{j=1}^{m} (\boldsymbol{k} \odot \boldsymbol{a}_j) = \boldsymbol{0}$ and none of $\boldsymbol{k}$'s entries is 0, then $\sum_{j=1}^{m} \boldsymbol{a}_j = \boldsymbol{0}$.

Lemma 1. *Let t be any positive integer. Then there does not exist n-dimensional vector $\boldsymbol{q}$ s.t.*

(1) $w \geq Qt$, and

(2) for any q_i $(i = 1, \ldots, n)$, $q_i \leq Q$,

where $w = W(\boldsymbol{q})$, and $Q = \lfloor \boldsymbol{q}/\mathcal{V}_t \rfloor + 1$.

Proof. We prove the proposition by induction on Q. If $Q = 1$, $C_t(\boldsymbol{q}) = 0$, which implies that there are at most $t - 1$ non-zero entries in $\boldsymbol{q}$; if $\boldsymbol{q}$ satisfies (2), then w is at most $t - 1$, so (1) cannot be satisfied.

Now suppose the proposition holds for $Q - 1$, but not for Q, i.e. there exists $\boldsymbol{q}$ which satisfies both (1) and (2). Note that $\boldsymbol{q}$ can have at most $t - 1$ entries that are larger than or equal to Q (otherwise those t entries that are larger than or equal to Q can be decreased Q times, so $C_t(\boldsymbol{q}) \geq Q$). Let $\boldsymbol{q'}$ be $\boldsymbol{q}$ with the largest t entries decreased, and $w' = W(\boldsymbol{q'})$. Then (1) $w' = w - t \geq (Q-1)t$, (2) for any q'_i $(i = 1, \ldots, n)$, $q'_i \leq Q - 1$ according to the above, and $C_t(\boldsymbol{q}) = Q - 2$. Therefore, $\boldsymbol{q'}$ is a counterexample for $Q - 1$, which contradicts our inductive hypothesis.

Lemma 2. *Let t be any non-negative integer, n be any positive integer, $\boldsymbol{q}$ be an n-dimensional vector, $w = W(\boldsymbol{q})$, $Q = C_{t+1}(\boldsymbol{q}) + 1$, and $\boldsymbol{k}$ be a w-dimensional vector where there are q_i i's as its entries $(i = 1, \ldots, n)$. Then for any w-dimensional vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_Q$, the set $V = \{\boldsymbol{k}^j \odot \boldsymbol{b}_i\}_{j \in \{0,\ldots,t\}, i \in [Q]}$ is linearly dependent.*

Proof. Let $M : w \times Q(t + 1)$ be the matrix whose columns are vectors in V. It is sufficient to show $rank(M) < Q(t + 1)$.

For $i = 1, \ldots, n$, there are q_i positions in $\boldsymbol{k}$ where the entry is i. Consider the corresponding rows in M; denote the $q_i \times Q(t+1)$ sub-matrix as M_i. Note that $rank(M_i) \leq Q$ since all its columns are multiples of its 1st, $[(t+1)+1]$-th, $\ldots$, $(Q-1)(t+1)$-th columns; therefore, for any $q_i > Q$, we can select Q rows of M_i forming matrix M_i' s.t. $rank(M_i') = rank(M_i)$. For all other q_i's, let $M_i' = M_i$. Let q_i' be the number of rows of M_i, i.e. for $q_i > Q$, let $q_i' = Q$, and for all other q_i's, let $q_i' = q_i$; then $q_i' \leq Q$ for all $i = 1, \ldots, n$. Let $w' = W(\boldsymbol{q'})$.

Now let $M' : w' \times Q(t+1)$ be the concatenation of $M_1, \ldots, M_n$. We can see that $rank(M') = rank(M)$. But $C_{t+1}(\boldsymbol{q'}) = C_{t+1}(\boldsymbol{q}) = Q - 1$. (The reason is as follows: obviously $C_{t+1}(\boldsymbol{q'}) \leq Q-1$. On the other hand, let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{Q-1} \in \mathcal{V}_{t+1}$ s.t. $\boldsymbol{v}_1 + \ldots + \boldsymbol{v}_{Q-1} \leq \boldsymbol{q}$. Clearly each entry of $\boldsymbol{v}_1 + \ldots + \boldsymbol{v}_{m-1}$ is at most $Q-1$; therefore, $\boldsymbol{v}_1 + \ldots + \boldsymbol{v}_{Q-1} \leq \boldsymbol{q'}$ since $\boldsymbol{q'}$ is simply $\boldsymbol{q}$ with entries greater than Q decreased to Q. That implies $C_{t+1}(\boldsymbol{q'}) \geq Q - 1$.) According to Lemma 1, we have $w' < Q(t+1)$. So $rank(W) = rank(W') \leq w' < Q(t+1)$.

Lemma 3. *Let t, n, $\boldsymbol{q}$, w, Q be the same with those in Lemma 2. Then there do not exist matrices $A : Q \times w$, $B : w \times Q$ and full-rank diagonal matrix $K : w \times w$ where there are q_i i's as its entries on the diagonal ($i = 1, \ldots, n$), s.t. $AB = I$ and $AKB = \ldots = AK^tB = O$, where I is the identity matrix and O is the zero matrix.*

Proof. Suppose $K = \begin{bmatrix} k_1 & & \\ & \ddots & \\ & & k_w \end{bmatrix}$. Let $\boldsymbol{a}_1^T, \ldots, \boldsymbol{a}_Q^T$ be the rows of A, $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_Q$ be the columns of B, and $\boldsymbol{k} = [k_1, \ldots, k_w]^T$. Then $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_Q, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_Q, \boldsymbol{k}$ are all w-dimension column vectors, and all entries of $\boldsymbol{k}$ are non-zero.

Let $\boldsymbol{k}^j$ denote $[k_1^j, \ldots, k_w^j]^T$, and $V = \{\boldsymbol{k}^j \odot \boldsymbol{b}_i\}_{j \in \{0,\ldots,t\}, i \in [Q]}$. The conditions $AB = I$ and $AKB = \ldots = AK^tB = O$ can be rewritten as

$$\boldsymbol{a}_i^T \boldsymbol{b} = \begin{cases} 1 & (\boldsymbol{b} = \boldsymbol{b}_i) \\ 0 & (\boldsymbol{b} \in V \setminus \{\boldsymbol{b}_i\}) \end{cases} (i \in [Q]).$$

Therefore, $\boldsymbol{b}_i$ cannot be linearly expressed by $V \setminus \{\boldsymbol{b}_i\}$.

We claim that $V \setminus \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_Q\} = \{\boldsymbol{k}^j \odot \boldsymbol{b}_i\}_{j \in [t], i \in [Q]}$ is linearly dependent. Otherwise since $\boldsymbol{b}_1$ cannot be linearly expressed by $V \setminus \{\boldsymbol{b}_1\}$, it cannot be linearly expressed by its subset $V \setminus \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_Q\}$ as well; therefore, adding $\boldsymbol{b}_1$ to $V \setminus \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_Q\}$, that is, $V \setminus \{\boldsymbol{b}_2, \ldots, \boldsymbol{b}_Q\}$, is still linearly independent. Similar with above, we can add $\boldsymbol{b}_2, \ldots, \boldsymbol{b}_Q$ to the set and remain its linear independency, i.e. V is also linearly independent. But this is impossible according to Lemma 2.

Since $\{\boldsymbol{k}^j \odot \boldsymbol{b}_i\}_{j \in [t], i \in [Q]}$ is linearly dependent, there exist x_{ij} ($j \in [t], i \in [Q]$), at least one of which is non-zero, s.t.

$$\sum_{j=1}^{t} \sum_{i=1}^{Q} x_{ij} \boldsymbol{k}^j \odot \boldsymbol{b}_i = \boldsymbol{0}.$$

Because none of $\boldsymbol{k}$'s entries is 0, we can derive

$$\sum_{j=1}^{t}\sum_{i=1}^{Q} x_{ij}\boldsymbol{k}^{j-1} \odot \boldsymbol{b}_i = \mathbf{0},$$

i.e.

$$\sum_{i=1}^{Q} x_{i1}\boldsymbol{b}_i + \sum_{j=1}^{t-1}\sum_{i=1}^{Q} x_{i,j+1}\boldsymbol{k}^{j} \odot \boldsymbol{b}_i = \mathbf{0}.$$

Recall that $\boldsymbol{b}_i$ cannot be linearly expressed by $V \setminus \{\boldsymbol{b}_i\}$, so $x_{11} = \ldots = x_{Q1} = 0$. We get

$$\sum_{j=1}^{t-1}\sum_{i=1}^{Q} x_{i,j+1}\boldsymbol{k}^{j} \odot \boldsymbol{b}_i = 0.$$

Using the same steps above, we can see $x_{12} = \ldots = x_{Q2} = 0$, then $x_{13} = \ldots = x_{Q3} = 0$, ..., finally $x_{1Q} = \ldots = x_{QQ} = 0$. That is, all x_{ij}'s ($j \in [t], i \in [Q]$) are 0, which contradicts the claim above.

Theorem 6. (generic group hardness of $(0, t, n, Q+1, Q)$-T-OMDH) *Let $\langle g \rangle$ be a generic group of prime order m. We use $\xi(a)$ for $a \in \mathbb{Z}_m$ to denote elements in $\langle g \rangle$, where $\xi(\cdot)$ is a random $1-1$ function mapping $\mathbb{Z}_m$ to bitstrings of sufficient size. Let $\mathcal{A}$ be an algorithm which can query the following two oracles:*

- *Group operation oracle, which on input $(\xi(a_1), \xi(a_2))$ and an operation, either $+$ or $-$, respectively outputs $\xi(a_1 + a_2)$ or $\xi(a_1 - a_2)$;*
- *Oracle $\mathsf{TOMDH}_p(\cdot,\cdot)$, where $p(\cdot)$ is a t-degree polynomial over $\mathbb{Z}_m$, which on input $(k, \xi(a))$ for $k \in [n]$ outputs $\xi(a \cdot p(k))$.*

If $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{TOMDH}_p(\cdot,\cdot)}(t, n, Q, r, m)$ is the probability that $\mathcal{A}(\xi(1), \xi(u_1), \ldots, \xi(u_{Q+1}))$ outputs $(\xi(u_1 \cdot p(0)), \ldots, \xi(u_{Q+1} \cdot p(0)))$ after making r group operation queries and q_i queries to $\mathsf{TOMDH}_p(i, \cdot)$ s.t. $C_{t+1}(\boldsymbol{q}) \leq Q$, then

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{TOMDH}_p(\cdot,\cdot)}(t, n, Q, r, m) \leq \frac{(w+1)(w+Q+r+2)^2 + 4}{2m},$$

where $w = W(\boldsymbol{q})$ (i.e. w is the total number of queries to the T-OMDH oracle), and the probability goes over the random choice of t-degree polynomial p, the randomness of $\mathcal{A}$ and the randomness of oracle $\xi(\cdot)$.

Proof. The proof goes by construction of an algorithm $\mathcal{B}$ which simulates the real challenger while interacting with $\mathcal{A}$. $\mathcal{B}$ maintains a list $T := \{(F_s, \xi_s)\}_{s=1,\ldots,\sigma}$, where $F_s(U_1, \ldots, U_{Q+1}, A_0, A_1, \ldots, A_t)$ is a polynomial of degree at most w, and ξ_s's are random distinct elements in $\langle g \rangle$. At the beginning, $\mathcal{B}$ sets $\sigma := Q + 2$ and initializes T by setting $F_1 := 1, F_2 := u_1, \ldots, F_{Q+2} := u_{Q+1}$, and picks $\xi_1, \ldots, \xi_{Q+2}$ as random distinct elements in $\langle g \rangle$ and $a_0, a_1, \ldots, a_t \leftarrow_{\mathrm{R}} \mathbb{F}$. $\mathcal{B}$ sends $\xi_1, \ldots, \xi_{Q+2}$ to $\mathcal{A}$ as $\xi(1), \xi(u_1), \ldots, \xi(u_{Q+1})$. Then $\mathcal{A}$ can make the following two types of queries to $\mathcal{B}$ (we assume that $\mathcal{A}$ only makes oracle queries on values that are previously obtained from $\mathcal{B}$):

– group operation query: $\mathcal{A}$ inputs two indices s_1 and s_2, as well as an operation (either a multiplication or a division). $\mathcal{B}$ computes $F_{\sigma+1} := F_{s_1} + F_{s_2}$ if the operation is a multiplication or $F_{\sigma+1} := F_{s_1} - F_{s_2}$ if the operation is a division. If $\exists\ t \leq \sigma$ s.t. $F_s = F_{\sigma+1}$, then $\mathcal{B}$ outputs ξ_t to $\mathcal{A}$. Otherwise $\mathcal{B}$ picks a group element $\xi_{\sigma+1}$ which is different from ξ_i for all $i \leq \sigma$, outputs $\xi_{\sigma+1}$ to $\mathcal{A}$, and sets σ++.
– $\mathsf{TOMDH}_p(\cdot,\cdot)$ oracle query: $\mathcal{A}$ inputs a $k \in [n]$ and an index $s \in [\sigma]$. Then $\mathcal{B}$ sets $F_{\sigma+1} := (\boldsymbol{L}_k^T \boldsymbol{a}) \cdot F_s$ and $\xi_{\sigma+1}$ to a random value which is different from ξ_s for all $s \leq \sigma$, outputs $\xi_{\sigma+1}$ to $\mathcal{A}$, and sets σ++, where $\boldsymbol{L}_k = [1, k, \ldots, k^t]^T$ and $\boldsymbol{a} = [a_0, a_1, \ldots, a_t]^T$.

$\mathcal{A}$ finally outputs $(F_{s_1}, \ldots, F_{s_{Q+1}})$, and it wins if $F_{s_i} = u_i \cdot a_0$ for all $i \in [Q+1]$.

Now we analyze the probability that $\mathcal{A}$ succeeds for a random assignment of $(u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t)$. First, note that the output of $\mathcal{A}$ comes from the two types of oracle queries listed above; therefore, for every $s \in \{s_1, \ldots, s_{Q+1}\}$, F_s is a linear combination of $v_1, \ldots, v_w$, $u_1, \ldots, u_{Q+1}$ and 1, where v_i $(i = 1, \ldots, w)$ is the value $\mathcal{A}$ obtains from the ith $\mathsf{TOMDH}_p(\cdot,\cdot)$ oracle query. That is,

$$F_s = \sum_{i=1}^{w} \alpha_i^{(s)} v_i + \sum_{i=0}^{Q+1} \gamma_i^{(s)} u_i,$$

where

$$v_i = \sum_{Z \subseteq [i] s.t. i \in Z} \left[\left(\sum_{j=0}^{Q+1} \beta_{jZ}^{(i)} u_j \right) \prod_{l \in Z} (\boldsymbol{L}_{k_l}^T \boldsymbol{a}) \right]$$

(we set $u_0 = 1$ in the two equations above), where α_i's, γ_i's and β_{jZ}'s are all elements in $\mathbb{Z}_m$ specified by $\mathcal{A}$. (Suppose that in the ith $\mathsf{TOMDH}_p(\cdot,\cdot)$ oracle query, $\mathcal{A}$'s second input is k_i; then $\boldsymbol{L}_{k_i}^T \boldsymbol{a}$ must appear in the output. That is why $i \in Z$ holds in expression of w_i.)

Recall that $\mathcal{A}$ wins if and only if $F_{s_i} = u_i \cdot a_0$ for all $i = 1, \ldots, Q+1$. Suppose that there exists an $i \in [Q+1]$ s.t. $\deg(F_{s_i}) > 1$ but $\mathcal{A}$ still wins, i.e. $F_{s_i} = u_i \cdot a_0$ (we view F_{s_i} as a polynomial of a_0 here). Since $\deg(u_i \cdot a_0) = 1$, the only possibility that this occurs is that the polynomials F_{t_i} and $u_i \cdot a_0$ evaluates to the same value for random $u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t$. Also note that $\deg(F_{s_i}) \leq w$, thus $\deg(F_{s_i} - u_i \cdot a_0) \leq w$; therefore, if the above occurs for a certain i, either (i) $1 \leq \deg(F_{s_i} - u_i \cdot a_0) \leq w$, and random a_0 is a solution of $F_{s_i} - u_i \cdot a_0$, or (ii) $F_{s_i} - u_i \cdot a_0$ is zero polynomial for fixed $u_1, \ldots, u_{Q+1}$ chosen from random. The probability of (i) is at most w/m, while the probability of (ii) is at most $1/p$. Since there are $Q+1$ possible values of i, the probability that there exists an $i \in [Q+1]$ s.t. $\deg(F_{s_i}) > 1$ but $\mathcal{A}$ still wins is at most $\frac{(w+1)(Q+1)}{m}$.

Next consider the case where $\deg(F_{s_i}) \leq 1$ for all $i = 1, \ldots, Q+1$. Let v_i' be v_i with all terms whose degree greater than 1 eliminated, that is, v_i' only remains

the single term in v_i where $Z = \{i\}$, i.e.

$$v_i' = \left(\sum_{j=0}^{Q+1} \beta_{j\{1\}}^{(i)} u_j \right) (\boldsymbol{L}_{k_i}^T \boldsymbol{a}).$$

Then

$$F_s = \sum_{i=1}^{w} \alpha_i^{(s)} v_i' + \sum_{i=0}^{w+1} \gamma_i^{(s)} u_i.$$

We rewrite the expression of F_s in matrix form below. Note that since $\deg(v_i') \leq 1$, all $\beta_{jZ}^{(i)}$'s for $Z \neq \{1\}$ does not appear in the expression of v_i'; therefore, we denote $\beta_j^{(i)}$ as $\beta_{j\{1\}}^{(i)}$:

$$\begin{bmatrix} F_{s_1} \\ \vdots \\ F_{s_{Q+1}} \end{bmatrix} = A \begin{bmatrix} v_1' \\ \vdots \\ v_w' \end{bmatrix} + C\boldsymbol{u} + \begin{bmatrix} \gamma_0^{(s_1)} \\ \vdots \\ \gamma_0^{(s_{Q+1})} \end{bmatrix}, \tag{1}$$

$$\begin{bmatrix} v_1' \\ \vdots \\ v_w' \end{bmatrix} = (B\boldsymbol{u} + \boldsymbol{b}_0) \odot \begin{bmatrix} \boldsymbol{L}_{k_1}^T \boldsymbol{a} \\ \vdots \\ \boldsymbol{L}_{k_w}^T \boldsymbol{a} \end{bmatrix}, \tag{2}$$

where

$$A = \begin{bmatrix} \alpha_1^{(s_1)} & \dots & \alpha_w^{(s_1)} \\ \vdots & & \vdots \\ \alpha_1^{(s_{Q+1})} & \dots & \alpha_Q^{(s_{Q+1})} \end{bmatrix}_{(Q+1)\times w}, B = \begin{bmatrix} \beta_1^{(1)} & \dots & \beta_{Q+1}^{(1)} \\ \vdots & & \vdots \\ \beta_1^{(w)} & \dots & \beta_{Q+1}^{(w)} \end{bmatrix}_{w\times(Q+1)},$$

$$C = \begin{bmatrix} \gamma_1^{(1)} & \dots & \gamma_{Q+1}^{(1)} \\ \vdots & & \vdots \\ \gamma_1^{(Q+1)} & \dots & \gamma_{Q+1}^{(Q+1)} \end{bmatrix}_{(Q+1)\times(Q+1)}, \boldsymbol{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_{Q+1} \end{bmatrix}, \boldsymbol{b}_0 = \begin{bmatrix} \beta_0^{(1)} \\ \vdots \\ \beta_0^{(w)} \end{bmatrix}.$$

Let $\boldsymbol{b} = B\boldsymbol{u} + \boldsymbol{b}_0$. Substituting (2) into (1), we get

$$\begin{bmatrix} F_{s_1} \\ \vdots \\ F_{s_{Q+1}} \end{bmatrix} = A \left(\boldsymbol{b} \odot \begin{bmatrix} \boldsymbol{L}_{k_1}^T \boldsymbol{a} \\ \vdots \\ \boldsymbol{L}_{k_w}^T \boldsymbol{a} \end{bmatrix} \right) + C\boldsymbol{u} + \begin{bmatrix} \gamma_0^{(s_1)} \\ \vdots \\ \gamma_0^{(s_{Q+1})} \end{bmatrix}.$$

Note that

$$\begin{bmatrix} \boldsymbol{L}_{k_1}^T \boldsymbol{a} \\ \vdots \\ \boldsymbol{L}_{k_w}^T \boldsymbol{a} \end{bmatrix} = \begin{bmatrix} a_0 + a_1 k_1 + \ldots + a_t k_1^t \\ \vdots \\ a_0 + a_w k_w + \ldots + a_t k_w^t \end{bmatrix} = a_0 \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} + a_1 \begin{bmatrix} k_1 \\ \vdots \\ k_w \end{bmatrix} + \ldots + a_t \begin{bmatrix} k_1^t \\ \vdots \\ k_w^t \end{bmatrix},$$

using the properties of Hadamard product:

$$\begin{bmatrix} F_{s_1} \\ \vdots \\ F_{s_{Q+1}} \end{bmatrix} = A(a_0 I\boldsymbol{b} + a_1 K\boldsymbol{b} + \ldots + a_t K^t\boldsymbol{b}) + C\boldsymbol{u} + \begin{bmatrix} \gamma_0^{(s_1)} \\ \vdots \\ \gamma_0^{(s_{Q+1})} \end{bmatrix}$$

$$= a_0 A\boldsymbol{b} + a_1 AK\boldsymbol{b} + \ldots + a_t AK^t\boldsymbol{b} + C\boldsymbol{u} + \begin{bmatrix} \gamma_0^{(s_1)} \\ \vdots \\ \gamma_0^{(s_{Q+1})} \end{bmatrix},$$

$$\text{where } K = \begin{bmatrix} k_1 & & \\ & \ddots & \\ & & k_w \end{bmatrix}_{w \times w}.$$

View the right side as a linear function of $a_0, a_1 \ldots, a_t$, that is, consider the right side for some fixed random $u_1, \ldots, u_{w+1}$. If $\mathcal{A}$ wins, then

$$\begin{bmatrix} F_{s_1} \\ \vdots \\ F_{s_{Q+1}} \end{bmatrix} = a_0\boldsymbol{u},$$

comparing the two equations, we have

$$A\boldsymbol{b} = \boldsymbol{u}, AK\boldsymbol{b} = \ldots = AK^t\boldsymbol{b} = \boldsymbol{0},$$

except for the case where the two linear functions evaluate to the same value for random $a_0, a_1, \ldots, a_t \in \mathbb{Z}_m$. The probability that this occurs is at most $1/p$.

Implementing the definition of $\boldsymbol{b}$ back, we get

$$AB\boldsymbol{u} + A\boldsymbol{b}_0 = \boldsymbol{u}, \text{ i.e. } (AB - I)\boldsymbol{u} + A\boldsymbol{b}_0 = \boldsymbol{0},$$

$$AKB\boldsymbol{u} + AK\boldsymbol{b}_0 = \ldots = AK^tB\boldsymbol{u} + AK^t\boldsymbol{b}_0 = \boldsymbol{0}.$$

According to Lemma 3, there is at least one of $AB - I, AKB, \ldots, AK^tB$ which is not O. Then there exists at least one row of one of the matrices above which is not $\boldsymbol{0}$; let it be the i-th row of the t_0-th matrix ($t_0 \in \{0, \ldots, t\}$). Denote that row as $\boldsymbol{z}^T$. Then we have

$$\boldsymbol{z}^T\boldsymbol{u} + AK^{t_0}\beta_0^{(i)} = \boldsymbol{0}$$

for random $u_1, \ldots, u_{Q+1} \in \mathbb{Z}_m$. The probability that this occurs is at most $1/p$.

In sum, we have proved that the probability that $\mathcal{A}$ succeeds in the game interacting with $\mathcal{B}$ is at most

$$\frac{(w+1)(Q+1)}{m} + \frac{1}{m} + \frac{1}{m} = \frac{(w+1)(Q+1)+2}{m}.$$

The difference between $\mathcal{A}$'s views in the interaction with $\mathcal{B}$ and with the real challenger of the T-OMDH assumption in the generic group model appears when there exist s_1 and s_2 s.t. $F_{s_1}(u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t) = F_{s_2}(u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t)$ for random $u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t$, but the polynomials F_{s_1} and F_{s_2} are not the same. There are $\binom{\sigma}{2}$ possible (s_1, s_2) pairs, and for each such pair, the probability that $F_{s_1}(u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t) = F_{s_2}(u_1, \ldots, u_{Q+1}, a_0, a_1, \ldots, a_t)$ is at most $(w+1)/m$. Thus, the probability that the event above occurs is at most $\binom{\sigma}{2} \cdot (w+1)/m$. Also note that each time $\mathcal{A}$ queries one of the two oracles, σ either remains the same or increases by 1; there are at most $r + w$ such queries, and $\sigma = Q + 2$ at the beginning. So $\sigma \leq w + Q + r + 2$.

Therefore, we have proved that the probability that $\mathcal{A}$ breaks the T-OMDH assumption where $t' = 0$ in the generic group model is at most

$$\frac{(w+1)(Q+1)+2}{m} + \binom{w+Q+r+2}{2} \cdot \frac{w+1}{m} \leq \frac{(w+1)(w+Q+r+2)^2+4}{2m}.$$

□

Next we consider the general case, i.e. $t' > 0$ and $\mathsf{B} \neq \varnothing$:

Theorem 7. (generic group hardness of $(t', t, n, Q+1, Q)$-T-OMDH) *Let $\langle g \rangle$ and $\xi(\cdot)$ be the same with those in Theorem 6. Let B be any t-element subset of $[n]$. $\mathcal{A}$ is given $\xi(1), \xi(u_1), \ldots, \xi(u_{Q+1})$. Then $\mathcal{A}$ outputs a map $F : \mathsf{B} \to \mathbb{Z}_m$. After that, $\mathcal{A}$ can query the group operation oracle and the T-OMDH oracle as in Theorem 6, with the exception that $p(\cdot)$ is a t-degree polynomial over $\mathbb{Z}_m$ s.t. $p(\alpha) = F(\alpha)$ for all $\alpha \in \mathsf{B}$. $\mathcal{A}$ wins if it outputs $(\xi(u_1 \cdot p(0)), \ldots, \xi(u_{Q+1} \cdot p(0)))$.*

The bound on adversarial advantage, i.e. on probability that $\mathcal{A}(\xi(1), \xi(u_1), \ldots, \xi(u_{Q+1}))$ outputs $(\xi(u_1 \cdot p(0)), \ldots, \xi(u_{Q+1} \cdot p(0)))$ after making r group operation queries and q_i queries to $\mathsf{TOMDH}_p(i, \cdot)$ for $i \notin \mathsf{B}$ s.t. $C_{t-t'+1}(\boldsymbol{q}) \leq Q$, is exactly the same as the bound on the adversarial advantage in Theorem 6.

Proof. The proof is an extension of that of Theorem 6, so we only provide a sketch. At the beginning of the simulated game, $\mathcal{A}$ chooses $F(\alpha) = \boldsymbol{L}_i^T \boldsymbol{a}$ $(\alpha \in \mathsf{B})$. Since $\mathcal{A}$ knows t' of $\boldsymbol{L}_i^T \boldsymbol{a}$'s, there are $t+1$ variables and t' linear equations (which are linearly independent), so there are $t - t' + 1$ free variables; in particular, $a_0, \ldots, a_{t-t'}$ are still independently random from $\mathcal{A}$'s view. The whole argument holds until the following step:

$$A\boldsymbol{b} = \boldsymbol{u}, AK\boldsymbol{b} = \ldots = AK^t\boldsymbol{b} = \boldsymbol{0},$$

where t should be replaced by $t - t'$. After that, the argument still holds if we replace t by $t - t'$. □

Theorem 8. *Suppose furthermore that $\langle g \rangle$ is a gap group, that is, $\mathcal{A}$ can make the following type of oracle queries as well:*

- *DDH oracle, which on input $\xi(a_1), \xi(a_2), \xi(a_3), \xi(a_4)$ which are different with each other, outputs 1 if $a_1 a_4 = a_2 a_3$, and 0 otherwise. $\mathcal{A}$ can make such queries at most q times.*

Then the bound on adversarial advantage of Theorem 7 still holds, with an upper-bound modified to $\mathsf{Adv}_{\mathcal{A}} \leq \frac{(w+1)(w+Q+r+2)^2+4}{2m} + \frac{(2w+1)q}{m}$.

Proof. We construct an algorithm $\mathcal{B}'$ which is exactly the same with $\mathcal{B}$ in the previous proof, except that $\mathcal{A}$ can make the following oracle queries to $\mathcal{B}'$ as well:

- DDH oracle query: $\mathcal{A}$ inputs four different indices s_1, s_2, s_3 and s_4. $\mathcal{B}'$ outputs 1 to $\mathcal{A}$ if $F_{s_1}F_{s_4} = F_{s_2}F_{s_3}$, and 0 otherwise.

Now the difference of $\mathcal{A}$'s views in the interaction with $\mathcal{B}$ and in the interaction with the real challenger appears in addition when there exist four different s_1, s_2, s_3, s_4 s.t.

$$F_{s_1}(u_1,\ldots,u_{Q+1},a_0,a_1,\ldots,a_t)F_{s_4}(u_1,\ldots,u_{Q+1},a_0,a_1,\ldots,a_t) = F_{s_2}(u_1,\ldots,u_{Q+1},a_0,a_1,\ldots,a_t)F_{s_3}(u_1,\ldots,u_{Q+1},a_0,a_1,\ldots,a_t)$$

for random $u_1,\ldots,u_{Q+1},a_0,a_1,\ldots,a_t$, but the polynomials $F_{s_1}F_{s_4}$ and $F_{s_2}F_{s_3}$ are not the same. Note that $\deg(F_{s_1}F_{s_4} - F_{s_2}F_{s_3}) < 2w$, and there are at most q such polynomials evaluated, so the probability that the above event occurs is at most $(2w+1)q/m$. All other proof steps remain as in the proof of Theorem 7. □

B Proof of Theorem 3 of Section 3.3

Proof. We present a simulator SIM in Figure 7, and we show that for any polynomial $\mathcal{Z}$, SIM creates a view indistinguishable from the real world. Without loss of generality, we assume that $\mathcal{A}$ is a "dummy" adversary that merely passes over all messages it gets to $\mathcal{Z}$.

The simulator SIM operates as follows. In general it follows the real world protocol execution with the following important differences. First, it selects a number of random group elements $g_1,\ldots,g_N$ ahead of time and stores them. It uses those elements to respond to $H_1(\cdot)$ queries (as opposed to answering such queries on the fly, as it happens in the real world execution with a random oracle). Second, in the case of queries to the $H_2(\cdot)$ oracle, they are answered on the fly except when it happens that they are of the relevant form $(x, H_1(x)^k)$ for some arbitrary k and an x which is already defined in the $H_1(\cdot)$ table; note that this is detected by SIM due to the fact that it has full control the $H_1(\cdot)$ table. For such queries the simulator uses the $\mathcal{F}_{\text{TOPRF}}$ interface to answer them employing suitably the table pointers p (either by creating new ones or by utilizing ones that have been defined already). This evaluation is also the only moment where the simulator may fail. A fail event happens when $\mathcal{F}_{\text{TOPRF}}$ refuses to answer a request for evaluation by the simulator. One of the important features of SIM is the recording of triples of the form (p, g_d, g_d^k) for relevant vales of the form $y_p = g^k$ which occur during the execution. In the analysis that follows we will show that the failure event will imply the computation of sufficiently many such triples so that the T-OMDH assumption is violated.

1. Pick $r_1, \ldots, r_N \leftarrow_{\mathrm{R}} \mathbb{Z}_m$, and set $g_1 := g^{r_1}, \ldots, g_N := g^{r_N}$. Set counter $D := 1$.
2. Every time when there is a fresh query x to $H_1(\cdot)$, answer it with g_D, record (x, r_D, g_D), and set $D{+}{+}$.
3. Upon receiving $(\textsc{Init}, sid, S, \mathcal{SI} = \{S_1, \ldots, S_n\}, p)$ from $\mathcal{F}_{\mathrm{TOPRF}}$, if no record $\langle sid, \ldots\rangle$ exists, mark S as active, pick $k \leftarrow_{\mathrm{R}} \mathbb{F}$, record $\langle sid, S, \mathcal{SI}, p, k\rangle$, define $y_p := g^k$ and send $(\textsc{Init}, sid, S, \mathcal{SI})$ to $\mathcal{A}$ using the interface of $\mathcal{F}_{\mathrm{DKG}}$. In the other case, mark S as active and send $(\textsc{Init}, sid, S, \mathcal{SI})$ to $\mathcal{A}$ using the interface of $\mathcal{F}_{\mathrm{DKG}}$.
4. Upon receiving $(\textsc{Init}, sid, \mathcal{SI}, s)$ in the $\mathcal{F}_{\mathrm{DKG}}$ interface from a corrupted S, if a record of the form $\langle sid, P, \mathcal{SI}, p, \ldots\rangle$ exists, mark S as active, record $\langle \mathcal{A}, S, s\rangle$ and send $(\textsc{Init}, sid, \mathcal{A}, S)$ to $\mathcal{A}$.
5. Upon receiving $(\textsc{InitComplete}, sid, S)$ in the $\mathcal{F}_{\mathrm{DKG}}$ interface from $\mathcal{A}$, check that S is active, retrieve $\langle sid, P, \mathcal{SI}, p, k\rangle$, check if all servers in $\mathcal{SI}$ are active and $S \in \mathcal{SI}$, and if no record $\langle sid, \mathcal{SI}, k_1, \ldots, k_n\rangle$ exists, create it by setting $k_i = p(i)$ for each $S_i \in \mathcal{SI}$ where p is a random polynomial subject to the restriction $p(i) = s_i$ for each record of the form $\langle \mathcal{A}, S_i, s_i\rangle$. Finally, record $\langle sid, S_i, \mathcal{SI}, i, k_i\rangle$ and send $(\textsc{InitComplete}, sid, S)$ to $\mathcal{F}_{\mathrm{TOPRF}}$.
6. On $(\textsc{Eval}, sid, ssid, U, \mathcal{SE})$ from $\mathcal{F}_{\mathrm{TOPRF}}$, record $\langle U, sid, ssid, \mathcal{SE}, r_D, g_D\rangle$, send $(\textsc{Send}, (sid, ssid, 1), U, S, (\mathcal{SE}, g_D))$ to $\mathcal{A}$ for each $S \in \mathcal{SE}$, and set $D{+}{+}$.
7. On $(\textsc{Sent}, (sid, ssid, 1), U, S_i, (\mathcal{SE}, a))$ from $\mathcal{A}$, recover $(sid, S_i, \mathcal{SI}, i, k_i)$ (ignore if it does not exist), compute interpolation coefficient λ_i corresponding to index i and subset $\mathcal{SE}$ of set $\mathcal{SI}$, set $b_i := a^{\lambda_i k_i}$, and send $(\textsc{Send}, (sid, ssid, 2), U, b_i)$ to $\mathcal{A}$ and $(\textsc{SndrComplete}, sid, ssid, S_i)$ to $\mathcal{F}_{\mathrm{TOPRF}}$.
8. When $(\textsc{Sent}, (sid, ssid, 2), S_{i_j}, U, b_j)$ for all $S_j \in \mathcal{SE}$ defined in a record $\langle U, sid, ssid, \mathcal{SE}, r_d, g_d\rangle$ have been received from $\mathcal{A}$, compute $b := \prod_{S_j} b_j$ and find p s.t. $b^{1/r_d} = y_p$. If such p does not exist choose a unique label p and set $y_p := b^{1/r_d}$ and send $(\textsc{Init}, sid, \mathcal{A}^*, p)$ to $\mathcal{F}_{\mathrm{TOPRF}}$. Record (p, g_d, b) and send $(\textsc{RcvComplete}, sid, ssid, U, p)$ to $\mathcal{F}_{\mathrm{TOPRF}}$.
9. Every time when there is a fresh query (x, u) to $H_2(\cdot, \cdot)$, if there is no tuple (x, r_d, g_d) stored in step 2 then set $H_2(x, u)$ to a random string in $\{0,1\}^\ell$ and return this value. Otherwise, if there is p, s.t. $y_p = u^{1/r_d}$ then store (p, g_d, u). If it does not exist choose unique label p, set $y_p := u^{1/r_d}$, and send $(\textsc{Init}, sid, \mathcal{A}^*, p)$ to $\mathcal{F}$. Pick a new unique label $ssid$. Send $(\textsc{SndrComplete}, sid, ssid, S)$ for the t' malicious servers S, followed by $(\textsc{Eval}, sid, ssid, (\mathcal{A}^*)^{t+1}, x)$ to $\mathcal{F}$ followed by $(\textsc{RcvComplete}, sid, ssid, \mathcal{A}^*, p)$. If $\mathcal{F}$ ignores this message abort and output $\textsc{Fail}$. Otherwise after receiving $\mathcal{F}$'s response $(\textsc{Eval}, sid, ssid, \rho)$, set $H_2(x, u) := \rho$.

Fig. 7. The Simulator SIM for the Threshold 2HashTDH Protocol.

Based on the way the simulation is defined, we conclude that if FAIL does not happen, $\mathcal{Z}$'s view in the real world and the simulated world are indistinguishable. Now we upper-bound Pr[FAIL]. Recall that FAIL occurs in the first bullet of step 9, when $H_2(x,u)$ is queried, and there exists triples (x, r_d, g_d) and $\langle sid, \mathcal{SI}, p, k\rangle$ stored previously such that $u = g_d^k$. Therefore, every triggering of FAIL is associated to a specific session sid. It follows that we can bound the distance between real and ideal world using q_T hybrids and the reduction to the T-OMDH assumption which works for a given sid as follows:

On input of a (Gap) T-OMDH instance $(Q, g, g_1, ..., g_N)$, initialize $q_1, \ldots, q_n$ to 0 and revise algorithm SIM to interact with oracle TOMDH_p as follows:

- In step 1, only set $D := 1$ and omit all other steps.
- In step 2 and step 6, use the instance elements $g_1, \ldots, g_N$ instead of random elements in $\langle g\rangle$ as $g_1, \ldots, g_N$. Furthermore, since there is no r_D, only record (x, g_D) in step 2 and $\langle U, sid, ssid, \mathcal{SE}, g_D\rangle$ in step 6.
- In step 3, omit the choice of k and record simply $\langle sid, S, \mathcal{SI}, p\rangle$ (omitting k).
- In step 4, when it receives $(\text{INIT}, sid, \mathcal{SI}, s)$ from server S_i and server S_i is marked as active, add s to the i-th location of the F vector. When F has all corrupt, t' in number, locations complete, set σ to be the complete view of the SIM so far, generate (F, σ) in a special (private) output tape.
- In step 5, in the case that no record $\langle sid, \mathcal{SI}, k_1, \ldots, k_n\rangle$ exists, send $(0, g)$ in order to receive the public-key g^k from the $\mathsf{TOMDH}_p(\cdot,\cdot)$ oracle and record $y_p := (g, g^k)$ where p is recorded in step 3 inside the record $\langle sid, S, \mathcal{SI}, p\rangle$.
- In step 6, the value r_D is not recorded (it is unknown).
- In step 7, use a (i, a) query to the $\mathsf{TOMDH}_p(\cdot,\cdot)$ oracle in order to compensate for the lack of knowledge of k_i values for honest servers S_i. Set q_i++.
- In step 8, if a new pointer p^* is to be created, the value $y_p = b^{1/r_d}$ cannot be computed, therefore we store instead $y_p = (g_d, b)$. Testing the predicate "$b^{1/r_d} = y_{p^*}$" for some pointer $y_{p^*} = \langle g_{d^*}, b^*\rangle$ can be accomplished by using the $\mathsf{DDH}(\cdot,\cdot,\cdot,\cdot)$ oracle on (g_{d^*}, b^*, g_d, b).
- In step 9, finding p such that $y_p = u^{1/r_d}$ where $y_p = \langle a_p, b_p\rangle$, can be done by using the $\mathsf{DDH}(\cdot,\cdot,\cdot,\cdot)$ oracle on (a_p, b_p, g_d, u) as above. New pointers are handled by storing $y_p = \langle g_d, u\rangle$.
  If the event FAIL is triggered, output in a special output tape, all pairs (g_d, b) collected from triples (p, g_d, b) that were recorded in steps 8 and 9.

We now show a construction of adversary $\mathcal{A}'$ against the T-OMDH assumption using the above modified simulator. The adversary $\mathcal{A}'$, given $[g, g_1, \ldots, g_N]$, simulates the UC execution with the dummy adversary $\mathcal{A}$, the simulator SIM modified as above and the environment $\mathcal{Z}$, until the moment that the simulator produces the special private output (F, σ). Observe that at this moment no $\mathsf{TOMDH}_p(\cdot,\cdot)$ queries have been issued. $\mathcal{A}'$ appends to σ the random coins of $\mathcal{Z}$ up until this moment and produces (F, σ) as the output. In the second stage $\mathcal{A}'$ receives σ as the input and is thus capable of continuing the UC execution with $\mathcal{Z}$, dummy $\mathcal{A}$ and SIM while enjoying now access to the $\mathsf{TOMDH}_p(\cdot,\cdot)$ with a polynomial p which is suitably defined based on the adversarial servers' values.

In this way the oracle queries of SIM can be served using the access that $\mathcal{A}'$ has to the $\mathsf{TOMDH}_p(\cdot,\cdot)$ and $\mathsf{DDH}(\cdot,\cdot,\cdot,\cdot)$ oracles. Finally, $\mathcal{A}'$ outputs $(J, \boldsymbol{V})$ so that for each pair (g_d, b) recorded in the special output tape of SIM, J contains d and b is included in $\boldsymbol{V}$ (in case no pairs are found $\mathcal{A}'$ fails).

We finally argue that $\mathcal{A}'$ will break the T-OMDH assumption with the same probability of success as the FAIL event. Recall that the event FAIL in session sid is associated with a pointer p and corresponds to the refusal of the ideal functionality $\mathcal{F}_{\text{TOPRF}}$ to respond to the simulator's $(\text{EVAL}, sid, ssid, (\mathcal{A}^*)^{t+1}, x)$ request for some x followed by $(\text{RCVCOMPLETE}, sid, ssid, \mathcal{A}^*, p)$. This can happen only if $|\{S \mid \mathsf{tx}(p, S) > 0\}| \leq t$, i.e. there are not enough servers with positive ticket counters. The ticket counters are incremented by $\mathcal{F}_{\text{TOPRF}}$ whenever a $(\text{SNDRCOMPLETE}, sid, ssid, S)$ message is delivered by SIM at step 7, which, in turn is a reaction to the delivery of a message $(\mathcal{SE}, a)$ to the respective server, originating from a user. In the reduction this only happens when an (i, a) query that corresponds to that server is made to the $\mathsf{TOMDH}_p(\cdot,\cdot)$ oracle.

We consider the triples of the form (p, g_d, u) that are recorded in step 8 and step 9 of the modified simulator. First observe that they are all of the form (p, g_d, g_d^k) for the public-key g^k of the T-OMDH assumption. Let Q be the total number of $\mathsf{TOMDH}_p(\cdot,\cdot)$ queries made by the reduction - which correspond to the number of $(\text{SNDRCOMPLETE}, sid, ssid, S_i)$ messages - and Q' be the total number of triples recorded. A failure event suggests that Q' has exceeded $C_{t-t'+1}(\boldsymbol{q})$. □

C Proof of Theorem 4 of Section 4

Proof. For any environment $\mathcal{Z}$ and any adversary $\mathcal{A}$, we construct a simulator SIM as in Figure 8. Again, without loss of generality, we assume that $\mathcal{A}$ is a "dummy" adversary that merely passes over all messages it gets to $\mathcal{Z}$.

First of all, note that SIM assigns an $H(\cdot)$ value to a certain string in steps 5, 10, 11 and 12; if there is a conflict in such assignments, that is, when SIM is going to assign an $H(\cdot)$ value, it finds out that it has already been assigned to a different value, SIM's will output FAIL. We show that such case can only occur with negligible probability:

- Step 5: Here $H(T(p, \mathsf{pw}))$ is set to $[C|K]$ if at least $t+1$ servers in $\mathcal{SI}$ are corrupt. Suppose SIM finds in this step that $H(T(p, \mathsf{pw}))$ is already assigned to another value (i) in step 10: According to the syntax of $\mathcal{F}_{\text{PPSS}}$, Reconstruction cannot be proceeded before Initialization, so this is impossible. (ii) in step 11: If step 11 is proceeded before step 5, there is no C found, and SIM will ignore U^* and $\mathcal{A}$'s message. So there will be no assignment. (iii) in step 12: Unless and until $\mathcal{Z}$ queries $T(p, \mathsf{pw})$, $T(p, \mathsf{pw})$ is a random string in $\{0,1\}^\ell$ to $\mathcal{Z}$, and the probability that $\mathcal{Z}$ queries $H(T(p, \mathsf{pw}))$ is negligible. Once $\mathcal{Z}$ queries $T(p, \mathsf{pw})$ (note that this query can be done only in step 11), this case transfers to case (ii).
- Step 11: Here $H(T(p^*, x))$ is set to $[C|K]$ if there are at least $t+1$ servers in either $\mathsf{tested}(x)$ subset or corrupt server subset of $\mathcal{SI}$, and also $x = \mathsf{pw}$. Suppose SIM finds in this step that $H(T(p^*, \mathsf{pw}))$ is already assigned to

Initialize $\mathsf{count} := 0$ and $\mathsf{tx}(p, S) := 0$ for all pairs (p, S).

1. On $(\textsc{Init}, sid, U, \mathcal{SI})$ from $\mathcal{F}_{\text{PPSS}}$, ignore it if $|\mathcal{SI}| \neq n$ or tuple $\langle U, sid, \cdot, \cdot \rangle$ exists. Otherwise set count++, record $\langle U, sid, \mathcal{SI}, \mathsf{count} \rangle$, and send $(\textsc{Send}, (sid, 0), U, S, \mathcal{SI})$ to $\mathcal{A}$ for all $S \in \mathcal{SI}$. If $\mathcal{F}_{\text{PPSS}}$ sends (pw, K), record it.
2. On $(\textsc{Sent}, (sid, 0), S, \mathcal{SI})$ from $\mathcal{A}$ for some $S \in \mathcal{SI}$, mark S as "active" and send $(\textsc{Init}, sid, S, \mathcal{SI}, \mathsf{count})$ to $\mathcal{A}$.
3. On $(\textsc{InitComplete}, sid, S)$ from $\mathcal{A}$ for some $S \in \mathcal{SI}$, retrieve $\langle U, sid, \mathcal{SI}, \mathsf{count} \rangle$. Ignore the message if there is no such tuple, or not all servers in $\mathcal{SI}$ are active. Otherwise send $(\textsc{Send}, (sid, 1), S, U, \textsc{done})$ to $\mathcal{A}$.
4. On $(\textsc{Sent}, (sid, 1), S, U, \textsc{done})$ from $\mathcal{A}$ for all $S \in \mathcal{SI}$, send $(\textsc{Eval}, sid, 0, U, \mathcal{SE})$ to $\mathcal{A}$ for any $\mathcal{SE} \subseteq \mathcal{SI}$ s.t. $|\mathcal{SE}| = t + 1$.
5. On $(\textsc{RcvComplete}, sid, ssid, P, p^*)$ from $\mathcal{A}$, recover p corresponding to U as stored in step 1, and ignore this message if either of the following conditions fails: (i) if $p^* = p$ then $|\{S \mid \mathsf{tx}(p, S) > 0\}| > t$, (ii) if all servers in $\mathcal{SE}$ are honest then $p^* = p$. Otherwise pick $C \leftarrow_{\text{R}} \{0,1\}^\ell$, append C to tuple $\langle U, sid, \mathcal{SI}, p \rangle$ stored in step 1, and send $(\textsc{Send}, (sid, 2), U, S, C)$ to $\mathcal{A}$ for each $S \in \mathcal{SI}$. If there is a pair (pw, K) stored in step 1 and $T(p, \mathsf{pw})$ is defined, then also set $H(T(p, \mathsf{pw})) := [C|K]$. Otherwise also pick $K \leftarrow_{\text{R}} \{0,1\}^\ell$. Furthermore, if $p^* = p$ then also set $\mathsf{tx}(p^*, S)$−− for any $t + 1$ distinct S s.t. $\mathsf{tx}(p, S) > 0$.
6. On $(\textsc{Sent}, (sid, 2), U, S, C)$ from $\mathcal{A}$ for some $S \in \mathcal{SI}$, send $(\textsc{Send}, (sid, 3), S, U, \textsc{ack})$ to $\mathcal{A}$ and $(\textsc{SInit}, sid, S)$ to $\mathcal{F}_{\text{PPSS}}$.
7. On $(\textsc{Send}, (sid, 3), S, U, \textsc{ack})$ from $\mathcal{A}$ where $S \in \mathcal{SI}$, mark S as $\mathsf{complete}$. If all servers in $\mathcal{SI}$ are marked as $\mathsf{complete}$, send $(\textsc{UInit}, sid, K)$ to $\mathcal{F}_{\text{PPSS}}$.
8. On $(\textsc{Rec}, U, sid, ssid, \mathcal{SR})$ from $\mathcal{F}_{\text{PPSS}}$, record $\langle U, sid, ssid, \mathcal{SR} \rangle$ and send $(\textsc{Eval}, sid, ssid, U, \mathcal{SR})$ to $\mathcal{A}$.
9. On $(\textsc{SndrComplete}, sid, ssid, S)$ from $\mathcal{A}$, set $\mathsf{tx}(S)$++ and send $(\textsc{SRec}, sid, ssid, S)$ to $\mathcal{F}_{\text{PPSS}}$.
10. On $(\textsc{RcvComplete}, sid, ssid, U, p^*)$ and $(\textsc{Sent}, (sid, ssid, i, 0), S, U, C')$ from $\mathcal{A}$ for all $S \in \mathcal{SR}$, recover $\mathcal{SR}$ corresponding to U as stored in step 8 and p corresponding to U as stored in step 1 (ignore this message if no corresponding tuples exist), and ignore this message if either of the following conditions fails: (i) if $p^* = p$ then $|\{S \mid \mathsf{tx}(p, S) > 0\}| > t$, (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 s.t. $\mathsf{tx}(p, S) > 0$, and send $(\textsc{URec}, sid, \mathcal{SR}, \mathsf{flag}, \mathsf{pw}^*, K^*)$ to $\mathcal{F}_{\text{PPSS}}$ for $(\mathsf{flag}, \mathsf{pw}^*, K^*)$ set as follows:
    (a) If not all C''s are the same, set $(\mathsf{flag}, \mathsf{pw}^*, K^*) := (0, \perp, \perp)$.
    (b) Otherwise recover p corresponding to U and sid as stored in step 1. If $p^* = p$ and $C' = C$, set $(\mathsf{flag}, \mathsf{pw}^*, K^*) := (1, \perp, \perp)$.
    (c) Otherwise define X as the set of values x in the dictionary such that $T(p^*, x)$ is defined. For every $x \in X$ in lexicographic order, set $v' := T(p^*, x)$ and check if $C' = H_L(v')$. If so, set $K' := H_R(v')$ and $(\mathsf{flag}, \mathsf{pw}^*, K^*) := (2, x, K')$, and break the loop. If the above loop processes all $x \in X$ without breaking, set $(\mathsf{flag}, \mathsf{pw}^*, K^*) := (0, \perp, \perp)$.
11. On $(\textsc{Eval}, sid, ssid, \mathcal{SE}, x)$ from party $P \in \{U, \mathcal{A}\}$ and $(\textsc{RcvComplete}, sid, ssid, P, p^*)$ from $\mathcal{A}$, recover p corresponding to U as stored in step 1 and C corresponding to U as in step 2 (ignore this message if no corresponding tuples exist), and ignore this message if either of the following conditions fails: (i) if $p^* = p$ then $|\{S \mid \mathsf{tx}(p, S) > 0\}| > t$, (ii) if all servers in $\mathcal{SE}$ are honest then $p^* = p$. Otherwise pick $T(p^*, x) \leftarrow_{\text{R}} \{0,1\}^\ell$ if it has not been defined, and send $(\textsc{Eval}, sid, ssid, T(p^*, x))$ to $\mathcal{A}$. If $p^* = p$ then also set $\mathsf{tx}(p^*, S)$−− for any $t + 1$ distinct S s.t. $\mathsf{tx}(p, S) > 0$, add every $S \in \mathcal{SE}$ to $\mathsf{tested}(x)$, send $(\textsc{TestPwd}, sid, S, x)$ to $\mathcal{F}_{\text{PPSS}}$. If $\mathcal{F}_{\text{PPSS}}$ replies K, then also set $H(T(p^*, x)) := [C|K]$.
12. On x from $\mathcal{A}$ as a query to the $H(\cdot)$ oracle, send $H(x)$ to $\mathcal{A}$.
    $H(x)$ is computed as follows: If it has not been set to a specific value, pick $v \leftarrow_{\text{R}} \{0,1\}^{2\ell}$ and set $H(x) := v$.
    If there is a conflict in the assignment of $H(\cdot)$ values, output $\textsc{Fail}$.

another value (i) in step 5: Note that in step 11, C and K are exactly the same with those in step 5, so there is no possibility of conflict. (ii) in step 10: In step 10, the computation of $H(p^*, \mathsf{pw})$ may occur in case (c), where $T(p^*, \mathsf{pw})$ is already defined. However, $T(p^*, \mathsf{pw})$ can be defined only through querying it in step 11, and once it is queried, $H(p^*, \mathsf{pw})$ will be assigned to $[C|K]$ immediately. Therefore, it is impossible that SIM wants to set $T(p^*, \mathsf{pw})$ to some value after it has already been assigned in step 10. (iii) in step 12: Similar to case (iii) in the bullet above, in this case FAIL occurs with negligible probability.
- Steps 10 and 12: These two cases are trivial, since here $H(\cdot)$ is assigned to a certain value only if it has not been set previously; that is, there is no possibility that $H(\cdot)$ is assigned again after it has been assigned to another value.

Since we have proved that Pr[FAIL] is negligible, we assume below that FAIL does not occur.

Next we show that the real world and the simulated world are indistinguishable in $\mathcal{Z}$'s view by a sequence of games, the first one, $\mathbf{G}_0$, describes interactions with $\mathcal{Z}$ in the real world and the last one describes those in the simulated world. $\mathbf{G}_0$ is shown in Figure 9, where we make a number of simplifications explained below:

- For all messages input from and output to $\mathcal{Z}$, entries such as INIT and REC, and session IDs (i.e. *sid* and *ssid*) are omitted.
- There is no difference between the real world and the simulated world regarding DONE, ACK (sent from S to U in Initialization) and (SREC, *sid*, *ssid*) (output by S to $\mathcal{Z}$ in Reconstruction), so we omit these two messages below.
- $\mathcal{Z}$'s input in $\mathcal{F}_{\text{TOPRF}}$ queries includes (EVAL, *sid*, *ssid*, $\mathcal{SE}, x$) (sent to SIM via $\mathcal{A}$ or a corrupt user U^*) and (RCVCOMPLETE, *sid*, *ssid*, P, p^*) (sent to SIM via $\mathcal{A}$). The process proceeds only when $\mathcal{SE}$ and p^* satisfy the conditions listed as (i) and (ii) in Figure 8; therefore, if those conditions are not met, the case is trivial and we do not consider such cases below. Once the four conditions are satisfied, the output is computed as $T(p^*, \mathsf{pw}^*)$. Therefore, step 8 of SIM is essentially querying the $T(\cdot, \cdot)$ functions maintained by $\mathcal{F}_{\text{TOPRF}}$. Thus, we simplify the input to the function pointer p and the variable x, and the output to the function value $v = T(p, x)$, and omit all other messages and entries exchanged among the participating parties.
- $\mathcal{Z}$'s view in Reconstruction includes messages (REC, [...], pw') output by U, (SNDRCOMPLETE, [...]) output by S, and (RCVCOMPLETE, [...], p^*) output by U. As in the bullet above, if the whole process ends with U outputting either a string $K' \in \{0,1\}^\ell$ or FAIL, then $\mathcal{SR}$ is not related to the final result. Therefore, we do not show them below, and simplify $\mathcal{Z}$'s input to pw' and C'. Furthermore, U outputs FAIL immediately if U receives two different C''s, so this case is trivial. We only consider the other case, i.e. all C''s are the same, in the games.

Let $\mathbf{G}_1$ be a modification of $\mathbf{G}_0$, where Reconstruction is proceeded by the following:

Set $T(p, \cdot)$ to a random function onto $\{0,1\}^\ell$.

Initialization On $\mathcal{Z}$'s inputs $\mathcal{SI}$ and pw:

1. Set p to a fresh non-negative integer and compute $v := T(p, \mathsf{pw})$;
2. Output $K := H_L(v)$ and $C := H_R(v)$ to $\mathcal{Z}$.

Reconstruction On $\mathcal{Z}$'s inputs p^*, pw', and C':

1. Set $v' := T(p^*, \mathsf{pw}')$;
2. Check if $C' = H_L(v')$. If so, output $K' := H_R(v')$ to $\mathcal{Z}$. Otherwise output FAIL to $\mathcal{Z}$.

$\mathcal{F}_{\text{TOPRF}}$ *Query* On $\mathcal{Z}$'s inputs p^* and x:

1. Output $T(p^*, x)$ to $\mathcal{Z}$.

Hash Function Query On $\mathcal{Z}$'s input x:

1. Output $H(x)$ to $\mathcal{Z}$.

Fig. 9. $\mathbf{G}_0$: Security Game in the Real World.

- If $p^* = p$ and $C' = C$ (we denote such event as E_C below), then output K to $\mathcal{Z}$ if $\mathsf{pw}' = \mathsf{pw}$, and FAIL otherwise.
- Otherwise let X be the set of all x in the dictionary such that $T(p^*, x)$ is queried by $\mathcal{Z}$. Iterate through all $x \in X$ in lexicographic order and perform Reconstruction as in $\mathbf{G}_0$, with the exception that pw' is replaced by x; that is, compute $v' := T(p^*, x)$, check if $C' = H_L(v')$, and if so,
  - if $x = \mathsf{pw}'$, output $K' := H_R(v')$ to $\mathcal{Z}$,
  - otherwise output FAIL to $\mathcal{Z}$.

  In either case, break the loop. If the loop ends without a break (i.e. the check does not pass for every $x \in X$), output FAIL to $\mathcal{Z}$.

We compare $\mathcal{Z}$'s views in $\mathbf{G}_1$ and $\mathbf{G}_0$. Let K'_1 and K'_0 be the output at the end of Reconstruction in $\mathbf{G}_1$ and $\mathbf{G}_0$, respectively. Let event E be $K'_1 \neq K'_0$.

First of all, note that if $K'_0 = \text{FAIL}$, then $K'_1 = \text{FAIL}$ as well. This is equivalent to if $K'_1 \neq \text{FAIL}$, then $K'_0 \neq \text{FAIL}$. This is because: (i) If E_C occurs and $K'_1 \neq \text{FAIL}$, this means that $\mathsf{pw}' = \mathsf{pw}$, and in this case, $C = H_L(T(p, \mathsf{pw}))$, so the check in $\mathbf{G}_0$ passes; that is, $K'_0 \neq \text{FAIL}$. (ii) If E_C does not occur and $K'_1 \neq \text{FAIL}$, then $C' = H_L(T(p^*, \mathsf{pw}'))$, so the check in $\mathbf{G}_0$ passes; that is, $K'_0 \neq \text{FAIL}$. Therefore, E can only occur when $K'_0 \neq \text{FAIL}$.

Next, we break E into several sub-events:

- E_1: $E_C \wedge K'_1 \neq K'_0$.

In this case, $\mathsf{pw}' \neq \mathsf{pw}$ must hold (otherwise $K_1' = K_0' = K$ where K is the output in Initialization of $\mathbf{G}_1$ and $\mathbf{G}_0$), and since $K_0' \neq \text{FAIL}$, the check in $\mathbf{G}_0$ passes; that is, $C' = C = H_L(v')$. On the other hand, we know from Initialization that $C = H_L(v)$. Note that $v' = T(p, \mathsf{pw}')$, $v = T(p, \mathsf{pw})$, and $\mathsf{pw}' \neq \mathsf{pw}$, so v' and v are two independently random strings in $\{0,1\}^\ell$. Therefore, $\Pr[v' = v]$ is negligible, so (v', v) forms a collision of $H_L(\cdot)$ with overwhelming probability.

- E_2: $\neg E_C \wedge \mathsf{pw}' \notin X \wedge K_1' \neq K_0'$.
  First consider the case where $p^* = p$ and $\mathsf{pw}' = \mathsf{pw}$. If so, since $K_0' \neq \text{FAIL}$, we have $C' = H_L(T(p^*, \mathsf{pw}')) = H_L(T(p, \mathsf{pw})) = C$. However, since E_C does not occur and $p^* = p$, $C' \neq C$ must hold, which contradicts the former. Therefore, if E_2 occurs, we know $p^* \neq p$ or $\mathsf{pw}' \neq \mathsf{pw}$. In either case, since $\mathsf{pw}' \notin X$, that is, $\mathcal{Z}$ does not query $T(p^*, \mathsf{pw})$, $T(p^*, \mathsf{pw})$ is independently random from everything else in $\mathcal{Z}$'s view. Thus, the probability that $\mathcal{Z}$ comes up with a C' such that $C' = H_L(T(p^*, \mathsf{pw}))$ is negligible.
- E_3: $\neg E_C \wedge \mathsf{pw}' \in X \wedge K_1' \neq K_0'$.
  In this case, $\mathbf{G}_1$ will search X in lexicographic order, and once it comes to pw', it will find out that $C' = H_L(T(p^*, \mathsf{pw}'))$ (this is derived from $K_0' \neq \text{FAIL}$) and output $K_1' := H_R(T(p^*, \mathsf{pw}')) = K_0'$. Therefore, if E_3 occurs, there exists a $x < \mathsf{pw}'$ such that $C' = H_L(T(p^*, x))$. But $C' = H_L(T(p^*, \mathsf{pw}'))$ as well, so $(T(p^*, x), T(p^*, \mathsf{pw}'))$ forms a conflict of $H_L(\cdot)$ unless $T(p^*, x) = T(p^*, \mathsf{pw}')$; that probability that the latter occurs is negligible.

We can see that

$$E_1 \vee E_2 \vee E_3 = E,$$

so we have proved that $\Pr[E]$ is negligible; that is, $\mathcal{Z}$'s views in $\mathbf{G}_0$ and $\mathbf{G}_1$ are indistinguishable.

Let $\mathbf{G}_2$ be a modification of $\mathbf{G}_1$, where in Initialization, $[C|K]$ is picked at random from $\{0,1\}^{2\ell}$, and once $T(p, \mathsf{pw})$ is queried, set $H(T(p, \mathsf{pw})) := [C|K]$. We can see that $\mathbf{G}_2$ is essentially the same with the security game in the simulated world.

In $\mathbf{G}_1$, before $T(p, \mathsf{pw})$ is queried, it is random in $\mathcal{Z}$'s view, so the probability that $\mathcal{Z}$ queries $H(T(p, \mathsf{pw}))$ is negligible. If $\mathcal{Z}$ does not query $H(T(p, \mathsf{pw}))$, C and K are random strings in $\mathcal{Z}$'s view; that is what $\mathbf{G}_2$ does. After $T(p, \mathsf{pw})$ is queried, $\mathbf{G}_2$ are $\mathbf{G}_1$ are exactly the same. Therefore, $\mathbf{G}_2$ and $\mathbf{G}_1$ are identical in $\mathcal{Z}$'s view.

In sum, we have shown that $\mathcal{Z}$'s views in the real world ($\mathbf{G}_0$) and the simulated world ($\mathbf{G}_2$) are indistinguishable. □

D Functionality $\mathcal{F}_{\text{PPSS}}$

For completeness, we present in Figure 10 the formalization of PPSS as a UC functionality $\mathcal{F}_{\text{PPSS}}$, as defined in [19].

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

Initialization

1. On message $(\textsc{Init}, sid, \mathcal{SI}, \mathsf{pw})$ for $|\mathcal{SI}| = n$ from U, record $\langle \textsc{Init}, sid, \mathcal{SI}, \mathsf{pw}\rangle$ and send $(\textsc{Init}, U, sid, \mathcal{SI})$ to $\mathcal{A}^*$. (Ignore other Init commands.) Pick $K \leftarrow_{\mathrm{R}} \{0,1\}^{\ell}$, and if $|\mathcal{SI} \cap \mathsf{CorrSrv}| \geq t+1$ then send (K, pw) to $\mathcal{A}^*$.
2. Upon receiving $(\textsc{SInit}, sid, S)$ from $\mathcal{A}^*$, if record $\langle \textsc{Init}, U, sid, \mathcal{SI}, \mathsf{pw}\rangle$ exists and $S \in \mathcal{SI}$ then mark S as ACTIVE and send $(\textsc{SInit}, sid)$ to S.
3. Upon receiving $(\textsc{UInit}, sid)$ from $\mathcal{A}^*$, if record $\langle \textsc{Init}, U, sid, \mathcal{SI}, \mathsf{pw}\rangle$ exists and all servers in $\mathcal{SI}$ are marked ACTIVE then add K to $\langle \textsc{Init}, U, sid, \mathcal{SI}, \mathsf{pw}\rangle$ and send $(\textsc{UInit}, sid, K)$ to U.

Reconstruction

1. Upon receiving $(\textsc{Rec}, sid, ssid, \mathcal{SR}, \mathsf{pw}')$ for $|\mathcal{SR}| = t+1$ from U', retrieve record $\langle \textsc{Init}, U, sid, \mathcal{SI}, \mathsf{pw}, K\rangle$, record $\langle \textsc{Rec}, U', sid, ssid, \mathcal{SI}, \mathcal{SR}, \mathsf{pw}, \mathsf{pw}'\rangle$ and send $(\textsc{Rec}, U', sid, ssid, \mathcal{SR})$ to $\mathcal{A}^*$. Ignore future Rec commands involving the same $ssid$.
2. Upon receiving $(\textsc{SRec}, sid, ssid, S)$ from $\mathcal{A}^*$, if S is marked ACTIVE then set $\mathsf{tx}(S)$++ and send $(\textsc{SRec}, sid, ssid)$ to S.
3. Upon receiving $(\textsc{URec}, sid, ssid, \mathcal{SC}, \mathsf{flag}, \mathsf{pw}^*, K^*)$ for $|\mathcal{SC}| = t+1$ from $\mathcal{A}^*$, if a record $\langle \textsc{Rec}, U', sid, ssid, \mathcal{SI}, \mathcal{SR}, \mathsf{pw}, \mathsf{pw}', K\rangle$ exists such that $\mathcal{SR} \setminus \mathsf{CorrSrv} \subseteq \mathcal{SC}$ and $\mathsf{tx}(S) > 0$ for all S in $\mathcal{SC}$, then set $\mathsf{tx}(S)$−− for all such S and send $(\textsc{URec}, sid, ssid, \textsc{Res})$ to U' such that:
   (a) $\textsc{Res} := K$ if $(\mathsf{pw}' = \mathsf{pw}) \wedge (\mathcal{SC} \subseteq \mathcal{SI}) \wedge [(\mathsf{flag} = 1) \vee (\mathcal{SR} \cap \mathsf{CorrSrv} = \varnothing)]$;
   (b) $\textsc{Res} := K^*$ if $(\mathsf{pw}' = \mathsf{pw}^*) \wedge (\mathcal{SC} \subseteq \mathsf{CorrSrv}) \wedge (\mathsf{flag} = 2)$;
   (c) $\textsc{Res} :=$ Fail otherwise.

Password Test

Upon receiving $(\textsc{TestPwd}, sid, S_i, \mathsf{pw}^*)$ from $\mathcal{A}^*$, if $\mathsf{tx}(S_i) > 0$ then set $\mathsf{tested}(\mathsf{pw}^*) := \mathsf{tested}(\mathsf{pw}^*) \cup \{S_i\}$ and $\mathsf{tx}(S_i)$−−, retrieve $\langle \textsc{Init}, U, \mathcal{SI}, \mathsf{pw}, K\rangle$, 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.

Fig. 10. (t, n)-Threshold PPSS Functionality $\mathcal{F}_{\mathrm{PPSS}}$.