Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only model

Stanislaw Jarecki* Aggelos Kiayias† Hugo Krawczyk‡

Abstract

In a *Password-Protected Secret Sharing (PPSS)* scheme with parameters (t, n) (formalized by Bagherzandi et al. [2]), a user Alice stores secret information s among n servers so that she can later recover the information solely on the basis of her password. The security requirement is similar to a (t, n)-threshold secret sharing, i.e., Alice can recover her secret as long as she can communicate with $t + 1$ honest servers but an attacker gaining access to t servers cannot learn information about the secret. In particular, the system is secure against off-line attacks by an attacker controlling up to t servers. On the other hand, accounting for inevitable on-line attacks one allows the attacker an advantage proportional to the fraction of dictionary passwords tested in on-line interactions with the user and servers.

We present the first round-optimal PPSS scheme, requiring just one message from user to server, and from server to user, and that works in the password-only setting where users do not have access to an authenticated public key. The scheme uses an Oblivious PRF whose security we define using a UC-style ideal functionality and denote as V-OPRF due to its verifiability, and for which we show concrete, very practical realizations in the random oracle model, as well as standard-model instantiations. As an important application we use this scheme to build the first single-round password-only Threshold-PAKE protocol in the CRS and ROM models for arbitrary (t, n) parameters with no PKI requirements for any party (clients or servers) and no inter-server communication. Our T-PAKE protocols are built by combining suitable key exchange protocols on top of our V-OPRF-based PPSS schemes. We prove T-PAKE security via a generic composition theorem showing the security of any such composed protocol.

1 Introduction

Remarkably, passwords have become a fundamental pillar of electronic security. That's quite a high task for these low-entropy easily-memorable easily-guessed short character strings. In spite of repeated evidence of their vulnerability to misuse and attack, passwords are still in widespread use and will probably remain as such for a long while. The portability of passwords make them ubiquitous keys to access remote services, open computing devices, decrypt encrypted files, protect financial and medical information, etc. Replacing passwords with long keys requires storing these keys in devices that are not always available to the user and are themselves at risk of falling in adversarial hands, hence endangering these keys and the data they protect.

An increasingly common solution to the problem of data security and availability is to store the data itself, or at least the keys protecting its security, at a remote server, which in turn is accessed using a password. This requires full trust in this single server and the one password. In particular, compromising such a server (or just its password file) is sufficient to crack most passwords stored at it through an *off-line* dictionary attack. Indeed, loss of millions of passwords to such attacks

*U. California Irvine. Email: stasio@ics.uci.edu.

†National and Kapodistrian University of Athens. Email: aggelos@kiayias.com. Research partly supported by ERC project CODAMODA.

‡IBM Research. Email: hugo@ee.technion.ac.il

are common news nowadays [41]. Unfortunately, off-line attacks are unavoidable in single-server scenarios. A natural approach to solving this problem is to distribute the above trust over a set of servers, for example by sharing information among these servers using a secret sharing scheme. However, how does the user access these servers? Using the same password in each of these servers makes the off-line password recovery attack even worse (as it can be performed against any of these servers) while memorizing a different password for each server is impractical.

PPSS. The above problem and a framework for solution is captured by the notion of Password-Protected Secret Sharing (PPSS) that originates with the work of Ford and Kaliski [23] and Jablon [29] and recently formalized by Bagherzandi et al. [2]. In such a scheme, parametrized by a pair of variables (t, n), a user Alice has some secret information sc that she wants to store and protect, and be able to later access on the basis of a single password pw. (Secret sc can represent any form of information, but it is best to think of it as a cryptographic key which protects some cryptographic capability.) The scheme has an *initialization phase* where Alice communicates with each one of a set of n servers $S_1, \ldots, S_n$ after which each server S_i stores some information ω_i associated with user Alice (ω_i is a function of the secret sc, the password pw and server name S_i). When Alice needs to retrieve sc, she performs a *reconstruction protocol* by interacting with a subset of at least $t + 1$ servers where the only input from Alice is her password pw.

The main requirements from this protocol are, informally: (i) an attacker breaking into t servers cannot gain any information on sc other than by correctly guessing Alice's password and running an on-line attack with it (more on this below). It follows, in particular, that off-line attacks on the password are not possible as long as the attacker has not compromised more than t servers. In this case, the only avenue of attack against the secrecy of sc is for the attacker to select one value pw' from a given dictionary D of passwords (from which the user has selected a password at random) and check its validity by interacting with the user and servers using pw' as the password. If the overall number of interactions between the attacker and the user, and between the attacker and the servers, is q then we allow the attacker to break the semantic security of sc with advantage $q/|D|$ (plus negligible). Moreover, we will require that "testing" a guessed password by impersonating the user to the servers will require interacting with $t + 1$ different servers. (ii) Soundness: Even the compromise of all servers cannot lead to the user reconstructing the wrong secret except with probability proportional to the number of *on-line* interactions between the attacker and the user. This is a necessary exception as the attacker can isolate the user and simulate a run with the servers with a password pw' and secret sc' chosen by the adversary; what's required is that *only if* pw' happens to be the user's password will the attack succeed. Additionally, a desirable property is (iii) Robustness: Alice can correctly reconstruct sc as long as (a) no more than t servers are corrupted and (b) Alice communicates without disruptions with at least $t + 1$ honest servers. Note that robustness can only be achieved if $2t + 1 \leq n$ while the other properties do not impose such intrinsic limitation.

T-PAKE. While PPSS schemes have many uses such as for retrieving keys, credentials, data, and so on, the main PPSS application is for bootstrapping a *Threshold Password-Authenticated Key Exchange (T-PAKE)* [37]. In a (t, n) T-PAKE protocol, a user with a single password is to establish n authenticated keys with n servers, such that security of the keys established with uncorrupted servers is guaranteed as long as there are no more than t corrupted servers. PPSS schemes make it possible to build T-PAKE protocols by combining the PPSS scheme with a regular key exchange protocol in a modular and generic way. This allows one to focus on the PPSS design which by virtue of being a much simpler primitive, e.g., avoiding the intricacies of the security of (password) authenticated key exchange protocols, is likely to result in simpler and stronger solutions, as is indeed demonstrated by our results below.

Prior Work and Our Contributions

For the general case of (t, n) parameters, Bagherzandi et al. [2] showed a PPSS scheme in the random oracle model (ROM) where the reconstruction protocol involves three messages between the user and a subset of $t+1$ servers (effectively 4 messages in the typical case that the user initiates the interaction). However, if any of these servers deviate from the correct execution of the protocol, a protocol needs to be re-run with a new subset of servers, which potentially increases the number of protocol rounds to $O(n)$. Another shortcoming of the PPSS solution from [2] is that it is secure only in the PKI model, where the user can authenticate the public keys of the servers. Indeed, if the attacker can induce the user to run the protocol on an incorrect server's public key, the protocol of [2] becomes completely insecure. Thus, [2] leaves at least two open questions: Do PPSS protocols with *optimal single-round communication* exist (i.e., requiring a single message from user to server and single message from server to user), and can such protocols work in the *password-only model*, namely when the user does not have a guaranteed authentic public key.

Our main contribution is a PPSS protocol with optimal single-round communication (in ROM) which works in the password-only model. Concurrently to our work, Camenisch et al. [10] present a PPSS for the general (t, n) setting which also works in the password-only model (and ROM). However, their protocol sends 10 messages between the user and each server and its total communication complexity is $O(n^2)$. Moreover, its robustness is fragile in the same way as that of [2], i.e. the user runs the reconstruction protocol with a chosen subset of $t + 1$ players, and the protocol must be re-started if any server in this chosen group deviates. By contrast, the protocol we present has 2 messages and $O(n \log n)$ communication complexity, which can be further reduced to $O(n)$ if the user caches $O(n)$ data between reconstruction protocol instances. Our protocol also has stronger robustness guarantee, namely Alice is guaranteed to recover her shared secret sc in the single protocol instance as long as it has unobstructed communication with at least $t + 1$ honest servers and if $2t + 1 \leq n$. On the minus side, unlike [10] who formalize a UC functionality for PPSS (which they call "TPASS") and whose protocol realizes this functionality, we model the security of a PPSS scheme in the password-only setting with a game-based notion. Still, we show that our game-based security notion is strong enough to imply the security of a natural T-PAKE construction built on top of a PPSS scheme.

Since our PPSS construction is based on a novel version of so-called Oblivious Pseudorandom Function (OPRF) [24], our contributions are threefold touching on three distinct elements, OPRF's, PPSS, and T-PAKE's, which we discuss separately below.

OPRF. The basic building block of our PPSS construction is a *Verifiable* Oblivious PRF (V-OPRF). Oblivious PRF (OPRF) was defined [24, 30] as a protocol between two parties, a server and a user, where the first holds the key k for a PRF function f while the latter holds an argument x on which $f_k(\cdot)$ should be evaluated. At the end of the protocol the user learns $f_k(x)$ and is convinced that such value is properly evaluated while the server learns nothing. Formalizing the OPRF primitive in a way that can serve our application is not trivial. Indeed, the intuitive definition of OPRF [24, 30] as the secure computation of a two-party functionality which on input pair (k, x) returns an output pair $(\perp, f_k(x))$, is limiting for at least three reasons: (1) It does not imply security when several OPRF instances are executed concurrently, as is the case in our PPSS construction; (2) It does not apply to our setting where the existence of authenticated channels cannot be assumed; and (3) It is not clear how to instantiate such functionality in the concurrent setting without online extractable zero-knowledge proofs of knowledge, which would add a significant overhead to any OPRF instantiation.

We overcome these issues via a novel formalization of the *verifiable* version of the OPRF primitive, V-OPRF, as an ideal functionality in the Universal Composability (UC) framework [13] for

which we show several very efficient instantiations. Expressing V-OPRF in the UC framework is a delicate task, especially in the setting of interest to us where there are no authenticated channels. Our formalism enforces that the server who generates the PRF key k also produces a function descriptor π, which fixes a determnististic function f_π. (For honest servers, π is a commitment to k and the fixed function f_π is equal to the PRF f_k.) Then, in any (non-rejecting) execution of the V-OPRF protocol executed given the function descriptor π, the V-OPRF functionality verifies that the user's output is computed as $f_\pi(x)$. This ensures, in particular, that repeated V-OPRF runs with the same parameter π and same input x always produce the same output. In other words, the V-OPRF functionality ensures *consistency* between V-OPRF instances executed under the same function descriptor π as well as *verifiability* that the output value is computed using the committed function f_π.

Our UC V-OPRF formalization bears interesting similarities to UC blind signatures [35, 22, 1]. In a nutshell, instead of on-line extraction of argument x from the (potentially malicious) client in every V-OPRF instance, a V-OPRF functionality issues a *ticket* for every instance executed under a given descriptor π. The user (or adversary) can then use these tickets to evaluate function f_π on inputs of their choice, but with the constraint that m tickets cannot be used to compute f_π values on more than m distinct arguments. Not surprisingly given this similarity, we observe that an efficient realization of V-OPRF can be achieved (in ROM) by hashing a deterministic blind signature-message pair.

We present three highly efficient variants of this design strategy, which provide three *single-round* V-OPRF instantiations in ROM, and we prove them UC-secure under "one-more" type of assumptions [4, 31]. Specificially, we show such V-OPRF instantiations in ROM under a one-more Gap DH assumption on any group of prime order, a similar assmuption on the group with a bilinear map, and a one-more RSA assumption. We also provide an efficient standard model V-OPRF construction for the Naor-Reingold PRF [40], based on the honest-but-curious OPRF protocol given by [25]. Our protocol has two rounds (four messages) and is secure under Strong-RSA and the Decisional Composite Residuosity assumptions. Note that a single round protocol is theoretically feasible in the standard (CRS) model, e.g., by encrypting the user input with fully homomorphic encryption and having each server apply a PRF program over the ciphertext and supply a UC NIZK proof, e.g. [28], that this computation was performed correctly using a function committed in π. However, such generic solution is likely to be much less efficient compared to the Naor-Reingold based construction we present.

We note that the UC formalization of the Verifiable Oblivious PRF functionality that is at the core of our security treatment is likely to have applications beyond this work. Indeed, OPRF's have been shown to be useful in a variety of scenarios, including Searchable Symmetric Encryption (SSE) schemes, e.g. [18, 14], and secure two-party computation of Set Intersection [24, 30, 31].

PPSS. Armed with our V-OPRF constructions, we proceed to solving PPSS. Our PPSS protocol is *password-only* in the Common Reference String (CRS) model, i.e. the user needs no other inputs except of her password and a CRS string defining an instance of a non-malleable commitment scheme, which can be embedded in the user's V-OPRF software. Our PPSS protocol is *single-round* in the hybrid model where parties can access the V-OPRF functionality. Given the V-OPRF instantiations discussed above, this implies three different instantiations of a single-round (i.e. two-message) PPSS schemes in ROM based on different one-more type of assumptions, and a four-message PPSS scheme in the CRS model.

Our PPSS construction follows the strategy of the early protocols of Ford and Kaliski [23] and Jablon [29] who treated the case of $t = n$: Secret-share the secret sc into shares $(s_1, \ldots, s_n)$, let each server S_i pick key k_i for a PRF f, and let $\mathbf{c} = (e_1, \ldots, e_n)$ where e_i is an encryption of s_i under

$\rho_i = f_{k_i}(\mathsf{pw})$. Each server S_i stores[1] $(k_i, \mathbf{c})$, and in the reconstruction protocol the user re-computes each ρ_i via an instance of a V-OPRF protocol with each server S_i on its input pw and S_i's input k_i. If the user also gets string $\mathbf{c}$ from the servers, the user can decrypt shares s_i using the ρ_i's and interpolate these shares to reconstruct sc. The first thing to note is that ciphertexts e_i must *not* be committing to the encryption key ρ_i. Otherwise, an adversary could test a password guess pw^* in an interaction with a single V-OPRF instance (instead of $t+1$ instances with $t+1$ different servers), by computing $\rho_i^* = f_{k_i}(\mathsf{pw}^*)$ and testing if decryption of e_i under ρ_i^* returns a plausible share value. We prevent such tests by sharing sc over a binary extension field $\mathbb{F} = GF(2^\ell)$, choosing a PRF f which maps onto ℓ-bit strings, and setting e_i to $s_i \oplus f_{k_i}(\mathsf{pw})$. Secondly, the above simple protocol can allow a malicious server S_i to find the user's password pw if S_i is not forced to use the same function f_{π_i} in each V-OPRF instance. Consider the OPRF protocol of [24] for the Naor-Reingold PRF $f_{k_i}(x) = g^v$ where $v = k_{i,0} \cdot \prod_{x_j=1} k_{i,j}$ for $k_i = (k_{i,0}, \ldots, k_{i,\ell})$ [40]. If in some PPSS instance, a misbehaving S_i uses key k_i' which differs from k_i on one index j, i.e. in one component $k_{i,j}$, S_i can conclude that the j-th bit of pw is 0 if the user recovers its secret correctly from such PPSS instance. Note that the adversary can learn whether user's secret is reconstructed correctly by observing any higher-level protocol which uses this secret, e.g. a T-PAKE protocol discussed below. We counter this attack by using the verifiability property of our V-OPRF functionality, which ensures that S_i computes the function committed in π_i, and by extending the user-related information stored by each server to $\omega = (\boldsymbol{\pi}, \mathbf{c}, C)$ where $\boldsymbol{\pi}$ is a vector of function descriptors $\pi_1, \ldots, \pi_n$ of each server, and C is a non-malleable commitment to the values $\boldsymbol{\pi}$, $\mathbf{c}$ and user's password pw. This commitment is the basis for ensuring that the on-line attacker playing the role of the servers can test at most one password guess per one reconstruction protocol instance.

With our efficient instantiations of V-OPRF in ROM we achieve a remarkably efficient reconstruction protocol. The user runs an optimal 2-message protocol with $t + 1$ (or more) servers, and in the case of our V-OPRF construction based on the one-more Gap DH assumption on a prime-order group, the protocol involves just 2 exponentiations by the server and $2t + 3$ multi-exponentiations for the user, employing the optimized ROM-based NIZK for discrete logarithm equality of [15], plus one hashing onto the prime-order subgroup, a few symmetric key operations, and a polynomial interpolation operation. The (one-time) initialization stage is also very efficient, involving $2n + 1$ exponentiations for the user and 3 exponentiations per each server. Note that there is no inter-server communication in the protocol and that the user can communicate with each server independently, so it can be done in parallel and/or in any order without the servers being aware of each other. Moreover, the user can initiate the V-OPRF protocol with more than $t + 1$ servers, and it will reconstruct secret sc as long as $t + 1$ contacted servers reply with correct triple $\omega = (\boldsymbol{\pi}, \mathbf{c}, C)$ and complete the V-OPRF instances on function descriptors π_i in $\boldsymbol{\pi}$, provided that the honest servers constitute the majority in the group of servers with whom the user initiates the reconstruction protocol.

Note that a PPSS protocol in the password-only setting enjoys the following security hedging property. While avoiding the need to rely on the authenticity of the servers' public keys held by the user is an important security property, when such public keys are available they can add significant security, because they render on-line attacks against a user ineffective and strengthen both the security and the soundness properties of the PPSS scheme.[2] Thus, to get the benefits of both worlds, i.e. with and without the correctly functioning PKI, our password-only PPSS protocol can be easily amended to achieve the following: If the user happens to have the public keys of the

[1] Note that this requires $O(n)$ storage but it is trivial to bring down to $O(\log n)$ using Merkle trees.

[2] Namely, they remove factor q_U from eq. 1 in Def. 1, and they make the upper-bound on the soundness error (i.e. the probability that any $\mathsf{U}_{\mathsf{Rec}}$ instance outputs K' which is neither $\perp$ nor K output in $\mathsf{U}_{\mathsf{Init}}$) to be negligible. (See the definitions of PPSS security and soundness in Section 4.)

scheme	$(t+1, n)$ range	ROM/std	client	inter-server	msgs	total comm.	computation C | S
BJKS[8]	$(2, 2)$	ROM	PKI	PKI	7	O(1)	O(1)
KMTG [32]	$(2, 2)$	Std/ROM	CRS	sec.chan.	≥ 5	O(1)	O(1)
CLN [11]	$(2, 2)$	Std/ROM	CRS	PKI	8	O(1)	O(1)
DRG [21]	$t < n/3$	Std	CRS	sec.chan.	≥ 12	$O(n^3)$	$O(1) \mid O(n^2)$
MSJ [37]	any	ROM	PKI	PKI	7	$O(n^2)$	$O(1) \mid O(n)$
BJSL [2]	any	ROM	PKI	PKI	3	$O(n)$	$8t + 17 \mid 16$
CLLN [10]	any	ROM	Std	PKI	10	$O(n^2)$	$14t + 24 \mid 7t + 28$
Our PPSS #1	any	ROM	CRS	none	2	$O(n)$	$2t + 3 \mid 3$
Our PPSS #2	any	Std	CRS	none	4	$O(n\ell)$	$O(n\ell) \mid O(l)$

Figure 1: Comparison between PPSS/T-PAKE schemes. "Our PPSS #1" and "Our PPSS #2" refer to our PPSS construction of Section 5 with V-OPRF instantiated, respectively, with protocol 2HashDH-NIZK of Section 3 (this instantiation is spelled out in Appendix B), and with protocol NR-V-OPRF of Section 3.2. The last column counts (multi)exponentiations in a prime-order group performed by the client and each server in the reconstruction protocol (except for "our PPSS #2" where exponentiations are modulo a Paillier modulus). The costs in the last four rows refer to an optimistic scenario with no adversarial interference. With worst-case adversarial interference, assuming $n = 2t + 1$, for schemes MSJ and BJSL all costs (including rounds and communication complexity) grow by the factor of $t + 1$, while for our schemes in the last two rows only the client costs grow, and only by the factor of 2. The "total comm." column counts the number of transmitted group elements and objects of length polynomial in the security parameter, like public-key signatures. Variable ℓ denotes the length of the password string (or its hash).

servers, she gets the additional security benefits stated above. However, if she does not have access to servers' public keys, or if some of these keys have been replaced by fake ones, she still enjoys the full security of the password-only setting.

T-PAKE. When composed with regular key exchange protocols, our PPSS scheme leads to the most efficient T-PAKE protocols to date even when compared to protocols that assume that the user carries a public key that it can use to authenticate the servers. Figure 1 summarizes the state of the art in T-PAKE protocols and how our protocols compare to this prior work. Interestingly, while there is a large body of work on single-server PAKE protocols (e.g. [5, 33, 34, 7]) that has produced remarkable schemes, including one-round password-only protocols in the standard model, threshold PAKE has seen less progress, with most protocols showing disadvantages in different areas as shown in the above table. In particular, before our work, no single-round (t, n)-PAKE protocol was known, not even in the ROM. Most protocols assume a public key carried by the client (making them non password-only) and all assume secure channels (or PKI) between servers. To improve complexity, several works treated the specific case of $n = t = 2$ but even in these cases no one-round protocol was known and all require inter-server secure channels. Our work improves on all these parameters achieving the best known properties in all the aspects reflected in the table.

In particular, *we achieve single-round password-only protocol in the CRS and ROM models for arbitrary (t, n) parameters with no PKI requirements for any party and no inter-server communication* (secure communication is only assumed when a user first registers with the servers). In addition, the protocol is computationally very efficient (and more so than any of the previous protocols, even for the (2,2) case). We also exhibit a password-only standard-model implementation of our scheme requiring two rounds of communication (4 messages in total) between client and servers. Our T-PAKE protocols are built by combining (existing) suitable key exchange protocols

on top of our V-OPRF-based PPSS scheme. We prove T-PAKE security via a generic composition theorem showing the security of any such composed protocol.

Organization. In Section 2 we present the formalization of the V-OPRFfunctionality in the UC setting and in Section 3 we present efficient realizations of this functionality in the random-oracle model as well as in the standard (CRS) model. In Section 4 we define and formalize the PPSS primitive in the password-only model and in Section 5 we present a realization of such a scheme based on the V-OPRFfunctionality (i.e., the hybrid UC model). Finally in Section 6 we consider T-PAKE schemes obtained by composing a PPSS scheme with a regular key-exchange protocol and prove a general security composition theorem. We illustrate the advantages of this composition approach by showing how to obtain single-round T-PAKE protocols in the password-only model with the remarkable properties discussed above. For concreteness, Appendix B includes a full specification of our most efficient instantiation of the PPSS and T-PAKE protocols.

2 Functionality $\mathcal{F}_{\text{VOPRF}}$

Functionality $\mathcal{F}_{\text{VOPRF}}$

Key generation: Upon receiving (KeyGen, sid) from sender S, forward (KeyGen, sid, S) to adversary $\mathcal{A}^*$.

Upon receiving $(\text{Parameter}, sid, S, \pi, M)$ from $\mathcal{A}^*$, ignore this call if $\mathsf{param}(S)$ is already defined. Otherwise, set $\mathsf{param}(S) = \langle\pi\rangle$ and initialize $\mathsf{tickets}(\pi) = 0$, and $\mathsf{hist}(\pi)$ to the empty string. If S is honest send $(\text{Parameter}, sid, \pi)$ to party S, else parse M as a polynomial-size circuit with ℓ-bit output and insert (π, M) in $\mathsf{CorruptParams}$.

V-OPRFevaluation: Upon receiving (Eval, sid, S, x) from party U for sender S, record (U, x) and forward (Eval, sid, U, S) to $\mathcal{A}^*$.

Upon receiving $(\text{SenderComplete}, sid, S)$ from $\mathcal{A}^*$ for some honest S output $(\text{SenderComplete}, sid)$ to party S and set $\mathsf{tickets}(\pi) = \mathsf{tickets}(\pi)+1$ for π s.t. $\langle\pi\rangle = \mathsf{param}(S)$.

Upon receiving $(\text{UserComplete}, sid, U, \pi, \mathsf{flag})$ from $\mathcal{A}^*$, recover (U, x) and proceed as follows:

- If $\mathsf{flag} = \top$ and $\langle\pi\rangle = \mathsf{param}(S)$ for some honest S then: If $\mathsf{tickets}(\pi) \leq 0$ ignore the UserComplete request of $\mathcal{A}^*$, otherwise proceed as follows. If $\mathsf{hist}(\pi)$ includes a pair $\langle x, \rho'\rangle$, set $\rho = \rho'$, else sample ρ at random from $\{0,1\}^\ell$ and enter $\langle x, \rho\rangle$ into $\mathsf{hist}(\pi)$. Set $\mathsf{tickets}(\pi) = \mathsf{tickets}(\pi) - 1$ and output (Eval, π, ρ) to party U.
- Else, if $\mathsf{flag} = \bot$ then return $(\text{Eval}, \pi, \bot)$ to U.
- Else, if $\mathsf{flag} = \top$ and π is such that $(\pi, M) \in \mathsf{CorruptParams}$ for some circuit M, compute $\rho = M(x)$ and enter $\langle x, \rho\rangle$ in $\mathsf{hist}(\pi)$. Output (Eval, π, ρ) to party U.

Figure 2: Verifiable Oblivious PRF functionality $\mathcal{F}_{\text{VOPRF}}$.

We introduce the $\mathcal{F}_{\text{VOPRF}}$ functionality in Figure 2. It represents the notion of a "verifiable oblivious pseudorandom function" as discussed in the introduction. We refer to the holder of function's key as the *sender* and the party providing the input as the *user*, and identify them with S and U monikers, respectively. The functionality is reactive allowing senders and users to interact multiple times. For simplicity, and without loss of generality, we will specify $\mathcal{F}_{\text{VOPRF}}$ assuming each sender S registers a unique function, identified via a parameter π, and each user U evaluates a single

value x. We will refer to the parameter π as a "function descriptor". (In all our implementations of $\mathcal{F}_{\text{VOPRF}}$ functionality, π is a commitment to a PRF key.)

The $\mathcal{F}_{\text{VOPRF}}$ functionality can be thought of as a collection of tables that are indexed by "labels" denoted by the function parameters π. Users may obtain values from these tables on inputs x of their choice without leaking any information about these inputs (and corresponding outputs) to the adversary. $\mathcal{F}_{\text{VOPRF}}$ generates these tables dynamically and fills them with random values on demand. Each table is associated by the functionality with a specific sender. In addition to the tables registered to honest senders, the adversary is allowed to register with $\mathcal{F}_{\text{VOPRF}}$ its own tables. Interacting with an adversary-registered table does not jeopardize the privacy of the user's input but naturally $\mathcal{F}_{\text{VOPRF}}$ will provide no pseudorandomness guarantee for the output derived from such tables. However, $\mathcal{F}_{\text{VOPRF}}$ will ensure that all adversarial tables are completely determined according to a deterministic function that is committed by the adversary at the time of the table's initialization in the form of a circuit M.

The V-OPRF evaluation cycles through three activations of the functionality: Eval that defines the input to a V-OPRF computation and two activations, SenderComplete and UserComplete, that signal the completion of the V-OPRF computation by a sender and user, respectively. Refer to Figure 2 for the details of each such activation. Next, we present the rationale for the actions triggered by these activations.

A major consideration in our definition of $\mathcal{F}_{\text{VOPRF}}$ is to avoid the need for input extractability (from dishonest users) in the real-world realizations of the functionality. Such need is common in UC-defined functionalities but in our case it would disqualify the more efficient instantiations of $\mathcal{F}_{\text{VOPRF}}$ presented here. Thus, instead of resorting to input extraction requirements, we define a "ticket mechanism" that increases a ticket upon function evaluation at a sender and decreases it when this value is computed at the user (or the adversary). The functionality guarantees that tickets remain non-negative, namely, for any function parameter π registered with a honest sender S, the number of inputs on which users compute the function π is no more than the number of evaluations of the function at S. Thus, in $\mathcal{F}_{\text{VOPRF}}$, each time that a user completes an interaction with an honest sender, the functionality increases the ticket count associated with the corresponding table of the sender indicating that there is a certain value of that table that is currently being transmitted. Such tickets are redeemed within $\mathcal{F}_{\text{VOPRF}}$ via either one of two possible routes: the first is when an honest user completes her interaction and receives the value, the other is when the ideal world adversary decides to obtain such a value. This second occurrence reflects the fact that a real world adversary may generate an arbitrary number of malicious users that interact with honest servers and hence the ideal world adversary should be capable of probing the honest tables maintained by $\mathcal{F}_{\text{VOPRF}}$ (but only up to the number of tickets issued for the given table). Thanks to the ticket mechanism, in the case of malicious users, the ideal world adversary has to merely inform $\mathcal{F}_{\text{VOPRF}}$ that an honest sender completed a user interaction (via a SenderComplete message) without the necessity of submitting a proper Eval action on behalf of the malicious user. Regarding party corruption, the functionality keeps track of honest and corrupted parties and makes decisions according to their status. We only consider static corruptions.

Another important aspect of our $\mathcal{F}_{\text{VOPRF}}$ formalism is the way we handle the 1-1 relationship between a sender S and its function parameter π, where S is used to identify a sender and π describes this sender's committed function. The unique sender-function binding that is known to the functionality cannot be enforced in a real-world setting where users cannot validate such a binding as is the case when no authenticated channels (or other forms of authenticated information) are available to the user. Since these settings are common in our applications, we define $\mathcal{F}_{\text{VOPRF}}$ so that the user can provide a name of a sender whose function it intends to compute but the result returned to the user applies a function π determined by the attacker, and possibly different than the

function associated to the requested S. For example, in the PPSS application this corresponds to a setting where the user is given a network address for contacting a server but she cannot validate that this is indeed a server that holds a share of her secret, let alone test which function parameter corresponds to this server. The PPSS protocol needs to make sure that the ability of the attacker to re-route requests to different servers does not break the security of the PPSS scheme. Therefore, while in an EVAL call the value of S is specified, the USERCOMPLETE response that sets the output for this call lets the adversary choose π even if this is different than the parameter assigned to S by $\mathcal{F}_{\text{VOPRF}}$.

In spite of the above, note that if $\mathcal{F}_{\text{VOPRF}}$ is used in a context where the user knows a-priori a correspondence between S and π, the user can reject responses that are not consistent with it. For instance, suppose the user knows that π corresponds to S and after sending (EVAL, sid, S, x) to $\mathcal{F}_{\text{VOPRF}}$, she receives a successful output (π', ρ) with $\pi \neq \pi'$. In this case the user can choose to abort, despite the fact that $\mathcal{F}_{\text{VOPRF}}$ successfully completes. In our PPSS constructions in the $\mathcal{F}_{\text{VOPRF}}$-hybrid model, we make essential use of this ability of the user to abort during PPSS initialization upon an identified mismatch with an expected value of π. Finally, note that $\mathcal{F}_{\text{VOPRF}}$ guarantees that the value ρ obtained by the user is in the table π even though such table may not have been the user's original target. This provides $\mathcal{F}_{\text{VOPRF}}$ with a verifiability property but one that is verifier-dependent and may not be transferable to others; in particular, it is a weaker guarantee than the verifiability propery of verifable random functions [39].

3 Efficient Realizations of $\mathcal{F}_{\text{VOPRF}}$

In this section we show efficient realizations of the $\mathcal{F}_{\text{VOPRF}}$ functionality which we later use as the basis for our instantiations of the PPSS and T-PAKE schemes. We present constructions in the Random Oracle model as well as in the CRS model.

3.1 Realizing $\mathcal{F}_{\text{VOPRF}}$ in the Random Oracle Model

We present a class of constructions for realizing $\mathcal{F}_{\text{VOPRF}}$ in the random oracle model. Our constructions share the following general structure: the receiver hashes and blinds her input and requests the sender's secret-key application on this blinded value. The receiver verifies the sender's response and then obtains the V-OPRF output via applying a second hash function. Due to the double hashing action (which is essential in the security proof) we term the constructions with the "2Hash" prefix.

The 2HashDH construction relies on a group of prime order m with a generator denoted g. The secret key is k chosen at random in $\mathbb{Z}_m$, and the public key y is set to $y = g^k$. The function is defined as $f_k(x) = H_2(\pi, x, H_1(x)^k)$, where $\pi = (g, m, y)$ and H_1 and H_2 are hash functions onto, respectively, the group $\langle g \rangle$ and $\{0,1\}^\lambda$ where λ is a security parameter. The associated V-OPRF protocol is simple: the client sends $a = H_1(x)^r$ to the sender and the sender after checking that $a \in \langle g \rangle$, it responds by $(y, b = a^k)$. The protocol terminates with the client returning the value $H_2(\pi, x, b^{1/r})$ after checking that the tuple $\langle g, y, a, b \rangle$ is a valid DDH tuple (which implies that $\langle g, y, H_1(x), b^{1/r} \rangle$ is a valid DDH tuple). There are two alternatives for the client to achieve the latter test. First assuming that a $\mathcal{DH}_{g,y}(\cdot,\cdot)$ oracle is available to the client (such oracle returns 1 if and only if when given (a, b) it holds that $\log_a b = \log_g y$ and can be implemented e.g., if a bilinear map exists for the underlying group) it is possible to test directly the tuple $\langle g, y, H_1(x), b^{1/r} \rangle$. If such an oracle is not available to the client then the sender transmits a NIZK for equality of discrete logarithms, specifically showing that the tuple $\langle g, y, a, b \rangle$ satisfies the relation $\log_g y = \log_a b$. This

is a standard protocol that we recall for completeness: the sender selects $t \in_R \mathbb{Z}_m$ and computes $w = H_3(g, y, a, b, g^t, a^t)$ as well as $s = t + w \cdot k \bmod m$. The proof is the pair $\zeta = (w, s)$ which we denote as $\mathtt{NIZK}_{\mathtt{EQ}}^{H_3}[g, y, a, b]$. The receiver verifies ζ by testing $w = H_3(g, y, a, b, g^s y^{-w}, a^s b^{-w})$. We present this latter NIZK version as our first $\mathcal{F}_{\text{VOPRF}}$ realization in figure 3.

Parameters: generator g of cyclic group of order m and three hash functions $H_1(\cdot), H_2(\cdot), H_3(\cdot)$.

Key Generation: Upon receiving (KeyGen, *sid*), pick $k \in_R \mathbb{Z}_m$ and set $y = g^k$ and $\pi = y$. Return (Parameter, *sid*, π).

V-OPRF Evaluation: Upon receiving (Eval, *sid*, S, x) pick $r \in_R \mathbb{Z}_m$ and transmit $H_1(x)^r$ to S.

Upon receiving a message $a = H_1(x)^r$ from some network entity U check if $a \in \langle g \rangle$, compute $b = a^k$, and $\zeta = \mathtt{NIZK}_{\mathtt{EQ}}^{H_3}[g, y, a, b]$. Send to U the message $\langle y, b, \zeta \rangle$.

Upon receiving a message $\langle y, b, \zeta \rangle$ from a party S verify the NIZK ζ and that $b \in \langle g \rangle$. If the tests pass return (Eval, $y, H_2(y, x, b^{1/r})$), else return (Eval, $y, \perp$).

Figure 3: Protocol 2HashDH-NIZK.

We will argue the security of the construction employing the following assumption: the (N, Q) One-more Gap DH assumption, states that for any PPT $\mathcal{A}$ it holds that the following probability is negligible:

$$\mathsf{Prob}[\mathcal{A}^{(\cdot)^k, \mathsf{DDH}(\cdot,\cdot,\cdot,\cdot)}(g, g^k, g_1, \ldots, g_N) = \{(g_{j_s}, g_{j_s}^k) \mid s = 1, \ldots, Q+1\}]$$

where Q is the number of queries that $\mathcal{A}$ poses to the $(\cdot)^k$ oracle. The probability is taken over all choices of $g^k, g_1, \ldots, g_N$ which are assumed to be random elements of $\langle g \rangle$. We denote by $\epsilon_{\mathsf{omdh},G}(N, Q)$ the maximum advantage of any PPT adversary against the assumption.

Theorem 1 *The 2HashDH-NIZK protocol over a group G of order m UC-realizes $\mathcal{F}_{\text{VOPRF}}$ per Fig. 2 in the random oracle model assuming (i) the existence of PRF functions, (ii) the (N, Q) One-More Gap DH assumption on G where Q is the number of V-OPRF executions and $N = Q + q_1$ where q_1 is the number of $H_1(\cdot)$ queries.*

More precisely, for any adversary against 2HashDH-NIZK there is an ideal-world adversary (simulator) that produces a view in the ideal world that no environment can distinguish with advantage better than $q_S \cdot \epsilon_{\mathsf{omdh},G}(N, Q) + q_3/m^2 + 2 \cdot q_U/m + N^2/m + \epsilon_{\mathsf{PRF}}(q_2)$ where q_S is the number of senders, q_U the number of users, q_2, q_3 are the number of queries to the $H_2(\cdot), H_3(\cdot)$ oracle respectively and $\epsilon_{\mathsf{PRF}}(q_2)$ is the security of the PRF function against polynomial-time adversaries posing q_2 queries.

Proof: In Figure 2 we describe the simulator SIM that interacts with the adversary $\mathcal{A}$ and the ideal functionality $\mathcal{F}_{\text{VOPRF}}$. We assume without loss of generality that $\mathcal{A}$ is a dummy adversary, i.e., it is merely a pass-through machine that outsources all its computation to the environment $\mathcal{Z}$. We describe the simulator in Figure 2. The simulator uses a PRF function family $\{\mathsf{f}_{\mathsf{k}(\cdot)}\}_{\mathsf{k}}$ that has output range equal to the output range of the $\mathcal{F}_{\text{VOPRF}}$. For simplicity, we will first assume that $\mathsf{f}_{\mathsf{k}}(\cdot)$ is substituted by a random function $R(\cdot)$ and we will prove that the simulator can simulate the view of any environment $\mathcal{Z}^{H_1(\cdot), H_2(\cdot), H_3(\cdot)}$ that interacts with protocol 2HashDH-NIZK. In this argument $R(\cdot)$ will be a random function that it will not be available directly as an oracle to the

environment $\mathcal{Z}$ who will access it implicitly via its own random oracles. After we establish this fact (which will be the main challenge of the proof), we will substitute $R(\cdot)$ back to $\mathsf{f}_\mathsf{k}(\cdot)$ and thus demonstrate that 2HashDH-NIZK UC-realizes $\mathcal{F}_{\text{VOPRF}}$ in the random oracle model.

One can examine that SIM is a valid ideal model adversary against the UC functionality $\mathcal{F}_{\text{VOPRF}}$ and that the simulation of $\mathcal{A}$ by SIM is identical to a real run by $\mathcal{A}$ with two possible exceptions: The choice of outputs for honest and dishonest users and the fact that SIM may end with output FAIL, FAIL$'$ in Steps 5 and 7 (such events obviously do not exist in the execution of $\mathcal{A}$ in the real world). Thus, we need to show that the outputs of honest users in the simulated run are identical to those assigned by $\mathcal{F}_{\text{VOPRF}}$ to these users, the dishonest users' view is indistinguishable between real and ideal and that the events FAIL, FAIL$'$ happen with negligible probability.

We start with an analysis of the NIZK employed in the protocol. We observe that in step 5.I, the simulator will output FAIL if it happens that $c \neq v^k$. This happens when an honest user receives a message (y, b', ζ'), the proof ζ' is valid but it happens that $(b')^{1/r} \neq v^k$ where v, r come from the values (v, a, r) that SIM has stored for that user instance. Observe that $\langle g, y, v, c\rangle$ is a DDH tuple if and only if $\langle g, y, a, b'\rangle$ is a DDH tuple. In the identified circumstance we have that the verification equation $w = H_3(g, y, a, b', g^s y^{-w}, a^s (b')^{-w})$ passes but $\langle g, y, v, c\rangle$ is not a DDH tuple. Suppose that $(g, y, a, b', g^s y^{-w}, a^s (b')^{-w})$ was never queried to the oracle H_3 at the time that the honest user (played by SIM) performs the verification step. The probability of the verification equation to be successful is $1/m$. On the other hand, suppose that $(g, y, a, b', e_1 = g^s y^{-w}, e_2 = a^s (b')^{-w})$ was queried by the adversary at some point prior to delivery to the honest user. It is easy to see that given $\langle g, y, a, b'\rangle$ is not a DDH tuple and w is random over $\{0,1\}^m$ the probability of success is $1/m$. We conclude that the event fail happens with probability at most $2/m$.

We continue an analysis of the outputs assigned by SIM to honest users in order to prove their equivalence to the $\mathcal{F}_{\text{VOPRF}}$ output. These outputs in the real world are produced by the hash function H_2 on inputs of the form (π, x, u) for $u = h^k$ where h is the response to $H_1(x)$ and k is the secret key defined in π. In the simulated world the values h are elements g_j taken from SIM's gseq sequence. Inputs to H_2 which are not of the above form are answered at random by SIM.

The consistency between outputs of honest users in the ideal world and the simulated world needs to be ensured for the cases handled in step 5 of SIM. In the ideal world, these are cases where the user receives output (triggered by the UserComplete commands) while in the real world they correspond to cases where users receive the correct value b', namely, $b' = a^k$ where a was chosen by the user and k is defined by the incoming parameter π. The parameter π can correspond to a honest sender or a corrupt sender, two cases hadled by Step 5.I and 5.III, respectively.

In the case of step 5.I, SIM simulates the actions of user U by generating the outgoing value a as g_j^r for random r and g_j chosen from the gseq sequence. If at any point the input x of user U is queried from H_1, SIM responds (according to SIM Step 2) with the next available element of gseq, say $g_{j'}$. This, however, will be different than the above g_j chosen to generate U's value a. Yet, this inconsistency is perfectly hidden from the environment and the attacker since all they see is g_j^r for r that remains independently random from the adversarial views.

Continuing with step 5.I, SIM will use the UserComplete to ask $\mathcal{F}_{\text{VOPRF}}$ to deliver output to the user. Contrary to the real world setting, SIM will ask $\mathcal{F}_{\text{VOPRF}}$ to fix the value of the PRF function on this x without necessarily fixing the corresponding value in the table of H_2 that corresponds to $(\pi, x, g_{j'})$. This generates a situation that does not happen in the real world but however is undetectable to the adversary and environment. This is due to the fact that the only way to detect it is via querying $(\pi, x, g_{j'})$ to $H_2(\cdot)$. In such a case (if it ever happens), SIM using the power of the $\mathcal{DH}(\cdot)$ oracle will be able to detect that such a critical value is queried to $H_2(\cdot)$ and as described in Step 7, it will query the functionality $\mathcal{F}_{\text{VOPRF}}$ on this input via a rapid succession of Eval, UserComplete commands on behalf of the corrupted user U^* in order to match the proper

1. Upon receiving (KeyGen, sid, S) from $\mathcal{F}_{\text{VOPRF}}$, it computes $y = g^k$ for a random k drawn from $\mathbb{Z}_m$ and returns (Parameter, $sid, S, \pi = y, \top$) to $\mathcal{F}_{\text{VOPRF}}$. It stores (S, k).

2. SIM chooses a sequence $\mathsf{gseq} = (g_1, \ldots, g_N)$ of random elements in $\langle g \rangle$ with $g_j = g^{r_j}$ which it uses for answering H_1 queries and for choosing g_j values in Step 3 below. SIM initializes a counter j at 1 and when an element from gseq is needed, g_j is selected and counter j is increased by 1. In addition, SIM maintains a table for the random oracle H_1. It answers a new query by selecting an element from gseq as above (repeated queries get the same answer). Tables for H_2, H_3 are also maintained as explained below.

3. Upon receiving (Eval, sid, U, S) from $\mathcal{F}_{\text{VOPRF}}$. Let g_j be the next unused value in gseq. The simulator picks a random element r of $\mathbb{Z}_m$, stores $\langle U, v = g_j, a = g_j^r, r \rangle$ and sends $a = g_j^r$ to $\mathcal{A}$ as the user U's message intended for S.

4. Upon receiving a message a' from $\mathcal{A}$ as a message from an honest user U to honest sender S, it recovers the corresponding $\langle U, v, a, r \rangle$ stored entry, retrieves S's function descriptor $\pi = (y)$ with $y = g^k$, checks that $a' \in \langle g \rangle$ and computes $b = (a')^k$ and a proper NIZK ζ (it ignores the message if $a' \notin \langle g \rangle$). It augments the user U's entry to $\langle U, v, a, r, a', S, b, \zeta \rangle$, returns to $\mathcal{A}$ the value (π, b, ζ) as the response of S intended for the user U, and simultaneously returns (SenderComplete, sid, S) to $\mathcal{F}_{\text{VOPRF}}$.

5. Upon receiving a message (π, b', ζ') from $\mathcal{A}$ as a message from some sender back to user U, it recovers the corresponding stored entry for U which is either $\langle U, v, a, r, a', S, b, \zeta \rangle$ or $\langle U, v, a, r \rangle$, checks $b' \in \langle g \rangle$ and computes $c = (b')^{1/r}$. It parses $\pi = (y)$ and proceeds as follows.

   Case I. If ζ' is valid and the parameter π is registered to some honest sender, then it checks that $c = v^k$ for the corresponding key k and thus $b' = v^{rk} = a^k$. In this occasion we have a proper completion of the user protocol and $\mathcal{S}$ transmits (UserComplete, $sid, U, \pi, \top$) to $\mathcal{F}_{\text{VOPRF}}$. Given that $v = g_j$ for some j the simulator records the triple $(\pi, j, c) = (\pi, j, g_j^k)$. If $c \neq v^k$ then return FAIL to the environment.

   Case II. If ζ' is invalid and the parameter π is registered to some honest sender, then $\mathcal{S}$ transmits (UserComplete, $sid, U, \pi, \perp$) to $\mathcal{F}_{\text{VOPRF}}$.

   Case III. If π is not registered with any honest sender then if π has not been seen before, it picks a new sender tag S, stores (S, π) and sends (Parameter, $sid, S, \pi, M^{\mathsf{f_k}(\cdot)}$) to $\mathcal{F}_{\text{VOPRF}}$ where $M^{\mathsf{f_k}(\cdot)}$ is the TM that given x returns $\mathsf{f_k}(\pi, x)$. Then, it forwards (UserComplete, $sid, U, \pi, \mathsf{flag}$) to $\mathcal{F}_{\text{VOPRF}}$ where $\mathsf{flag} = \top, \perp$ depending on whether ζ' is valid or invalid.

6. Upon receiving a message a' from $\mathcal{A}$ as a message from a corrupt user U for honest sender S, it checks $a' \in \langle g \rangle$ and returns to $\mathcal{A}$ the value $(\pi, b = (a')^k, \zeta)$, where π, is the parameter of S and ζ a proper NIZK and sends to $\mathcal{F}_{\text{VOPRF}}$ the message (SenderComplete, sid, S).

7. It maintains a table for the random oracle $H_2(\cdot)$ defining responses to queries (as usual, repeated queries are answered identically). Entries for queries not previously answered are set to random elements of $\{0,1\}^\ell$, except in the following case. If the query is of the form (π, x, u), where $\pi = (y)$ and for some j the pair (x, g_j) belongs to the $H_1(\cdot)$ table and (g, y, g_j, u) is a DDH-tuple (which can be tested via $u = y^{r_j}$), then:

   Case I. If $\pi = (y) = (g^k)$ is a parameter assigned to an honest sender S, then SIM sends (Eval, sid, S^*, x) on behalf of a corrupted user U^* directed to an arbitrary sender S^* and immediately follows it with (UserComplete, $sid, U^*, \pi, \top$). If the UserComplete is ignored then SIM outputs FAIL$'$ and terminates. Otherwise it sets the $H_2(\cdot)$ query to be equal to the response of $\mathcal{F}_{\text{VOPRF}}$ and records the triple $(\pi, j, u) = (\pi, j, g_j^k)$.

   Case II. If $\pi = (y)$ is not assigned to an honest sender, SIM sets its response to the $H_2(\cdot)$ query to be equal to $\mathsf{f_k}(\pi, x)$.

Figure 4: The Simulator $\mathsf{SIM}^{\mathsf{f_k}(\cdot)}$ for the 2Hash-NIZK protocol where $\{\mathsf{f_k}(\cdot)\}_\mathsf{k}$ is a PRF.

output that was (potentially earlier) assigned to x by the $\mathcal{F}_{\text{VOPRF}}$. The $\mathcal{DH}(\cdot)$ oracle is available to SIM since it selects the value key k for each sender. Note that the EVAL, USERCOMPLETE rapid command succession issued on behalf of the corrupted user U^* also sets the value of the function on x if this value was not set before. Therefore, there is no problem of SIM having to set H_2 on a point for which $\mathcal{F}_{\text{VOPRF}}$ will only generate an output in the future.

In summary it is evident that SIM jointly with $\mathcal{F}_{\text{VOPRF}}$ pick values for the table of $H_2(\cdot)$ and SIM is capable of querying $\mathcal{F}_{\text{VOPRF}}$ whenever it needs to answer a $H_2(\cdot)$ query that potentially is set already by the ideal functionality via a rapid succession of EVAL, USERCOMPLETE commands on behalf of a corrupted user. Note that getting a response from $\mathcal{F}_{\text{VOPRF}}$ on a query of the form $(\pi, x, H_1(x)^k)$ is always contingent on $\mathsf{tickets}(\pi) > 0$ but we will show below (with the analysis of the FAIL$'$ event) that the probability that this condition is violated is negligible.

The case of 5.III corresponds to an honest user U on input x completing an interaction with a corrupt sender with parameter π' (and corresponding exponent k'). In this case, $\mathcal{F}_{\text{VOPRF}}$ sets the value of the function on x using the Turing machine M provided by the attacker and which is defined by SIM in step 5.III. The way this TM is defined ensures that the value given to this point is consistent with previous values of the function table corresponding to π'. In particular, if H_2 was *previously* set by SIM for the triple $(\pi', x, H_1(x)^{k'})$ then $\mathcal{F}_{\text{VOPRF}}$ will set the function to this value. This is important since it is possible for the attacker to query this input before a USERCOMPLETE for this x is issued (e.g., even before a user in the ideal world queried x). Note that the ideal functionality chooses values for inputs not already set by H_2 and therefore SIM needs a way to query the value of the function on x when presented with input $(\pi', x, H_1(x)^{k'})$. For this task, SIM uses the rapid succession of EVAL, USERCOMPLETE on behalf of the corrupted user U^* in step 7.

Regarding cases where PRF output is not generated, we have the case at step 5.II where SIM asks from $\mathcal{F}_{\text{VOPRF}}$ to return $\perp$ to the user. Finally, we consider the output generated by the sender which is a mere acknowledgement symbol produced in step 4 or 6. The generation of these messages coincides with the termination of the sender to user communication in the real world (as simulated by SIM) hence also indistinguishable from real world execution.

Finally, the adversary is able to produce PRF values via the control of adversarial users. The view of dishonest users is provided in two different steps in the operation of SIM. First in step 6, SIM responds to an adversarial user in a way that is indistinguishable to the real world operation of an honest sender. Second, an adversarial user may seek to complete the evaluation of a PRF value via a suitable query to the $H_2(\cdot)$ oracle. As stated already, SIM, via the $\mathcal{DH}(\cdot)$ oracle, has the ability to detect such critical queries to the $H_2(\cdot)$ oracle and using a rapid succession of EVAL, USERCOMPLETE command on behalf of a corrupted user U^* it can respond properly in a way that is indistinguishable from real world execution. As mentioned the successful simulation of this step, is contingent on not producing the event FAIL ever which is something we argue about next.

Note. The above hypothetical lines of attack where the attacker presents "problematic" queries based on the knowledge of honest users' inputs are possible in the UC setting. The environment knows (and even chooses) these queries so it can communicate them to the attacker before or after the user was activated with them.

We now show that the probability of SIM outputting FAIL$'$ in step 7 is related to the One-More DH assumption and is therefore negligible for groups where this assumption holds.

Note that FAIL$'$ happens when SIM calls $\mathcal{F}_{\text{VOPRF}}$ with a rapid succession of EVAL, USERCOMPLETE commands with argument π on behalf of corrupted user U^* but the value of $\mathsf{tickets}(\pi)$ is below 1. We observe several properties of the variable tickets. It is only defined and operated for honest π (i.e., π owned by a honest sender), its value is never less than zero (it starts at zero and it is

decreased only if its current value is positive). The only command in $\mathcal{F}_{\text{VOPRF}}$ that triggers an increase is SENDERCOMPLETE which SIM calls in Steps 4 and 6. These calls cover all cases where SIM computes a $(\cdot)^k$ operation for some honest parameter k. The only $\mathcal{F}_{\text{VOPRF}}$ command that triggers a decrease of tickets is USERCOMPLETE with $M = \top$ and an honest function parameter π. SIM calls this command in Steps 5.I and Step 7.I, respectively. In each of these calls, SIM records a triple (π, g_j, g_j^k) for g_j's from the gseq sequence. In the case of step 5.I, $\mathsf{DDH}(g, y, v, c)$ holds where $v = g_j$ for g_j chosen and recorded as v in Step 3. In the case of step 7.I, the g_j value is an element from gseq that was chosen as the output of $H_1(x)$ upon a query by the adversary $\mathcal{A}$ on input x. Note that the sets of g_j's used in the 5.I and 7.I cases respectively correspond to different index sets within the gseq sequence.

Now, assume a run of SIM where event FAIL$'$ occurs for $\mathsf{tickets}(\pi), \pi = (y = g^k)$. Denote by Q the total number of times $\mathsf{tickets}(\pi)$ is increased in that run. Consider the call to USERCOMPLETE by SIM that produces the FAIL$'$ event. Let (π, x, v) be the query to H_2 that generates this call and let g_j^* be such that $g_{j^*} = H_1(x)$ and $v = g_{j^*}^k$. At the time the call happens the value of $\mathsf{tickets}(\pi)$ is zero; hence there were Q instances where $\mathsf{tickets}(\pi)$ was increased and Q instances where it was decreased. By the above observations, and assuming there are no collisions between the gseq elements, these instances correspond to exactly Q invocations of $(\cdot)^k$ and to the production of Q triples (π, j, g_j^k) by SIM with different indexes j. The latter triples do not include $(\pi, j^*, v = g_{j^*}^k)$, hence considering this triple we have a total of $Q + 1$ triples (π, j, g_j^k) produced by SIM with only Q invocations of $(\cdot)^k$. Given that the event of a collision among the gseq elements is bounded by N^2/m we can condition the simulation on the complement of this event with a failure probability bounded by that (negligible) amount.

Thus, we can build an attacker against the One-More DH (OMDH) assumption as follows. Let $(q, g, y, g_1, \ldots, g_N)$ define an instance of the One-More DH problem. We run the above simulator SIM against the given adversary, choosing one of the honest senders at random and setting its parameter $\pi^* = (y = g^{k^*})$ using the inputs from the above OMDH instance. In addition, we set the gseq sequence to the values $g_1, \ldots, g_N$ from that instance too. SIM runs as before except that each time a $(\cdot)^{k^*}$ computation is required we invoke the given OMDH oracle. The NIZK proofs on behalf of the $(\cdot)^{k*}$ sender will have to be simulated by exploiting the fact that SIM controls the $H_3(\cdot)$ oracle. Such proof is generated by choosing s, w at random and inserting in the $H_3(\cdot)$ table the value $\langle (g, y, a, b', g^s y^{-w}, a^s b'^{-w}), w \rangle$. The simulator may fail here in case such value was already submitted by the adversary to the oracle $H_3(\cdot)$ (and thus corresponded to a different output). This means that If q_3 is the number of queries to the $H_3(\cdot)$ oracle performed, there is a probability q_3/m^2 to encounter such event (independently of the choice of the sender).

Finally observe that at step 5.III and step 7 the SIM is supposed to verify that a certain sequence of the form (g, y, g_j, u) is a DDH-tuple. In the original version of the simulator this was feasible due to the fact that the simulator knows the value $r_j = \log_g g_j$. Nevertheless, now that the value g_j is determined according to the OMDH challenge this validity test cannot be supported by SIM in the same way. However, observe that given that the simulator acts as an OMDH attacker in the Gap setting it therefore has access to an $\mathsf{DDH}(\cdot)$ oracle. Hence SIM can invoke this oracle to complete these tests.

By the above analysis, the probability that we find $Q + 1$ distinct pairs $(j, g_j^{k^*})$ when we only queried Q times the OMDH oracle $(\cdot)^{k^*}$, is the same as the FAIL$'$ probability. Hence, this probability is upper bounded by the assumed security of the OMDH instance. Note that since the simulator guesses the sender to plug in the OMDH challenge a factor of q_S will be also accrued.

We complete the argument by summarizing the steps that are required in order to show the upper bound in the distance between the real and ideal world. First we have three types of failing

events: (i) failing the simulation of the NIZK proofs for honest senders, (ii) accepting a invalid statement from an impersonated honest sender (the event FAIL in the simulation), and (iii) collision between the gseq elements. These three events have probability upper bounds $q_3/m^2, 2q_U/m, N^2/m$ respectively. Conditional on any of them not happening, we can argue a bound for the probability α of FAIL$'$. As shown above with probability α/q_S we successfully break the OMDH assumption, hence $\alpha \leq q_S \cdot \epsilon_{\mathsf{omdh},G}(N, Q)$. This completes the proof that $\mathsf{SIM}^{R(\cdot)}$ interacting with the ideal functionality simulates the view of $\mathcal{Z}^{H_1(\cdot),H_2(\cdot),H_3(\cdot)}$. We complete the proof of the theorem by substituting back $R(\cdot)$ with the PRF $\mathsf{f}_\mathsf{k}(\cdot)$. Based on the PRF function property we can easily show that the statistical distance between the executions of $\mathcal{Z}^{H_1(\cdot),H_2(\cdot),H_3(\cdot)}$ when interacting with $\mathsf{SIM}^{R(\cdot)}$ and $\mathcal{Z}^{H_1(\cdot),H_2(\cdot),H_3(\cdot)}$ when interacting with $\mathsf{SIM}^{\mathsf{f}_\mathsf{k}(\cdot)}$ is bounded by $\epsilon_{\mathsf{PRF}}(q_2)$. □

Parameters: generator g of cyclic group of order q and two hash functions $H_1(\cdot), H_2(\cdot)$

Key Generation: Upon receiving (KEYGEN, *sid*), pick $k \in_R \mathbb{Z}_m$ and set $y = g^k$ and $\pi = y$. Return (PARAMETER, *sid*, π).

V-OPRF Evaluation: Upon receiving (EVAL, *sid*, S, x) pick $r \in_R \mathbb{Z}_m$ and transmit $H_1(x)^r$ to S.

Upon receiving a message $a = H_1(x)^r$ from some network entity U compute $b = a^k$. Send to U the message $\langle y, b\rangle$.

Upon receiving a message $\langle y, b\rangle$ from a party U test that $\mathsf{DDH}(g, y, a, b)$ is true. If the test passes return (EVAL, $y, H_2(y, x, b^{1/r})$, else return (EVAL, $y, \perp$).

Figure 5: Protocol 2HashDH-GAP.

Parameters: Public-key length 1^λ and two hash functions $H_1(\cdot), H_2(\cdot)$.

Key Generation: Upon receiving (KEYGEN, *sid*), pick e, n RSA public-key and return (PARAMETER, *sid*, π), for $\pi = (e, n)$. Store locally $d = e^{-1} \bmod \phi(n)$.

V-OPRF Evaluation: Upon receiving (EVAL, *sid*, S, x) request the parameters of S.

Upon receiving a message asking the parameter from some U transmit $\pi = (e, n)$.

Upon receiving $\pi = (e, n)$ from some S compute $a = H_1(x)r^e \bmod n$ and transmit a back to S.

Upon receiving a from some U compute $b = a^d \bmod n$ and send to U the message $\langle \pi, b\rangle$.

Upon receiving a message $\langle \pi, b\rangle$ from some S with $\pi = (e, n)$ test that $b^e \bmod n = a$ is true. If the test passes return (EVAL, $n, e, H_2(n, e, x, b \cdot r^{-1} \bmod n)$), else return (EVAL, $y, \perp$).

Figure 6: Protocol 2HashRSA.

Following the same logic as the above proof we can prove that two further protocols, 2HashDH-GAP and 2HashRSA, shown in the two figures above, are secure respectively under the One-More Diffie-Hellman Assumption over gap-groups and under the One-More RSA Assumption.

Corollary 2 *The 2HashDH-GAP protocol over a gap group G of order m (resp. the 2HashRSA protocol) UC-realizes $\mathcal{F}_{\text{VOPRF}}$ per Fig. 2 in the random oracle model assuming the (N, Q) One-*

More DH assumption on group G (resp. the One-More RSA assumption) where Q is the number of V-OPRF executions and $N = Q + q_1$ where q_1 is the number of $H_1(\cdot)$ queries.

More precisely, for any adversary there is an ideal-world adversary (simulator) that produces a view in the ideal world that no environment can distinguish with advantage better than $q_S \cdot \epsilon(N, Q) + \epsilon_{\mathsf{PRF}}(q_2)$ where $\epsilon(N, Q)$ is the advantage of the corresponding assumption, q_S is the number of senders, q_2 is the number of queries to the $H_2(\cdot)$ oracle and $\epsilon_{\mathsf{PRF}}(q_2)$ is the security of the PRF function against polynomial-time adversaries posing q_2 queries. In the case of the One-More DH assumption the collision probability N^2/m is also added to the above.

Proof: We provide a brief sketch of the proof. The simulation argument is essentially the same as that of theorem 1 with some simplifications. First due to the fact that honest users (and the simulator SIM) can verify the validity of the response the sender, the use of NIZK's and NIZK simulation is obviated. This affects step 4, where SIM does not need to compute ζ as well as step 5, where SIM decides the three cases based on the validity of the sender's response (via checking either the bilinear map equation or the verification of the RSA signature). The event FAIL in step 5.I is infeasible since the verification check precludes it entirely (so the soundness of the underlying interaction is perfectly upheld). Similarly, step 7 is simplified given that SIM can recognize the critical area of the domain of $H_2(\cdot)$ using the verification check. This has the simplification that it is unnecessary to program the random oracle $H_1(\cdot)$ and the responses can be raw random values from the appropriate domain depending on the scheme (either group elements of $\langle g \rangle$ or $\mathbb{Z}_n^*$). The event FAIL$'$ will still of relevance and will provide the reduction to the corresponding one-more assumption. Note that during this step the simulation of NIZK's on behalf of the selected sender whose parameter is used for the reduction is also obviated. □

3.2 Realizing $\mathcal{F}_{\text{VOPRF}}$ in the Common Reference String model

We next demonstrate how to efficiently realize $\mathcal{F}_{\text{VOPRF}}$ in the common reference string model without assuming random oracles. Our construction requires 4 communication moves and communication complexity of $O(\ell \cdot \lambda)$ where λ is the security parameter and ℓ the number of bits in the domain of the PRF. The construction builds on top of the Naor Reingold (NR) PRF [40] that has been suggested before in the context of V-OPRF [25] but without providing verifiability or a UC-level of security. We recall briefly the NR function: the sender's key is equal to a secret vector $a_1, \ldots, a_\ell \in \mathbb{Z}_m$ and the function is defined as $g^{\prod_{j=1}^{\ell} a_j^{x_j}}$ where $x_1, \ldots, x_\ell$ are the ℓ bits of the input string $x \in \{0,1\}^\ell$. Building an V-OPRF from the NR function is suggested in [25] by having the sender transmit $A = g^{1/r_1 \ldots r_\ell}$ for some random masking values $r_1, \ldots, r_\ell \in \mathbb{Z}_m$. Subsequently the sender and receiver engage in ℓ oblivious transfers from which the receiver obtains the ℓ values $a_j^{x_j} r_j \bmod m$ where $x_j \in \{0,1\}$ is the private input of the receiver. Given these values and A it is straightforward for the receiver to extract the value $g^{\prod_{j=1}^{\ell} a_j^{x_j}}$.

It is worth noting that the above construction cannot realize $\mathcal{F}_{\text{VOPRF}}$ even when composed on top of a UC-secure oblivious transfer protocol. The main reason is that the sender is not required to commit to a function by providing a function descriptor π against which the receiver can verify the protocol output (see the explicit attack mentioned in the introduction). It follows that in order to implement $\mathcal{F}_{\text{VOPRF}}$ we need to provide a sender parameter π against which the sender will have to demonstrate the correctness of the computed value. We can easily achieve this as follows. The sender determines its public parameter to be a sequence $g^{a_0}, g^{a_1}, \ldots, g^{a_\ell}$. The underlying PRF function is then defined as $g^{a_0 \prod_{j=1}^{\ell} a_j^{x_j}}$ and a proof of correct evaluation is also added to the

requirements of the sender side. Note that the revelation of π does not hurt the pseudorandomness of the underlying PRF function since without loss of generality the parameters $g^{a_0}, \ldots, g^{a_\ell}$ can be thought of as $\ell+1$ evaluations of a NR-PRF with domain $\{0,1\}^{\ell+1}$. We combine the above ideas in a protocol that uses 4 communication moves and relies on the existence of a short common random string.

The main idea of the protocol is as follows. In the first move, the receiver encrypts using Paillier encryption a sequence of "indicator ciphertexts" containing either $(0,1)$ or $(1,0)$ according the bits of $x \in \{0,1\}^\ell$. The receiver sends those ciphertexts and commits to a first move $\zeta_{R,1}$ of a Σ-proof for the consistency of these ciphertexts. The sender chooses the blinding factors $r_1, \ldots, r_\ell$ and prepares the values $a'_j = a_j r_j \bmod m$ for $j = 1, \ldots, \ell$. The indicator ciphertexts are subsequently used by the sender on the pairs of values (r_j, a'_j), an operation that results in a sequence of ℓ pairs of ciphertexts so that each pair contains the values $\{0, r_j a_j^{x_j} \bmod m\}$). However, the sender should not send these ciphertexts yet. Indeed, the sender runs the risk that the original ciphertexts pairs submitted by the receiver are malformed. For this reason, the sender simply commits to the ciphertext pairs (as well as to the first move $\zeta_{S,1}$ of a Σ-protocol for the correctness of these ciphertexts) and provides a challenge for the Σ-protocol started by the receiver. The commitment used by the sender is a perfectly hiding equivocal commitment; at this stage it ensures that no information is leaked. The receiver now opens the commitment for its proof in the first move and the ciphertexts that enable the calculation of the values $\{r_j a_j^{x_j} \bmod m\}_{j=1}^{\ell}$ which together with A facilitate the computation of the PRF value. It also provides the final move of the proof $\zeta_{R,2}$. The sender verifies the proof and opens its own commitment along with its final move $\zeta_{S,2}$. Observe that it is possible for the simulator to extract any malicious sender's key by equivocating the commitment and cheating in the proof of consistency of the indicator ciphertexts. Specifically, the simulator can choose all ciphertexts to encode the value 1, something that will permit the recovery of the sequence of values r_j, a'_j. In this case, the simulator can extract the sender's key by computing $a_j = a'_j (r_j)^{-1} \bmod m$ for $j = 1, \ldots, \ell$.

A parallel composition of a number of Σ-proofs are necessary for the completion of the protocol. We first describe the necessary Σ-protocol components. We use the notation C_a for some $a \in \mathbb{Z}_m$ to denote a commitment of the form $g^r h^a$ (where the values g, h will be placed in the *crs*). Furthermore we use $\mathsf{enc}(\cdot)$ for the variant of Paillier encryption [43] that was proposed in [12]. Ciphertexts are of the form $\mathsf{enc}(m;\rho) = \langle g^\rho, y^\rho (1+n)^m \rangle$ where all operations are taken modulo $\tilde{n}^2$ (we do not require the CCA2 secure version). As in the case of Paillier the scheme is additively homomorphic and can be decrypted by the application of $\log_g y$. We note that we will place the modulus $\tilde{n}$ in the *crs* and we will select it to be "safe", i.e., a product of two safe primes. This does not prevent a party from generating a public-key g, y as it can choose random $x \in [0, \lfloor n/4 \rfloor]$ and set $y = g^x \bmod \tilde{n}^2$. However it should be noted that the small order elements $\{-1, 1\}$ in $\mathbb{Z}^*_{\tilde{n}^2}$ are known to all parties and hence it is possible to choose public-keys of the form $y = -g^x \bmod \tilde{n}^2$ or ciphertexts of the form $\langle g^r, -h^r (1+n)^m \rangle$. This can be problematic in some cases, e.g., when a party computes a ciphertext $\psi' = (\mathsf{enc}(v;r))^a \cdot \mathsf{enc}(0;r)$ (here "$\cdot$" is component-wise multiplication within $\mathbb{Z}^*_{\tilde{n}^2}$). Note that assuming an honestly generated ψ, the value ψ' would be identically distributed to a ciphertext encrypting $v \cdot a \bmod \tilde{n}$, however for a maliciously generated ciphertext ψ the computation of ψ' potentially leaks a bit of a. This can be easily fixed by squaring all untrusted elements of $\mathbb{Z}^*_{\tilde{n}^2}$. Note that this will introduce a factor of 2 in all plaintexts but this can be easily removed after decryption by the receiver since $\gcd(2, n) = 1$.

We present the protocol in Figure 7. For readability we abstract some of the operations including the ciphertext encryption and decryption operation as well as the zero-knowledge proofs executed by the two players. Below we list in detail the Σ-protocols that we require in the construction.

Parameters: Public-key length 1^λ, parameters for a perfectly binding commitment $(\mathsf{com}, \mathsf{open})$ over $\mathbb{Z}_m$, verification key vk for a digital signature scheme, safe RSA moduli $\overline{n}$, $\tilde{n} = pq$ with $p = 2p' + 1, q = 2q' + 1$ and element $\tilde{g}$ of order $p'q'$ in $\mathbb{Z}^*_{\tilde{n}^2}$. Denote $\mathsf{enc}(v; \rho) = (\tilde{g}^\rho, \tilde{h}^\rho(1+n)^v)$.

Key Generation: Pick $a_0, a_1, \ldots, a_\ell \in_R \mathbb{Z}_m$ and set $\pi = (g^{a_0}, \ldots, g^{a_\ell})$.

V-OPRF Evaluation: Upon receiving (EVAL, sid, S, x) choose pk for $\mathsf{enc}(\cdot)$, a one-time signature key vk_R and form the values $\psi_{0,j} = \mathsf{enc}(x_j; \rho_{0,j})$ and $\psi_{1,j} = \mathsf{enc}(1 - x_j; \rho_{0,j})$. Prepare a first step of a proof $\zeta_{R,1}$ that $\psi_{0,j}, \psi_{1,j}$ are correctly formed, i.e., they encode $\{x_i, 1 - x_i\}$ and transmit $(\mathsf{pk}, \psi, \mathsf{vk}_R, \zeta_{R,1})$.

Upon receiving a message $(\mathsf{pk}, \psi = \langle \psi_{0,j}, \psi_{1,j} \rangle_{j=1}^{\ell}, \mathsf{vk}_R, \zeta_{R,1})$ from some network entity U, choose one-time signature key vk_S and $r_1, \ldots, r_\ell \in_R \mathbb{Z}_m$ and compute $a'_j = (a_j \cdot r_j \bmod m)$ and $\tilde{\psi}_{0,j} = \mathsf{enc}(x_j; \rho_{0,j})^{r_j} \cdot \mathsf{enc}(0; \tilde{\rho}_{0,j})$. $\tilde{\psi}_{1,j} = \mathsf{enc}(a'_j; \rho_{1,j})^{r_j} \cdot \mathsf{enc}(0; \tilde{\rho}_{1,j})$. Then, compute the commitments: C_{a_j}, C_{r_j}, $C_{a'_j}$ and the value $A = g^{a_0/\overline{r}_\ell}$ as well as the first move of the proof that they are correctly formed as $\zeta_{S,1}$. Collect all commitments and values $\tilde{\psi} = (\tilde{\psi}_{0,1}, \tilde{\psi}_{1,1}, \ldots, \tilde{\psi}_{0,\ell}, \tilde{\psi}_{1,\ell}, A)$ as well as $\zeta_{S,1}$ and commit to them into C_S. Finally transmit $(C_S = \mathsf{com}(\pi, \tilde{\psi}, \zeta_{S,1}), ch_S, \mathsf{vk}_S)$, where ch_S is the sender's challenge.

Upon receiving $\langle C_S, ch_S, \mathsf{vk}_S \rangle$ from the network entity S compute the response $\zeta_{R,2}$ to the proof of well formedness for ψ as well as a signature σ_R under vk_R on the whole communication transcript; transmit $(\mathsf{open}(C_R), \zeta_{R,2}, ch_R, \sigma_R)$ where ch_R is the receiver's challenge.

Upon receiving $\langle \zeta_{R,2}, ch_R, \sigma_R \rangle$ from party U verify the proof $[\zeta_{R,1}, ch_S, \zeta_{R,2}]$ as well as the signature σ_R and if the test passes then compute the proof $\zeta_{S,2}$ using ch_R as the challenge; send to U the message $\langle \mathsf{open}(C_S), \zeta_{S,2}, \sigma_S \rangle$ where the signature σ_S signs the communication transcript under vk_S.

Upon receiving a message $\langle \mathsf{open}(C_S), \zeta_{S,2}, \sigma_S \rangle$ from a party S verify the proof $[\zeta_{S,1}, ch_R, \zeta_{S,2}]$, and the signature σ under vk_S. If the test passes, decrypt all ciphertexts $(\tilde{\psi}_{0,1}, \tilde{\psi}_{1,1}, \ldots, \tilde{\psi}_{0,\ell}, \tilde{\psi}_{1,\ell})$ in order to obtain the plaintexts $\{0, a'_1, \ldots, a'_\ell\}$ and return $(\text{EVAL}, \pi, A^{a'_1 \cdot \ldots \cdot a'_\ell})$. Else, return $(\text{EVAL}, \pi, \perp)$.

Figure 7: Protocol NR-V-OPRF: The Naor-Reingold based $\mathcal{F}_{\text{VOPRF}}$.

1. A proof of knowledge Π_{sig} of a signature on a public-message w.r.t. a given public-key vk. Many digital signature schemes accept such proof of knowledge, including the standard DSA algorithm [42], however since we will utilize the Strong-RSA assumption we opt to employ as an underlying digital signature scheme of Cramer-Shoup [17].

2. A proof Π_{mult} that C_a, C_b, C_c satisfy $c = a \cdot b \bmod m$. A Σ-protocol for this is standard, see [16].

3. A proof Π_{cross} that the commitment C_a and the ciphertext $\psi' = \psi^a \cdot \mathsf{enc}(0; \rho) \bmod n^2$ are consistent for some *integer* a where ψ is a given ciphertext. Observe that the relation here is discrete-log based, and hence a Σ-protocol can be constructed using a so-called generalized Schnorr proof, see [9]. Such proofs rely on integer commitments [26, 19] and have a special soundness property that only holds computationally under the Strong-RSA assumption with respect to the RSA modulus $\overline{n}$ that is placed in the *crs*. Note that for this protocol, the soundness property that is provided cannot preclude the existence of small order elements of $\mathbb{Z}^*_{\tilde{n}^2}$ to be present in ψ' (note -1 is the only such element that is computable by the parties since the factorization of $\tilde{n}$ is unknown); nevertheless this small soundness deviation is inconsequential. In addition it cannot be precluded that ψ' contains a value that is of the form $a + \mu \cdot m$ for some small integer μ. This soundness deviation is also inconsequential for our purposes: when the ciphertext ψ' will be decrypted, the plaintext will be reduced modulo m.

4. A proof $\Pi_{\mathsf{mult}'}$ that C_a and g^x, g^c satisfy that $c = a \cdot x \bmod m$. A Σ protocol for this can be easily constructed from the protocol of case 1 above.

5. A proof that the ciphertext $\mathsf{enc}(a)$ is valid and encrypts a specific value a (denoted by Π^a_{cons}) or some value $a \in \{0, 1\}$ (denoted by $\Pi^{\{0,1\}}_{\mathsf{cons}}$). A Σ protocol can be easily derived from standard proofs of consistency for Paillier ciphertexts, see [3] and also [12].

We now detail how the $(\zeta_{S,1}, \zeta_{S,2})$ and $(\zeta_{R,1}, \zeta_{R,2})$ proofs are determined (refer to figure 7 for the protocol).

For the sender's proof $(\zeta_{S,1}, \zeta_{S,2})$, we utilize the compiler of [27] to obtain a universally simulation sound zero-knowledge proof from a Σ-protocol. Specifically, the proof $(\zeta_{S,1}, \zeta_{S,2})$ will be a Σ-protocol that either demonstrates correctness of a sequence of statements detailed below or the knowledge of a signature on the one-time signature key vk_S. First, the sender executes of Π_{mult} for the triple $C_{a_j}, C_{r_j}, C_{a'_j}$ for $j = 1, \ldots, \ell$. For each $\tilde{\psi}_{0,j}$ he will show that it is appropriately computed based on $\psi_{0,j}$ and C_{r_j} executing Π_{cross} and respectively for $\tilde{\psi}_{1,j}$ he will show that it is appropriately computed based on $\psi_{1,j}$ and $C_{a'_j}$. For each $j > 1$, the sender also computes $\overline{r}_j = r_1 \ldots r_j \bmod m$ with $\overline{r}_1 = r_1$ and produces the commitments $C_{\overline{r}_j}$. He will also commit to $C_{\overline{r}_{j-1} \cdot r_j}$ and will show that this value is consistent with $C_{\overline{r}_{j-1}}$ and C_{r_j} by executing Π_{mult} for each triple $j = 2, \ldots, \ell$. Finally, the sender shows that that the value v committed in $C_{\overline{r_j}}$ satisfies that $v = \log_g A \cdot a_0 \bmod m$ using the proof $\Pi_{\mathsf{mult}'}$.

We now turn to the case of the receiver proof $(\zeta_{R,1}, \zeta_{R,2})$. As before the receiver will perform an OR-proof to show that she either knows a signature on vk_R under vk or 3ℓ Σ-protocols defined below. Recall, that the receiver will need to show that the ciphertext pairs $(\psi_{0,j}, \psi_{1,j})$ encrypt the two values $\{0, 1\}$ (in some arbitrary hidden order). This can be done via executing the proof $\Pi^{\{0,1\}}_{\mathsf{cons}}$ for each ciphertext of $(\psi_{0,j}, \psi_{1,j})$. Then it will combine the two ciphertexts $\psi_{0,1,j} = \psi_{0,j} \cdot \psi_{1,j} \cdot \mathsf{enc}(0; \rho_j) \bmod \tilde{n}^2$ and show that the ciphertext $\psi_{0,1,j}$ contains the value 1 via the proof $\Pi^{\{1\}}_{\mathsf{cons}}$.

Theorem 3 *The NR-V-OPRF protocol UC-realizes $\mathcal{F}_{\text{VOPRF}}$ per Fig. 2 in the common reference string model assuming the (i) strong-RSA assumption, (ii) Decisional composite residuosity assumption, (iii) the hiding and binding property of the commitment* $\mathsf{com}, \mathsf{open}$.

Proof: We sketch the simulation argument. With respect to the CRS, our simulator SIM will possess the signing key sk that corresponds to vk, and the factorization of the modulus $\tilde{n}$. This knowledge endows SIM with important capabilities: (i) first using the signing key sk, SIM can fake all proofs it performs on behalf of an honest sender or receiver. Using this capability means that it is easy for SIM to craft a fake proof that the sequence of ciphertexts $\psi_{0,j} = \mathsf{enc}(1; \rho_{0,j}), \psi_{1,j} = \mathsf{enc}(1; \rho_{1,j})$ contain the $\{0, 1\}$ elements. Observe that this leads to a complete loss of privacy for any malicious sender: the PRF keys $a_1, \ldots, a_\ell$ can be reconstructed. The use of the Decisional-Composite Residuosity assumption is critical here as in this step the simulator deviates from the real world by removing the dependency to honest user input and fixing the ciphertext to constant plaintext values. Given this extractability capability, SIM is capable of registering the precise NR instance that is associated with each parameter π; specifically, for all malicious parameters, SIM can construct the TM that implements the NR function for $a_1, \ldots, a_\ell$ and g^{a_0} and deposit it at $\mathcal{F}_{\text{VOPRF}}$ with a $(\textsc{Parameter}, \cdot)$ statement. This ensures that $\mathcal{F}_{\text{VOPRF}}$ will produce the correct output to honest users interacting with malicious sender parameters. (ii) Using the signing key sk that corresponds to vk the simulator can also fake the consistency proof on behalf of any honest sender. This is a critical feature since the values $a_1, \ldots, a_\ell$ will be unknown to SIM when it is transformed to a reduction for the pseudorandomness of the Naor-Reingold PRF. The techniques of [27] apply here and we can show that the adversary will be unable to generate an invalid proof even after seeing an unbounded number of fake proofs generated by SIM. The special soundness of the sub-protocols in $(\zeta_{S,1}, \zeta_{S,2})$ is relevant here since in case of a fake proof the simulator will fork a random malicious sender and obtain via special soundness a forgery on the underlying signature scheme. Nevertheless, this step is complicated by the fact that due to proof Π_{cross} special soundness is only computationally ensured. However it is possible to modify the forking argument of [27] and consider two events as follows: either the generation of a forgery is produced via special soundness, or the recovery of an integer value that prevents witness reconstruction in Π_{cross} takes place. In this latter case it follows that one can directly break the Strong-RSA assumption w.r.t. the *crs* modulus used for integer commitments (cf. [26, 19, 9]). Due to the fact that a digital signature scheme based on Strong-RSA is used we deduce that simulation-soundness is preserved under the strong-RSA assumption. Using this simulation strategy, SIM can interact with any malicious user on behalf of an honest sender. Nevertheless, there is still an important consideration requiring another simulator capability: (iii) given that SIM will be providing a completely fake transcript to the adversary on behalf of an honest sender, it must ensure that proper output to the malicious users will be provided in all cases. This can be achieved via extracting the malicious user input from ciphertexts $\psi_{0,j}, \psi_{1,j}$ utilizing the global encryption trapdoor enabled by the knowledge of the factorization of $\tilde{n}$. It follows that for any malicious user that completes an interaction with an honest sender, SIM will be capable of querying $\mathcal{F}_{\text{VOPRF}}$ via a rapid succession of Eval, UserComplete on behalf of a corrupted user in order to obtain the proper output. It is straightforward to see that it is possible for SIM to choose the value A so that the output of the malicious user will be equal to target value ρ. □

4 Password-Protected Secret Sharing: Definitions

Our definition of PPSS adapts the PPSS notion of [2] to the CRS model, but also re-defines PPSS in terms of a *key derivation mechanism* rather than an encryption-style notion used in [2]. In other

words, rather than used directly to semantically protect any message, a PPSS will generate and protect a random key. This change allows for better modularity, because the resulting key can be used not only for message encryption (and authentication), but also e.g. for an Authenticated Key Exchange. A Password-Protected Secret Sharing (PPSS) scheme in the CRS model is a protocol involving $n+1$ parties, a user U, and n servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$. A PPSS scheme is a tuple $(\mathsf{ParGen}, \mathsf{SKeyGen}, \mathsf{Init}, \mathsf{Rec})$, where ParGen and $\mathsf{SKeyGen}$ are randomized algorithms and Init and Rec are multi-paty protocols with the following syntax:

- Algorithm ParGen generates public parameters CRS for a given security parameter τ.
- Each S_i runs $\mathsf{SKeyGen}(\mathsf{CRS})$ to generate its private state σ_i and a public parameter π_i.
- Protocol Init is executed by U and servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$, where U runs algorithm $\mathsf{U}_{\mathsf{Init}}$ on inputs a password $\mathsf{pw} \in \{0,1\}^\tau$, global parameters CRS, and a vector of server's public parameters $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$, while each S_i runs algorithm $\mathsf{S}_{\mathsf{Init}}$ on input $(\mathsf{CRS}, \sigma_i, \pi_i)$. The outputs of Init is a τ-bit key K for U and a user-specific information ω_i for each server S_i.
- Protocol Rec is executed by U and servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$, where U runs algorithm $\mathsf{U}_{\mathsf{Rec}}$ on (CRS, pw), and each S_i runs algorithm $\mathsf{S}_{\mathsf{Rec}}$ on $(\mathsf{CRS}, \sigma_i, \pi_i, \omega_i)$. Protocol Rec generates no output for the servers, while U outputs K' which is either a τ-bit string or a rejection symbol $\perp$.

The *correctness* requirement on a PPSS is that Rec returns the same key K which was generated in Init, i.e. that for any τ, any CRS generated by $\mathsf{ParGen}(1^\tau)$, any (σ_i, π_i) output by n instances of $\mathsf{SKeyGen}(\mathsf{CRS})$, and any $\mathsf{pw} \in \{0,1\}^\tau$, if $(K, \omega_1, \ldots, \omega_n)$ is the vector of outputs of Init executed on inputs $(\mathsf{pw}, \mathsf{CRS}, \boldsymbol{\pi})$ for U and $(\mathsf{CRS}, \sigma_i, \pi_i)$ for each S_i, then U's local output in an instance of Rec executed on inputs $(\mathsf{CRS}, \mathsf{pw})$ for U and $(\mathsf{CRS}, \sigma_i, \pi_i, \omega_i)$ for each S_i, is equal to K.

Server's User-Related State. We stress that the state $(\sigma_i, \pi_i, \omega_i)$ of each server S_i is stored *for each user separately*, and the PPSS security notion we define below assumes that each S_i stores a separate $(\sigma_i, \pi_i, \omega_i)$ tuple for each user account. Indeed, the security of the PPSS protocol we present in Section 5 would be decreased if S_i re-uses the same OPRF key, stored in σ_i in this PPSS protocol, across multiple user accounts. (Technically, the adversary would get additional oracle access to the same $\mathsf{S}^{\diamond}_{\mathsf{Rec}}$ oracle, see below, for each user account on which the server re-uses the same (σ_i, π_i) pair.) Consequently, if S_i wants to provide PPSS service to multiple users, it has to generate a separate (σ_i, π_i) pair for each user (these per-user keys can be derived internally by S_i using a PRF and a global PRF key applied to a user's identifier).

Security. We define security of a PPSS scheme in terms of adversary's advantage in distinguishing the key K output by U from a random string. We assume that the adversary sees CRS and the vector of server's public parameters $\boldsymbol{\pi}$ used in the initialization instance, as well as the private states $\boldsymbol{\sigma}_{\mathsf{B}} \triangleq \{\sigma_i\}_{i \in \mathsf{B}}$ and $\boldsymbol{\omega}_{\mathsf{B}} \triangleq \{\omega_i\}_{i \in \mathsf{B}}$ for some set B of corrupted servers, and that it has concurrent oracle access to instances of $\mathsf{U}_{\mathsf{Rec}}(\mathsf{CRS}, \mathsf{pw})$ and $\mathsf{S}_{\mathsf{Rec}}(\mathsf{CRS}, \sigma_i, \pi_i, \omega_i)$, for i in $\overline{\mathsf{B}} \triangleq \{1, ..., n\} \setminus \mathsf{B}$. We denote as $\mathsf{U}^{\diamond}_{\mathsf{Rec}}(\mathsf{CRS}, \mathsf{pw}, b, K^{(0)})$ an oracle which executes the interactive algorithm $\mathsf{U}_{\mathsf{Rec}}(\mathsf{CRS}, \mathsf{pw})$, and when this algorithm terminates with a local output K, the $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ oracle (re-)sets K to $K^{(0)}$ if $b = 0$ and $K \neq \perp$, and then returns K to the caller. However, if $b = 1$ or $K = \perp$ then the caller receives the unmodified value K (to which we will refer as $K^{(1)}$) as it was output by the $\mathsf{U}_{\mathsf{Rec}}$ instance. We denote as $\mathsf{S}^{\diamond}_{\mathsf{Rec}}(\mathsf{CRS}, \boldsymbol{\sigma}_{\overline{\mathsf{B}}}, \boldsymbol{\pi}, \boldsymbol{\omega}_{\overline{\mathsf{B}}})$ an oracle which on input $i \in \overline{\mathsf{B}}$ executes the interactive algorithm $\mathsf{S}_{\mathsf{Rec}}(\mathsf{CRS}, \sigma_i, \pi_i, \omega_i)$.

Intuitively, we should call an (t, n)-threshold PPSS scheme secure if for any password dictionary D, if pw is randomly chosen in D then the adversary's advantage in distinguishing the PPSS-protected key K from a random string (i.e., guessing b) is at most negligibly above $1/|D|$, the

probability of guessing the password, times $q_u + \lfloor q_s/(t-t'+1) \rfloor$, where q_u and q_s are the numbers, respectively, of the $\mathsf{U}_{\mathsf{Rec}}$ and $\mathsf{S}_{\mathsf{Rec}}$ protocol instances the adversary can interact with, and $t' \leq t$ is the number of corrupted servers. Factor $1/|D| * \lfloor q_s/(t-t'+1) \rfloor$ corresponds to an inherent vulnerability due to on-line dictionary attacks: An adversary who learns the shares of $t' \leq t$ servers can test any password $\tilde{\mathsf{pw}}$ in D by running the user's protocol on $\tilde{\mathsf{pw}}$ interacting with $t-t'+1$ uncorrupted servers. Factor $1/|D| * q_u$ corresponds to an inherent vulnerability of password-authenticated protocols in the CRS model, because the adversary can run the initialization protocol Init on a password guess $\tilde{\mathsf{pw}}$ and then run the servers' protocol interacting with the user: If the user does not reject (by outputting $\perp$), the adversary can conclude that $\tilde{\mathsf{pw}} = \mathsf{pw}$.

To make the PPSS notion easier to use in applications it is important that the adversary sees the key-pseudorandomness challenge, either a real key or a random key, already after the initialization protocol Init, rather than only when this key is reconstructed in protocol Rec. (Our own Threshold PAKE protocol in Section 6 relies on this property of a PPSS scheme.) To make sure that the PPSS-protected key remains pseudorandom in each key usage, whether after the initialization or after each reconstruction instance, we let the adversary see the key generated by Init as well as the key(s) output by every Rec instance. That is, at the end of Init and after each Rec instance the attacker is given $K^{(0)}$ if $b = 0$, but if $b = 1$ then the attacker is given the actual value of the key output by, respectively, $\mathsf{U}_{\mathsf{Init}}$ or $\mathsf{U}_{\mathsf{Rec}}$. Note that $K^{(0)}$ does not change across different reconstructions because it is fixed at the start of the experiment, while $K^{(1)}$ is determined by the actual outputs of $\mathsf{U}_{\mathsf{Init}}$ and $\mathsf{U}_{\mathsf{Rec}}$ instances. Importantly, note that this definition implies that in the real execution the reconstruction instances must output the same key that was created in the initialization or the attacker can trivially guess b. We further discuss this *soundness* property below.

Definition 1 *A PPSS scheme is* (T, q_u, q_s, ϵ)-secure *(for fixed threshold parameters* (t, n) *if for any* $D \subseteq \{0,1\}^\tau$*, any set* B *s.t.* $t' \leq t$ *where* $t' \triangleq |\mathsf{B}|$*, and any algorithm* $\mathcal{A}$ *with running time* T*, we have*

$$\mathbf{Adv}^{\mathsf{ppss}}_{\mathcal{A}} \leq \left(q_u + \left\lfloor \frac{q_s}{t-t'+1} \right\rfloor \right) * \frac{1}{|D|} + \epsilon \tag{1}$$

where $\mathbf{Adv}^{\mathsf{ppss}}_{\mathcal{A}} = |p^{(1)}_{\mathcal{A}} - p^{(0)}_{\mathcal{A}}|$ *and* $p^{(b)}_{\mathcal{A}} = \Pr[b' = 1]$ *in the following experiment defined for* $b \in \{0,1\}$*:*
(1) Choose pw *at random in* D*, generate* $\mathsf{CRS} \leftarrow \mathsf{ParGen}(1^\tau)$ *and* $(\sigma_i, \pi_i) \leftarrow \mathsf{SKeyGen}(\mathsf{CRS})$ *for* $i \in \overline{\mathsf{B}}$*. Give* CRS *and* $\{\pi_i\}_{i \in \overline{\mathsf{B}}}$ *to* $\mathcal{A}$ *and let* $\mathcal{A}$ *generate* $\{\pi_i\}_{i \in \mathsf{B}}$*.*
(2) Run an instance of Init *between* U*, which executes protocol* $\mathsf{U}_{\mathsf{Init}}(\mathsf{CRS}, \mathsf{pw}, \pi_1, \ldots, \pi_n)$*, and the servers, where each* S_i *for* $i \in \overline{\mathsf{B}}$ *executes protocol* $\mathsf{S}_{\mathsf{Init}}(\mathsf{CRS}, \sigma_i, \pi_i)$*, while servers* S_i *for* $i \in \mathsf{B}$ *are controlled by adversary* $\mathcal{A}$*. The protocol proceeds on public channels, with* $\mathcal{A}$ *playing a man-in-the-middle on all communications. Denote* U*'s output in this* Init *instance as* $K^{(1)}$*, and denote* S_i*'s output, for* $i \in \overline{\mathsf{B}}$*, as* ω_i*. Choose* $K^{(0)}$ *at random in* $\{0,1\}^\tau$*. Give key* $K^{(b)}$ *to* $\mathcal{A}$*.*
(3) Let $\mathcal{A}$ *interact with* q_u *instances of* $\mathsf{U}^{\diamond}_{\mathsf{Rec}}(\mathsf{CRS}, \mathsf{pw}, b, K^{(0)})$ *and* q_s *instances of* $\mathsf{S}^{\diamond}_{\mathsf{Rec}}(\mathsf{CRS}, \boldsymbol{\sigma}_{\overline{\mathsf{B}}}, \boldsymbol{\pi}, \boldsymbol{\omega}_{\overline{\mathsf{B}}})$*. Let* b' *be the final output of* $\mathcal{A}$*.*

Secure Initialization. Note that in the above definition we assume that in the initalization protocol U runs the $\mathsf{U}_{\mathsf{Init}}$ procedure on input a vector of public parameters $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ where π_i for each $i \in \overline{\mathsf{B}}$ is the true output of $\mathsf{SKeyGen}$ executed by server S_i. In other words, we assume that the user runs the initialization procedure on correct (i.e. authentic) values π_i for the honest servers. This is equivalent to assuming that the user can authenticate, e.g. via the PKI, the servers with whom it wants to initialize the PPSS scheme. The requirement of authenticated channels between the user and the honest servers *during the initialiation protocol* is indeed necessesary, or otherwise the adversary would be able to pose as $t+1$ servers among $\mathsf{S}_1, \ldots, \mathsf{S}_n$ and recover U's

secret sc from the initialization protocol. (A similar assumption on authenticity of servers' public keys in the initialization is also made in [10].)

Soundness. The above definition captures also a *soundness* property of a PPSS scheme, because it implies an upper-bound on the probability that an adversary causes any $\mathsf{U}_{\mathsf{Rec}}$ instance to output $K' \notin \{K, \perp\}$ where K was an output of $\mathsf{U}_{\mathsf{Init}}$. Assume algorithm $\mathcal{A}$ which outputs 0 if every key returned by $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ oracle is either equal to $K^{(b)}$ which was output by $\mathsf{U}_{\mathsf{Init}}$, or to $\perp$. Note that in the security experiment with $b = 0$, oracle $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ always returns $K^{(0)}$ or $\perp$, so $p^{(0)}_{\mathcal{A}} = 0$. The security definition implies that $p^{(1)}_{\mathcal{A}} \leq \left(q_u + \left\lfloor \frac{q_s}{t-t'+1} \right\rfloor\right) * \frac{1}{|D|} + \epsilon$, hence this is also an upper-bound on the probability that any $\mathsf{U}_{\mathsf{Rec}}$ instance outputs K' which is neither $\perp$ nor K output in $\mathsf{U}_{\mathsf{Init}}$.

Robustness. Another desirable property of a PPSS scheme is robustness, which requires that a user reconstructs the key created in Init as long as (1) it has unobstructed communication channels with at least $t + 1$ non-corrupt servers (both in the Init and the Rec protocols), and (2) the upper-bound on the number of corrupt servers, t, satisfies the constraint that $2t + 1 \leq n$. (These two constraints ensure that the majority of servers U accesses without obstruction in both the Init and the Rec protocols is uncorrupted.) This property is distinct from soundness in that it assumes that the adversary lets the user communicate with $t + 1$ non-corrupt servers without interference. On the other hand, it requires not only that the user does not reconstruct an incorrect key, i.e. that $K' \in \{K, \perp\}$, but that it does output the correct one, i.e. that $K' = K$. We call a (t, n)-threshold PPSS scheme *robust* if, except for negligible probability, $\mathsf{U}_{\mathsf{Rec}}$ outputs the key K generated by $\mathsf{U}_{\mathsf{Init}}$ as long as the set B of corrupted servers *and* corrupted communication links to honest servers which U contacts in the $\mathsf{U}_{\mathsf{Rec}}$ protocol, is less than $t+1$, while the number of uncorrupted servers whom U contacts over uncorrupted links is at least $t+1$. Note that this implies that $|\mathsf{B}| < \min(t+1, n-t)$, which means that $|\mathsf{B}|$ can be equal to t only if $t < n/2$, a restriction which is not imposed by either the security or the soundness properties above. We stress that when considering robustness we need to count in set B both corrupt servers and corrupted links to honest servers. This is in contrast to the definition of security (and soundness) above, where B stands only for corrupted servers, and the adversary is assumed to play a man-in-the-middle on all communications.

5 A PPSS protocol in the $\mathcal{F}_{\text{VOPRF}}$-hybrid world

We show a PPSS protocol based on any realization of the $\mathcal{F}_{\text{VOPRF}}$ functionality. The protocol is shown in Figure 8 in the $\mathcal{F}_{\text{VOPRF}}$-hybrid model (a specific instantiation based on the 2HashDH-NIZK V-OPRF of Section 3.1 is presented in Appendix B). The protocol is secure in the CRS model and it assumes a pseudorandom generator and a computationally hiding, computationally binding, and non-malleable (with respect to decommitment) commitment scheme, which can be realized e.g. by a CCA-secure public key encryption, or by a hashing the message together with a random nonce in ROM. The simplified version of the protocol in Figure 8 works as follows: In SKeyGen, each server S_i picks its public parameter π_i as the V-OPRF function descriptor, which in all our V-OPRF instantiations is a commitment to the private key of the underlying PRF (see Section 2). In protocol Init, on U's inputs a password pw and a vector of function descriptors $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$, which are authentically delivered to the user, user U picks a random key K, secret-shares it into shares $s_1, \ldots, s_n$, and then encrypts each s_i using one-time pad encryption under key $\rho_i = F_{\pi_i}(\mathsf{pw})$, computed in a V-OPRF instance with server S_i. The vector of function descriptors $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ and the ciphertexts $\mathbf{c} = (c_1, \ldots, c_n)$, where $c_i = s_i \oplus F_{\pi_i}(\mathsf{pw})$, is public, and given to each server. At reconstruction the servers send these two vectors to U, who can recover $t + 1$ shares s_i, and

interpolate them to recover K, after $t+1$ V-OPRF instances in which U recomputes the values $\rho_i = F_{\pi_i}(\mathsf{pw})$ for $t+1$ different i's.

As we explain in the introduction, if the attacker learns whether the receiver recovers the shared key K correctly, the above protocol would enable a malicious server S_i to get information about user's password pw (including recovering it completely using binary search in an OPRF based on the Naor-Reingold PRF), by manipulating the function descriptor π_i in each OPRF instance executed by S_i in this reconstruction protocol. In fact, in the PPSS security model defined in section 4, the above simplified protocol allows a malicious server to recover π through an off-line dictionary search after a single instance of PPSS reconstruction. Note that our PPSS security model reveals the whole key K output in a PPSS reconstruction to the adversary, which models putting this key to an arbitrary usage by the higher-level protocol, e.g. by the T-PAKE scheme built from PPSS in Section 6. Now, if $\mathcal{A}$ sends to $\mathsf{U}_{\mathsf{Rec}}$ a vector of function descriptors π_i^* which correspond to PRF keys k_i^* which $\mathcal{A}$ creates, and if $\mathcal{A}$ learns the key K output by this $\mathsf{U}_{\mathsf{Rec}}$ instance, then $\mathcal{A}$ can stage an off-line dictionary attack running the user's reconstruction algorithm for every guess $\tilde{\mathsf{pw}}$ in the password dictionary D, and locally computing values $F_{\pi_i^*}(\tilde{\mathsf{pw}})$ using the PRF keys k_i^*. This is yet another reason why we need to extend the above protocol by adding a non-malleable commitment C that binds user's password pw to the reconstructed secret K. We accomplish this binding as follows: The CRS string will include an instance of a non-malleable commitment scheme COM. In the initialization procedure, the user secret-shares not the key K directly, but a random value s, and then it uses s as a PRG seed to derive the key K together with the commitment randomness r, and sets each state ω_i given to S_i to $(\boldsymbol{\pi}, \mathbf{c}, C)$ where $C = \mathsf{COM}((\mathsf{pw}, \boldsymbol{\pi}, \mathbf{c}); r)$. By the binding property of commitment COM, the adversary playing the role of the servers must commit to a password guess $\tilde{\mathsf{pw}}$ in value C it sends to the user, and the reconstruction procedure rejects unless the guess was right, i.e. unless $\tilde{\mathsf{pw}} = \mathsf{pw}$, disabling the off-line dictionary attack above. We need the non-malleability of the commitment scheme to forestall the possibility that the adversary modifies either the vector of function descriptors $\boldsymbol{\pi}$ or the ciphertexts $\mathbf{c}$, and hence in particular modifies the reconstructed key K, without guessing the password.

Communication Complexity. In Figure 8 we show a PPSS scheme whose communication complexity is $O(n^2 \cdot \mathrm{poly}(\tau))$ where τ is a security parameter, because the protocol starts with each server S_i sending to U a tuple ω which contains n function descriptors π_i and n field elements c_i. The reason we do this is simplicity, plus we suspect that in most applications the number of servers n will be small enough that the $O(n^2)$ cost of this communication will not be significant in practice. However, for large n we can reduce the communication to $O(n \log n)$ using a Merkle Tree hash [38]. Each server S_i would then send only its own π_i, c_i values together with the co-path in the hash tree which allows U to agree on the set of $t+1$ servers whose tree co-paths hash to the same root value. In practice U could also cash the ω vector as it does not change between Rec protocol instances, in which case the communication cost becomes $O(n)$.

Security Discussion. Before formally analyzing the security of the PPSS protocol in Figure 8, we summarize the capabilities of the adversary defined by the security game, and see what these capabilities are for the case of the above PPSS scheme: **(1)** In the server initialization stage, the adversary registers the corrupt servers' parameters π_i and the corresponding Turing machines M_{π_i}, for $i \in \mathsf{B}$, with the $\mathcal{F}_{\mathrm{VOPRF}}$ functionality. This effectively sets values $\rho_i = F_{\pi_i}(\mathsf{pw})$ which U computes via V-OPRF evaluation during the initialization protocol as $\rho_i = M_{\pi_i}(\mathsf{pw})$ for $i \in \mathsf{B}$, while for $i \in \overline{\mathsf{B}}$, ρ_i's are sampled by $\mathcal{F}_{\mathrm{VOPRF}}$ as random τ-bit strings. After completing all the V-OPRF instances, the adversary then sees vector $\omega = (\boldsymbol{\pi}, \mathbf{c}, C)$ which U sends to every S_i. **(2)** The adversary can then interact with q_u user instances $\mathsf{U}(\mathsf{CRS}, \mathsf{pw})$. In each such execution, which we equate with a single execution of Step 2 in the U protocol in Figure 8, the user settles on some

Parameters: Security parameters τ and ℓ, binary extension field $\mathbb{F} = GF(2^\ell)$, session ID $sid = (\mathsf{S}_1, \ldots, \mathsf{S}_n)$, threshold parameters $t, n \in \mathbb{N}$.

$\underline{\mathsf{ParGen}(\tau)}$: Sets CRS as an instance of a non-malleable commitment COM.

$\underline{\mathsf{SKeyGen}(\mathsf{CRS})}$: S_i sends $(\textsc{KeyGen}, sid)$ to $\mathcal{F}_{\text{VOPRF}}$ and sets π_i to π it receives in the response $(\textsc{Parameter}, sid, \pi)$ from $\mathcal{F}_{\text{VOPRF}}$. The private state σ_i of S_i is the unique handle "S_i" has to the V-OPRF function F_{π_i} implemented by the ideal $\mathcal{F}_{\text{VOPRF}}$ functionality. (In all our V-OPRF instantiations σ_i is a PRF key and the function descriptor π_i is a commitment to it.)

$\underline{\mathsf{U}_{\mathsf{Init}}(\mathsf{CRS}, \mathsf{pw}, \pi_1, \ldots, \pi_n) \rightleftharpoons \{\mathsf{S}_{\mathsf{Init}}(\mathsf{CRS}, \sigma_i, \pi_i)\}_{i=1}^n}$:
Step 1. User U picks $s \leftarrow \mathbb{F}$ and generates $(s_1, \ldots, s_n)$ as a (t, n) Shamir's secret-sharing of s over field $\mathbb{F}$. (Indices $0, 1, \ldots, n$ used in Shamir's secret-sharing are encoded as some distinct field elements $\langle 0 \rangle_{\mathbb{F}}, \langle 1 \rangle_{\mathbb{F}}, \ldots, \langle n \rangle_{\mathbb{F}}$.) For $i = 1$ to n, U sends $(\textsc{Eval}, sid, \mathsf{S}_i, \mathsf{pw})$ to $\mathcal{F}_{\text{VOPRF}}$.
Step 2. User U collects $\mathcal{F}_{\text{VOPRF}}$ responses $(\pi'_1, \rho_1), \ldots, (\pi'_n, \rho_n)$, and aborts if $\pi'_i \neq \pi_i$ for any i. If all parameters π'_i match those in the inputs, U computes $c_i \leftarrow s_i \oplus \rho_i$ for $i = 1$ to n, $\mathbf{c} \leftarrow (c_1, \ldots, c_n)$, $\boldsymbol{\pi} \leftarrow (\pi_1, \ldots, \pi_n)$, $[r||K] \leftarrow G(s)$, $C \leftarrow \mathsf{COM}((\mathsf{pw}, \boldsymbol{\pi}, \mathbf{c}); r)$, sends $\omega = (\boldsymbol{\pi}, \mathbf{c}, C)$ to each S_i, and outputs K as a local output.

$\underline{\mathsf{U}_{\mathsf{Rec}}(\mathsf{CRS}, \mathsf{pw}) \rightleftharpoons \{\mathsf{S}_{\mathsf{Rec}}(\mathsf{CRS}, \sigma_i, \pi_i, \omega_i)\}_{i \in S}}$:
For each $i = 1, \ldots, n$, user U sends $(\textsc{Eval}, sid, \mathsf{S}_i, \mathsf{pw})$ to $\mathcal{F}_{\text{VOPRF}}$ and initiates a run of the protocol Rec with S_i.
Each S_i responds by sending ω_i to U and $(\textsc{SenderComplete}, sid, \mathsf{S}_i)$ to $\mathcal{F}_{\text{VOPRF}}$, and U collects $\mathcal{F}_{\text{VOPRF}}$ responses (π'_i, ρ_i) and ω_i for each $i \in S$.
Let S be a subset of servers such that: (i) $|S| = t + 1$; (ii) there exists $\omega = (\boldsymbol{\pi}, \mathbf{c}, C)$ with $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ and $\mathbf{c} = (c_1, \ldots, c_n)$ such that $\omega_i = \omega$ for all $S_i \in S$; (iii) for all $S_i \in S$, $\pi'_i = \pi_i$ and $\rho_i \neq \perp$. If no such subset exists output $\perp$ and halt.
Reconstruction: Set $u_i \leftarrow c_i \oplus \rho_i$ for all $i \in S$. Interpolate points $\{(\langle i \rangle_{\mathbb{F}}, u_i)\}_{i \in S}$ with a polynomial $U \in \mathbb{F}[x]$, and set $s \leftarrow U(\langle 0 \rangle_{\mathbb{F}})$. Compute $[r||K] \leftarrow G(s)$. If $\mathsf{COM}((\mathsf{pw}, \boldsymbol{\pi}, \mathbf{c}); r) = C$ then output K, else output $\perp$.

Figure 8: A PPSS scheme in the $\mathcal{F}_{\text{VOPRF}}$-hybrid-model.

$\omega^* = (\boldsymbol{\pi}^*, \mathbf{c}^*, C^*)$ tuple, which w.l.o.g. is decided by the adversary. When the user issues the Eval requests to the $\mathcal{F}_{\text{VOPRF}}$, it will except the returned π_i's to match those specified in $\boldsymbol{\pi}^*$. Since we do not assume authenticated channels, the adversary can choose $\boldsymbol{\pi}^*$ vector at will. Some π_i^*'s in $\boldsymbol{\pi}^*$ can match π_i's of uncorrupted servers, others can be chosen by the adversary, e.g. they can correspond to the keys of the corrupted servers (but there is no limit on how many such keys the adversary effectively uses). By the rules of the $\mathcal{F}_{\text{VOPRF}}$ functionality, whenever $\pi_i^* = \pi_i$ for some uncorrupted server S_i, the adversary can only choose whether or not to allow the Eval request to output the correct PRF value $\rho_i = F_{\pi_i}(\mathsf{pw})$, or $\perp$. For π_i^*'s chosen by the adversary, the adversary decides on function $F_{\pi_i^*}$ in the form of a Turing Machine $M_{\pi_i^*}$ which will be evaluated on pw by $\mathcal{F}_{\text{VOPRF}}$. By the rules of $\mathcal{F}_{\text{VOPRF}}$, every function descriptor π has a fixed Turing machine M_π corresponding to it, but the adversary can use different π_i^*'s in different $\mathsf{U}(\mathsf{CRS}, \mathsf{pw})$ calls, including the π_i's it chose for the corrupted servers, $i \in \mathsf{B}$, during the server initialization stage. **(3)** The adversary can also interact with q_s instances of uncorrupted servers. Each server S_i will provide the string ω and engage in the server-side V-OPRF protocol, which enables the adversary to evaluate function F_{π_i} on an input of his choice.

Intuitively, if we show that the adversary cannot learn anything useful from interacting with

the user, then in the $\mathcal{F}_{\text{VOPRF}}$-hybrid world values $c_1, \ldots, c_n$ hide the secret-shared "master secret" value s as long as the adversary does not engage $t-t'+1$ distinct uncorrupted servers in a V-OPRF protocol on input the correctly guessed user's password pw. This is because, due to the properties of $\mathcal{F}_{\text{VOPRF}}$, the adversary learns only irrelevant information in every V-OPRF instance to which he inputs $x \neq \mathsf{pw}$. Consequently, if the adversary does not make $t-t'+1$ server queries on the correct pw, the only information related to pw and s in the adversary's view is the commitment C. However, assuming the pseudorandomness of G, which is used to derive r from s, and assuming computational hiding of the commitment scheme, this commitment offers no help, except for negligible probability in deriving any additional information on pw and/or s or K. Therefore, for pw chosen at random the probability that there are $t-t'+1$ server queries on pw among q_s queries the adversary makes, is at most $\lfloor q_s/(t-t'+1) \rfloor * (1/|D|)$ plus a negligible factor. As for the interactions with the user, the only way such interaction might return anything but $\perp$ is when the adversary sends to the user a tuple $\omega^* = (\boldsymbol{\pi}^*, \mathbf{c}^*, C^*)$ s.t. C^* is a commitment to the correct password pw. If $C^* = C$ then the adversary is also bound to the V-OPRF parameters $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ and ciphertexts $\mathbf{c} = (c_1, \ldots, c_n)$ which were committed in C in the initialization protocol. By the consistency of the $\mathcal{F}_{\text{VOPRF}}$ functionality, if the user runs a V-OPRF protocol with server S_i on input pw, and it rejects any response (π_i', ρ_i) s.t. $\pi_i' \neq \pi_i$, then the only two possible values for ρ_i are $\perp$ and $c_i \oplus s_i$ where s_i is the secret-sharing share U picked in the Init protocol. Therefore the only secret the user can reconstruct in this case, without breaking the binding property of the commitment scheme, is the value s created in the initialization, which leads the user to reconstruct the key K created in the initialization. On the other hand, by the non-malleability of the commitment scheme, the probability that the adversary creates a new commitment $C^* \neq C$ which commits to the same value pw in any of the q_u user sessions, is at most $q_u * (1/|D|)$ plus a negligible factor.

5.1 Security proof for the PPSS construction.

We present the formal argument for the security of the PPSS protocol from Figure 8 that formalizes the intuition given above. In particular, the two key events defined below, $\mathcal{E}_S$ and $\mathcal{E}_U$, correspond to the two succesful attack conditions discussed earlier: that the adversary queries $t-t'+1$ uncorrupted servers on the correct password pw and that the adversary initializes a user instance on ω^* containing $C^* \neq C$ which is a committment to pw.

Theorem 4 (PPSS Security) *Assuming commitment scheme* COM *is computationally hiding, computationally binding, and non-malleable (with respect to decommitment), and that G is a pseudorandom generator, the PPSS scheme in Figure 8 is (T, q_u, q_s, ϵ)-secure for $\epsilon = \epsilon_H + \epsilon_B + q_u \cdot \epsilon_{NM} + 4\epsilon_G$, where ϵ_H, ϵ_B, ϵ_{NM} and ϵ_G are the bounds implied by, respectively, computational hiding of* COM*, copmutational binding of* COM*, non-malleability of* COM *with respect to decommitment, and the pseudorandomness of G, on input sizes implied by the usage of* COM *and G in the PPSS scheme, for adversaries whose time is bounded by T plus the time taken by a single instance of* Init*, q_u instances of* $\mathsf{U}_{\mathsf{Rec}}$*, and q_s instances of* $\mathsf{S}_{\mathsf{Rec}}$*.*

Proof: Take any n, t s.t. $t < n$, and let $\mathcal{A}$ be an adversarial algorithm in the PPSS security experiment, whose running time is bounded by T, and who corrupts a set B of $t' \leq t$ servers. Let T' be T plus the time taken by a single instance of Init, q_u instances of $\mathsf{U}_{\mathsf{Rec}}$ and q_s instances of $\mathsf{S}_{\mathsf{Rec}}$. Recall that the security experiment starts by choosing pw at random in D, where D is an arbitrary subset of τ-bit strings, choosing b as a random bit, $K^{(0)}$ as a random τ-bit string, and generating CRS, which in the case of our PPSS protocol consists of a description of the extension field $\mathbb{F} = GF(2^\ell)$, session parameter $sid = (\mathsf{S}_1, \ldots, \mathsf{S}_n)$, and integers t, n.

Server Initialization. Adversary $\mathcal{A}$ first participates in the servers' initialization step. Namely, $\mathcal{A}$ receives $(\text{KEYGEN}, sid, \mathsf{S}_i)$ from $\mathcal{F}_{\text{VOPRF}}$ for every $i \notin \mathsf{B}$ and sets π_i via command $(\text{PARAMETER}, sid, \mathsf{S}_i, \pi_i)$ to $\mathcal{F}_{\text{VOPRF}}$. Note that parameters π_i for $i \notin \mathsf{B}$ serve as tags which identify functions F_{π_i}, but that each F_{π_i} for $i \in \overline{\mathsf{B}}$ is a random function: Its values are randomly sampled by $\mathcal{F}_{\text{VOPRF}}$ in response to $\mathcal{A}$'s $(\text{USERCOMPLETE}, sid, \mathsf{U}, \pi_i, \top)$ calls In addition, for every $i \in \mathsf{B}$, $\mathcal{A}$ registers the parameters of the corrupted server S_i, by issuing the $(\text{PARAMETER}, sid, \mathsf{S}_i, \pi_i, M_{\pi_i})$ command, thus registering some Turing Machine M_{π_i} corresponding to S_i's parameter π_i. Let $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ be the vector of parameters defined in this step.

Initialization Protocol. Without loss of generality we can consider $\mathcal{A}$'s which do not cause the instance of $\mathsf{U}_{\mathsf{Init}}(\mathsf{CRS}, \mathsf{pw}, \boldsymbol{\pi})$ executing in the Init protocol to abort (as in that case $\mathcal{A}$ gets no information about b). By the rules of $\mathcal{F}_{\text{VOPRF}}$ functionality, the only way U can not abort is if $\mathcal{A}$ lets U evaluate the V-OPRF on pw for all n parameters π_i in $\boldsymbol{\pi}$. In other words, U's ρ_i values will be set as $\rho_i \leftarrow M_{\pi_i}(\mathsf{pw})$ for each $i \in \mathsf{B}$ and for $i \in \overline{\mathsf{B}}$ values $\rho_i = F_{\pi_i}(\mathsf{pw})$ will be sampled as random τ-bit strings by $\mathcal{F}_{\text{VOPRF}}$. The initialization defines $\omega = (\boldsymbol{\pi}, \mathbf{c}, C)$ where $\mathbf{c} = (c_1, \ldots, c_n)$ for $c_i = \rho_i \oplus s_i$ for all i, and $C = \mathsf{COM}((\boldsymbol{\pi}, \mathbf{c}, \mathsf{pw}); r)$ for $[r||K] = G(s)$, and $\mathcal{A}$ gets ω and K if $b = 1$ or $K^{(0)}$ if $b = 0$.

User and Server Queries. $\mathcal{A}$ then interacts with q_u instances of user oracle $\mathsf{U}^{\diamond}_{\mathsf{Rec}}(\mathsf{CRS}, \mathsf{pw}, b, K^{(0)})$ and q_s instances of server oracle $\mathsf{S}^{\diamond}_{\mathsf{Rec}}(\mathsf{CRS}, \boldsymbol{\sigma}_{\overline{\mathsf{B}}}, \boldsymbol{\pi}, \omega)$, and decides its output bit b'. Recall that in each instance of $\mathsf{U}_{\mathsf{Rec}}(\mathsf{CRS}, \mathsf{pw})$ triggered by $\mathsf{U}^{\diamond}_{\mathsf{Rec}}(\mathsf{CRS}, \mathsf{pw}, b, K^{(0)})$, the adversary effectively decides on $\omega^* = (\boldsymbol{\pi}^*, \mathbf{c}^*, C^*)$ and a set of server indexes S which $\mathsf{U}_{\mathsf{Rec}}$ will use in the reconstruction protocol. Let $\boldsymbol{\pi}^* = (\pi_1^*, \ldots, \pi_n^*)$ and $\mathbf{c}^* = (c_1^*, \ldots, c_n^*)$. Recall that by the $\mathcal{F}_{\text{VOPRF}}$ rules, the only thing that $\mathcal{A}$ can decide with respect to each V-OPRF instance which U will execute, is whether it results in U getting the value $\rho_i = F_{\pi_i^*}(\mathsf{pw})$ (if $\mathcal{A}$ lets this V-OPRF instance proceed on π_i^*) or that U will strike out this index i from S (if $\mathcal{A}$ blocks this instance or makes it execute on some function descriptor different than π_i^*). Since $\mathcal{A}$ learns no further information during this process beyond its view when it decided on ω^* and S, we can assume that $\mathcal{A}$ straight away decides on the final make-up of S for which U reconstructs $\rho_i = F_{\pi_i^*}(\mathsf{pw})$ values, and that $|S| = t + 1$ since otherwise U aborts.

Let $K^{(1)}$ denote K output by U in the Init instance. Let G_0 denote the game defined by the security experiment in the PPSS security definition. Let $\mathsf{UncParams}$ denote the set of parameters of the uncorrupted servers, $\{\pi_i\}_{i \in \overline{\mathsf{B}}}$.

Define event $\mathcal{E}_S$ as $\mathcal{A}$ making a $(\text{USERCOMPLETE}, sid, U^*, \pi_i, \top)$ query where $(\text{EVAL}, sid, S^*, \tilde{\mathsf{pw}})$ was previously submitted by corrupted user U^* and $\tilde{\mathsf{pw}} = \mathsf{pw}$ for $t - t' + 1$ distinct parameters $\pi_i \in \mathsf{UncParams}$. Note that if $\mathcal{E}_S$ happens the adversary can learn that its guess $\tilde{\mathsf{pw}}$ for pw was correct, and thus recover K: Recall that $\mathcal{A}$ knows $t' = |\mathsf{B}|$ turing machines M_{π_i} which define functions F_{π_i}, so $\mathcal{A}$ can locally compute $F_{\pi_i}(\tilde{\mathsf{pw}})$ for all $\tilde{\mathsf{pw}} \in D$ (assuming w.l.o.g. that D is small enough that this can be done within time T). For each such $\tilde{\mathsf{pw}}$, if $\mathcal{A}$ additionally learn $t - t' + 1$ values $f_{\pi_i}(\tilde{\mathsf{pw}})$ for $t - t' + 1$ different parameters $\pi_i \in \mathsf{UncParams}$, then $\mathcal{A}$ can xor these values with corresponding c_i's in $\mathbf{c}$, interpolate the resulting points with a polynomial $\tilde{U}$, compute $\tilde{s} \leftarrow \tilde{U}(0)$, derive $[\tilde{r}||\tilde{K}] \leftarrow G(\tilde{s})$, and confirm if $\tilde{\mathsf{pw}} = \mathsf{pw}$ and $\tilde{K} = K^{(1)}$ by checking if $C = \mathsf{COM}((\boldsymbol{\pi}, \mathbf{c}, \tilde{\mathsf{pw}}); \tilde{r})$.

Define event $\mathcal{E}_U$ that some instance of $\mathsf{U}_{\mathsf{Rec}}(\mathsf{CRS}, \mathsf{pw})$ triggered by $\mathcal{A}$'s call to the $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ oracle with input $\omega^* = (\boldsymbol{\pi}^*, \mathbf{c}^*, C^*)$ s.t. (1) $\omega* \neq \omega$, (2) $\mathcal{A}$ allows some set S of $t + 1$ V-OPRF instances triggered by this $\mathsf{U}_{\mathsf{Rec}}$ instance to proceed correctly on parameters π_i^* in $\boldsymbol{\pi}^*$, and (3) $\mathsf{U}_{\mathsf{Rec}}$ outputs $K \neq \perp$ in this instance. Note that this happens if and only if $C^* = \mathsf{COM}((\boldsymbol{\pi}^*, \mathbf{c}^*, \mathsf{pw}); r)$ for r computed by $\mathsf{U}_{\mathsf{Rec}}$'s procedure on inputs pw, $\boldsymbol{\pi}^*, \mathbf{c}^*$ and values $\rho_i = F_{\pi_i^*}(\mathsf{pw})$ for $i \in S$ computed via the V-OPRF instances. Note that in our PPSS protocol U chooses the $(t+1)$-element set S arbitrarily among the servers who respond correctly in their V-OPRF instances, but w.l.o.g. we can assume that S is chosen by $\mathcal{A}$ together with ω^*. The reason this event is crucial is that if $\mathcal{E}_U$ happens

then the adversary who created C^* with the knowledge of $(\tilde{\mathsf{pw}}, r)$ s.t. $C^* = \mathsf{COM}((\boldsymbol{\pi}^*, \mathbf{c}^*, \tilde{\mathsf{pw}}); r)$, can conclude that $\tilde{\mathsf{pw}} = \mathsf{pw}$ (or there is a break in commitment binding), in which case $\mathcal{A}$ could turn around and use pw in $t - t' + 1$ V-OPRF sessions with uncorrupted servers, which allows $\mathcal{A}$ to learn $K^{(1)}$ as described above. We will break even $\mathcal{E}_U$ into mutually exclusive components, $\mathcal{E}_{U,C}$ and $\mathcal{E}_{U,NC}$, where $\mathcal{E}_{U,C}$ is defined as the event that $\mathcal{E}_U$ happens for C^* in ω^* s.t. $C^* = C$, while $\mathcal{E}_{U,NC}$ is the case of cE_U for $C^* \neq C$. Note that if $\omega^* \neq \omega$ and $C^* = C$ then $(\boldsymbol{\pi}^*, \mathbf{c}^*) \neq (\boldsymbol{\pi}, \mathbf{c})$ therefore event $\mathcal{E}_{U,C}$ corresponds to a break in commitment binding, whereas event $\mathcal{E}_{U,NC}$ will correspond to a break in commitment non-malleability.

Consider a modification G_1 of game G_0, in which the $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ oracle returns $K^{(b)}$ whenever an instance of $\mathsf{U}_{\mathsf{Rec}}$ is started on $\omega^* = \omega$ and there exists a set S of $t + 1$ servers for which $\mathcal{A}$ allows the V-OPRF instances to proceed on π_i's in ω, and otherwise the $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ oracle returns $\perp$. Note that under condition $\neg\mathcal{E}_U$ game G_1 is identical to G_0.

Consider a modification G_2 of game G_1 in which the $\mathsf{U}_{\mathsf{Init}}$ algorithm creates $\vec{s}$ as a (t, n) Shamir's secret-sharing of 0 in $\mathbb{F}$ instead of s. However, s is still used to generate $[r||K] \leftarrow G(s)$. Note that under condition $\neg\mathcal{E}_S$, game G_2 is identical to G_1: Assuming $\neg\mathcal{E}_S$, adversary's view includes information about at most t shares in $\vec{s} = (s_1, \ldots, s_n)$ defined in $\mathsf{U}_{\mathsf{Init}}$, even if $\mathcal{A}$ knew (or guessed) pw. Note that given pw, $\mathcal{A}$ can compute $\rho_i = F_{\pi_i}(\mathsf{pw})$ for the t' indexes $i \in \mathsf{B}$, and under $\neg\mathcal{E}_S$ it can compute up to $t - t'$ values $\rho_i = F_{\pi_i}(\mathsf{pw})$ for $i \in \overline{\mathsf{B}}$, but this is the only information on $\vec{s}$ it sees. By $\mathcal{F}_{\text{VOPRF}}$ rules, values of $F_{\pi_i}(x)$ for $x \neq \mathsf{pw}$ and $\pi_i \in \mathsf{UncParams}$ are independent from values $F_{\pi_i}(\mathsf{pw})$, and in game G_1 the user oracle $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ does not depend on the information gained from the V-OPRF evaluations on pw: It replies $\perp$ or $K^{(b)}$ based solely on whether $\omega^* = \omega$ and whether $\mathcal{A}$ allowed $|S| = t + 1$ V-OPRF instances to go through correctly, so $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ oracle execution can be perfectly simulated without information on $F_{\pi_i}(\mathsf{pw})$'s. Let $\mathcal{E} = \mathcal{E}_S \cup \mathcal{E}_U$. It follows that under condition $\neg\mathcal{E}$, games G_2 and G_0 are identical. Therefore $\Pr[\mathcal{E} \text{ in } G_0] = \Pr[\mathcal{E} \text{ in } G_2]$ and $\Pr[\neg\mathcal{E} \text{ in } G_0] = \Pr[\neg\mathcal{E} \text{ in } G_2]$. Moreover, since G_0 and G_1 are identical under $\neg\mathcal{E}$ we also get that $\Pr[b' = b \mid \neg\mathcal{E} \text{ in } G_0] = \Pr[b' = b \mid \neg\mathcal{E} \text{ in } G_2]$. Combining the last two equations implies that

$$\Pr[b' = b \wedge \neg\mathcal{E} \text{ in } G_0] = \Pr[b' = b \wedge \neg\mathcal{E} \text{ in } G_2] \tag{2}$$

Let G_3 be a modification of G_2 in which $[r||K]$ is chosen as a random bistring rather than as $G(s)$. Consider two events, $\mathcal{E}$ and $b = b' \wedge \neg\mathcal{E}$. We claim that if G is a (T', ϵ_G)-secure PRG, i.e. no algorithm running in time T' has advantage better than ϵ_G in distinguishing $G(s)$ for random s in $\mathbb{F}$ from a random string, then

$$\Pr[\mathcal{E} \text{ in } G_2] \leq \Pr[\mathcal{E} \text{ in } G_3] + \epsilon_G \tag{3}$$

$$\Pr[b = b' \wedge \neg\mathcal{E} \text{ in } G_2] \leq \Pr[b = b' \wedge \neg\mathcal{E} \text{ in } G_3] + \epsilon_G \tag{4}$$

In both cases the reduction $\mathcal{R}$, given the PRG challenge $[r||K]$, picks pw at random in D, b as a random bit, $\vec{s}$ as a secret-sharing of 0, as in game G_2, and $K^{(0)}$ as a random string. $\mathcal{R}$ then assigns $K^{(1)} \leftarrow K$, and goes through the server initialization step and the Init protocol with $\mathcal{A}$, performing the steps of $\mathcal{F}_{\text{VOPRF}}$, which determines $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ and $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_n)$, and therefore also $\mathbf{c} = (c_1, \ldots, c_n)$. $\mathcal{R}$ then computes $C = \mathsf{COM}((\boldsymbol{\pi}, \mathbf{c}, \mathsf{pw}); r)$ and gives $\omega = (\boldsymbol{\pi}, \mathbf{c}, C)$ and $K^{(b)}$ to $\mathcal{A}$. $\mathcal{R}$ then performs the code of $\mathsf{S}^{\diamond}_{\mathsf{Rec}}$ and $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ oracles, executing the code of $\mathcal{F}_{\text{VOPRF}}$ whenever $\mathcal{F}_{\text{VOPRF}}$ is called, consulting $\mathcal{A}$ whenever needed according to the rules of $\mathcal{F}_{\text{VOPRF}}$. Importantly, $\mathcal{R}$ does not immediately answer the $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ calls as in game G_1, i.e. based only on whether $\omega^* = \omega$ and $\mathcal{A}$ allows $t + 1$ V-OPRF calls to proceed on the parameters in $\boldsymbol{\pi}$. Instead, if $\omega^* \neq \omega$ and yet $\mathcal{A}$ allows $t + 1$ V-OPRF calls to proceed on the parameters in $\boldsymbol{\pi}^*$ in ω^*, $\mathcal{R}$ follows the $\mathsf{U}_{\mathsf{Rec}}$ algorithm and tests whether event $\mathcal{E}_U$ occurs. If it does, $\mathcal{R}$ outputs 1 and halts. Otherwise, it outputs $\perp$. Note that this way $\mathcal{R}$ can test for $\mathcal{E}_U$, but until it occurs $\mathcal{R}$ proceeds as G_2 does. $\mathcal{R}$ can also

output 1 whenever $\mathcal{E}_S$ occurs, which it can test for because $\mathcal{R}$ knows pw and it monitors $\mathcal{A}$'s calls to $\mathcal{F}_{\text{VOPRF}}$. Finally, $\mathcal{R}$ can also test whether $b' = b$ when $\mathcal{A}$ halts. Therefore $\mathcal{R}$ can test both for event $\mathcal{E}$ and for event $b = b' \wedge \neg\mathcal{E}$, implying both inequalities above.

Note that in game G_3, the adversary gets no information about bit b because both $K^{(0)}$ and $K^{(1)}$ are random strings independent of everything else. Therefore the probability that $b' = b$ is independent of $\mathcal{A}$'s view, and therefore it is independent of any event, e.g. $\neg\mathcal{E}$, which is a function of $\mathcal{A}$'s view. It follows that

$$\Pr[b' = b \wedge \neg\mathcal{E} \text{ in } G_3] = \Pr[b' = b \text{ in } G_3] \cdot \Pr[\neg\mathcal{E} \text{ in } G_3] = 1/2 \cdot \Pr[\neg\mathcal{E} \text{ in } G_3] \tag{5}$$

Consider game G_4 in which the $\mathsf{U}_{\mathsf{Init}}$ procedure skips generating s_i's, ignores ρ_i values it receives from $\mathcal{F}_{\text{VOPRF}}$, and picks values $c_1, \ldots, c_n$ directly as n random τ-bit strings. Under condition $\neg\mathcal{E}_S$, game G_4 is identical to game G_3 because unless $\mathcal{E}_S$ occurs, game G_3 reveals to $\mathcal{A}$ information about at most t values $\rho_i = F_{\pi_i}(\mathsf{pw})$ which mask the secret-shares s_i.

We will argue that if the commitment scheme COM is (T', ϵ_B)-binding then the following inequality holds:

$$\Pr[\mathcal{E}_{U,C} \wedge \neg\mathcal{E}_S \text{ in } G_4] \leq \epsilon_B \tag{6}$$

The reduction $\mathcal{R}$ from commitment binding is straightforward: Reduction $\mathcal{R}$, on input CRS, follows the server initialization procedure, interacting with $\mathcal{A}$ and performing the code of $\mathcal{F}_{\text{VOPRF}}$, to choose elements in $\boldsymbol{\pi}$, then it picks random pw in D, random r, and random elements in $\mathbf{c}$, as is done in G_4, computes $C \leftarrow \mathsf{COM}((\mathsf{pw}, \boldsymbol{\pi}, \mathbf{c}); r)$ and sends it to $\mathcal{A}$ and to all S_i's for $i \in \overline{\mathsf{B}}$. When $\mathcal{A}$ makes queries to $\mathsf{S}^{\diamond}_{\mathsf{Rec}}$ and $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ oracles, these are serviced by $\mathcal{R}$. In particular, $\mathcal{R}$ can test if event $\mathcal{E}_S$ happens. Also, if $\mathcal{A}$ triggers a $\mathsf{U}_{\mathsf{Rec}}$ instance, $\mathcal{R}$ performs the code of U to verify if event $\mathcal{E}_{U,C}$ happens. If it does, $\mathcal{R}$ can output two different decommitments of C.

Secondly, we will argue that if COM is (T', ϵ_{NM})-non-malleable (with respect to decommitment) [20] then:

$$\Pr[\mathcal{E}_{U,NC} \wedge \neg\mathcal{E}_S \text{ in } G_4] \leq q_u/|D| + q_u \cdot \epsilon_{NM} \tag{7}$$

The non-malleability assumption implies that the probability of the following event is at most $1/|D| + \epsilon_{NM}$ for any algorithm $\mathcal{R}$ running within time T': On input a randomly generated CRS and a commitment $C = \mathsf{COM}((z, \mathsf{pw}); r)$, where pw is chosen at random in D, r is random, and z is generated (by some algorithm) independently from pw, r, $\mathcal{R}$ outputs (z', r', C^*) s.t. $C^* \neq C$ and $C^* = \mathsf{Com}((z', \mathsf{pw}); r')$. Let $\mathcal{E}^i_{U,NC}$ stands for even $\mathcal{E}_{U,NC}$ occuring in the i-th instance of $\mathsf{U}_{\mathsf{Rec}}$ which $\mathcal{A}$ triggers. We will argue that for any i, the probability of $\mathcal{E}^i_{U,NC} \wedge \neq \mathcal{E}_S$ is upper-bounded by $1/|D| + \epsilon_{NM}$. Since $\mathcal{E}_{U,NC}$ is a union of q_u events $\mathcal{E}^i_{U,NC}$'s, inequality 7 fill follow. The reduction $\mathcal{R}$, on randomly generated CRS, follows the server initialization procedures which determines elements in $\boldsymbol{\pi}$, then it chooses elements in $\mathbf{c}$ at random and sends $\boldsymbol{\pi}, \mathbf{c}$ to the COM non-malleability challenger, which chooses pw at random in D and a random r and returns C computed as $C \leftarrow \mathsf{COM}((\boldsymbol{\pi}, \mathbf{c}, \mathsf{pw}); r)$. Given C, $\mathcal{R}$ can simulate the rest of game G_4 to $\mathcal{A}$ (assuming $\neq \mathcal{E}_S$), until the i-th query $\mathcal{A}$ makes to $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ oracle, i.e. the i-th instance of $\mathsf{U}_{\mathsf{Rec}}$. When that instance is triggered, $\mathcal{A}$ triggers it on some input $\omega^* = (\boldsymbol{\pi}^*, \mathbf{c}^*, C^*)$. If $C^* = C$, $\mathcal{R}$ aborts. Otherwise $\mathcal{R}$ asks the non-malleability challenger to provide a decommitment to C, i.e. $\mathcal{R}$ receives (pw, r). Having received pw from the non-malleability challenger, $\mathcal{R}$ can run $\mathsf{U}_{\mathsf{Rec}}$ protocol on ω^* a(and on set S determined w.l.o.g. by $\mathcal{A}$). If $\mathcal{E}^i_{U,NC}$ happens, then $\mathcal{R}$ computes r' s.t. $C^* = \mathsf{COM}((\boldsymbol{\pi}^*, \mathbf{c}^*, \mathsf{pw}); r')$ and therefore can provide decommitment $(\boldsymbol{\pi}^*, \mathbf{c}^*, \mathsf{pw}, r')$ to the non-malleability challenger. As argued above, this implies inequality 7.

Since $\mathcal{E}_U = \mathcal{E}_{U,C} \cup \mathcal{E}_{U,NC}$, inequalities 6 and 7 together imply that $\Pr[\mathcal{E}_U \wedge \neq \mathcal{E}_S] \leq q_u/|D| + q_u \cdot \epsilon_{NM} + \epsilon_B$. Moreover, since under condition $\neg\mathcal{E}_S$ game G_4 is identical to G_3, this iimplies:

$$\Pr[\mathcal{E}_U \wedge \neg\mathcal{E}_S \text{ in } G_3] \leq q_u/|D| + q_u \cdot \epsilon_{NM} + \epsilon_B \tag{8}$$

Consider game G_5 which proceeds as G_4 except that C is a commitment to a fixed bitstring. If the commitment scheme COM is (T', ϵ_H)-hiding, an easy reduction shows that $\Pr[\mathcal{E}_S \text{ in } G_4]$ is upper-bounded by $\Pr[\mathcal{E}_S \text{ in } G_5] + \epsilon_H$. Since $\mathcal{A}$'s view in game G_5 does not depend on pw, it follows that $\Pr[\mathcal{E}_S \text{ in } G_5] \leq \lfloor q_s/(t - t' + 1) \rfloor * (1/|D|)$. Taking the two facts together we conclude that $\Pr[\mathcal{E}_S \text{ in } G_4] \leq \lfloor q_s/(t - t' + 1) \rfloor * (1/|D|) + \epsilon_H$. Finally, since under condition $\neg\mathcal{E}_S$ game G_4 is identical to G_3, we get that

$$\Pr[\mathcal{E}_S \text{ in } G_3] \leq \lfloor q_s/(t - t' + 1) \rfloor * (1/|D|) + \epsilon_H \tag{9}$$

The theorem follows because $\mathbf{Adv}^{\mathsf{ppss}}_{\mathcal{A}} = |\Pr[b' = 1 \mid b = 1] - \Pr[b' = 1 \mid b = 0]| = 2 \cdot \Pr[b' = b] - 1$, and because we can bound $\Pr[b' = b]$ in G_0 as follows:

$$\begin{aligned}
\Pr[b' = b \text{ in } G_0] &\leq \Pr[\mathcal{E} \text{ in } G_0] + \Pr[b' = b \wedge \neg\mathcal{E} \text{ in } G_0] && (10)\\
&= \Pr[\mathcal{E} \text{ in } G_2] + \Pr[b' = b \wedge \neg\mathcal{E} \text{ in } G_2] && (11)\\
&\leq \Pr[\mathcal{E} \text{ in } G_3] + \Pr[b' = b \wedge \neg\mathcal{E} \text{ in } G_3] + 2\epsilon_G && (12)\\
&= \Pr[\mathcal{E} \text{ in } G_3] + 1/2 \cdot \Pr[\neg\mathcal{E} \text{ in } G_3] + 2\epsilon_G && (13)\\
&= 1/2 + 1/2 \cdot \Pr[\mathcal{E} \text{ in } G_3] + 2\epsilon_G && (14)\\
&\leq 1/2 \left(1 + \left(q_u + \left\lfloor \frac{q_s}{t - t' + 1} \right\rfloor\right) \cdot \frac{1}{|D|} + \epsilon_H + \epsilon_B + q_u \cdot \epsilon_{NM} + 4\epsilon_G\right) && (15)
\end{aligned}$$

Where equation (11) is implied by equation (2) and by the fact that G_0 and G_2 are identical under $\neg\mathcal{E}$; inequality (12) is implied by inequalities (3) and (4); equality (13) is implied by equality (5); and inequality (15) is implied by the fact that $\mathcal{E} = \mathcal{E}_U \vee \mathcal{E}_S$ is a union of disjoint events $(\mathcal{E}_U \wedge \neg\mathcal{E}_S)$ and $\mathcal{E}_S$, whose probability in G_3 is upper-bounded by inequalities (8) and (9). □

Robustness. The PPSS scheme in Figure 8 is robust: If the number of honest players a user instance $\mathsf{U}_{\mathsf{Rec}}$ interacts with is at least $t + 1$, this instance will run on the tuple ω which was generated by $\mathsf{U}_{\mathsf{Init}}$. By the rules of the $\mathcal{F}_{\text{VOPRF}}$ functionality, the only thing a corrupt server S_i for $i \in \mathsf{B}$ can do is for U to output either $\perp$ or $F_{\pi_i}(\mathsf{pw})$. In either case, since U will reconstruct correct values $F_{\pi_i}(\mathsf{pw})$ for at least $t+1$ honest servers $i \in \mathsf{G}$, the user will reconstruct the same polynomial which was created in the secret-sharing step of $\mathsf{U}_{\mathsf{Init}}$, and hence the user will reconstruct the same $[r||K]$ string, use it verify the commitment C in ω, and output K.

Note that our robustness notion requires $\mathsf{U}_{\mathsf{Rec}}$ to output the correct key only if $\mathsf{U}_{\mathsf{Rec}}$ connects without obstructions with at least $t + 1$ honest servers and with at most t corrupted connections, namely connections where the incoming information is corrupted. However, we can extend this property without limiting the number of *corrupted connections* over which $\mathsf{U}_{\mathsf{Rec}}$ tries to connect to the servers. Note that when an instance of $\mathsf{U}_{\mathsf{Rec}}$ rejects after receiving consistent ω values from at least $t + 1$ servers, it can do so for two reasons: Either the V-OPRF protocol went through with fewer than $t+1$ servers for the parameters specified in the $\boldsymbol{\pi}$ vector in ω, or the V-OPRF protocol succeeded for at least $t + 1$ parameters in $\boldsymbol{\pi}$ but the commitment verification step failed. In the latter case U can conclude that ω was wrong, and it can run another instance of $\mathsf{U}_{\mathsf{Rec}}$ excluding the servers which supplied this wrong ω in the previous $\mathsf{U}_{\mathsf{Rec}}$ instance. In the first case, U can just re-run the $\mathsf{U}_{\mathsf{Rec}}$ with a larger set of servers. In this way the user will eventually find the set of uncorrupted $t + 1$ servers with unobstructed communication channels (if such set exists), and run a $\mathsf{U}_{\mathsf{Rec}}$ instance with a server subset in which they constitute a majority.

6 From PPSS to Single-Round T-PAKE

In this section we show that the composition of a secure PPSS with a (regular) key-exchange (KE) scheme results in a secure T-PAKE protocol. As a consequence, we can combine the PPSS scheme from Section 5, implemented with the 2Hash-DH-NIZK V-OPRF, together with a simple one-round key-exchange protocol to obtain a very efficient one-round T-PAKE protocol in the password-only CRS model (no PKI or secure channels between servers are assumed) with arbitrary threshold parameters. A full specification of this protocol is presented in Appendix B.

We start by recalling the security models of T-PAKE and Authenticated Key Exchange (with a detailed T-PAKE model presented in Appendix A). Then we define a generic composition of PPSS and KE protocols to obtain T-PAKE schemes from which the one-round protocols are derived. Finally, we prove the security of the T-PAKE protocols obtained via the above generic composition and establish the security bounds of such protocols when instantiated with our V-OPRF-based PPSS scheme.

6.1 Security Models for T-PAKE and KE

We want to show that a PPSS scheme can be composed with a KE protocol to obtain a secure T-PAKE scheme. For this we need security models for both T-PAKE and Authenticated Key Exchange (KE). In Appendix A we present a detailed model for T-PAKE security based on the work of MacKenzie et al. [37] which in turn extends the PAKE model of Bellare et al. [5] to the threshold case. Our model introduces some modifications intended at some simplifications as well as for a more careful counting of online password-guessing attempts by the attacker.

For KE security (i.e., a key exchange setting where parties have strong random secrets and there is no threshold component) we consider a model similar to the one used for T-PAKE but stripped off of the elements related to the threshold setting (e.g., the client ring and parameters (t, n)) and the password setting (e.g., adversary's advantage is required to be negligible). The model obtained after stripping off the threshold elements is essentially a close variant of [5] while further eliminating the password elements results on a variant of the [6] model (with matching sessions defined via matching session identifiers rather than by equality of transcripts). Note that while T-PAKE protocols assume a client-server model, PAKE and KE protocols do not usually assume that and neither do we in our treatment here.

Client keystores. A PPSS scheme allows a client C to retrieve a secret key from a set of servers I_C. In order to augment the PPSS scheme into a T-PAKE protocol, the client will use the retrieved secret, call it K_C, as a way to bootstrap KE sessions with its servers. Usually, however, there is more information that the client needs for completing the KE, such as servers identities, public keys, certificates, additional secret keys, etc. In this case, C will store, and later retrieve, this extra information at the same set of servers I_C, and will use the secret K_C to protect (encrypt and authenticate) this information. We refer to the information stored at the servers as a keystore and denote it by $\mathsf{keystore}_C$. When defining a KE protocol for use in conjunction with a PPSS protocol (for obtaining a T-PAKE) one needs to specify the information included in the keystore and how this information is protected using the client's secret K_C. We use $\boldsymbol{KE}^+$ to refer to a KE protocol augmented with a definition of a keystore.

6.2 A generic T-PAKE protocol from PPSS and KE

Given a PPSS scheme $\boldsymbol{P} = (\mathsf{ParGen}, \mathsf{SKeyGen}, \mathsf{Init}, \mathsf{Rec})$ with threshold parameters (t, n) and a key exchange protocol (with clients keystores) $\boldsymbol{KE}^+$, we build a T-PAKE protocol $\boldsymbol{T}$ as follows. The

protocol participants are a set of clients and a set of servers. Each client C is associated with a subset of n servers called the client's ring and denoted as $I_C = \{\mathsf{S}_1, \ldots, \mathsf{S}_n\}$ (for simplicity, we assume global parameters (t, n) but one could have different parameters for different clients).

Protocol $\boldsymbol{T}$ initialization. The following operations are assumed to be performed at initialization and communication run over secure channels (these procedures can be used to initialized new parties at any time but we assume for simplicity that they are initialized at the onset of the protocol execution). They represent operations performed once when a user register for and joins the system.

1. A global *crs* with a given security parameter κ is generated using ParGen. This *crs* is input to each procedure below (except for actions specific to the key exchange protocol $\boldsymbol{P}$).

2. For each client C, an instance of the PPSS scheme $\boldsymbol{P}$ is initialized between C and each server in I_C. Namely, each server $\mathsf{S}_i \in I_C$ runs the $\mathsf{SKeyGen}$ procedure to create its secret and public parameters, $\sigma_i(C)$ and $\pi_i(C)$, and then Init is run between C and these servers. The latter procedure uses as input the client's password pw_C and the servers' parameters, and it generates a random κ-long key K_C and per-server client information $\omega_i(C)$ stored at server S_i. We note that a server can participate in the ring of more than one client. However, we assume the parameters $\sigma_i(C), \pi_i(C)$ to be unique and independent for each client.

3. In addition to the the above initialization of $\boldsymbol{P}$ between each client and its servers' ring, all parties initialize and share any information required by the key exchange protocol $\boldsymbol{KE}$ (including parties' shared and/or public keys assumed by $\boldsymbol{KE}$ - although no trusted third party is assumed). From this information, and according to the specification of protocol $\boldsymbol{KE}^+$, the client C produces keystores $\mathsf{keystore}_i(C)$, $i = 1, \ldots, n$, and stores them at the corresponding server S_i together with ω_i. The generation of keystores uses the key K_C created in step 2 above as the client local generating key. Each server S_i also stores $\zeta_i(C)$ that includes the long-term state information required for running $\boldsymbol{KE}$ with client C (this may include keying material used by S_i with all its clients, e.g., a private-public key pair).

   Note: To ensure modular composition, we assume that the server and client keys/parameters for the key exchange protocol $\boldsymbol{KE}$ are chosen independently of those of the PPSS scheme.

4. At the end of the initialization stage, each server $\mathsf{S}_i \in I_C$ stores the tuple $(\sigma_i, \pi_i, \omega_i, \zeta_i, \mathsf{keystore}_i)$ associated with C while the client only stores its password pw. As said, servers may participate in the ring of more than one client in which case they store the above information for each such client.

Establishing sessions. Here clients and servers interact over insecure, adversarially controlled channels as specified in the T-PAKE security model (Appendix A).

1. When a client instance Π_i^C is created with input $I \subset I_C$ via a $\mathsf{send}(C, i, \mathsf{null}, I, \mathsf{init})$ query, C performs the PPSS Rec procedure with the servers in I and also retrieves the $\mathsf{keystore}(C)$ values stored at these servers. If the PPSS reconstruction fails (i.e., its output is $\perp$), Π_i^C aborts, otherwise C uses the reconstructed key K_C to process the retrieved keystore values.

2. Π_i^C runs a $\boldsymbol{KE}$-session with each $\mathsf{S}_1 \in I$ using the reconstructed key K_C and the information from the retrieved keystore $\mathsf{keystore}_C$, while server S_i uses the long-term state $\zeta_i(C)$.

6.3 Single-round T-PAKE from single-round PPSS

We illustrate the above composition of PPSS and regular KE protocols for obtaining T-PAKE protocols with two examples. In particular, they show how using our single-round PPSS scheme from Section 5 (implemented with the 2Hash-DH V-OPRF from Section 3.1) one obtains very efficient single-round T-PAKE protocols in the CRS (password-only) model with no PKI or secure channel requirements for clients or servers (other than the assumed secure initialization phase) and with arbitrary (t, n) threshold parameters.

T-PAKE via PPSS and symmetric-key KE. Let $\boldsymbol{P}$ be a (t, n)-PPSS protocol in the CRS model. To bootstrap a (t, n)-TPAKE protocol using $\boldsymbol{P}$, each server S_i, $i = 1, \ldots, n$, generates its state pair (σ_i, π_i) and runs with client C the Init procedure of protocol $\boldsymbol{P}$. As a result a user's secret, which we call K_C, is (t, n)-secret-shared among these servers under the protection of the PPSS scheme and the client's password pw. Next, client C uses key K_C to compute n keys $K_i = f_{K_C}(i), i = 1, \ldots, n$, where f is a pseudorandom function[3], and transmits each K_i (protected under the secure communication assumed at initialization) to the corresponding S_i who stores K_i in its client-specific $\zeta_i(C)$ state. In this case, the client's keystore is empty. Later, when a T-PAKE session at C is invoked, C runs the Rec procedure of protocol $\boldsymbol{P}$ with a sufficient number of servers to obtain K_C. C uses K_C to compute $K_1, \ldots, K_n$ and uses these keys as shared keys with the corresponding servers to exchange a session key. Any KE protocol that assumes pre-shared keys between pairs of parties can be used for this purpose. For example, C and S_i can compute their session key as $f_{K_i}(n_C, n_{\mathsf{S}_i}, id_C, id_{\mathsf{S}_i})$ where id_C, id_{S_i} stand for the identities of C and S_i respectively, and n_C, n_{S_i} are nonces exchanged between these parties that also serve as session identifiers. Note that when using a one-round PPSS scheme, the exchange of nonces can be piggybacked on top of the PPSS messages hence *preserving the single round complexity of the protocol* (with one additional message from C to S_i if key confirmation is desired). A full specification of this protocol based on the 2HashDH-NIZK V-OPRF of Section 3.1 is presented in Appendix B. One can also add forward secrecy to the protocol by using the shared key to authenticate a Diffie-Hellman exchange (also piggybacked on top of the two PPSS messages to preserve the single-round complexity).

T-PAKE via PPSS and public-key KE. The above scheme provides a full T-PAKE protocol with very little extra cost over the PPSS scheme. Its relative drawback is (as in any pre-shared key scheme) that the server needs to keep a per-client secret[4] and also that it requires secrecy for the transmission of key K_i to S_i (otherwise, our PPSS scheme only needs authenticated channels during initialization). To avoid these secrecy requirements, key exchange protocols based on public keys of the parties can be accommodated on top of a PPSS as follows. At initialization, the client generates a pair of private and public keys and stores in its $\mathsf{keystore}$ the client's private key (encrypted under a key derived from K_C), the client's public key, and each server's public key (which C learns during the secure initialization). In addition, all the information in $\mathsf{keystore}$ is authenticated with an authentication key derived from K_C. C stores $\mathsf{keystore}$ with all servers in its ring. In addition, each server S_i in C's ring stores C's public key in $\zeta_i(C)$. When a T-PAKE session is invoked at C, the client retrieves $\mathsf{keystore}$ from the servers and, after reconstructing K_C, uses this key to check the integrity of $\mathsf{keystore}$ and to decrypt its private key. With this information and the (authenticated) public keys of the servers contained in $\mathsf{keystore}$, C is ready to perform the key exchange protocol. Similarly, the servers can use C's public key stored in $\zeta_i(C)$ to bootstrap the public-key based

[3] Index i is used to uniquely identify server S_i but this can be replaced with actual (unique) server identities.

[4] We assume servers have per-client secret parameters (to limit the opportunities for online attacks against any given client) but an implementation can keep a single "master key" at the server from which per-client keys are derived. This is not the case for the key K_i shared with each client in the above protocol that needs to be stored separately.

key exchange. In particular, using a *single-round implicitly-authenticated protocol*, where the KE messages do not depend on the parties' private or public key (such as HMQV [36]), one obtains a single-round T-PAKE by piggy-backing the KE messages on top of the PPSS ones.

6.4 The Security of PPSS-derived T-PAKE

Here we prove that the generic composition described in Section 6.3 of a PPSS protocol $\boldsymbol{P}$ and a (regular) key exchange protocol $\boldsymbol{KE}$ (augmented with client keystores) results in a secure T-PAKE protocol $\boldsymbol{T}$. We consider an attacker $\mathcal{T}$ against the T-PAKE protocol $\boldsymbol{T}$ which targets a particular client. Namely, the attacker's goal is to break the security of the T-PAKE protocol in an instance of this client. This modeling has two reasons. One is to capture the extreme scenario in which the attacker corrupts all clients except one, in which case the protocol should still ensure security for the one uncorrupted party. Second, since the main challenge in building PAKE protocols is to show that there is no shortcut to breaking security other than attempting online guessing attacks against the victim, focusing on one client let us count the number of such attempts against this specific client.

Theorem 5 *Let $\boldsymbol{T}$ be a T-PAKE protocol built from a PPSS scheme $\boldsymbol{P}$ with parameters (t, n) and a key exchange protocol $\boldsymbol{KE}$ as described in Section 6.3. Assume that any attacker running time $\boldsymbol{t}_{\mathsf{ppss}}$ (resp. $\boldsymbol{t}_{\mathsf{ke}}$) has advantage at most $\mathsf{Adv}^{\mathsf{ppss}}$ (resp. $\mathsf{Adv}^{\mathsf{ke}}$) in breaking the security of $\boldsymbol{P}$ (resp. $\boldsymbol{KE}$). Then for any attacker $\mathcal{T}$ against $\boldsymbol{T}$ running time $\min\{\boldsymbol{t}_{\mathsf{ppss}}/2, \boldsymbol{t}_{\mathsf{ke}}/2\}$ we have*

1. *$Pr[\mathcal{T}$ succeeds$] \leq 1/2 + \mathsf{Adv}^{\mathsf{ppss}} + \mathsf{Adv}^{\mathsf{ke}}$. More specifically,*

$$Pr[\mathcal{T} \text{ succeeds}] \leq \frac{1}{2} + \left(q_{\mathsf{rog}}(\mathsf{U}) + \frac{q_{\mathsf{rog}}(I_{\mathsf{U}})}{t - t' + 1} \right) \cdot \frac{1}{|D|} + \mathsf{Adv}^{\mathsf{ke}}$$

   where D is the password dictionary, $q_{\mathsf{rog}}(\mathsf{U})$ is the number of rogue $\boldsymbol{P}$-send queries with U as recipient, $q_{\mathsf{rog}}(I_{\mathsf{U}})$ the number of rogue $\boldsymbol{P}$-send queries with sender U and recipient $\mathsf{S} \in I_{\mathsf{U}}$, and t' is the number of corrupted parties in I_{U} ($\boldsymbol{P}$-send refers to send queries related to the actions of subprotocol $\boldsymbol{P}$).

2. *Protocol $\boldsymbol{T}$ is correct, i.e., matching sessions compute the same session keys, except with probability ϵ_{ppss}.*

Proof: We first prove a security bound for the composed protocol and then we show correctness.

Given an attacker $\mathcal{T}$ against the composed T-PAKE protocol $\boldsymbol{T}$, we build an attacker $\mathcal{P}$ against the underlying PPSS scheme $\boldsymbol{P}$. Attacker $\mathcal{T}$ targets a specific uncorrupted client, that we will denote by U, and whom will be chosen by $\mathcal{T}$ as the test session's client. Other parties in protocol $\boldsymbol{T}$ can be corrupted and controlled by the attacker or uncorrupted and simulated by $\mathcal{P}$.

The goal of $\mathcal{P}$ is to attack a given instance $(\mathsf{U}, \{\mathsf{S}'_1, \ldots, \mathsf{S}'_n\})$ of the PPSS protocol $\boldsymbol{P}$. It starts by orchestrating a run of the $\boldsymbol{T}$ protocol in which it embeds the given PPSS instance and runs attacker $\mathcal{T}$ against it. $\mathcal{P}$ uses party U from the PPSS instance to simulate the actions of party U in protocol $\boldsymbol{T}$, and it uses servers $\{\mathsf{S}'_1, \ldots, \mathsf{S}'_n\}$ from the PPSS instance to simulate the servers in U's ring I_{U}. Specifically, $\mathcal{P}$ uses its $\boldsymbol{P}$-oracle $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ to simulate the actions of U and uses the oracle $\mathsf{S}^{\diamond}_{\mathsf{Rec}}$ to simulate the actions of servers in I_{U}. More precisely, $\mathcal{P}$ uses the server oracles only for simulating the actions of the servers in I_{U} that are uncorrupted. Servers in I_{U} that are corrupted by $\mathcal{T}$ in $\boldsymbol{T}$ are also corrupted by $\mathcal{P}$ in $\boldsymbol{P}$. In addition, $\mathcal{P}$ schedules actions and delivers messages in $\boldsymbol{P}$ in accordance to the actions and messages of $\mathcal{T}$ in $\boldsymbol{T}$.

Parties in $\boldsymbol{T}$ other than the client U and servers in I_{U} are either corrupted and controlled by $\mathcal{T}$ or they are fully simulated by $\mathcal{P}$ who chooses their secret keys and public parameters. In particular, $\mathcal{P}$ chooses the passwords of uncorrupted clients other than U and also any secret keys generated by uncorrupted servers in the set I_{U} for use with clients other than U (the secret keys used by these servers with U are defined in the PPSS game and unknown to $\mathcal{P}$).

$\mathcal{P}$ starts the simulation with the initialization procedure in $\boldsymbol{T}$, initializing all the uncorrupted parties. For initializing U and its ring I_{U}, $\mathcal{P}$ follows the initialization procedure of instance $(\mathsf{U}, \{\mathsf{S}'_1, \ldots, \mathsf{S}'_n\})$ in $\boldsymbol{P}$ via the $\mathsf{U}^{\diamond}_{\mathsf{Rec}}$ and $\mathsf{S}^{\diamond}_{\mathsf{Rec}}$ oracles. It then obtains the real-or-random challenge key K^* from the PPSS experiment and uses K^* to generate any keys and information needed to produce $\mathsf{keystore}_i(\mathsf{U})$, including choosing any private/public keys for $\mathsf{S}'_1, \ldots, \mathsf{S}'_n$ as required by the key-exchange protocol $\boldsymbol{KE}$.

When an instance Π^{U}_i of U is created by $\mathcal{T}$ with server set $I \subset I_{\mathsf{U}}$, $\mathcal{P}$ triggers a reconstruction procedure in $\boldsymbol{P}$ between U and the servers in I, and simulates this actions in $\boldsymbol{T}$. Let K^*_i be the challenge key given to $\mathcal{P}$ by the PPSS experiment as the real-or-random key for the reconstructed key in the PPSS run used to create Π^{U}_i (recall that in our model the PPSS adversary receives a real-or-random challenge key K^* at the onset of the protocol and also such a key with each reconstruction). If $K^*_i = \perp$ then Π^{U}_i aborts. Otherwise $\mathcal{P}$ uses K^*_i as the reconstructed key for all the actions of instance Π^{U}_i for establishing $\boldsymbol{KE}$ sessions with the servers in the specified set I (including keystore verification).

In addition, $\mathcal{P}$ responds to all send and reveal queries issued by $\mathcal{T}$ during the protocol. $\mathcal{P}$ can do so as it has full information of the secrets and internal state of uncorrupted parties. This is the case also for information related to protocol $\boldsymbol{KE}$ generated in U instances since $\mathcal{P}$ chooses all the $\boldsymbol{KE}$-related information for U and the uncorrupted subset of I_{U}.

When $\mathcal{T}$ issues the test session at an instance of U or with U as the peer to the session (we may assume, wlog, that $\mathcal{T}$ follows the rules in choosing the test session), $\mathcal{P}$ responds to the query by choosing a bit v at random and outputting the corresponding session key (which $\mathcal{P}$ knows) if $v = 1$ and outputting a random key otherwise.

Eventually, when $\mathcal{T}$ stops and outputs its guess v', $\mathcal{P}$ stops and outputs $b' = 1$ if $v' = v$ (indicating a guess for "real") and outputs $b' = 0$ otherwise.

We consider two cases depending on the outcome of the real-or-random PPSS experiment, namely, whether $b = 1$ or $b = 0$.

- If $b = 1$ then all keys K^*_i provided to $\mathcal{P}$ by the PPSS experiment for each reconstruction are the real ones, namely, those output by the reconstruction procedure. (Note that these keys are all equal to each other and equal to the original K^* key received by $\mathcal{P}$ at the onset of the protocol, except if soundness was broken in one of these reconstructions; but even in these cases $\mathcal{P}$ uses the provided K^*_i for simulating the corresponding instance). Hence, in this case, the simulation of the $\boldsymbol{T}$ game by $\mathcal{P}$ is perfect and the probability of $\mathcal{T}$ to win in the simulated game is exactly the same as the probability of $\mathcal{T}$ to win against a real run of protocol $\boldsymbol{T}$.

- If $b = 0$ then K^* is random (and independent of the key initialized by U) and $K^*_i = K^*$ for all instances Π^{U}_i (except for instances where the reconstruction procedure outputs $\perp$). In this case, we have a hybrid T-PAKE/KE game where all sessions involving clients other than U perfectly follow the $\boldsymbol{T}$ protocol while the sessions of U run as a pure KE+ protocol (with a random key K_{U} and correct $\mathsf{keystore}_i(\mathsf{U})$). We will refer to this game as hybrid and will analyze it below.

Based on these observations, we analyze the success probability of $\mathcal{P}$ in the above simulation.

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{ppss}}_{\mathcal{P}} &= 2 \cdot Pr\left[\mathcal{P} \text{ wins } \boldsymbol{P}\right] - 1 \\
&= 2 \cdot (Pr\left[\mathcal{P} \text{ wins and } K^* \text{ real}\right] + Pr\left[\mathcal{P} \text{ wins and } K^* \text{ random}\right]) - 1 \\
&= Pr\left[\mathcal{P} \text{ wins : } K^* \text{ real}\right] + Pr\left[\mathcal{P} \text{ wins : } K^* \text{ random}\right] - 1 \\
&= Pr\left[\mathcal{P} \text{ outputs 1: } K^* \text{ real}\right] + Pr\left[\mathcal{P} \text{ outputs 0 : } K^* \text{ random}\right] - 1 \\
&= Pr\left[\mathcal{T} \text{ wins: } K^* \text{ real}\right] + Pr\left[\mathcal{T} \text{ loses: } K^* \text{ random}\right] - 1 \\
&= Pr\left[\mathcal{T} \text{ wins in } \boldsymbol{T}\right] - Pr\left[\mathcal{T} \text{ wins: } K^* \text{ random}\right] \\
&= Pr\left[\mathcal{T} \text{ wins } \boldsymbol{T}\right] - Pr\left[\mathcal{T} \text{ wins in } \mathsf{hybrid}\right]
\end{aligned}$$

Thus,

$$Pr\left[\mathcal{T} \text{ wins } \boldsymbol{T}\right] = \mathsf{Adv}^{\mathsf{ppss}}_{\mathcal{P}} + Pr\left[\mathcal{T} \text{ wins in } \mathsf{hybrid}\right]$$

Thus, to complete the proof of security we need to show that $Pr\left[\mathcal{T} \text{ wins in } \mathsf{hybrid}\right] \leq 1/2 + \mathsf{Adv}^{\mathsf{ke}}$. For this we first recall the hybrid game. The participants are the same parties as in the T-PAKE protocol $\boldsymbol{T}$ and they all follow that protocol except for party U and its ring of servers I_{U}. Sessions with U are generated independently of the PPSS component of $\boldsymbol{T}$ but rather follow the key exchange protocol $\boldsymbol{KE}$ with key material generated in accordance to the $\boldsymbol{KE}^+$ spec. Namely, the keystore of U is generated and verified on the basis of a random generating key K^*_{U} (chosen independently of any other action in $\boldsymbol{T}$), hence resulting in key material exactly as prescribed in $\boldsymbol{KE}$. The keys for uncorrupted servers in I_{U} are generated according to protocol $\boldsymbol{KE}$ and validated (in the case that $\boldsymbol{KE}$ uses public keys) via the verification of U's keystore.

It is now easy to see that an attacker $\mathcal{T}$ against the hybrid game can be transformed into an attacker $\mathcal{A}$ against protocol $\boldsymbol{KE}$. $\mathcal{A}$ will run against an instance of protocol $\boldsymbol{KE}$ with parties U and the servers in the set I_{U}; we denote this instance by $(\mathsf{U}, I_{\mathsf{U}})_{\mathsf{ke}}$. $\mathcal{A}$ invokes $\mathcal{T}$ on a simulated run of hybrid where $\mathcal{A}$ controls all the parties not corrupted by $\mathcal{T}$. For parties other than U and the set I_{U}, $\mathcal{A}$ chooses all secret information and can answer all queries by $\mathcal{T}$. For U and the uncorrupted servers in I_{U}, $\mathcal{A}$ chooses all the information corresponding to the PPSS component, i.e., U's password, the servers' keys, and the key K_{U} initialized at U by the PPSS protocol $\boldsymbol{P}$ - note that in the hybrid game K_{U} and K^*_{U} are chosen independently.

For all the session actions in hybrid that follow the $\boldsymbol{KE}$ specification and have U as a sender or receiver, $\mathcal{A}$ uses her $(\mathsf{U}, I_{\mathsf{U}})_{\mathsf{ke}}$ instance to generate the corresponding messages and transfer them to the simulation of the hybrid run. Any queries by $\mathcal{T}$ regarding these sessions are responded by $\mathcal{A}$ by issuing the corresponding queries to $(\mathsf{U}, I_{\mathsf{U}})_{\mathsf{ke}}$. In particular, when $\mathcal{T}$ chooses a test session in the simulated run of hybrid with client U and one of the uncorrupted servers in I_{U}, $\mathcal{A}$ issues a test session query at the corresponding session in the $\boldsymbol{KE}$ instance $(\mathsf{U}, I_{\mathsf{U}})_{\mathsf{ke}}$ and returns the answer to $\mathcal{T}$. When $\mathcal{T}$ exits with bit b', $\mathcal{A}$ outputs the same bit and stops.

The above represents a perfect simulation by $\mathcal{A}$ of a run of $\mathcal{T}$ against the hybrid game where the sessions of U are exactly the same as the ones generated by the $\boldsymbol{KE}$ instance $(\mathsf{U}, I_{\mathsf{U}})_{\mathsf{ke}}$. In particular, the value of the key in the test session is the same in the simulated hybrid game and in the run of the $\boldsymbol{KE}$ instance $(\mathsf{U}, I_{\mathsf{U}})_{\mathsf{ke}}$, and so the response to the test query in hybrid is correct if and only if the same bit is correct in $\boldsymbol{KE}$. We get that the probability of $\mathcal{A}$ to guess the right bit in the $\boldsymbol{KE}$ game against instance $(\mathsf{U}, I_{\mathsf{U}})_{\mathsf{ke}}$ is the same as the probability of $\mathcal{T}$ guessing it in the hybrid game. In other words, $Pr\left[\mathcal{T} \text{ wins in } \mathsf{hybrid}\right] = Pr\left[\mathcal{A} \text{ wins in } \boldsymbol{KE}\right] =\leq 1/2 + \mathsf{Adv}^{\mathsf{ke}}_{\mathcal{A}}/2 \leq 1/2 + \mathsf{Adv}^{\mathsf{ke}}$.

This completes the proof of security. Note that the explicit bound in the theorem in terms of q_{rog} only considers rogue send messages from the $\boldsymbol{P}$ component of the protocol. Indeed, this comes

from the $\mathsf{Adv}^{\boldsymbol{P}}$ term in the bound expression hence it only needs to count rogue messages sent during the initialization and reconstruction procedures of U.

To show correctness, we observe that as long as soundness is not violated then the reconstructed key is either $\perp$ (indicating a reconstruction failure, in which case the instance aborts and no sessions are created) or it is the correct key K_{U}, namely the one initialized by U. In the latter case, if the retrieved keystores $\mathsf{keystore}_i(\mathsf{U})$ values pass validation (using K_{U}), then U reconstructs the correct key material and the correctness of $\boldsymbol{T}$ follows from the correctness of the KE protocol $\boldsymbol{KE}$. Thus, we have that the protocol is correct except in the event of a successful soundness attack against $\mathcal{P}$'s PPSS instance. The probability of such event is the same as $\mathsf{Adv}^{\mathsf{ppss}}$ (see section 4). □

Application to the V-OPRF-based PPSS scheme. We note that when one applies the bound in the above theorem to a scheme that uses the V-OPRF-based PPSS scheme from Section 5, the term $q_{\mathsf{rog}}(\mathsf{U})$ can be reduced by a factor of $t - t' + 1$ since it takes such a number of messages sent to U with a common rogue value of ω_i for the attacker to test one password.

References

[1] M. Abe and M. Ohkubo. A framework for universally composable non-committing blind signatures. In M. Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 435–450. Springer, 2009.

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

[3] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard. Practical multi-candidate election system. In A. D. Kshemkalyani and N. Shavit, editors, *PODC*, pages 274–283. ACM, 2001.

[4] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *J. Cryptology*, 16(3):185–215, 2003.

[5] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. pages 139–155, 2000.

[6] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[7] F. Benhamouda, O. Blazy, C. Chevalier, D. Pointcheval, and D. Vergnaud. New techniques for sphfs and efficient one-round pake protocols. In R. Canetti and J. A. Garay, editors, *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 449–475. Springer, 2013.

[8] J. Brainard, A. Juels, B. Kaliski, and M. Szydlo. A new two-server approach for authentication with short secrets. In *12th USENIX Security Symp*, pages 201–213, 2003.

[9] J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized schnorr proofs. In A. Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 425–442. Springer, 2009.

[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 J. A. Garay and R. Gennaro, editors, *CRYPTO (2)*, volume 8617 of *Lecture Notes in Computer Science*, pages 256–275. Springer, 2014.

[11] J. Camenisch, A. Lysyanskaya, and G. Neven. Practical yet universally composable two-server password-authenticated secret sharing. In *ACM Conference on Computer and Communications Security*, pages 525–536, 2012.

[12] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, pages 126–144, 2003.

[13] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. pages 136–145, 2001.

[14] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. Crypto'2013. Cryptology ePrint Archive, Report 2013/169, Mar. 2013. `http://eprint.iacr.org/2013/169`.

[15] S. Chow, C. Ma, and J. Weng. Zero-knowledge argument for simultaneous discrete logarithms. In M. Thai and S. Sahni, editors, *Computing and Combinatorics*, volume 6196 of *Lecture Notes in Computer Science*, pages 520–529. Springer Berlin Heidelberg, 2010.

[16] R. Cramer and I. Damgård. Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free? In H. Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 424–441. Springer, 1998.

[17] R. Cramer and V. Shoup. Signature schemes based on the strong rsa assumption. In *ACM Conference on Computer and Communications Security*, pages 46–51, 1999.

[18] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. pages 79–88, 2006.

[19] I. Damgård and E. Fujisaki. A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In *ASIACRYPT'02*, volume 2501 of *LNCS*, pages 125–142. Springer, 2002.

[20] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Non-interactive and non-malleable commitment. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 141–150, New York, NY, USA, 1998. ACM.

[21] M. Di Raimondo and R. Gennaro. Provably secure threshold password-authenticated key exchange. *J. Comput. Syst. Sci.*, 72(6):978–1001, 2006.

[22] M. Fischlin. Round-optimal composable blind signatures in the common reference string model. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 60–77. Springer, 2006.

[23] W. Ford and B. S. K. Jr. Server-assisted generation of a strong secret from a password. In *WETICE*, pages 176–180, 2000.

[24] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. pages 303–324, 2005.

[25] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In J. Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.

[26] E. Fujisaki and T. Okamoto. Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations. In *CRYPTO '97*, volume 1294 of *LNCS*, pages 16–30, 1997.

[27] J. A. Garay, P. D. MacKenzie, and K. Yang. Strengthening zero-knowledge protocols using signatures. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2003.

[28] C. Gentry, J. Groth, Y. Ishai, C. Peikert, A. Sahai, and A. Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *Journal of Cryptology*, pages 1–24, 2014.

[29] D. Jablon. Password authentication using multiple servers. In *CT-RSA'01: RSA Cryptographers' Track*, pages 344–360. Springer-Verlag, 2001.

[30] S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. pages 577–594, 2009.

[31] S. Jarecki and X. Liu. Fast secure computation of set intersection. pages 418–435, 2010.

[32] J. Katz, P. Mackenzie, G. Taban, and V. Gligor. Two-server password-only authenticated key exchange. In *Proc. Applied Cryptography and Network Security ACNS05*, 2005.

[33] J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques*, 2001.

[34] J. Katz and V. Vaikuntanathan. Round-optimal password-based authenticated key exchange. *J. Cryptology*, 26(4):714–743, 2013.

[35] A. Kiayias and H.-S. Zhou. Equivocal blind signatures and adaptive uc-security. In R. Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 340–355. Springer, 2008.

[36] H. Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In *CRYPTO*, pages 546–566, 2005.

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

[38] R. Merkle. A digital signature based on a conventional encryption function. In C. Pomerance, editor, *Advances in Cryptology CRYPTO 87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer Berlin Heidelberg, 1988.

[39] S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable random functions. pages 120–130, 1999.

[40] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS*, pages 458–467. IEEE Computer Society, 1997.

[41] New York Times. Russian Hackers Amass Over a Billion Internet Passwords. `http://www.nytimes.com/2014/08/06/technology/russian-gang-said-to-amass-more-than-a-billion-stolen-internet-credentials.html?_r=0`, August 5, 2015.

[42] NIST. Digital signature standard (DSS). Technical Report 169. National Institute for Standards and Technology, August 30, 1991.

[43] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

A T-PAKE Security Model

We present a security model for T-PAKE protocols based on the model of MacKenzie, Shrimpton and Jakobsson [37] which in turn is based on the PAKE model of Bellare, Pointcheval and Rogaway [5]. The model extends authenticated key exchange models to account for the inherent vulnerability of password protocols to online guessing attacks and to formalize the notion that other than these attacks the adversary's advantage should be negligible as in regular KE protocols.

We change some aspects of the formalism with respect to [37] to carefully capture the minimally possible security loss due to online guessing attempts as well as to simplify and/or refine some properties of the model. See remark on changes at the end.

Protocol participants. There are two types of parties participating in a T-PAKE protocol, clients and servers, individually denoted C and S, respectively. Each client has a secret password pw chosen at random from a fixed dictionary D (other source distributions for passwords are possible but the above choice simplifies presentation). Parties may have additional keying material (e.g., certified public keys of each other) but the basic model does not assume such keys. Each client C will have an associated set of servers, denoted I_C and referred to as C's ring, with which C is initialized to share its password-based authentication capability. Associated to C's ring I_C are the parameters t, n where $n = |I_C|$ and t is the maximal number of servers in I_C that can be corrupted by an

adversary before C's security is compromised. In particular, C will need to interact with at least $t+1$ servers before it can establish keys with any server in I_C. The parameters (t, n) can differ from client to client, but for simplicity we fix them as global parameters for all clients. We also assume that a client C will only establish session keys with servers in its ring I_C. This restriction simplifies the model but it can be lifted, although establishing keys with servers outside the ring would still require prior interaction with servers in I_C.

Protocol execution. A T-PAKE protocol has two phases: initialization and key exchange. In the initialization phase each client C chooses a random password pw_C from a dictionary D and then it interacts with each server S in I_C producing output $\omega_C(\mathsf{S})$ that S (but not C) stores. In the CRS model, C only stores its password (fixed global parameters such the CRS, t, n, can be seen as part of C's code). *Initialization is assumed to be executed securely, e.g., over secure channels.* In the key exchange phase, clients interact with servers over insecure (adversary-controlled) channels to establish session keys. Each client may execute the protocol multiple times with different sets of servers in its ring and in a concurrent fashion. Each such execution defines a client instance, denoted Π_i^C, where i serves to differentiate between instances of same C. A client instance is associated with a subset $I \subset I_C$ of size at least $t+1$, where the intention is to establish session keys with all the servers in I. Servers also have instances denoted Π_i^{S} with each instance associated with a single client.

Each party's instance produces one or more sessions, a local object to the party capturing the execution of the protocol with another party, referred to as the session's peer and denoted by a session variable *pid*. A client instance Π_i^C produces $|I|$ sessions, one with each server S in the set I associated with this instance as a peer. A session for instance Π_i^{S} is unique and has a client as the peer. The output of a session at party P is a session key sk which is set to $\perp$ if the party aborts the session (e.g., in case that authentication fails, a misformed message is received, etc.) When a session outputs $\mathsf{sk} \neq \perp$ we say the session accepts. Sessions have unique identifiers, denoted sid, defined by the protocol. This is usually defined as some session-specific fresh information transmitted during the session such as the concatenation of random nonces or other information in the protocol transcript (sometimes defined as the full transcript of a protocol execution) plus the identities of involved parties. The sid main purpose is to define a correspondence between sessions as formally captured by the notion of *matching sessions* defined below.

T-PAKE Security. To define security we consider a probabilistic attacker $\mathcal{A}$ which schedules all actions in the protocol and controls all communication channels with full ability to transport, modify, inject or drop messages at will. The model defines the following queries with which the adversary interacts with the protocol's parties.

$\mathsf{send}(P, i, P', M, \mathsf{tag})$: causes message M to be delivered to instance Π_i^P purportedly coming from P', with an optional value tag used to denote a message context or identifier. In response to a send query the instance takes the actions specified by the protocol and outputs a message given to $\mathcal{A}$. When a session accepts, a message indicating acceptance is provided to $\mathcal{A}$. A send query can also be used to create a new instance of party P. Specifically, $\mathsf{send}(C, i, \mathsf{null}, I, \mathsf{init})$ creates a new instance i of client C with the message specifying the set of servers $I \subset I_C$ with which C is to share new session keys. Similarly, $\mathsf{send}(\mathsf{S}, i, C, M, \mathsf{init})$ creates a new instance Π_i^{S} with incoming message M and intended peer C. *Note that this formalism assumes that protocol exchanges are initiated by clients with servers as responders (which is the operational setting in T-PAKE).*

$\mathsf{reveal}(C, i, \mathsf{S})$: if instance Π_i^C has accepted a session with peer S, this query causes the output of the corresponding session key sk; otherwise the output is $\perp$.

$\mathsf{reveal}(\mathsf{S}, i)$: if instance Π_i^{S} has accepted, this query causes the output of the corresponding session key sk; otherwise the output is $\perp$.

$\mathsf{test}(C, i, \mathsf{S})$: if instance Π_i^C has accepted a session with peer S, this query causes Π_i^C to flip a random bit b. If $b = 1$ the corresponding session key sk is output and if $b = 0$ a string drawn uniformly from the space of session keys is output.

$\mathsf{test}(\mathsf{S}, i)$: if instance Π_i^{S} has accepted, this query causes Π_i^{S} to flip a random bit b. If $b = 1$ the corresponding session key sk is output and if $b = 0$ a string drawn uniformly from the space of session keys is output.

We note that reveal queries are used to model an adversary who obtains information on session keys in some sessions while the test query is a technical tool to define security based on the indistinguishability of session keys from random. A test query may be asked at any time during the execution of the protocol, but may only be asked once (either from C or S).

Corruptions. The attacker $\mathcal{A}$ can corrupt parties meaning that the party is fully controlled by $\mathcal{A}$ (and all its secret information is chosen and/or known to $\mathcal{A}$). Since we only consider *static corruptions* we can assume without loss of generality that corruptions are chosen by $\mathcal{A}$ at the onset of the protocol. Both clients and servers can be corrupted by $\mathcal{A}$. In addition, a client C is considered corrupted if more than t of the servers in I_C are corrupted.

Rogue send queries: The following notion is needed to accurately capture and count attempts at online password guesses, and is used in our definition of security. We say that a $\mathsf{send}(P, i, P', M, \mathsf{tag})$ query is rogue if it was not generated and/or delivered according to the specification of the protocol, namely, the message M has been changed or injected by the attacker, or the delivery order differs from what is stipulated by the protocol. We also consider as rogue any $\mathsf{send}(P, i, P', M, \mathsf{tag})$ query where P is uncorrupted and P' is corrupted. In addition, we say that a rogue $\mathsf{send}(P, i, P', M, \mathsf{tag})$ query to be against client C if the query is rogue and one of the following two conditions hold: $P = C$ and P' is a server in I_C or $P' = C$ and P is a server in I_C. (We assume that a $\mathsf{send}(P, i, P', M, \mathsf{tag})$ query for which P is a client and P' is not in I_P or where P is a server and P is not in $I_{P'}$ are discarded by the recipient.)

Matching sessions. A session in instance Π_i^C and a session in instance Π_j^{S} are said to be matching if both have the same session identifier sid, the first has $pid = \mathsf{S}$, the second has $pid = C$ and both have accepted.

Fresh sessions. A session at instance Π_i^P with peer P' is called fresh if neither P or P' are corrupted and no reveal query was issued against the session or against its matching session, if such session exists.

Correctness. Matching sessions between uncorrupted peers output the same session key.

Attacker's advantage. Let $\boldsymbol{T}$ be a T-PAKE protocol and $\mathcal{A}$ be an attacker with the above capabilities running against $\boldsymbol{T}$. Assume that $\mathcal{A}$ issues a single test query against a fresh session at a client or server and ends its run with an output bit b'. We say that $\mathcal{A}$ wins if $b' = b$ where b is the bit chosen internally by the test session. The advantage of $\mathcal{A}$ against T is defined as

$$\mathsf{Adv}_{\mathcal{A}}^{\boldsymbol{T}} = 2 \cdot Pr\left[\mathcal{A} \text{ wins against } \boldsymbol{T}\right] - 1.$$

We are now ready to define security of a T-PAKE protocol. The definition requires correctness as well as a small attacker's advantage beyond the inevitable online guessing attacks. Below we say that C is the "test session's client" if the test session belongs to a instance Π_i^P where $P = C$ or P is a server and C is the sessions peer.

Definition 2 *A T-PAKE protocol* $\boldsymbol{T}$ *with threshold parameters* (t, n) *is* $(t, n, T, q_{\mathsf{rog}}, \epsilon)$-$\mathsf{secure}$ *if it is correct and for any password dictionary* D*, and any attacker* $\mathcal{A}$ *that runs in time* T *and issues at most* q_{rog} *rogue* send *queries against the test session's client, it holds that* $\mathsf{Adv}^{\boldsymbol{T}}_{\mathcal{A}} \leq q_{\mathsf{rog}}/|D| + \epsilon$.

The above bound based on q_{rog} represents a liberal choice. It accepts a protocol where the attacker can test a different password with each protocol message the attacker tampers with and which has the test session's client as sender or receiver. This is needed in order to keep the definition as general as possible without making any assumptions on the mechanics of the T-PAKE protocol. We will see that for PPSS-based schemes as those we build one can do significantly better, requiring several rogue send messages to test a single password. Another aspect of the definition that is worth highlighting is the requirement to only count rogue send's that have the target client C as sender or recipient. This is intended to force the attacker to disclose the identity of the client being attacked in an online guessing attempt. This allows servers to count rogue messages targeted at a particular client and limit the number of such messages. This leads to the requirement in our V-OPRF-based T-PAKE schemes that servers use independent per-client V-OPRFkeys (note that adding the client identity as an input to the V-OPRFwould not solve this issue).

Note (changes with respect to the model of MacKenzie et al [37]). For reference, we list some of the changes we made to the model of [37]. These include allowing multiple sessions under a single client instance, the removal of the "Execute" command (which was used to represent sessions in which the attacker does not modify the messages or the sessions flow but which fails to account for adversarial scheduling of sessions and the interleaving between messages of different sessions), the definition of rogue send queries (and counting only these queries targeted at the test client), the modeling of server rings that can change from client to client (which allows to deal with individually compromised rings rather than having all system compromised when $t + 1$ servers are corrupted), and adding an explicit correctness requirement to avoid trivial protocols (without it, a protocol where each party chooses an independent random key as the session key would be secure).

B DH-based Instantiation of the PPSS and T-PAKE Protocols

For illustration and for the reader's convenience we describe in Figure 9 the specific instantiation of the PPSS and T-PAKE protocols based on the 2HashDH-NIZK V-OPRF, with the NIZK for DL equality implemented as in [15], and a symmetric-key KE scheme. We comment on some of our choices for this illustration. The initialization is presented for the case in which the client generates the servers' V-OPRF keys and computes all the values in the ω vector by itself. Another option, more in line with the formal description of the PPSS protocol from Figure 8, is for the servers to choose their own V-OPRF keys and engage in an V-OPRF computation with the client for generating the pads used to encrypt the shares s_i. One advantage of the latter option is that servers can save in the amount of secret memory and derive the V-OPRF keys for each user U using a single key MK and a PRF F, i.e., as $k_{\mathsf{U}} = F_{MK}(\mathsf{U})$ (we are abusing the symbols U and S_i to denote the identities of these parties). This option is more useful with a PK-based KE, where servers do not need to store user-specific secrets (in contrast, the protocol from Figure 9 requires the server storing the session key with each user). User performance during reconstruction is improved by choosing a common value ρ for blinding the $H_1(\mathsf{pw})$ value sent to all servers. We stress that while we specifiy the actions of honest servers, corrupted ones can deviate from the protocol in any way they choose to. Finally, note that the protocol as presented does not include an explicit authentication mechanism. This can be easily added, for example, by server S_i adding the value

$f_{K_i}(0, \mu_i, \mu_i'))$ to its message and by U adding a third message with value $f_{K_i}(1, \mu_i', \mu_i))$ (in this case, the session key could be derived as $SK_i = f_{K_i}(2, \mu_i, \mu_i', \mathsf{U}, \mathsf{S}_i)$).

Parties: User U, Servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$.

Public parameters and components: Security parameters τ and ℓ, threshold parameters $t, n \in \mathbb{N}, t \leq n$, field $\mathbb{F} = GF(2^\ell)$, cyclic group of prime order m with generator g; hash functions H_1, H_2, H_3, H_4, H_5 with ranges $\langle g \rangle, \{0,1\}^\ell, \{0,1\}^\tau, Z_m, Z_m$, respectively; pseudorandom generator G and pseudorandom function family f.

Initialization (secure channels between U and each server S_i are assumed only through initialization): User U performs the following steps:

1. Chooses $s \in_R \mathbb{F}$ and generates shares $(s_1, \ldots, s_n)$ as a (t, n) Shamir's secret-sharing of s over field $\mathbb{F}$.
2. For $i = 1, \ldots, n$, U chooses value $k_i \in_R Z_m$ and sets $\pi_i = g^{k_i}$ and $c_i = s_i \oplus H_2(\pi_i, \mathsf{pw}, (H_1(\mathsf{pw}))^{k_i})$.
3. Sets $\mathbf{c} = (c_1, \ldots, c_n)$, $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$, $[r||K] = G(s)$, $C = H_3(r, \mathsf{pw}, \boldsymbol{\pi}, \mathbf{c})$; For $i = 1, \ldots, n$, sets $K_i = f_K(\mathsf{S}_i)$.
4. For $i = 1, \ldots, n$, sends to server S_i the values $\omega = (\boldsymbol{\pi}, \mathbf{c}, C), k_i, K_i$.
5. U memorizes pw and erases all other information.

Each server S_i, $i = 1, \ldots, n$, stores $\omega, k_i, y_i = g^{k_i}, K_i$ in its U-specific storage ζ_i.

Reconstruction/Key Exchange (black text describes the reconstruction protocol; red additions are for key exchange):

– User U initiates a key exchange session with servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$ by sending to each S_i the value $a = (H_1(\mathsf{pw}))^\rho$ with $\rho \in_R \mathbb{Z}_m$ and a nonce $\mu_i \in_R \{0,1\}^\tau$.

– Upon receiving (a, μ_i), server S_i checks that $a \in \langle g \rangle$ and if so, S_i retrieves k_i and $y_i = g^{k_i}$ from its U-related storage $\zeta_i(\mathsf{U})$, picks $z \in_R \mathbb{Z}_m$, and computes $b_i = a^{k_i}$, $\gamma = H_4(g, y_i, a, b_i)$, $v_i = H_5(g, y_i, a, b_i, (g \cdot a^\gamma)^z)$, and $u_i = z + v_i \cdot k_i \bmod m$. S_i sends to U the values y_i, b_i, u_i, v_i as well as a nonce $\mu_i' \in_R \{0,1\}^\tau$ and the value ω_i stored in $\zeta_i(\mathsf{U})$. S_i computes the session key with U as $SK_i = f_{K_i}(\mu_i, \mu_i', \mathsf{U}, \mathsf{S}_i)$.

– Upon receiving values $b_i, u_i, v_i, \omega_i, \mu_i'$ from S_i, U proceeds as follows:

- U chooses a subset of servers S for which the following conditions hold: (i) there is a value $\omega = (\boldsymbol{\pi}, \mathbf{c}, C)$ with $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ and $\mathbf{c} = (c_1, \ldots, c_n)$ such that $\omega_i = \omega$ for all $\mathsf{S}_i \in S$; (ii) $y_i = \pi_i$ for all $\mathsf{S}_i \in S$; (iii) $b_i \in \langle g \rangle$ and the equality $v_i = H_5(g, y_i, a, b_i, (g \cdot a^\gamma)^{u_i} \cdot (y_i \cdot b_i^\gamma)^{-v_i})$ for $\gamma = H_4(g, y_i, a, b_i)$ holds for all $\mathsf{S}_i \in S$; (iv) $|S| = t + 1$.
- If no such subset exists U aborts. Otherwise, set $s_i = c_i \oplus H_2(y_i, \mathsf{pw}, b_i^{1/\rho})$, for each $\mathsf{S}_i \in S$, and reconstruct s from these s_i shares using polynomial interpolation.
- Compute $[r||K] = G(s)$. If $C \neq H_3(r, \mathsf{pw}, \boldsymbol{\pi}, \mathbf{c})$ then U aborts.
- For each $\mathsf{S}_i \in S$, set $K_i = f_K(\mathsf{S}_i)$ and compute $SK_i = f_{K_i}(\mu_i, \mu_i', \mathsf{U}, \mathsf{S}_i)$.

Figure 9: DH-based PPSS and T-PAKE Protocols (red color indicates key-exchange specific operations on top of PPSS)