# Password Protected Universal Thresholdizer

Sabyasachi Dutta[1], Partha Sarathi Roy[2],
Reihaneh Safavi-Naini[3], and Willy Susilo[2]

[1] Department of Computer Science and Engineering
SRM University, AP
Neeru Konda, Amaravati, Andhra Pradesh 522502, India
sabyasachi.d@srmap.edu.in
[2] Institute of Cybersecurity and Cryptology
School of Computing and Information Technology
University of Wollongong
Northfields Avenue, Wollongong NSW 2522, Australia
{partha,wsusilo}@uow.edu.au
[3] Department of Computer Science
University of Calgary
2500 University Drive, Calgary, NW T2N 1N4, Canada
rei@ucalgary.ca

**Abstract.** Universal thresholdizer (UT) was proposed by Boneh et al. in CRYPTO'18 as a general framework for thresholdizing non-threshold cryptographic primitives where a set of $N$ servers, each gets a share such that any set of $k$ servers, each produce a partial result, which can be combined to generate the final result. In many applications of threshold cryptography such as the protection of private keys in a digital wallet, the combining operation of partial results must be protected. In this paper, we extend the UT framework to include password authentication for such protection. We formalize the notion of password protected universal thresholdizer (PPUT) that requires the knowledge of a password to execute the protocol, propose a general construction of PPUT, and prove its security. Our construction uses threshold password authenticated key exchange (TPAKE) with simulation-based security as one of the main building blocks. We define simulation-based security of TPAKE in standalone model and give a construction using threshold fully-homomorphic encryption. As an application of PPUT, we propose a new primitive called password protected threshold signature. All the proposed constructions are secure in the standard model, and can be instantiated from lattices.

## 1 Introduction

Threshold cryptography aims to distribute a cryptographic task among $N$ servers such that any $k$ out of $N$ servers can perform the task, without the need to reconstruct the secret key. An important example of threshold cryptosystems is threshold digital signature [13,14] where the task of signing a message is distributed among $N$ servers such that each server can generate a partial signature on the message using their share, and any $k$ partial signatures can be combined to

generate a full signature on the message. A signed message can be verified by the verification key of the signature scheme. An important application of threshold signatures in digital wallets is distributing shares of private key among servers, and using partial signatures that are generated by a subset of servers to construct the signed message. A second example is threshold decryption of a secure public key encryption system [12,13] where the decryption task is distributed among $N$ servers such that any set of size $k$ out of $N$ servers, can decrypt the ciphertext. In CRYPTO'18, Boneh et al. [9] provided a general framework, called "Universal Thresholdizer" ($\mathsf{UT}$), that gives a general construction for thresholdizing many non-threshold cryptographic schemes. A universal thresholdizer takes the secret key of a cryptographic function as input and produces secret states of the servers, allowing them to compute partial evaluations of the function such that any $k$ such evaluations produce the final result.

In practice, a user must authenticate themselves to the servers to execute the protocol and obtain the results. A direct way of achieving this goal is for the user to establish an individual password with each server and use the passwords for authentication when the task needs to be initiated. However, one needs to carefully define the security model for authentication and the thresholdized function to guarantee the security of the final results, and manage the significant burden of storing and using $N$ distinct passwords in practice. Password authenticated cryptographic systems, such as threshold password authenticated key exchange ($\mathsf{TPAKE}$) [1,11,20,23] and password protected secret sharing ($\mathsf{PPSS}$) [4,18,19,10,2,24] aim to integrate password authentication into the threshold cryptographic task using a single password for user authentication with all $N$ servers. Security requirements of these systems must also model protection against the low-entropy password. For example in $\mathsf{TPAKE}$, the goal is establishing authenticated keys between the user and each server, and the extra password security requirement is that the best an adversary can do is "guess" the password correctly and try to impersonate the user. A similar requirement exists in $\mathsf{PPSS}$ that stores the shares of a secret at $N$ servers such that any $k$ shares can generate the secret that can be combined by a user who knows the password to recover the secret. Furthermore, security must ensure that the best strategy for an adversary is to guess the password and attack the system. More specifically, the security guarantee for any password-based scheme is any PPT impersonation attack is bounded by $O(1/|\mathbf{D}|) + \mathsf{negl}(\lambda)$ for some security parameter $\lambda$ and dictionary $\mathbf{D}$.

### 1.1 Our Work

In this paper, our goal is to equip the universal thresholdizer ($\mathsf{UT}$) of [9] with password authentication. We propose a general construction that applies to many cryptographic tasks, converting them into a password protected functionality to authenticate a user based on its password. As an application of our generic construction, we present a password authenticated threshold signature that can be used for distributed storage of wallet-key appropriate for signing transactions in blockchain technology.

We design a general framework of password protected universal thresholdizer (PPUT) which can be used to thresholdize cryptographic primitives and require password authentication of users to servers to execute a cryptographic task. From a very high level, each server in a subset of size greater than or equal to a threshold value participating in the protocol, computes "partial evaluation" of the functionality which are sent to the user and only a legitimate user holding the correct password can combine them to output the end result. To this end, we define a simulation-based security of PPUT to ensure that the partial evaluations do not reveal any information about the secret states used in the system. We define simulation-based security of PPUT following the simulation-based security of UT by Boneh et al. [9]. Note that simulation-security of the UT proposed in [9] only protects against semi-honest adversaries. However, any password-authenticated system requires protection against malicious adversaries and hence we model the simulation-based security of PPUT against malicious adversaries in stand-alone model. Our construction of PPUT uses UT of [9], and a threshold password-authenticated key exchange (TPAKE) to introduce user authentication feature into the UT and provide security against malicious adversaries. We reduce the security of our PPUT construction to the security of TPAKE which necessitates the design of simulation-based security of the underlying TPAKE. To this end, we first propose a Real-Ideal world simulation-based security for TPAKE in stand-alone model following the simulation-based security model of PAKE by Goldreich and Lindell [17]. In the proposed security models of TPAKE and PPUT, we consider the uniform distribution of passwords over the dictionary. Any TPAKE realizing the proposed security definition can fit into the proposed design rationale of PPUT construction. To the best of our knowledge, a quantum-safe construction of TPAKE was not available except [24] which only outlined a generic methodology to obtain TPAKE; but neither a formal description of the security model nor a security reduction was provided. In this paper, we exhibit a construction of TPAKE using the threshold fully homomorphic encryption (TFHE) by Boneh et al. [9] to realize the proposed definition, which is also of independent interest and may have further applications to construct other quantum-safe password protected systems.

In the proposed PPUT, keeping the spirit of password protected systems, the user only needs to remember the password and does not need any secure storage after the one-time Setup phase. Our protocol ensures that if the user's password is correct, it obtains a valid output; otherwise, it does not learn anything. Finally, as an application, we propose a new primitive called password protected threshold signature (PPTS) with construction using an EUF-CMA secure signature scheme and proposed PPUT. It is worth mentioning that all of our proposed constructions can be instantiated based on the hardness assumptions from lattices secure in the standard model, which makes them quantum-safe.

**Outline of our approach.** Our basic approach to constructing PPUT is the following: (*i*) the user uses the UT to obtain a thresholdized version of the cryptographic task; (*ii*) uses a TPAKE to establish an authenticated shared key between the user and each of the servers; (*iii*) each server in a set of $k$ (out of $N$) servers

encrypts partial evaluations using their shared secret key with the user, and send the encrypted partial evaluations to the user. The user decrypts the received ciphertexts, and combines them to achieve the final result. While TPAKE solves the problem of establishing a secret key between a server and a user, where the user authenticates with a password, it is not immediate whether the problem of creating a general framework for thresholdizing non-threshold cryptographic primitives allowing user authentication is feasible by a straightforward combination of UT and TPAKE. We emphasize that using the existing security models of TPAKE, it is not clear how to define PPUT and its simulation-based security. A significant redesign of the security model of TPAKE is required to make the construction of PPUT satisfy its desired security properties. In the following, we first describe the basic design rationale of our TPAKE construction using TFHE which only provides security against semi-honest adversaries, and then discuss its extension which provides protection against malicious adversaries. Subsequently, we describe overviews of PPUT and PPTS constructions.

**Threshold password authenticated key exchange (TPAKE).** The basic idea of TPAKE is to encrypt the secret key of a CCA secure encryption scheme together with the password using the TFHE, and distribute the shares of the secret key of the TFHE among the servers. To establish authenticated shared key, the TPAKE protocol starts by running an interactive authentication procedure between the servers and the user, and if successful allows the user to recover the secret key of the CCA secure encryption scheme. Now each server can individually select a random secret key, and encrypt it with the public key of the encryption scheme. The user can then recover the keys and the TPAKE is completed. The protocol has the advantage that the user does not need to store the secret key of the encryption scheme. The system however is vulnerable to malicious attacks. The protocol uses a simulation sound NIZK and signature scheme to satisfy the simulation-based security against malicious adversaries.

In more details, TPAKE consists of two algorithms: ($\mathtt{TPAKE.Reg}, \mathtt{TPAKE.Login}$) involving user $\mathbf{U}$ and $N$ servers $\mathbf{S}_1, \ldots, \mathbf{S}_N$. The registration algorithm is performed by a user $\mathbf{U}$ with the input of a password $\mathsf{pw}$ and the Login is an interactive protocol between user $\mathbf{U}$ with input $\widetilde{\mathsf{pw}}$ and a subset of $k$ servers $\mathbf{S}_i$ with their secret state $\mathsf{st}_i$. User $\mathbf{U}$ has a public key, private key pair $(\mathtt{pk}^{\mathtt{cca}}, \mathtt{sk}^{\mathtt{cca}})$ corresponding to a CCA secure encryption scheme $(\mathsf{Enc}^{cca}, \mathsf{Dec}^{cca})$ and runs the $\mathtt{TFHE.Setup}$ of $k$-out-of-$N$ TFHE scheme to obtain $(\mathtt{pk}^{\mathtt{tfhe}}, \mathtt{sk}_1^{\mathtt{tfhe}}, \ldots, \mathtt{sk}_N^{\mathtt{tfhe}})$. The secret keys $\mathtt{sk}_i^{\mathtt{tfhe}}$'s constitute the secret states of the $i^{th}$ server. The encryptions of $\mathsf{pw}$ and $\mathtt{sk}^{\mathtt{cca}}$ with respect to the public key $\mathtt{pk}^{\mathtt{tfhe}}$ and the encrypted values $\mathbf{C}_{\mathsf{pw}}, \mathbf{C}_{\mathtt{sk}^{\mathtt{cca}}}$ are kept public.

Servers initiate the Login by sending encryptions w.r.t $\mathtt{pk}^{\mathtt{tfhe}}$ of random elements to the user (viz. $\mathbf{C}_{R_j}$'s for random $R_j$'s) to mask the encryption of password to resist offline dictionary attack in case adversary tries to impersonate the valid user. The user then replies to every server with fresh encryption of its password $\mathbf{C}_{\widetilde{\mathsf{pw}}}$, an encryption $\mathbf{C}_X$ of a randomly chosen $X$ and the set of $\{\mathbf{C}_{R_j}\}$. We shall see later that $X$ is randomly chosen to perfectly hide the secret $\mathtt{sk}^{\mathtt{cca}}$ from the eavesdropper during the $\mathtt{TFHE.FinDec}$. Each of the servers computes

the encryption $\mathbf{CF}_j$ of $\sum_j R_j(\widetilde{\mathsf{pw}} - \mathsf{pw}) + \mathtt{sk}^{\mathtt{cca}} + X$ from the ciphertexts, obtained from the user during the protocol and from the public values, utilizing the TFHE.Eval algorithm for computation on encrypted values. The servers then run TFHE.PartDec using their secret state $\mathtt{sk}_{\mathtt{i}}^{\mathtt{tfhe}}$ and $\mathtt{pk}^{\mathtt{tfhe}}$ on $\mathbf{CF}_i$ to output partial decryption $\mathbf{W}_i$ of $\mathbf{CF}_i$. The user then retrieves $\mathtt{sk}^{\mathtt{cca}} + X$ using the TFHE.FinDec algorithm and removes the randomly chosen $X$ to obtain $\mathtt{sk}^{\mathtt{cca}}$. Note that, the TFHE.FinDec algorithm only takes $\mathbf{W}_i$'s and $\mathtt{pk}^{\mathtt{tfhe}}$ as inputs and hence masking by a random $X$ is crucial to hide the secret from the adversary. Once the secret key is recovered by the user, the key exchange is easy - each server $\mathbf{S}_j$ participating in the protocol chooses a uniformly random key $\mathsf{key}_j$ from a predefined key space and sends the encryption $\mathtt{Enc}^{\mathtt{cca}}{}_{\mathtt{pk}^{\mathtt{cca}}}(\mathsf{key}_j)$ to the user who can now decrypt (using $\mathtt{sk}^{\mathtt{cca}}$) to obtain the common key $\mathsf{key}_j$ with $\mathbf{S}_j$.

We can easily see that the above preliminary design cannot resist malicious adversarial attacks. To provide security against malicious adversaries in the absence of secure channels, we use a simulation sound NIZK and to protect against adversarial tampering of protocol messages, use EUF-CMA signature scheme. In each transmission, each participating server signs their corresponding messages using a EUF-CMA signature to protect against any adversarial tampering of the protocol messages. For this, the user includes the signing keys of servers in their secret states $\mathsf{st}_i$'s and includes all the corresponding verification keys into pp during the registration phase. In the preliminary design, the user is supposed to reply with an encryption of its password with fresh randomness. However, a malicious user may re-randomize the encrypted password stored in pp which will ultimately prove it as a legitimate user. To resist this attack, we force the user to forward fresh encryption of the password. To this end, the legitimate user generates another set of TFHE keys $((\widehat{\mathtt{pk}^{\mathtt{tfhe}}}, \widehat{\mathtt{sk}_1^{\mathtt{tfhe}}}, \ldots, \widehat{\mathtt{sk}_N^{\mathtt{tfhe}}}))$[4] during the registration and append the public key $\widehat{\mathtt{pk}^{\mathtt{tfhe}}}$ in pp. The user needs to encrypt the same password, using $\mathtt{pk}^{\mathtt{tfhe}}$ and $\widehat{\mathtt{pk}^{\mathtt{tfhe}}}$ with the same randomness and sends the two encryptions along with the "proof" of same randomness using a simulation sound NIZK. Lastly, we note that malicious server(s) may try to convince the user with a different secret $sk' \neq \mathtt{sk}^{\mathtt{cca}}$. The user signs the secret message $\mathtt{sk}^{\mathtt{cca}}$ during registration phase using EUF-CMA signature to output $\sigma$. Finally, the user encrypts $\mathtt{sk}^{\mathtt{cca}}||\sigma$ instead of just encrypting $\mathtt{sk}^{\mathtt{cca}}$. At the end of the recovery phase, the user checks the veracity of the recovered secret by the verification key of signature. The user needs to add the verification key of the user's signature in pp during the execution of registration.

If no server $\mathbf{S}_j$ abort the process, each one of them randomly chooses a key $\mathsf{key}_j$, signs the key with the EUF-CMA Signature to produce $\sigma_j$, encrypts $(\mathsf{key}_j, \sigma_j)$ with $\mathtt{pk}^{\mathtt{cca}}$ to output $e_j$ and sends $e_j$ to the user along with $\mathbf{W}_j$. The user decrypts $e_j$ to obtain $(\mathsf{key}_j, \sigma_j)$ and verifies whether $\sigma_j$ is a valid signature

[4] The secret keys $\widehat{\mathtt{sk}_{\mathtt{i}}^{\mathtt{tfhe}}}$'s do not play any role at any point during the execution of Login. The user neither requires $\widehat{\mathtt{sk}_{\mathtt{i}}^{\mathtt{tfhe}}}$'s to be shared and stored among the servers nor to keep them with itself - it can delete $\widehat{\mathtt{sk}_{\mathtt{i}}^{\mathtt{tfhe}}}$'s after executing the Registration.

of $\mathsf{key}_j$. If so, then the user locally outputs $\mathsf{key}_j$ as its common secret key with server $\mathbf{S}_j$.

**Password protected Universal thresholdizer (PPUT).** We use the proposed TPAKE to define a primitive called a password protected universal thresholdizer (PPUT) that can be used to thresholdize existing systems including signatures (Section 5) with the property that a user is authenticated by a set of servers. The resulting systems are secure threshold systems that also provide robustness guarantees against malicious share holders.

A PPUT scheme consists of the following algorithms: `PPUT.Setup`, `PPUT.Eval`, `PPUT.Verify`, `PPUT.Combine`. The setup of PPUT takes input a secret message $x$, a password pw and runs `TPAKE.Reg` and `UT.Setup` to generate a set of secret states $S_1, \ldots, S_N$, which are distributed to $N$ servers. In the evaluation phase, on input a circuit $C$ and a password pw (by the user), a threshold number of servers authenticate a user as a legitimate one by triggering `TPAKE.Login`. Each server independently computes partial evaluation $Y_i$ of $C(x)$ using their secret state $S_i$ by running the `UT.Eval`. Note that, the usage of TFHE in the proposed construction allows us to input any circuit. To transmit the partial evaluation only to the intended authenticated user, each server emulates a secure channel by encrypting $Y_i$ (using a CPA secure symmetric key encryption) by the key established during `TPAKE.Login`. We denote encryption of $Y_i$ by $y_i$. To achieve robustness, we execute the Verify algorithm of PPUT which essentially runs the `UT.Verify` algorithm on $Y_i$. Hence, the encrypted value $y_i$ needs to be decrypted first to obtain $Y_i$. Only the authenticated user with the correct password pw can now perform decryption procedure using the established key, and also can check whether $Y_i$ was computed correctly and (simultaneously) that $y_i$ is not tampered by a channel adversary. We derive the robustness as a consequence of the robustness property of the UT scheme of Boneh et al. [9]. For any $k$ many evaluations $y_i$'s which are input to the `PPUT.Combine` algorithm, only the authenticated user can obtain the corresponding (plaintext) $Y_i$'s and combine them to correctly produce $Y = C(x)$ using the combine algorithm of underlying UT. This makes the combining algorithm of the PPUT password protected.

**Password protected Threshold Signature (PPTS).** We now use PPUT to propose password protected threshold signature (PPTS) which distributes the task of generating a signature among a set of servers on a user supplied message where the user authenticates itself to the set of servers using a password. The PPTS scheme is a tuple (Init, `PPTS.PartSign`, `PPTS.PartSignVerify`, `PPTS.Combine`, `PPTS.Vrfy`). The proposed construction uses PPUT on an EUF-CMA signature scheme `Sig` = (S.KeyGen, S.Sign, S.Verify). The verification key svk of the signature scheme is also used as the verification key for the PPTS and the signing key ssk is used as the secret input to the `PPUT.Setup` algorithm. Except the `PPTS.Vrfy` (which is a public verification), all other algorithms take the pw of the user as an input. In order to trigger the signing procedure and user authentication `PPTS.PartSign` requires the password pw of the user as an input. Similarly, to resist an illegitimate user from obtaining a valid signature on a message, the `PPTS.Combine` algorithm must include pw as an input. On the other hand,

to protect the legitimate user from accepting any maliciously generated partial signature(s) for computing the signature, we use the PPTS.PartSignVerify algorithm to detect whether the servers computed the partial evaluations honestly. PPTS.PartSign, PPTS.PartSignVerify, PPTS.Combine run respectively PPUT.Eval, PPUT.Verify, PPUT.Combine which necessitates the pw to be used as inputs to the first three algorithms. PPTS.Vrfy runs S.Verify which keeps the PPTS.Vrfy algorithm public. We can instantiate PPTS using our lattice-based PPUT from section 4 and lattice-based adaptive EUF-CMA signature scheme from [3].

## 2 Preliminaries

We present definitions and properties of some important primitives.

### 2.1 Secret Sharing

Originally proposed by Blakley [7] and Shamir [26], a secret sharing system is used to distribute a secret among a set of servers $\mathcal{S}$ such that authorized subsets of servers can reconstruct the secret, and unauthorized set will not learn anything. Let $\mathbb{A}$ be a subset of the power set, $2^{\mathcal{S}}$, that specifies the subsets of servers that form an authorized set; *i.e.* the set of their shares can recover the secret. A subset $F \subset \mathcal{S}$ which is not in $\mathbb{A}$, *i.e.* $F \notin \mathbb{A}$, is called an unauthorized (or, forbidden) set and the set of shares $(\mathsf{sh}_u)_{u \in F}$ will be independent of secret $S$. We take the set of all unauthorized sets to be the complement of $\mathbb{A}$ *i.e.*, $\mathbb{A}^c$. This means that for any subset $X \subset \mathcal{S}$, either it is an authorized set or an unauthorized set. Such specifications of authorized and unauthorized subsets define an access structure on $\mathcal{S}$. We also consider those access structures which satisfy the following condition: if $A_1 \in \mathbb{A}$ and $A_1 \subseteq A_2$, then $A_2 \in \mathbb{A}$. This condition is referred to as *monotonicity* of access structures. In this paper, we mainly focus on the $k$-out-of-$N$ threshold access structures where any set of size at least $k$ is authorized. However, our definitions will be for any access structures to capture the generality.

**Definition 1 ($(\mathbb{A}, N, \mathcal{M})$ Secret Sharing Scheme).** *Let $\mathcal{S}$ be a set of $N$ servers labeled by $[N] = \{1, 2, \ldots, N\}$. Let $\mathbb{A}$ be an access structure on these $N$ servers. A secret sharing scheme $\Pi$ for $\mathbb{A}$ consists of a pair of algorithms $(\mathsf{Gen}, \mathit{Rec})$. $\mathsf{Gen}$ is a randomized algorithm that gets as input a secret $S$ (from a domain of secrets $\mathcal{M}$ with at least two elements), $\mathbb{A}$ and the number of servers $N$, and generates $N$ shares $(\mathsf{sh}_1, \ldots, \mathsf{sh}_N) \longleftarrow \mathsf{Gen}(S)$. $\mathit{Rec}$ is a deterministic algorithm that gets as input the shares of a subset $B$ of servers and outputs a string. The requirements for defining a secret sharing scheme are as follows:*

- Correctness: *If $\{\mathsf{sh}_u\}_u = \mathsf{Gen}(S; R)$ for some $R$, then for any $B \in \mathbb{A}$, we always have $\mathit{Rec}(\{\mathsf{sh}_u\}_{u \in B}) = S$.*
- Perfect privacy: *For any two distinct secrets $s_0 \neq s_1$ in $\mathcal{M}$ and for any distinguisher $D$ with output in $\{0, 1\}$, it must hold that for any unauthorized set $F$*

$$|Pr[D(\mathsf{Gen}(s_0)_F) = 1] - Pr[D(\mathsf{Gen}(s_1)_F) = 1]| = 0.$$

### 2.2 Signature Scheme

**Definition 2 (Signature Scheme).** *A signature scheme* $\mathtt{Sig}$ *is a tuple of algorithms* $\mathtt{Sig} = (\mathsf{S.KeyGen}, \mathsf{S.Sign}, \mathsf{S.Verify})$ *defined as follows*:

- $\mathsf{S.KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$*: On input the security parameter* $\lambda$*, the key generation algorithm outputs a signing key* $\mathsf{sk}$ *and a verification key* $\mathsf{vk}$*.*
- $\mathsf{S.Sign}(\mathsf{msg}, \mathsf{sk}) \to \sigma$*: On input a signing key* $\mathsf{sk}$*, and a message* $\mathsf{msg} \in \{0,1\}^*$*, the signing algorithm outputs a signature* $\sigma$*.*
- $\mathsf{S.Verify}(\mathsf{vk}, \mathsf{msg}, \sigma) \to \{0,1\}$*: On input a verification key* $\mathsf{vk}$*, a message* $\mathsf{msg}$ *and, a signature* $\sigma$*, the verification algorithm accepts or rejects.*

We require a signature scheme $\mathtt{Sig}$ to satisfy the following correctness and security properties.

**Definition 3 (Correctness).** *We say that a signature scheme* $\mathtt{Sig}$ *is correct if for all* $\lambda \in \mathbb{N}$*,* $\mathsf{msg} \in M$*,* $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{S.KeyGen}(1^\lambda)$*, we have that*

$$\Pr[\mathsf{S.Verify}(\mathsf{vk}, \mathsf{S.Sign}(\mathsf{msg}, \mathsf{sk}) = 1)] = 1.$$

**Definition 4 (Unforgeability).** *We say that a signature scheme satisfies unforgeability or* EUF-CMA *security if for any PPT adversary* $\mathcal{A}$*, the following experiment* $\mathtt{Expt}_{\mathcal{A},\mathtt{Sig},uf}(1^\lambda)$ *outputs* 1 *with negligible probability:*

$\mathtt{Expt}_{\mathcal{A},\mathtt{Sig},uf}(1^\lambda)$ :

- *The challenger runs* $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{S.KeyGen}(1^\lambda)$*, and provides* $\mathsf{vk}$ *to* $\mathcal{A}$*.*
- $\mathcal{A}$ *issues a polynomial number of adaptive queries* $\mathsf{msg}$*. For each query, the challenger computes* $\sigma \leftarrow \mathsf{S.Sign}(\mathsf{msg}, \mathsf{sk})$ *and provides* $\sigma$ *to* $\mathcal{A}$*.*
- *At the end of the experiment,* $\mathcal{A}$ *outputs a forgery* $(\mathsf{msg}^*, \sigma^*)$*. The experiment outputs* 1 *if* $\mathsf{S.Verify}(\mathsf{vk}, \mathsf{msg}^*, \sigma^*) = 1$ *and* $\mathsf{msg}^*$ *was not previously queried as a signing query.*

### 2.3 Threshold Fully Homomorphic Encryption

**Definition 5 (Threshold Fully Homomorphic Encryption (TFHE) [9]).** *Let* $\mathcal{S} = \{\mathbf{S}_1, \cdots, \mathbf{S}_N\}$ *be a set of servers and let* $\mathbb{S}$ *be a class of efficient access structures on* $\mathcal{S}$*. A threshold fully homomorphic encryption scheme for* $\mathbb{S}$ *is a tuple of PPT algorithms* $\mathsf{TFHE} = (\mathtt{TFHE.Setup}, \mathtt{TFHE.Encrypt}, \mathtt{TFHE.Eval}, \mathtt{TFHE.PartDec}, \mathtt{TFHE.FinDec})$ *with the following properties:*

- $\mathtt{TFHE.Setup}(1^\lambda, 1^d, \mathbb{A}) \to (\mathtt{pk}, \mathtt{sk}_1, \cdots, \mathtt{sk}_N)$ : *On input the security parameter* $\lambda$*, a depth bound* $d$*, an access structure* $\mathbb{A}$*, the setup algorithm outputs a public key* $\mathtt{pk}$*, and a set of secret key shares* $\mathtt{sk}_1, \cdots, \mathtt{sk}_N$*.*
- $\mathtt{TFHE.Encrypt}(\mathtt{pk}, \mu) \to \mathtt{ct}$ : *On input a public key* $\mathtt{pk}$*, and a single bit plaintext* $\mu \in \{0,1\}$*, the encryption algorithm outputs a ciphertext* $\mathtt{ct}$*.*
- $\mathtt{TFHE.Eval}(\mathtt{pk}, C, \mathtt{ct}_1, \cdots, \mathtt{ct}_k) \to \widehat{\mathtt{ct}}$ : *On input a public key* $\mathtt{pk}$*, circuit* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most* $d$*, and a set of ciphertexts* $\mathtt{ct}_1, \cdots, \mathtt{ct}_k$*, the evaluation algorithm outputs a cipehrtext* $\widehat{\mathtt{ct}}$*.*

- $\texttt{TFHE.PartDec}(\texttt{pk}, \texttt{ct}, \texttt{sk}_\texttt{i}) \to \texttt{p}_\texttt{i}$ : *On input a public key* $\texttt{pk}$*, a ciphertext* $\texttt{ct}$*, and a secret key share* $\texttt{sk}_\texttt{i}$*, the partial decryption algorithm outputs a partial decryption* $\texttt{p}_\texttt{i}$ *related to the server* $\mathbf{S}_i$*.*
- $\texttt{TFHE.FinDec}(\texttt{pk}, B) \to \hat{\mu}$ : *On input a public key* $\texttt{pk}$*, and a set* $B = \{\texttt{p}_\texttt{i}\}_{i \in S}$ *for some* $S \subseteq \{\mathbf{S}_1, \cdots, \mathbf{S}_N\}$*, the final decryption algorithm outputs a plaintext* $\hat{\mu} \in \{0, 1, \perp\}$*.*

We require that a TFHE scheme satisfies compactness, correctness, and security.

**Definition 6 (Compactness).** *We say that a* TFHE *scheme is compact if there exists polynomials* $poly_1(\cdot)$ *and* $poly_2(\cdot)$ *such that for all* $\lambda$*, depth bound* $d$*, circuit* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most* $d$*, and* $\mu \in \{0,1\}$*, the following holds: for* $(\texttt{pk}, \texttt{sk}_1, \cdots, \texttt{sk}_\texttt{N}) \leftarrow \texttt{TFHE.Setup}(1^\lambda, 1^d, \mathbb{A})$*,* $\texttt{ct} \leftarrow \texttt{TFHE.Encrypt}(\texttt{pk}, \mu)$*,* $\hat{\texttt{ct}} \leftarrow \texttt{TFHE.Eval}(\texttt{pk}, C, \texttt{ct}_1, \cdots, \texttt{ct}_\texttt{k})$*,* $\texttt{p}_\texttt{j} \leftarrow \texttt{TFHE.PartDec}(\texttt{pk}, \texttt{ct}, \texttt{sk}_\texttt{j})$ *for any* $j \in [N]$*,* $|\hat{ct}| \leq poly_1(\lambda, d)$ *and* $|p_j| \leq poly_2(\lambda, d, N)$*.*

**Definition 7 (Evaluation Correctness).** *We say that a* TFHE *scheme satisfies evaluation correctness if for all* $\lambda$*, depth bound* $d$*, access structure* $\mathbb{A}$*, circuit* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most* $d$*,* $Q \in \mathbb{A}$*, and* $\mu_i \in \{0,1\}$ *for* $i \in [k]$*, the following holds: for* $(\texttt{pk}, \texttt{sk}_1, \cdots, \texttt{sk}_\texttt{N}) \leftarrow \texttt{TFHE.Setup}(1^\lambda, 1^d, \mathbb{A})$*,* $\texttt{ct}_i \leftarrow \texttt{TFHE.Encrypt}(\texttt{pk}, \mu_i)$ *for* $i \in [k]$*,* $\hat{\texttt{ct}} \leftarrow \texttt{TFHE.Eval}(\texttt{pk}, C, \texttt{ct}_1, \cdots, \texttt{ct}_\texttt{k})$*,*

$$\Pr[\texttt{TFHE.FinDec}(\texttt{pk}, \{\texttt{TFHE.PartDec}(\texttt{pk}, \texttt{ct}, \texttt{sk}_\texttt{i})\}_{i \in Q}) = C(\mu_1, \cdots, \mu_k)] = 1 - \mathsf{negl}(\lambda).$$

**Definition 8 (Simulation Security).** *We say that a* TFHE *scheme satisfies simulation security if for all* $\lambda$*, depth bound* $d$*, access structure* $\mathbb{A}$*, the following holds: there exists a stateful PPT algorithm* $Sim = (Sim_1, Sim_2)$ *such that for any PPT adversary* $\mathcal{A}$*, the following experiments* $\texttt{Expt}_{\mathcal{A}, Real}(1^\lambda, 1^d)$ *and* $\texttt{Expt}_{\mathcal{A}, Ideal}(1^\lambda, 1^d)$ *are computationally indistinguishable:*

$\texttt{Expt}_{\mathcal{A}, Real}(1^\lambda, 1^d)$ :

1. *On input the security parameter* $1^\lambda$*, circuit depth* $1^d$*, the adversary* $\mathcal{A}$ *outputs an access structure* $\mathbb{A} \in \mathbb{S}$*.*
2. *The challenger runs* $(\texttt{pp}, s_1, \cdots, s_N) \leftarrow \texttt{TFHE.Setup}(1^\lambda, 1^d, \mathbb{A})$ *and provides* $\texttt{pk}$ *to* $\mathcal{A}$*.*
3. $\mathcal{A}$ *outputs a maximal forbidden set* $S^* \subset \{\mathbf{S}_1, \cdots, \mathbf{S}_N\}$ *and messages* $\{\mu_1, \cdots, \mu_k\} \in \{0,1\}$*.*
4. *The challenger provides the shares* $\{\texttt{sk}_j\}_{j \in S^*}$ *and* $\texttt{TFHE.Encrypt}(\texttt{pk}, \mu_i)$ *for* $i \in [k]$ *to* $\mathcal{A}$*.*
5. $\mathcal{A}$ *submits a polynomial numbers of adaptive queries of the form* $(S \subset \{\mathbf{S}_1, \cdots, \mathbf{S}_N\}, C)$ *for circuits* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most* $d$*. For each query, the challenger computes* $\hat{\texttt{ct}} \leftarrow \texttt{TFHE.Eval}(\texttt{pk}, C, \texttt{ct}_1, \cdots, \texttt{ct}_\texttt{k})$*, and provides* $\{\texttt{TFHE.PartDec}(\texttt{pk}, \hat{\texttt{ct}}, \texttt{sk}_\texttt{i})\}_{i \in S}$ *to* $\mathcal{A}$*.*
6. *At the end of the experiment,* $\mathcal{A}$ *outputs a distinguishing bit* $b$*.*

$\texttt{Expt}_{\mathcal{A}, Ideal}(1^\lambda, 1^d)$ :

1. *On input the security parameter* $1^\lambda$*, circuit depth* $1^d$*, the adversary* $\mathcal{A}$ *outputs an access structure* $\mathbb{A} \in \mathbb{S}$.
2. *The challenger runs* $(\mathsf{pp}, s_1, \cdots, s_N, \mathsf{st}) \leftarrow Sim_1(1^\lambda, 1^d, \mathbb{A})$ *and provides* $\mathtt{pk}$ *to* $\mathcal{A}$.
3. $\mathcal{A}$ *outputs a maximal forbidden set* $S^* \subset \{\mathbf{S}_1, \cdots, \mathbf{S}_N\}$ *and messages* $\{\mu_1, \cdots, \mu_k\} \in \{0,1\}$.
4. *The challenger provides the shares* $\{\mathtt{sk}_j\}_{j \in S^*}$ *and* $\mathtt{TFHE.Encrypt}(\mathtt{pk}, \mu_i)$ *for* $i \in [k]$ *to* $\mathcal{A}$.
5. $\mathcal{A}$ *submits a polynomial numbers of adaptive queries of the form* $(S \subset \{\mathbf{S}_1, \cdots, \mathbf{S}_N\}, C)$ *for circuits* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most* $d$*. For each query, the challenger runs the simulator* $\{\mathtt{p}_i\}_{i \in S} \leftarrow Sim_2(C, \{\mathtt{ct_1}, \cdots, \mathtt{ct_k}\}, C(\mu_1, \cdots, \mu_k), S, \mathsf{st})$*, and provides* $\{\mathtt{p}_i\}_{i \in S}$ *to* $\mathcal{A}$.
6. *At the end of the experiment,* $\mathcal{A}$ *outputs a distinguishing bit* $b$.

### 2.4 Universal Thresholdizer

**Definition 9 (Universal Thresholdizer [9]).** *Let* $\mathcal{S} = \{\mathbf{S}_1, \cdots, \mathbf{S}_N\}$ *be a set of servers and let* $\mathbb{S}$ *be a class of efficient access structures on* $\mathcal{S}$*. A universal thresholdizer scheme for* $\mathbb{S}$ *is a tuple of PPT algorithms* $\mathsf{UT} = (\mathtt{UT.Setup}, \mathtt{UT.Eval}, \mathtt{UT.Verify}, \mathtt{UT.Combine})$ *with the following properties:*

- $\mathtt{UT.Setup}(1^\lambda, 1^d, \mathbb{A}, x) \to (\mathsf{pp}, s_1, \cdots, s_N)$ : *On input the security parameter* $\lambda$*, a depth bound* $d$*, an access structure* $\mathbb{A}$ *and a message* $x \in \{0,1\}^k$*, the setup algorithm outputs the public parameters* $\mathsf{pp}$*, and a set of shares* $s_1, \cdots, s_N$.
- $\mathtt{UT.Eval}(\mathsf{pp}, s_i, C) \to y_i$ : *On input the public parameters* $\mathsf{pp}$*, a share* $s_i$*, and a circuit* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most* $d$*, the evaluation algorithm outputs a partial evaluation* $y_i$.
- $\mathtt{UT.Verify}(\mathsf{pp}, y_i, C) \to \{0,1\}$ : *On input the public parameters* $\mathsf{pp}$*, a partial evaluation* $y_i$*, and a circuit* $C : \{0,1\}^k \to \{0,1\}$*, the verification algorithm accepts or rejects.*
- $\mathtt{UT.Combine}(\mathsf{pp}, B) \to y$ : *On input the public parameters* $\mathsf{pp}$*, a set of partial evaluations* $B = \{y_i\}_{i \in Q}$ *where* $Q \in \mathbb{A}$*, the combining algorithm outputs the final evaluation* $y$.

We require that an $\mathsf{UT}$ scheme to satisfy compactness, correctness, robustness, and security.

**Definition 10 (Compactness).** *We say that a* $\mathsf{UT}$ *scheme is compact if there exists a polynomial* $poly(\cdot)$ *such that for all* $\lambda$*, depth bound* $d$*, circuit* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most* $d$*, and* $\mu \in \{0,1\}$*, the following holds: for* $(\mathsf{pp}, s_1, \cdots, s_N) \leftarrow \mathtt{UT.Setup}(1^\lambda, 1^d, \mathbb{A}, x)$*,* $y_i \leftarrow \mathtt{UT.Eval}(\mathsf{pp}, s_i, C)$ *for any* $i \in [N]$*,* $y_i \le poly(\lambda, d, N)$.

**Definition 11 (Evaluation Correctness).** *We say that a* $\mathsf{UT}$ *scheme is compact if for all* $\lambda$*, depth bound* $d$*, circuit* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most*

*d, access structure* $\mathbb{A}$*, message* $x \in \{0,1\}^k$*,* $Q \in \mathbb{A}$ *the following holds: for* $(\mathsf{pp}, s_1, \cdots, s_N) \leftarrow \mathtt{UT.Setup}(1^\lambda, 1^d, \mathbb{A}, x)$*,*

$$\Pr[\mathtt{UT.Combine}(\mathsf{pp}, \{\mathtt{UT.Eval}(\mathsf{pp}, s_i, C)\}_{i \in Q}) = C(x)] = 1 - \mathsf{negl}(\lambda).$$

**Definition 12 (Verification Correctness).** *We say that a* UT *scheme satisfies verification correctness if for all* $\lambda$*, depth bound* $d$*, password* pw*, access structure* $\mathbb{A}$*, message* $x \in \{0,1\}^k$*, and circuit* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most* $d$*, the following holds. For* $(\mathsf{pp}, s_1, \cdots, s_N) \leftarrow \mathtt{UT.Setup}(1^\lambda, 1^d, \mathbb{A}, x)$ *and* $y_i \leftarrow \mathtt{UT.Eval}(\mathsf{pp}, S_i, C)$ *for any* $i \in [N]$*, we have that*

$$\Pr[\mathtt{UT.Verify}(\mathsf{pp}, y_i, C) = 1] = 1.$$

**Definition 13 (Security of UT).** *We say that a* UT *scheme satisfies security if for all* $\lambda$*, and depth bound* $d$*, the following holds. There exists a PPT algorithm* $Sim = (Sim_1, Sim_2)$ *such that for any PPT adversary* $\mathcal{A}$*, we have that the following experiments* $\mathtt{Expt}_{\mathcal{A},UT,Real}(1^\lambda, 1^d)$ *and* $\mathtt{Expt}_{\mathcal{A},UT,Ideal}(1^\lambda, 1^d)$ *are computationally indistinguishable:*

$\mathtt{Expt}_{\mathcal{A},UT,Real}(1^\lambda, 1^d)$ :

1. *On input the security parameter* $1^\lambda$*, circuit depth* $1^d$*, the adversary* $\mathcal{A}$ *outputs an access structure* $\mathbb{A} \in \mathbb{S}$*, and a message* $x \in \{0,1\}^k$ .
2. *The challenger runs* $(\mathsf{pp}, s_1, \cdots, s_N) \leftarrow \mathtt{UT.Setup}(1^\lambda, 1^d, \mathbb{A}, x)$ *and provides* pp *to* $\mathcal{A}$*.*
3. $\mathcal{A}$ *outputs a maximal forbidden set* $S^* \subset \{\mathbf{S}_1, \cdots, \mathbf{S}_N\}$ *for* $\mathcal{A}$*.*
4. *The challenger provides the shares* $\{s_i\}_{i \in S^*}$ *to* $\mathcal{A}$*.*
5. $\mathcal{A}$ *issues a polynomial number of adaptive queries of the form* $(S \subset \{\mathbf{S}_1, \cdots, \mathbf{S}_N\}, C)$ *for circuits* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most* $d$*. For each query, the challenger provides* $\{y_i \leftarrow \mathtt{UT.Eval}(\mathsf{pp}, s_i, C)\}_{i \in S}$ *to* $\mathcal{A}$*.*
6. *At the end of the experiment,* $\mathcal{A}$ *outputs a distinguishing bit* $b$*.*

$\mathtt{Expt}_{\mathcal{A},UT,Ideal}(1^\lambda, 1^d)$ :

1. *On input the security parameter* $1^\lambda$*, circuit depth* $1^d$*, the adversary* $\mathcal{A}$ *outputs an access structure* $\mathbb{A} \in \mathbb{S}$*, and a message* $x \in \{0,1\}^k$ .
2. *The challenger runs simulator* $(\mathsf{pp}, s_1, \cdots, s_N) \leftarrow Sim_1(1^\lambda, 1^d, \mathbb{A})$ *and provides* pp *to* $\mathcal{A}$*.*
3. $\mathcal{A}$ *outputs a maximal forbidden set* $S^* \subset \{\mathbf{S}_1, \cdots, \mathbf{S}_N\}$ *for* $\mathcal{A}$*.*
4. *The challenger provides the shares* $\{s_i\}_{i \in S^*}$ *to* $\mathcal{A}$*.*
5. $\mathcal{A}$ *issues a polynomial number of adaptive queries of the form* $(S \subset \{\mathbf{S}_1, \cdots, \mathbf{S}_N\}, C)$ *for circuits* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most* $d$*. For each query, the challenger provides* $\{y_i \leftarrow Sim_2(\mathsf{pp}, C, C(x))\}_{i \in S}$ *to* $\mathcal{A}$*.*
6. *At the end of the experiment,* $\mathcal{A}$ *outputs a distinguishing bit* $b$*.*

**Definition 14 (Robustness of UT).** *We say that a* UT *scheme satisfies security if for all* $\lambda$*, and depth bound* $d$*, the following holds. For any PPT adversary* $\mathcal{A}$*, the following experiment* $\mathtt{Expt}_{\mathcal{A},UT,Robust}(1^\lambda, 1^d)$ *outputs* 1 *with negligible probability:*

$\mathtt{Expt}_{\mathcal{A},UT,Robust}(1^\lambda, 1^d)$ :

1. *On input the security parameter* $1^\lambda$*, circuit depth* $1^d$*, the adversary* $\mathcal{A}$ *outputs an access structure* $\mathbb{A} \in \mathbb{S}$*, and a message* $x \in \{0,1\}^k$ .
2. *The challenger runs* $(\mathsf{pp}, s_1, \cdots, s_N) \leftarrow \mathtt{UT.Setup}(1^\lambda, 1^d, \mathbb{A}, x)$ *and provides* $(\mathsf{pp}, s_1, \cdots, s_N)$ *to* $\mathcal{A}$.
3. $\mathcal{A}$ *outputs a fake partial evaluation* $y_i^*$.
4. *The challenger returns* 1 *if* $\mathtt{UT.Eval}(\mathsf{pp}, s_i, C) \neq y_i^*$ *and* $\mathtt{UT.Verify}(\mathsf{pp}, y_i^*, C) = 1$.

## 3 Threshold Password Authenticated Key Exchange

Password authenticated key exchange (PAKE) was proposed by Bellovin and Merritt [6]. Mackenzie et al. [20] first thresholdized the idea of PAKE and introduced TPAKE which required a threshold number of servers to authenticate a user and securely establish keys with them. Here, we first define simulation-based security of TPAKE in stand-alone model followed by a construction.

**Definition 15 ($(k,N)$-Threshold Password Authenticated Key Exchange).** *A $k$-out-of-$N$ threshold password authenticated key exchange protocol for a dictionary space* $\mathbf{D}$ *is a tuple* $(\mathtt{TPAKE.Reg}, \mathtt{TPAKE.Login})$ *involving user* $\mathbf{U}$ *and* $N$ *servers* $\mathbf{S}_1, \ldots, \mathbf{S}_N$ :

- $(\mathsf{pp}, \mathsf{st}_1, \ldots, \mathsf{st}_N) \longleftarrow \mathtt{TPAKE.Reg}(\mathsf{pw})$ : *On input a password* $\mathsf{pw} \in \mathbf{D}$ *it outputs* $(\mathsf{pp}, \mathsf{st}_1, \ldots, \mathsf{st}_N)$*, where* $\mathsf{pp}$ *is public parameters/state and* $\mathsf{st}_i$ *is the private state of server* $\mathbf{S}_i$.
- $\{\mathsf{key}_i\}_{i \in Q} / \perp \longleftarrow \mathtt{TPAKE.Login}(\widetilde{\mathsf{pw}}, \mathsf{pp}, \mathsf{st}_{i \in Q})$ : $\mathtt{TPAKE.Login}$ *is an interactive protocol between user* $\mathbf{U}$ *with input* $\widetilde{\mathsf{pw}}$ *and a subset of* $k$ *servers* $\mathbf{S}_i(\mathsf{st}_i)$ *indexed by* $Q \subset \{1, \ldots, N\}$ *such that at the end either* $\mathbf{U}$ *and* $\mathbf{S_i}$ *share a common key* $\mathsf{key}_i$ *from a key-space Key for all* $i \in Q$ *or* $\perp$ *is output.*

*Security of* TPAKE: Our definition for the task of authenticated key generation is based on the simulation paradigm. That is, we require that a secure protocol emulates an ideal execution of a key generation protocol. In such an ideal execution, a trusted party hands identical, uniformly distributed keys to the honest parties (in case of an honest user, it is a k-tuple of keys and for an honest server it is the one key that corresponds to the user). The only power given to the adversary in this ideal model is to prevent the trusted party from handing keys to one of the honest parties. We emphasize that in this ideal model of execution, the adversary learns nothing about the password and the established keys corresponding to the honest parties. However, if the user or a qualified set of servers is corrupted, then the adversary is given the power to fully determine the session key. The rationale for this is that the aim of key exchange is to enable honest parties to generate a key that is unknown to an external adversary. If the user or a qualified set of servers is corrupted, then the adversary will learn the generated key (because it is one of the legitimate participants), and so the security requirement is meaningless. Furthermore, $\mathcal{A}$ controls the communication line between the (honest) user and the (honest) servers. Thus, it can block all communication

between the user and servers, and cause any protocol to fail. This (unavoidable) adversarial capability is modeled in the functionality by letting $\mathcal{A}$ input a single bit $b$ indicating whether or not the execution is to be successful. Specifically, if $b = 1$ (i.e., success) then the user and participating servers receive session-key. On the other hand, if $b = 0$ then the servers receive a uniformly random string, whereas the user receives a special abort symbol $\perp$ instead[5]. In conclusion, the problem of $\mathsf{TPAKE}$ is cast as the following functionality:

$$(\{\mathsf{st}_i\}_{i\in Q}, \mathsf{pw}, b) \rightarrow \begin{cases} (\{\mathcal{U}_i\}_{i\in Q}, \{\mathcal{U}_i\}_{i\in Q}) & if\ b = 1, \\ & \mathbf{U}\ \&\ \{\mathbf{S}_i\}_{i\in Q}\ are\ honest; \\ (\perp, \{\mathcal{U}_i\}_{i\in Q}) & if\ b = 0, \\ & \mathbf{U}\ \&\ \{\mathbf{S}_i\}_{i\in Q}\ are\ honest; \\ (\perp, \{\mathcal{U}_i\}_{i\in F}) & if\ b = 0\ or\ 1, \\ & \{\mathbf{S}_i\}_{i\in F}; \\ (\{\mathcal{U}_i^*\}_{i\in Q}, \{\mathcal{U}_i^*\}_{i\in Q}) & if\ b = 1, \\ & \mathbf{U}\ or\ \{\mathbf{S}_i\}_{i\in Q}\ are\ dishonest; \end{cases}$$

where $\mathcal{U}_i$ denotes uniform random variable over the space $Key$ and $\mathcal{U}_i^*$ is adversarially chosen distribution over $Key$, $Q$ is a qualified set of servers, and $F$ is a forbidden set of servers.

*The Ideal Model* : Let $\mathbf{U}$ and $\{\mathbf{S}_1, \ldots, \mathbf{S}_N\}$ denote the user and the set of servers respectively. Let $\mathcal{A}$ denote a PPT adversary in the ideal model (with arbitrary auxiliary input $\mathsf{aux}$). An ideal-model execution proceeds in the following phases:

- **Setup:** A password $\mathsf{pw} \in_R \mathbf{D}$ is uniformly chosen from the dictionary and given to the user $\mathbf{U}$. $\mathsf{st}_1, \mathsf{st}_2, \cdots, \mathsf{st}_N$ are chosen and given to $\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_N$, respectively.
- **Sending inputs to trusted party:** If the user $\mathbf{U}$ is not dishonest, $\mathbf{U}$ sends the trusted party the password $\mathsf{pw}$ it has received in the setup stage; otherwise, $\mathbf{U}$ sends $\widehat{\mathsf{pw}}$ which is adversarially chosen. The adversary $\mathcal{A}$ sends either 1 (denoting a successful protocol execution) or 0 (denoting a failed protocol execution). Moreover, some servers may be captured by $\mathcal{A}$. The server $\mathbf{S}_i$ sends the trusted party $\mathsf{st}_i$ he has received in the setup stage if $\mathbf{S}_i$ is not captured by $\mathcal{A}$; Otherwise, server $\mathbf{S}_i$ sends the trusted party $\widehat{\mathsf{st}_i}$, which is adversarially chosen.
- **The trusted party answers all parties:** In the case $\mathcal{A}$ sends 1 and $\{\mathbf{S}_i\}_{i\in Q}$ are not corrupted, the trusted party chooses uniformly distributed strings $\mathsf{key}_1, \mathsf{key}_2, \cdots, \mathsf{key}_Q$ and sends $\mathsf{key}_i$ to both $\mathbf{U}$ and $\mathbf{S}_i$. In the case $\mathcal{A}$ sends 0, the trusted party sends $\perp$ to $\mathbf{U}$ and uniformly random string to $\{\mathbf{S}_i\}_{i\in Q}$. In the case only a forbidden set servers are corrupted, the trusted party sends

[5] This lack of symmetry in the definition is inherent as it is not possible to guarantee that the user and the servers both terminate with the same "success/failure bit". For sake of simplicity, we (arbitrarily) choose to have servers always receive a uniformly distributed session key and to have the user always output $\perp$ when $b = 0$.

$\perp$ to $\mathbf{U}$ and uniformly random string to $\{\mathbf{S}_i\}_{i\in Q}$. In all the above cases, $\mathcal{A}$ receives no output. In the case $\mathcal{A}$ sends 1 and $\{\mathbf{S}_i\}_{i\in Q}$ are corrupted or the user $\mathbf{U}$ is corrupted, the trusted party chooses strings $\mathsf{key}_1, \mathsf{key}_2, \cdots, \mathsf{key}_Q \in Key$ and sends $\mathsf{key}_i$ to both $\mathbf{U}$, $\{\mathbf{S}_i\}_{i\in Q}$ and $\mathcal{A}$.

The ideal distribution is defined as follows:

$$ideal_{\mathcal{A}}(\mathbf{D}, \mathsf{aux}) = (\mathsf{pw}, output(\mathbf{U}(\mathsf{pw})), \{output\mathbf{S}_i(\mathsf{st}_i)\}_{i\in Q}, output(\mathcal{A}(\mathsf{aux}))).$$

*The Real Model* : As in the ideal model, the real model begins with a setup stage in which the user $\mathbf{U}$ receives uniformly distributed password $\mathsf{pw} \in \mathbf{D}$, and $\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_N$ receive $\mathsf{st}_1, \mathsf{st}_2, \cdots, \mathsf{st}_N$, respectively. Then, the protocol is executed among $\mathbf{U}$ and $\{\mathbf{S}_i\}_{i\in Q}$ communicating via $\mathcal{A}$. The execution of this protocol is denoted $\mathcal{A}^{\mathbf{U}(\mathsf{pw}),\{\mathbf{S}_i(\mathsf{st}_i)\}_{i\in Q}}(\mathsf{aux})$ and we augment $\mathcal{A}'s$ view with the accept/reject decision bits (this decision bit denotes whether a party's private output is a session-key or $\perp$). The real distribution is defined as follows:

$$real_{\mathcal{A}}(\mathbf{D}, \mathsf{aux}) =$$

$$(\mathsf{pw}, output(\mathbf{U}(\mathsf{pw})), \{output\mathbf{S}_i(\mathsf{st}_i)\}_{i\in Q}, output(\mathcal{A}^{\mathbf{U}(\mathsf{pw}),\{\mathbf{S}_i(\mathsf{st}_i)\}_{i\in Q}}(\mathsf{aux})).$$

An important observation in the context of password-based security is that, during an online attack, an adversary can always attempt impersonation by simply guessing the secret password and participating in the protocol, claiming to be the user $\mathbf{U}$. If the adversary's guess is correct, then impersonation always succeeds (and, for example, the adversary knows the generated session-key). Furthermore, by executing the protocol with $k$ many servers, the adversary can verify whether or not its guess is correct, and thus can learn information about the password (e.g., it can rule out an incorrect guess from the list of possible passwords). Since the dictionary $\mathbf{D}$ may be small, this information learned by the adversary in a protocol execution may not be negligible at all. User and servers merely want to establish that they are talking to one another. Repeating an observation made above, we note that if the adversary initiates $\ell \leq |\mathbf{D}|$ instances of the $(k, N)$-TPAKE protocol, guessing a different password in each of them, then with probability $\ell/|\mathbf{D}|$ it will succeed in impersonating user to servers (and furthermore find the password). Therefore, in the context of a password-only setting, a $(k, N)$-TPAKE protocol is said to be secure if the above-mentioned ideal-model emulation results in an output distribution that can be distinguished from a real execution by (a gap of) at most $O(1/|\mathbf{D}|) + \mathsf{negl}(\lambda)$, where $\ell$ is the number of sessions initiated by the adversary.

**Definition 16.** *A protocol for threshold password authenticated key generation is secure if the following two requirements hold:*

1. *Semi-honest adversaries: For every PPT real-model semi-honest adversary $\mathcal{A}$ there exists a PPT ideal-model adversary $\mathcal{A}^*$ such that for every dictionary $\mathbf{D}$ and every auxiliary input $\mathsf{aux} \in \{0,1\}^{\mathsf{poly}(\lambda)}$ it should hold that $\{\mathtt{ideal}_{\mathcal{A}^*}(\mathbf{D}, \mathsf{aux})\}_{\mathbf{D},\mathsf{aux}} \approx_c \{\mathtt{real}_{\mathcal{A}}(\mathbf{D}, \mathsf{aux})\}_{\mathbf{D},\mathsf{aux}}$*

2. *Malicious adversaries: For every PPT real-model adversary $\mathcal{A}$ there exists a PPT ideal-model adversary $\mathcal{A}^*$ such that for every dictionary $\mathbf{D}$ and every auxiliary input $\mathsf{aux} \in \{0,1\}^{\mathsf{poly}(\lambda)}$ it should hold that* $\{\mathtt{ideal}_{\mathcal{A}^*}(\mathbf{D}, \mathsf{aux})\}_{\mathbf{D},\mathsf{aux}} \approx_{O(\frac{1}{\mathbf{D}})+\epsilon} \{\mathtt{real}_{\mathcal{A}}(\mathbf{D}, \mathsf{aux})\}_{\mathbf{D},\mathsf{aux}}$

### 3.1 A Construction of TPAKE from TFHE

In the following, we describe a construction of TPAKE using TFHE scheme of Boneh et al. [9], an EUF-CMA signature, a CCA secure public key encryption scheme, and a simulation sound NIZK (SS-NIZK). In Figure 1 we present the TPAKE.Reg algorithm and in Figure 2, the TPAKE.Login algorithm.

Parties: User $\mathbf{U}$, Servers $\mathbf{S}_1, \ldots, \mathbf{S}_N$.
Building Blocks: CCA secure encryption scheme $(\mathsf{Enc}^{cca}, \mathsf{Dec}^{cca})$; EUF-CMA Signature $(\mathsf{Sig}, \mathsf{Ver})$; and the $k$-out-of-$N$ TFHE scheme based on Shamir SS of [9].

TPAKE.Reg (secure channels between $\mathbf{U}$ and each $\mathbf{S}_i$ is assumed only for registration phase): On input a password $\mathsf{pw} \in \mathbf{D}$,

1. Run $\mathtt{Keygen}^{cca}$ to obtain $(\mathtt{pk}^{\mathtt{cca}}, \mathtt{sk}^{\mathtt{cca}})$
2. Run TFHE.Setup twice independently to obtain $(\mathtt{pk}^{\mathtt{tfhe}}, \mathtt{sk}_1^{\mathtt{tfhe}}, \ldots, \mathtt{sk}_N^{\mathtt{tfhe}})$ and $(\widehat{\mathtt{pk}^{\mathtt{tfhe}}}, \widehat{\mathtt{sk}_1^{\mathtt{tfhe}}}, \ldots, \widehat{\mathtt{sk}_N^{\mathtt{tfhe}}})$.
3. Generate EUF-CMA Signature algorithm $\{(\mathsf{Sig}_i, \mathsf{Ver}_i)\}_{i=1}^N$ for servers and $(\mathsf{Sig}_{\mathbf{U}}, \mathsf{Ver}_{\mathbf{U}})$ for user.
4. Compute $\sigma_u = \mathsf{Sig}_{\mathbf{U}}(\mathtt{sk}^{\mathtt{cca}})$ and run $\mathtt{TFHE.Enc}_{\mathtt{pk}^{\mathtt{tfhe}}}$ to encrypt $\mathsf{pw}$ and $\mathtt{sk}^{\mathtt{cca}} || \sigma_u$ and obtain $\mathbf{C}_{\mathsf{pw}}, \mathbf{C}_{\mathtt{sk}^{\mathtt{cca}}}$.
5. Output $(\mathsf{pp}, \mathsf{st}_1, \ldots, \mathsf{st}_N)$, where $\mathsf{pp} \leftarrow \{\mathtt{pk}^{\mathtt{tfhe}}, \widehat{\mathtt{pk}^{\mathtt{tfhe}}}, \mathtt{pk}^{\mathtt{cca}}, \mathbf{C}_{\mathsf{pw}}, \mathbf{C}_{\mathtt{sk}^{\mathtt{cca}}}, \{\mathsf{Ver}_i\}_{i=1}^N, \mathsf{Ver}_{\mathbf{U}}\}$, and $\{\mathsf{st}_i \leftarrow \{\mathtt{sk}_i^{\mathtt{tfhe}}, \mathsf{Sig}_i\}\}_{i=1}^N$.
6. $\mathbf{U}$ memorizes $\mathsf{pw}$ and erases all other information.

**Fig. 1.** $(k, N)$-TPAKE.Reg algorithm

**Security Analysis.**

**Lemma 1.** *For every PPT adversary $\mathcal{A}'$ interacting with servers $\mathbf{S}_1, \ldots, \mathbf{S}_{k-1}, \mathbf{S}_k$ there exists a non-interactive machine $\mathcal{A}''$, such that*

$$\{\mathsf{pw}, \mathsf{st}_1, \ldots, \mathsf{st}_{k-1}, \mathsf{st}_k, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathcal{U}, \mathit{output}(\mathcal{A}''(\mathsf{aux}))\} \approx$$
$$\{\mathsf{pw}, \mathsf{st}_1, \ldots, \mathsf{st}_{k-1}, \mathsf{st}_k, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathsf{key}_k, \mathit{output}(\mathcal{A}'^{\{\mathbf{S}_i(\mathsf{st}_i)\}_{1 \le i \le k}}(\mathsf{aux}))\}$$

*Proof.* Let the real model adversary $\mathcal{A}$ corrupts (without loss of generality) the first $k-1$ servers $\mathbf{S}_1, \ldots, \mathbf{S}_{k-1}$ and obtains their private states $\mathsf{st}_1, \ldots, \mathsf{st}_{k-1}$ in the beginning of the Login protocol. Also, suppose the user $\mathbf{U}$ interacts with $\mathbf{S}_1, \ldots, \mathbf{S}_{k-1}$ (corrupted servers) and $\mathbf{S}_k$ (uncorrupted server) during the Login phase for the sake of simplicity. Therefore, after the Login phase (if it succeeds

$\underline{\texttt{TPAKE.Login}}$: On input $\mathsf{pp}$ and secret states $\mathsf{st}_i$ of $k$ many servers indexed by $Q$ and password $\mathsf{pw} \in \mathbf{D}$, $\mathbf{U}$ and $\{\mathbf{S}_i\}_{i\in Q}$ perform the following steps

- $\mathsf{Server}$ 1: Server $\mathbf{S}_j$ (for each $j \in Q$) performs the following steps:
  1. Randomly choose $R_j$ and run $\texttt{TFHE.Enc}$ to encrypt $R_j$ w.r.t $\mathtt{pk}^{\mathtt{tfhe}}$:
     • Compute $\mathbf{C}_{R_j} = \texttt{TFHE.Enc}_{\mathtt{pk}^{\mathtt{tfhe}}}(R_j)$.
  2. Compute $\mathsf{Sig}_j(\mathbf{C}_{R_j})$.
  3. Send $\mathbf{C}_{R_j}$ and $\mathsf{Sig}_j(\mathbf{C}_{R_j})$ to the user $\mathbf{U}$.
- $\mathsf{User}$ 1: User $\mathbf{U}$ does the following:
  1. Check $\{\mathsf{Ver}_i(\mathsf{Sig}_i(\mathbf{C}_{\tilde{R}_i})) = \text{accept or reject}\}_{i\in Q}$.
     • If $\mathsf{Ver}_i(\mathsf{Sig}_i(\mathbf{C}_{\tilde{R}_i})) \neq \textit{accept}$ for any $i \in Q$, send $\perp$ to $\{\mathbf{S}_j\}_{j\in Q}$.
  2. Otherwise,
     (a) Randomly choose $X$ and Run $\texttt{TFHE.Enc}$
        • to encrypt $\widetilde{\mathsf{pw}}$ w.r.t $\mathtt{pk}^{\mathtt{tfhe}}$ and $\widehat{\mathtt{pk}^{\mathtt{tfhe}}}$ with the same randomness $\mathbf{R}_{\widetilde{\mathsf{pw}}}$ and obtain $\mathbf{C}_{\widetilde{\mathsf{pw}}}, \widehat{\mathbf{C}}_{\widetilde{\mathsf{pw}}}$
        • encrypt $X$ with respect to $\mathtt{pk}^{\mathtt{tfhe}}$ and obtain $\mathbf{C}_X$.
     (b) Compute $\{\pi_j \leftarrow \mathcal{P}[\mathcal{L}_{\mathbf{U}}^{\mathsf{pp}}]((\mathbf{C}_{\widetilde{\mathsf{pw}}}, \widehat{\mathbf{C}}_{\widetilde{\mathsf{pw}}}), (\widetilde{\mathsf{pw}}, \mathbf{R}_{\widetilde{\mathsf{pw}}}))\}_{j\in Q}$.
     (c) Send $(\mathbf{C}_{\widetilde{\mathsf{pw}}}, \widehat{\mathbf{C}}_{\widetilde{\mathsf{pw}}}, \mathbf{C}_X, \pi_j, \{\mathbf{C}_{\tilde{R}_j}, \mathsf{Sig}_j(\mathbf{C}_{\tilde{R}_j})\}_{j\in Q})$ to $\{\mathbf{S}_j\}_{j\in Q}$.
- $\mathsf{Server}$ 2: Server $\mathbf{S}_j$ (for all $j \in Q$) proceeds as follows:
  1. Check $\{\mathsf{Ver}_i(\mathsf{Sig}_i(\mathbf{C}_{\tilde{R}_i})) = \text{accept or reject}\}_{i\in Q}$.
     • If $\mathsf{Ver}_i(\mathsf{Sig}_i(\mathbf{C}_{\tilde{R}_i})) \neq \textit{accept}$ for any $i \in Q$, send $\perp$ to user $\mathbf{U}$.
  2. If $\mathcal{V}[\mathcal{L}_{\mathbf{U}}^{\mathsf{pp}}]((\mathbf{C}_{\widetilde{\mathsf{pw}}}, \widehat{\mathbf{C}}_{\widetilde{\mathsf{pw}}}), (\widetilde{\mathsf{pw}}, \mathbf{R}_{\widetilde{\mathsf{pw}}})) = \text{reject}$, send $\perp$ to user $\mathbf{U}$.
  3. Otherwise,
     • Using $\texttt{TFHE.Eval}(\mathtt{pk}^{\mathtt{tfhe}}, C, \{\mathbf{C}_{\tilde{R}_i}\}_{i\neq j}, \mathbf{C}_{R_j}, \mathbf{C}_{\widetilde{\mathsf{pw}}}, \mathbf{C}_{\mathsf{pw}}, \mathbf{C}_{sk^{cca}}, \mathbf{C}_X)$ compute $\mathbf{CF}_j = \left(\sum_{i\in Q\setminus\{j\}} \mathbf{C}_{\tilde{R}_i} + \mathbf{C}_{R_j}\right) \cdot \mathbf{C}_{\widetilde{\mathsf{pw}}} - \left(\sum_{i\in Q\setminus\{j\}} \mathbf{C}_{\tilde{R}_i} + \mathbf{C}_{R_j}\right) \cdot \mathbf{C}_{\mathsf{pw}} + (\mathbf{C}_{\mathsf{sk}^{\mathsf{cca}}} + \mathbf{C}_X)$.
     • Run $\texttt{TFHE.PartDec}(\mathtt{pk}^{\mathtt{tfhe}}, \mathbf{CF}_j, \mathsf{sk}_j^{\mathtt{tfhe}})$ to output $\mathbf{W}_j$.
     • compute $\mathsf{Sig}_j(\mathbf{W}_j)$
     • choose random $\mathsf{key}_j$; compute $\sigma_j \leftarrow \mathsf{Sig}_j(\mathsf{key}_j)$ and compute $c_j \leftarrow \mathtt{Enc}^{\mathtt{cca}}(\mathsf{key}_j, \sigma_j)$
  4. Send $\left(\mathbf{W}_j, \mathsf{Sig}_j(\mathbf{W}_j), c_j\right)$ to the user $\mathbf{U}$.
- $\mathsf{User}$ 2: User $\mathbf{U}$ performs the following steps:
  1. If receives $\perp$ from any server $\mathbf{S}_i$ (for $i \in Q$) then outputs $\perp$.
  2. Checks $\{\mathsf{Ver}_i(\mathsf{Sig}_i(W_i)) = \textit{accept or reject}\}_{i\in Q}$.
     • If any $\mathsf{Ver}_i(\mathsf{Sig}_i(W_i)) \neq \textit{accept}$ for $i \in Q$, outputs $\perp$.
  3. Otherwise,
     • $\mathtt{sk}^{\mathtt{cca}}||\sigma_u + X \leftarrow \texttt{TFHE.FinDec}(\mathtt{pk}^{\mathtt{tfhe}}, \{\mathbf{W_j}\}_{j\in Q})$.
     • compute $\mathtt{sk}^{\mathtt{cca}}||\sigma_u = (\mathtt{sk}^{\mathtt{cca}}||\sigma_u + X) - X$.
     • if $\mathsf{Ver}_{\mathbf{U}}(\sigma_u) = \text{reject}$ then output $\perp$; otherwise compute $(\mathsf{key}_j, \sigma_j) \leftarrow \mathtt{Dec}_{\mathtt{sk}^{\mathtt{cca}}}^{\mathtt{cca}}(c_j)$
     • if $\mathsf{Ver}_j(\sigma_j) = \textit{reject}$ for any $j$ output $\perp$; otherwise, output $\mathsf{key}_j$.

**Fig. 2.** $(k, N)$-$\texttt{TPAKE.Login}$ algorithm

and does not get aborted by the user) the adversary $\mathcal{A}$ will obtain the keys of the corrupted servers viz. $\mathsf{key}_1, \ldots, \mathsf{key}_{k-1}$. Thus, the real view $\{real_{\mathcal{A}}(\mathbf{D}, \mathsf{aux})\}$ always include $\{\mathsf{key}_1, \ldots, \mathsf{key}_{k-1}\}$. Also, note that the secret states $\mathsf{st}_1, \ldots, \mathsf{st}_{k-1}$ are under adversarial capture.

We note that it is enough to prove that for every PPT $\mathcal{A}'$,

$$\{\mathsf{pw}, \mathsf{st}_1, \ldots, \mathsf{st}_{k-1}, \mathsf{st}_k, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathsf{key}_k, \text{output}(\mathcal{A}'^{\{\mathbf{S}_i(\mathsf{st}_i)\}_{1\le i\le k}}(\mathsf{aux}))\} \approx$$
$$\{\mathsf{pw}, \mathsf{st}_1, \ldots, \mathsf{st}_{k-1}, \mathsf{st}_k, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathcal{U}, \text{output}(\mathcal{A}'^{\{\mathbf{S}_i(\mathsf{st}_i)_{1\le i\le k-1}, \mathbf{S}_k(\widetilde{\mathsf{st}_k})\}}(\mathsf{aux}))\}$$

The reason for this is that once we have the above then we can define the non-interactive $\mathcal{A}''$ as follows. It chooses random state $\widetilde{\mathsf{st}_k}$. Then $\mathcal{A}''$ can emulate an execution of $\mathcal{A}'^{\{\mathbf{S}_i(\mathsf{st}_i)_{1\le i\le k-1}, \mathbf{S}_k(\widetilde{\mathsf{st}_k})\}}(\mathsf{aux})$ by playing the role of server $\mathbf{S}_k$ as it selects $\widetilde{\mathsf{st}_k}$. Lastly, $\mathcal{A}''$ outputs whatever $\mathcal{A}'$ outputs. Therefore, the resulting output is distributed exactly like $\text{output}(\mathcal{A}'^{\{\mathbf{S}_i(\mathsf{st}_i)_{1\le i\le k-1}, \mathbf{S}_k(\widetilde{\mathsf{st}_k})\}}(\mathsf{aux}))$.

Now since the distributions

$$\{\mathsf{pw}, \mathsf{st}_1, \ldots, \mathsf{st}_{k-1}, \mathsf{st}_k, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathcal{U}, \text{output}(\mathcal{A}'^{\{\mathbf{S}_i(\mathsf{st}_i)_{1\le i\le k-1}, \mathbf{S}_k(\widetilde{\mathsf{st}_k})\}}(\mathsf{aux}))\}$$

and

$$\{\widetilde{\mathsf{pw}}, \mathsf{st}_1, \ldots, \mathsf{st}_{k-1}, \widetilde{\mathsf{st}_k}, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathcal{U}, \text{output}(\mathcal{A}'^{\{\mathbf{S}_i(\mathsf{st}_i)\}_{1\le i\le k}}(\mathsf{aux}))\}$$

are equivalent we only show that

$$\{\mathsf{pw}, \mathsf{st}_1, \ldots, \mathsf{st}_{k-1}, \mathsf{st}_k, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathsf{key}_k, \text{output}(\mathcal{A}'^{\{\mathbf{S}_i(\mathsf{st}_i)\}_{1\le i\le k}}(\mathsf{aux}))\} \approx$$
$$\{\widetilde{\mathsf{pw}}, \mathsf{st}_1, \ldots, \mathsf{st}_{k-1}, \widetilde{\mathsf{st}_k}, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathcal{U}, \text{output}(\mathcal{A}'^{\{\mathbf{S}_i(\mathsf{st}_i)\}_{1\le i\le k}}(\mathsf{aux}))\}.$$

Note that $\mathsf{key}_k$ is chosen uniformly randomly from the key space and thus is identically distributed as uniform $\mathcal{U}$. Thus, the distribution $\{\mathsf{pw}, \mathsf{st}_1, \ldots, \mathsf{st}_{k-1}, \mathsf{st}_k, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathsf{key}_k\}$ is $1 - \epsilon_{Sig_k}$ indistinguishable from $\{\widetilde{\mathsf{pw}}, \mathsf{st}_1, \ldots, \mathsf{st}_{k-1}, \widetilde{\mathsf{st}_k}, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathsf{key}_k\}$ due to the perfect privacy of Shamir secret sharing and EUF-CMA security of signature scheme.

This completes the proof.

**Lemma 2.** *For every PPT adversary $\mathcal{A}$ interacting with user $\mathbf{U}$ and servers $\mathbf{S}_1, \ldots, \mathbf{S}_{k-1}, \mathbf{S}_k$ there exists an $\mathcal{A}'$ interacting with only $\mathbf{S}_1, \ldots, \mathbf{S}_{k-1}, \mathbf{S}_k$, such that*

$$\{\mathsf{pw}, \mathsf{st}_1, \ldots, \mathsf{st}_{k-1}, \mathsf{st}_k, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathsf{key}_k, \textit{output}(\mathcal{A}'^{\{\mathbf{S}_i(\mathsf{st}_i)\}_{1\le i\le k}}(\mathsf{aux}))\} \approx$$
$$\{\mathsf{pw}, \mathsf{st}_1, \ldots, \mathsf{st}_{k-1}, \mathsf{st}_k, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathsf{key}_k, \textit{output}(\mathcal{A}^{\mathbf{U}(\mathsf{pw}), \{\mathbf{S}_i(\mathsf{st}_i)\}_{1\le i\le k}}(\mathsf{aux}))\}$$

*Proof.* As before, let $\mathcal{A}$ corrupt (without loss of generality) the first $k-1$ servers $\mathbf{S}_1, \ldots, \mathbf{S}_{k-1}$ and obtain their private states $\mathsf{st}_1, \ldots, \mathsf{st}_{k-1}$ in the beginning of

the Login protocol. Also, suppose the user $\mathbf{U}$ interacts with $\mathbf{S}_1, \ldots, \mathbf{S}_{k-1}$ and $\mathbf{S}_k$ during the Login phase.

We note that the keys are chosen uniformly randomly by the servers (in particular, honest $\mathbf{S}_k$ chooses $\mathsf{key}_k$ uniformly) and the user $\mathbf{U}$ ultimately outputs accept (or reject). This accept/reject bit can be easily simulated and noting that at the end of the protocol, the probability $\mathbf{U}$ accepts and yet the keys $\mathsf{key}_k$ (chosen by $\mathbf{S}_k$) and $\mathsf{key}'_k$ (obtained by $\mathbf{U}$) are different is at most $\epsilon_{Sig_{\mathbf{U}}} + \epsilon_{\mathsf{cca}}$.

Thus, what we need to show is that barring the step when $\mathbf{U}$ accepts or rejects; the entire interaction of adversary $\mathcal{A}^{\mathbf{U}(\mathsf{pw}),\{\mathbf{S}_i(\mathsf{st}_i)\}_{1\leq i\leq k}}(\mathsf{aux})$ can be simulated by an adversary $\mathcal{A}'$ who interacts only with $\{\mathbf{S}_i(\mathsf{st}_i)\}_{1\leq i\leq k}$. We first note that in the User 1 step, the user $\mathbf{U}$ uses a simulation-sound NIZK to prove his statement and all the rest of the sent values are encrypted values. Therefore, even if $\mathbf{U}$ uses some fixed $\mathsf{pw}' \in \mathbf{D}$ instead of $\mathsf{pw}$, the adversary $\mathcal{A}^{\mathbf{U}(\mathsf{pw}),\{\mathbf{S}_i(\mathsf{st}_i)\}_{1\leq i\leq k}}$ cannot distinguish between

$\{\mathsf{pw}, \mathsf{st}_1, \ldots, \mathsf{st}_{k-1}, \mathsf{st}_k, \text{output}(\mathcal{A}^{\mathbf{U}(\mathsf{pw}),\{\mathbf{S}_i(\mathsf{st}_i)\}_{1\leq i\leq k}}(\mathsf{aux}))\}$ and

$\{\mathsf{pw}, \mathsf{st}_1, \ldots, \mathsf{st}_{k-1}, \mathsf{st}_k, \text{output}(\mathcal{A}^{\mathbf{U}(\mathsf{pw}'),\{\mathbf{S}_i(\mathsf{st}_i)\}_{1\leq i\leq k}}(\mathsf{aux}))\}$.

Now, we can define our intended adversary $\mathcal{A}'$ as follows – it chooses a fixed password $\mathsf{pw}'$ and emulates $\mathcal{A}^{\mathbf{U}(\mathsf{pw}'),\{\mathbf{S}_i(\mathsf{st}_i)\}_{1\leq i\leq k}}$, while interacting with the servers $\{\mathbf{S}_i(\mathsf{st}_i)\}_{1\leq i\leq k}$ and using $\mathsf{pw}'$ while emulating $\mathbf{U}(\mathsf{pw})$. Ultimately, it outputs whatever $\mathcal{A}$ outputs.

Now the proof follows from the observation that we made in the beginning of the proof of Lemma 1 and thereby including the output keys.

**Theorem 1.** *For every PPT real-model adversary $\mathcal{A}$ there exists a PPT ideal-model adversary $\mathcal{A}^*$, such that $\{ideal_{\mathcal{A}^*}(\mathbf{D}, \mathsf{aux})\} \approx_{O(1/|\mathbf{D}|)+\epsilon} \{real_{\mathcal{A}}(\mathbf{D}, \mathsf{aux})\}$.*

*Proof.* Let the real model adversary $\mathcal{A}$ corrupts (without loss of generality) the first $k-1$ servers $\mathbf{S}_1, \ldots, \mathbf{S}_{k-1}$ and obtains their private states $\mathsf{st}_1, \ldots, \mathsf{st}_{k-1}$ in the beginning of the Login protocol. Also, suppose the user $\mathbf{U}$ interacts with $\mathbf{S}_1, \ldots, \mathbf{S}_{k-1}$ and $\mathbf{S}_k$ during the Login phase for the sake of simplicity, considering $k$ many servers are needed for the login to happen. Therefore, after the Login phase (if it succeeds and does not get aborted by the user) the adversary $\mathcal{A}$ will obtain the keys of the corrupted servers viz. $\mathsf{key}_1, \ldots, \mathsf{key}_{k-1}$. Thus, the real view $\{real_{\mathcal{A}}(\mathbf{D}, \mathsf{aux})\}$ always include $\{\mathsf{key}_1, \ldots, \mathsf{key}_{k-1}\}$.

Suppose for every PPT adversary $\mathcal{A}$, there exists a non-interactive machine $\mathcal{A}''$ such that

$$\{\mathsf{pw}, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathcal{U}, \text{output}(\mathcal{A}''(\mathsf{aux}))\} \approx_{O(1/|\mathbf{D}|)}$$
$$\{\mathsf{pw}, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathsf{key}_k, \text{output}(\mathcal{A}^{\mathbf{U}(\mathsf{pw}),\{\mathbf{S}_i(\mathsf{st}_i)\}_{1\leq i\leq k}}(\mathsf{aux}))\}.$$

We now start by describing the ideal model adversary $\mathcal{A}^*$. It first invokes the non-interactive machine $\mathcal{A}''$ and receives the output of $\mathcal{A}''$ (which contains $\mathcal{A}$'s view and in particular $\mathbf{U}$'s accept/reject bit), $\mathcal{A}^*$ sets the value of $b$ (the bit sent by it to the trusted party) as follows:

• If $\mathbf{U}$ accepted in the view output by $\mathcal{A}''$ , then $\mathcal{A}^*$ sends $b = 1$ to the trusted

party.
• If $\mathbf{U}$ rejected in the view output by $\mathcal{A}''$ , then $\mathcal{A}^*$ sends $b = 0$ to the trusted party.

Note that upon receiving $b = 1$, the trusted party hands the same uniformly distributed key to $\mathbf{S}_k$ and $\mathbf{U}$. On the other hand, upon receiving $b = 0$, the trusted party hands a uniformly distributed key to $\mathbf{S}_k$, and $\mathbf{U}$ receives $\perp$. Also, in each case, $\mathsf{key}_1, \ldots, \mathsf{key}_{k-1}$ are chosen uniformly by the trusted party and is handed over to $\mathbf{U}$ and $\mathbf{S}_1, \ldots, \mathbf{S}_{k-1}$. Finally, $\mathcal{A}^*$ halts and outputs the output of $\mathcal{A}''$.

Therefore, from the definition of $\mathcal{A}^*$, it follows that

$$\{\mathsf{pw}, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathcal{U}, \text{output}(\mathcal{A}^*)\} \approx$$

$$\{\mathsf{pw}, \mathsf{key}_1, \ldots, \mathsf{key}_{k-1}, \mathsf{key}_k, \text{output}(\mathcal{A}^{\mathbf{U}(\mathsf{pw}), \{\mathbf{S}_i(\mathsf{st}_i)\}_{1 \le i \le k}}(\mathsf{aux}))\} \cdot \cdots \cdot (**)$$

The distribution on the left is "almost" the ideal-model distribution and the right hand side distribution is "almost" the real-model distribution, where in both cases the only thing missing is user $\mathbf{U}$'s local output (which may or may not be equal to $\mathbf{S}_k$'s local output). We will now show that in case $\mathbf{U}$ rejects and this output is included into the distributions then a PPT distinguisher is able to tell the difference between ideal and real distributions is negligible. Later, we will show the same in case $\mathbf{U}$ accepts.

Suppose $D$ is any PPT distinguisher attempting to distinguish between the IDEAL and REAL distributions. When referring to $\mathbf{U}$'s decision (*i.e.* `accept` or `reject`), within the context of $ideal_{\mathcal{A}^*}$ , we mean $\mathbf{U}$'s decision as included in the emulated view of $\mathcal{A}$ (which is part of the output of $\mathcal{A}^*$, by definition of $\mathcal{A}^*$ ). More precisely, by the construction of $\mathcal{A}^*$, it holds that $\mathbf{U}$'s decision in the emulated view matches the output of $\mathbf{U}$ in the ideal-model; *i.e.* decision$\mathbf{U}$ = `reject` if and only if the output of $\mathbf{U}$ in the ideal-model is $\perp$. Thus,

$$|Pr[D(ideal_{\mathcal{A}^*}(\mathbf{D}, \mathsf{aux})) = 1 \ \& \ \mathbf{U} \ \texttt{rejects}] - Pr[D(real_{\mathcal{A}}(\mathbf{D}, \mathsf{aux})) = 1 \ \& \ \mathbf{U} \ \texttt{rejects}]|$$

$$= |Pr[D(\mathsf{pw}, \{\mathsf{key}_i\}_{i \in [k-1]}, \mathcal{U}, \perp, \text{output}(\mathcal{A}''(\mathsf{aux}))) \ \& \ \mathbf{U} \ \texttt{rejects}]$$

$$- Pr[D(\mathsf{pw}, \{\mathsf{key}_i\}_{i \in [k-1]}, \mathsf{key}_k, \perp, \text{output}(\mathcal{A}^{\mathbf{U}(\mathsf{pw}), \{\mathbf{S}_i(\mathsf{st}_i)\}_{1 \le i \le k}}(\mathsf{aux})) \ \& \ \mathbf{U} \ \texttt{rejects}]|.$$

We obtain the above from the protocol definition that states that when $\mathbf{U}$ rejects it outputs $\perp$, and from the construction of the ideal-model adversary $\mathcal{A}^*$ who sends $b = 0$ to the trusted party (causing $\mathbf{U}$'s output to be $\perp$) in the case that $\mathbf{U}$ rejects in the view output by $\mathcal{A}''$. Noting that $\mathcal{A}$'s view includes $\mathbf{U}$'s accept/reject decision bit (and thus implicitly $\mathbf{U}$'s output of $\perp$ in the case that $\mathbf{U}$ rejects). Combining the above with Eqn.$(**)$ (from the last page) we obtain,

$$|Pr[D(ideal_{\mathcal{A}^*}(\mathbf{D}, \mathsf{aux})) = 1 \ \& \ \mathbf{U} \ \texttt{rejects}]$$

$$- Pr[D(real_{\mathcal{A}}(\mathbf{D}, \mathsf{aux})) = 1 \ \& \ \mathbf{U} \ \texttt{rejects}]| \le O(1/|\mathbf{D}|) + \mathsf{negl}(\lambda).$$

Now we turn our attention to the case when $\mathbf{U}$ accepts. We will show that in case the adversary behaves maliciously during the protocol execution the distinguisher $D$ still cannot distinguish between ideal execution and real execution.

Note that, in the ideal execution, $Pr[\mathbf{U}\ \texttt{accepts}\ \&\ \mathcal{A}\ \textit{is malicious}] = 0$. This is due to the fact that in the simulated transcript output by $A''$, $\mathbf{U}$ accepts if and only if $\mathcal{A}$ acts passively. In a real execution, $\mathcal{A}$ acts maliciously and still $\mathbf{U}$ accepts is possible if $\mathcal{A}$ can

- break the $\mathsf{EUF\text{-}CMA}$ security of signature algorithm $(\mathsf{Sig}_i, \mathsf{Ver}_i)$ for any honest server or $(\mathsf{Sig}_{\mathbf{U}}, \mathsf{Ver}_{\mathbf{U}})$ for the user in $\texttt{TPAKE.Login}$ phase;
- break the $\mathsf{CPA}$ security of $\mathsf{TFHE}$ in $\texttt{TPAKE.Reg}$ or $\texttt{TPAKE.Login}$ phase;
- break $\mathsf{CCA}$ security of $(\mathtt{pk}^{\mathtt{cca}}, \mathtt{sk}^{\mathtt{cca}})$ in the round $\mathsf{Server}\ 2$;
- deceive a honest server during verification of NIZK in the round $\mathsf{Server}\ 2$.

Hence, in a real execution, ignoring constant multipliers and considering union bound, $\mathcal{A}$ acts maliciously and $\mathbf{U}$ accepts is possible only with probability at most $(O(\frac{1}{|\mathbf{D}|}) + \epsilon_{\mathsf{TFHE}} + \epsilon_{\mathsf{Sig}} + \epsilon_{\mathsf{SS}} + \epsilon_{\mathsf{Sig}_{\mathbf{U}}} + \epsilon_{cca})$. ■

**Lattice-based Instantiation in the Standard Model.** We instantiate using the CCA secure lattice based encryption scheme $(\mathsf{Enc}^{cca}, \mathsf{Dec}^{cca})$ [21] (based on the hardness of $\mathsf{dLWE}$ problem)**;** Adaptive $\mathsf{EUF\text{-}CMA}$ Signature $(\mathsf{Sig}, \mathsf{Ver})$ of [3]**;** the $k$-out-of-$N$ $\mathsf{TFHE}$ scheme [9] based on Shamir secret sharing with parameters $B$ (noise bound of $\mathsf{FHE}$) and $B_{sm}$ satisfying the conditions $B+(N!)^3 \cdot N \cdot B_{sm} \leq \frac{q}{4}$ and $B/B_{sm} = \mathsf{negl}(\lambda)$**;** and a Lattice-based $\mathsf{SS\text{-}NIZK}$. Peikert et al. [22, Theorem 5.4] proposed non-interactive zero-knowledge proof system for any NP language based on the hardness of $\mathsf{dLWE}$. Generic conversion proposed by Sahai [25] transform any ordinary non-interactive zero-knowledge proof system into $\mathsf{SS\text{-}NIZK}$. Thus, we have $\mathsf{SS\text{-}NIZK}$ based on the hardness of $\mathsf{dLWE}$ for the language $\mathcal{L}_{\mathbf{U}}^{\mathsf{pp}}$ corresponding to the one protocol message from User to Servers parameterized by public parameters $\mathsf{pp} \leftarrow \{\mathtt{pk}^{\mathtt{tfhe}}, \widehat{\mathtt{pk}^{\mathtt{tfhe}}}, \mathbf{C}_{\mathsf{pw}}, \mathbf{C}_{\mathtt{sk}^{\mathtt{cca}}}, \{\mathsf{Ver}_i\}_{i=1}^{N}\}$, where $\mathcal{L}_{\mathbf{U}}^{\mathsf{pp}} = \{(\mathbf{C}_{\widetilde{\mathsf{pw}}}, \widehat{\mathbf{C}}_{\widetilde{\mathsf{pw}}}) \mid \exists\ (\widetilde{\mathsf{pw}}, \mathbf{R}_{\widetilde{\mathsf{pw}}})$ where $\mathbf{C}_{\widetilde{\mathsf{pw}}} = \texttt{TFHE.Encrypt}(\mathtt{pk}^{\mathtt{tfhe}}, \widetilde{\mathsf{pw}}, \mathbf{R}_{\widetilde{\mathsf{pw}}})$ & $\bar{\mathbf{C}}_{\widetilde{\mathsf{pw}}} = \texttt{TFHE.Encrypt}(\widehat{\mathtt{pk}^{\mathtt{tfhe}}}, \widetilde{\mathsf{pw}}, \mathbf{R}_{\widetilde{\mathsf{pw}}})\}$.

## 4 Password Protected Universal Thresholdizer

We now propose a new primitive which makes the universal thresholdizer [9] password protected. A user can execute such a protocol if it inputs the password with which it has registered with the system. We begin with defining the primitive.

**Definition 17 (Password Protected Universal Thresholdizer).** *Let $\mathcal{S} = \{\mathbf{S}_1, \cdots, \mathbf{S}_N\}$ be a set of parties/servers, $\mathbf{U}$ be the user and let $\mathbb{S}$ be a class of efficient access structures on $\mathcal{S}$. A password protected universal thresholdizer scheme for $\mathsf{User}$ and $\mathbb{S}$ is a tuple of PPT algorithms $\mathsf{PPUT} = (\texttt{PPUT.Setup}, \texttt{PPUT.Eval}, \texttt{PPUT.Verify}, \texttt{PPUT.Combine})$ with the following properties:*

- $\texttt{PPUT.Setup}(1^\lambda, 1^d, \mathbb{A}, \mathsf{pw}, x) \rightarrow (\mathsf{pp}, S_1, \cdots, S_N)$ : *On input the security parameter $\lambda$, a depth bound $d$, an access structure $\mathbb{A} \in \mathbb{S}$, a password $\mathsf{pw} \in_R \mathbf{D}$ and a message $x \in \{0,1\}^k$, the setup algorithm outputs the public parameters $\mathsf{pp}$ and a set of shares $S_1, \cdots, S_N$ to $\mathbf{S}_1, \cdots, \mathbf{S}_N$.*

- $\texttt{PPUT.Eval}(\mathsf{pp}, S_i, \mathsf{pw}, C) \to y_i$ : *On input the public parameters* $\mathsf{pp}$*, a share* $s_i$*, and a circuit* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most* $d$ *and a password* $\mathsf{pw}$*, the evaluation algorithm outputs a partial evaluation* $y_i$ *to* $\mathbf{S}_i$*.*
- $\texttt{PPUT.Verify}(\mathsf{pp}, \mathsf{pw}, y_i, C) \to \{0,1\}$ : *On input the public parameters* $\mathsf{pp}$*, a password* $\mathsf{pw}$*, a partial evaluation* $y_i$*, and a circuit* $C : \{0,1\}^k \to \{0,1\}$*, the verification algorithm accepts or rejects.*
- $\texttt{PPUT.Combine}(\mathsf{pp}, \mathsf{pw}, B) \to y$ : *On input the public parameters* $\mathsf{pp}$*, a password* $\mathsf{pw}$*, a set of partial evaluations* $B = \{y_i\}_{i \in Q}$ *where* $Q \in \mathbb{A}$*, the combining algorithm outputs the final evaluation* $y$ *to the user* $\mathbf{U}$*.*

A PPUT is required to satisfy the following properties.

**Definition 18 (Compactness).** *We say that a* PPUT *scheme is compact if there exists a polynomial* $poly(\cdot)$ *such that for all* $\lambda$*, depth bound* $d$*, circuit* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most* $d$*,* $\mu \in \{0,1\}$ *and password* $\mathsf{pw}$*, the following holds. For* $(\mathsf{pp}, s_1, \cdots, s_N) \leftarrow \texttt{PPUT.Setup}(1^\lambda, 1^d, \mathbb{A}, \mathsf{pw}, x)$ *and* $y_i \leftarrow \texttt{PPUT.Eval}(\mathsf{pp}, S_i, \mathsf{pw}, C)$ *for any* $i \in [N]$*, we have* $|y_i| \leq poly(\lambda, d, N)$*.*

**Definition 19 (Evaluation Correctness).** *We say that a* PPUT *scheme satisfies evaluation correctness if for all* $\lambda$*, depth bound* $d$*, password* $\mathsf{pw}$*, access structure* $\mathbb{A}$*, message* $x \in \{0,1\}^k$*, circuit* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most* $d$*, and* $Q \in \mathbb{A}$*, the following condition holds.*
*For* $(\mathsf{pp}, s_1, \cdots, s_N) \leftarrow \texttt{PPUT.Setup}(1^\lambda, 1^d, \mathbb{A}, \mathsf{pw}, x)$*,*
$\Pr[\texttt{PPUT.Combine}(\mathsf{pp}, \{\texttt{PPUT.Eval}(\mathsf{pp}, s_i, \mathsf{pw}, C)\}_{i \in Q}) = C(x)] = 1 - \mathsf{negl}(\lambda)$.

**Definition 20 (Verification Correctness).** *We say that a* PPUT *scheme satisfies verification correctness if for all* $\lambda$*, depth bound* $d$*, password* $\mathsf{pw}$*, access structure* $\mathbb{A}$*, message* $x \in \{0,1\}^k$*, and circuit* $C : \{0,1\}^k \to \{0,1\}$ *of depth at most* $d$*, the following holds. For* $(\mathsf{pp}, s_1, \cdots, s_N) \leftarrow \texttt{PPUT.Setup}(1^\lambda, 1^d, \mathbb{A}, \mathsf{pw}, x)$ *and* $y_i \leftarrow \texttt{PPUT.Eval}(\mathsf{pp}, S_i, \mathsf{pw}, C)$ *for any* $i \in [N]$*, we have that*

$$\Pr[\texttt{PPUT.Verify}(\mathsf{pp}, \mathsf{pw}, y_i, C) = 1] = 1.$$

**Security of** PPUT: Our definition for the task of PPUT is based on the simulation paradigm. That is, we require that a secure protocol emulates an ideal execution of a PPUT protocol. In such an ideal execution, a trusted party hands partial circuit evaluations to the servers and the combination of evaluations to the user. The only power given to the adversary in this ideal model is to prevent the trusted party from handing outputs to one of the honest parties and the user. We emphasize that in this ideal model of execution, the adversary learns nothing about the password, partial circuit evaluations of honest participants, and combination of partial evaluation for the honest user. However, if the user or a qualified set of participants is corrupted, then the adversary is given the power to fully determine all the evaluations and combination. The rationale for this is that the aim of PPUT is to enable honest parties to evaluate the circuit partially and their combination by the honest user, which are unknown to an external adversary. If the user or a qualified set of participants is corrupted, then

the adversary will learn the combination of partial evaluations (because it is one of the legitimate participants), and so the security requirement is meaningless. Furthermore, $\mathcal{A}$ controls the communication line between the (honest) user and the (honest) participants. Thus, it can block all communication between the user and participants, and cause any protocol to fail. This (unavoidable) adversarial capability is modeled in the functionality by letting $\mathcal{A}$ input a single bit $b$ indicating whether or not the execution is to be successful. Specifically, if $b = 1$ (i.e., success) then the user and participating participants receive the desire outputs. On the other hand, if $b = 0$ then participants receive a uniformly random string, whereas the user receives a special abort symbol $\perp$ instead. *It is worth mentioning that* PPUT *is a single output functionality, and the participants always output some special symbol* $\lambda$. In conclusion, the problem of PPUT is cast as the following functionality:

$$(\{S_i\}_{i\in Q}, \mathsf{pw}, C, x, b) \rightarrow \begin{cases} ((C(x) = y), \lambda) & \text{if } b = 1, \\ & \mathbf{U} \ \& \ \{\mathbf{S}_i\}_{i\in Q} \textit{ are honest}; \\ (\perp, \lambda) & \text{if } b = 0, \\ & \mathbf{U} \ \& \ \{\mathbf{S}_i\}_{i\in Q} \textit{ are honest}; \\ (\perp, \lambda) & \text{if } b = 0 \text{ or } 1, \\ & \{\mathbf{S}_i\}_{i\in F}; \\ (y^*, \lambda) & \text{if } b = 1, \\ & \mathbf{U} \textit{ or } \{\mathbf{S}_i\}_{i\in Q} \textit{ are dishonest}; \end{cases}$$

where $Q \in \mathbb{A}$, $y^*$ is adversarially chosen output, and $F$ is a forbidden set of servers.

*The Ideal Model* : Let $\mathbf{U}$ and $\{\mathbf{S}_1, \ldots, \mathbf{S}_N\}$ denote the user and the set of servers respectively. Let $\mathcal{A}$ denote an PPT adversary in the ideal model (with arbitrary auxiliary input $\mathsf{aux}$). An ideal-model execution proceeds in the following phases:

- **Setup:** A password $\mathsf{pw} \in_R \mathbf{D}$ is uniformly chosen from the dictionary, $x$ and a circuit $C$ are given to the user $\mathbf{U}$. $S_1, S_2, \cdots, S_N$ are chosen and given to $\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_N$, respectively.
- **Sending inputs to trusted party:** If the user $\mathbf{U}$ is not dishonest, $\mathbf{U}$ sends the trusted party the password $\mathsf{pw}$, $x$ and the circuit $C$ it has received in the setup stage; otherwise, $\mathbf{U}$ sends $\widehat{\mathsf{pw}}$ and $\widehat{C}$ which are adversarially chosen. The adversary $\mathcal{A}$ sends either 1 (denoting a successful protocol execution) or 0 (denoting a failed protocol execution). Moreover, some participants may be captured by $\mathcal{A}$. The server $\mathbf{S}_i$ sends the trusted party $S_i$ he has received in the setup stage if $\mathbf{S}_i$ is not captured by $\mathcal{A}$; Otherwise, server $\mathbf{S}_i$ sends the trusted party $\widehat{S}_i$, which is adversarially chosen.
- **The trusted party answers all parties:** Without loss of generality, the trusted party only answers the user $\mathbf{U}$. In the case $\mathcal{A}$ sends 1 and $\{\mathbf{S_i}\}_{\mathbf{i}\in\mathbf{Q}}$ are not corrupted, the trusted party computes $C(x)$ and sends $y$ to $\mathbf{U}$. In the case $\mathcal{A}$ sends 0, the trusted party sends $\perp$ to $\mathbf{U}$. In the case only a forbidden set of servers is involved in the protocol, the trusted party sends $\perp$ to $\mathbf{U}$. In

the case $\mathcal{A}$ sends 1 and $\{\mathbf{S}_i\}_{i\in Q}$ are corrupted or the user $\mathbf{U}$ is corrupted, the trusted party sends $y^*$ to $\mathbf{U}$.

The ideal distribution is defined as follows:

$$ideal_{\mathcal{A}}(\mathbf{D}, \mathsf{aux}) = (\mathsf{pw}, output(\mathbf{U}(\mathsf{pw})), output(\mathcal{A}(\mathsf{aux}))).$$

*The Real Model* : As in the ideal model, the real model begins with a setup stage in which the user $\mathbf{U}$ receives uniformly distributed password $\mathsf{pw} \in \mathbf{D}$, and $\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_N$ receive $S_1, S_2, \cdots, S_N$, respectively. Then, the protocol is executed among $\mathbf{U}$ and $\{\mathbf{S}_i\}_{i\in Q}$ communicating via $\mathcal{A}$. The execution of this protocol is denoted by $\mathcal{A}^{\mathbf{U}(\mathsf{pw},C),\{\mathbf{S}_i(S_i)\}_{i\in Q}}(\mathsf{aux})$. The real distribution is defined as follows:

$$real_{\mathcal{A}}(\mathbf{D}, \mathsf{aux}) = (\mathsf{pw}, output(\mathbf{U}(\mathsf{pw})), output(\mathcal{A}^{\mathbf{U}(\mathsf{pw}),\{\mathbf{S}_i(S_i)\}_{i\in Q}}(\mathsf{aux})).$$

As in TPAKE, a PPUT protocol is said to be secure if the above-mentioned ideal-model emulation results in an output distribution that can be distinguished from a real execution by (a gap of) at most $O(1/|\mathbf{D}|) + \mathsf{negl}(\lambda)$.

**Definition 21 (Security).** *A* PPUT *is secure if the following two requirements hold:*

1. *Semi-honest adversaries: For every PPT real-model semi-honest adversary $\mathcal{A}$ there exists a PPT ideal-model adversary $\mathcal{A}^*$ such that for every dictionary $\mathbf{D}$ and every auxiliary input $\mathsf{aux} \in \{0,1\}^{\mathsf{poly}(\lambda)}$ it should hold that $\{\mathtt{ideal}_{\mathcal{A}^*}(\mathbf{D}, \mathsf{aux})\}_{\mathbf{D},\mathsf{aux}} \approx_c \{\mathtt{real}_{\mathcal{A}}(\mathbf{D}, \mathsf{aux})\}_{\mathbf{D},\mathsf{aux}}$*
2. *Malicious adversaries: For every PPT real-model adversary $\mathcal{A}$ there exists a PPT ideal-model adversary $\mathcal{A}^*$ such that for every dictionary $\mathbf{D}$ and every auxiliary input $\mathsf{aux} \in \{0,1\}^{\mathsf{poly}(\lambda)}$ it should hold that $\{\mathtt{ideal}_{\mathcal{A}^*}(\mathbf{D}, \mathsf{aux})\}_{\mathbf{D},\mathsf{aux}} \approx_{O(\frac{1}{\mathbf{D}})+\epsilon} \{\mathtt{real}_{\mathcal{A}}(\mathbf{D}, \mathsf{aux})\}_{\mathbf{D},\mathsf{aux}}$*

**Definition 22 (Robustness of PPUT).** *We say that a* PPUT *scheme satisfies robustness if for all $\lambda$, and depth bound $d$, the following holds. For any PPT adversary $\mathcal{A}$, the following experiment $\mathtt{Expt}_{\mathcal{A},PPUT,Robust}(1^\lambda, 1^d)$ outputs 1 with negligible probability:*

$\mathtt{Expt}_{\mathcal{A},PPUT,Robust}(1^\lambda, 1^d)$ :

1. *On input the security parameter $1^\lambda$, circuit depth $1^d$, the adversary $\mathcal{A}$ outputs $(k, N)$-threshold access structure, a message $x \in \{0,1\}^k$, and a password dictionary $\mathbf{D}$.*
2. *The challenger runs $(\mathsf{pp}, S_1, \cdots, S_N) \leftarrow \mathtt{PPUT.Setup}(1^\lambda, 1^d, (k, N), \mathbf{D}, x)$ and provides $(\mathsf{pp}, S_1, \cdots, S_N)$ and the password $\mathsf{pw} \in \mathbf{D}$ to $\mathcal{A}$.*
3. *$\mathcal{A}$ outputs a fake partial evaluation $y_i^*$.*
4. *The challenger returns 1 if $\mathtt{PPUT.Eval}(\mathsf{pp}, S_i, \mathsf{st}_i, \mathsf{pw}, C) \neq y_i^*$ and $\mathtt{PPUT.Verify}(\mathsf{pp}, \mathsf{pw}, y_i^*, C) = 1$.*

### 4.1 Construction of PPUT

In this section, we describe a generic construction of secure PPUT from TPAKE and UT. In Figure 3, we give the description of the construction. As building blocks, we use $(k, N)$-threshold UT, a $(k, n)$-TPAKE and a CPA secure symmetric encryption scheme. In Theorem 2 we prove the simulation security of our construction.

- $\texttt{PPUT.Setup}(1^\lambda, 1^d, (k, N), \mathbf{D}, x) \rightarrow (\mathsf{pp}, S_1, \cdots, S_N)$ :
  - User chooses password $\mathsf{pw} \leftarrow \mathbf{D}$.
  - Set $\mathsf{TPAKE}_{(k,N)} = (\texttt{TPAKE.Reg}\ ,\ \texttt{TPAKE.Login})$
  - Set $\mathsf{UT}_{(k,N)} = (\texttt{UT.Setup}, \texttt{UT.Eval}, \texttt{UT.Verify}, \texttt{UT.Combine})$,
  - A CPA secure symmetric-key encryption scheme $\mathcal{E} = (\mathsf{enc}, \mathsf{dec})$.
  - Execute $\texttt{UT.Setup}(1^\lambda, 1^d, (k, N), x)$ to obtain $(\mathsf{pp}', s_1, \cdots, s_N)$
  - Execute $\texttt{TPAKE.Reg}(\mathsf{pw})$ with $N$ servers $\mathbf{S}_1, \ldots, \mathbf{S}_N$ to obtain $(\mathsf{pp}'', \mathsf{st}_1, \mathsf{st}_2, \ldots, \mathsf{st}_N)$.
  - Store: $S_i = (s_i, \mathsf{st}_i)$ in $\mathbf{S}_i$ for all $1 \leq i \leq N$ and publish $\mathsf{pp} = \mathsf{pp}' \cup \mathsf{pp}''$.
- $\texttt{PPUT.Eval}(\mathsf{pp}, S_i, \mathsf{pw}, C) \rightarrow y_i$ :
  User contacts an available set $\{\mathbf{S}_{i_1}, \ldots, \mathbf{S}_{i_k}\}$ of servers indexed by $Q = \{i_1, \ldots, i_k\}$.
  - (User $\rightleftharpoons$ Servers)
    - Run $\texttt{TPAKE.Login}(\mathsf{pw})$ with servers indexed by $Q$
    - outputs: $(\mathsf{key}_{i_1}, \ldots, \mathsf{key}_{i_k}) \leftarrow \mathsf{User}(\mathsf{pw})$ and $\mathsf{key}_{i_j} \leftarrow \mathsf{Server}(\mathsf{st}_{i_j})$
    - Run: $\texttt{UT.Eval}(\mathsf{pp}, s_{i_j}, C)$ to output partial evaluation $Y_{i_j}$ for $i_j \in Q$ at the server side.
    - User receives $y_{i_j} \leftarrow \mathsf{enc}(\mathsf{key}_{i_j}, Y_{i_j})$ for $i_j \in Q$. // $\mathsf{key}_{i_j}$ is the key used for encryption.
- $\texttt{PPUT.Verify}(\mathsf{pp}, \mathsf{pw}, y_i, C) \rightarrow \{0, 1\}$ :
  - User runs $\mathsf{dec}(\mathsf{key}_j, y_{i_j})$ to obtain $Y_{i_j}$ for $i_j \in Q$
  - Run $\texttt{UT.Verify}$ to verify $\{Y_i\}_{i \in Q}$ and output 0 if any verification fails.
- $\texttt{PPUT.Combine}(\mathsf{pp}, \mathsf{pw}, \{y_i\}_{i \in Q}, Q) \rightarrow Y$ :
  - User obtains $Y_{i_j}$'s from $\mathsf{dec}(\mathsf{key}_{i_j}, y_{i_j})$.
  - User runs $\mathsf{UT.Combine}(\mathsf{pp}, Y_{i_j}, Q)$ to compute $Y$.
  - outputs $Y$.

**Fig. 3. Construction for Password Protected Universal Thresholdizer**

The compactness property (Definition 18), the evaluation correctness (Def. 19), verification correctness (Definition 20) and the robustness property (Definition 22) follow from the corresponding properties of the underlying UT. In the following, we prove the security.

**Security Analysis.**

**Theorem 2.** *Suppose* UT *and* TPAKE *satisfy simulation security. Then, the* PPUT *scheme from Fig. 3 satisfies security according to Definition 21.*

*Proof.* Our proof proceeds via a sequence of hybrid experiments between an adversary $\mathcal{A}$ and a challenger.

- $\mathtt{H}_0$ : This is the PPUT real security experiment $\mathtt{Expt}_{\mathcal{A},\mathsf{PPUT},Real}(1^\lambda, 1^d)$ between an adversary $\mathcal{A}$ and a challenger.
- $\mathtt{H}_1$ : For ease of the representation, we denote TPAKE simulator as Sim.TPAKE.Reg to simulate TPAKE.Reg and Sim.TPAKE.Login to simulate TPAKE.Login. $\mathtt{H}_1$ is same as $\mathtt{H}_0$, except that the challenger simulates TPAKE.Reg in PPUT.Setup. Specifically, on input $(k, N)$-threshold access structure, a random password $\bar{\mathsf{pw}}$ and a message $x \in \{0,1\}^k$ from $\mathcal{A}$, the challenger runs $\mathtt{PPUT.Setup}(1^\lambda, 1^d, (k, N), \mathbf{D}, x)$ by invoking $(\mathsf{st}_1, \mathsf{st}_2, \ldots, \mathsf{st}_N) \leftarrow \mathsf{Sim.TPAKE.Reg}(\bar{\mathsf{pw}})$, and $(\mathsf{pp}, s_1, \cdots, s_N) \leftarrow \mathtt{UT.Setup}(1^\lambda, 1^d, (k, N), x)$. By the simulation security of TPAKE, the hybrid experiments $\mathtt{H}_0$ and $\mathtt{H}_1$ are computationally indistinguishable.
- $\mathtt{H}_2$ : Same as $\mathtt{H}_1$, except that the challenger simulates TPAKE.Login in PPUT.Eval to compute $(\mathsf{key}^*_{i_1}, \ldots, \mathsf{key}^*_{i_j}) \leftarrow \mathsf{Sim.TPAKE.Login}(\bar{\mathsf{pw}})$. Challenger runs $\mathtt{UT.Eval}(\mathsf{pp}, s_{i_j}, C)$ in PPUT.Eval and computes $y^*_{i_j} \leftarrow \mathsf{enc}(\mathsf{key}^*_j, Y_{i_j})$ for $1 \leq j \leq |Q|$. Finally, the challenger runs PPUT.Verify and PPUT.Combine. By the simulation security of TPAKE, the hybrid experiments $\mathtt{H}_1$ and $\mathtt{H}_2$ are computationally indistinguishable.
- $\mathtt{H}_3$ : Same as $\mathtt{H}_2$, except that the challenger uses the zero string $\mathbf{0}$ instead of $x$ during setup. Specifically, on input a $(k, N)$-threshold access structure, and a message $x \in \{0,1\}^k$ from $\mathcal{A}$, the challenger ignores $x$ and runs setup as in $\mathtt{H}_2$ but with the zero string $\mathtt{UT.Setup}(1^\lambda, 1^d, (k, N), \mathbf{0})$. The challenger carries out the rest of the experiments as before. By the perfect security of the Shamir secret sharing scheme and the CPA security of encryption scheme, the hybrid experiments $\mathtt{H}_2$ and $\mathtt{H}_3$ are computationally indistinguishable. Hence, assuming the simulation security of TPAKE, the hybrid experiments $\mathtt{H}_3$ is computationally indistinguishable with $\mathtt{Expt}_{\mathcal{A},\mathsf{PPUT},Ideal}(1^\lambda, 1^d)$.

■

**Lattice-based Instantiation in the Standard Model.** We instantiate using the lattice-based TPAKE from section 3, lattice-based UT from [9], and lattice-based CPA secure symmetric-key encryption using a pseudorandom function (PRF) from [5].

## 5 Password Protected Threshold Signature: An Application

In a threshold signature scheme, the signing algorithm is delegated to a set of servers each holding a share such that for signing a message, each of the servers

creates a partial signature with its own share. A combining algorithm then combines the partial signatures to generate signature. Several important examples include thresholdizing RSA signatures [27], Schnorr signatures [**?**], (EC)DSA signatures [16,15,14] and BLS signatures [8]. We use password based authentication for verifying the correct user who with the knowledge of the correct password can execute the threshold signing process with an authorized set of servers. We first give a definition of a password protected threshold signature scheme followed by a construction.

**Definition 23 ($(k, N)$-Password Protected Threshold Signature).** *A $(k, N)$-*PPTS *is a tuple* $(\mathsf{Init}, \mathtt{PPTS.PartSign}, \mathtt{PPTS.PartSignVerify}, \mathtt{PPTS.Combine}, \mathtt{PPTS.Vrfy})$ *which involves user* $\mathbf{U}$ *and* $N$ *servers* $\mathbf{S}_1, \ldots, \mathbf{S}_N$ :

- $(\{\mathsf{sk}_i\}_{i=1}^N, \mathsf{vk}, \mathsf{pp}) \longleftarrow \mathsf{Init}(1^\lambda, (k, N), \mathbf{D})$ : *On input security parameter, number of servers $N$ and the dictionary of passwords $\mathbf{D}$ as inputs, it outputs secret states $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$ for the servers, a verification key* vk *and the public parameters* pp.
- $\Sigma_i/\perp \longleftarrow \mathtt{PPTS.PartSign}(\mathsf{pp}, \mathsf{pw}, \mathsf{msg}, \mathsf{sk}_i)$ : *It is an interactive protocol between user* $\mathbf{U}$ *and a subset of $k$ servers indexed by $Q \subset \{1, \ldots, N\}$ such that partial signature $\Sigma_i$ for the message* msg *is a function of the password* pw[6] *and is produced under the key* $\mathsf{sk}_i$.
- $1/0 \longleftarrow \mathtt{PPTS.PartSignVerify}(\mathsf{pp}, \mathsf{pw}, \mathsf{msg}, \Sigma_i)$*: On input a message* msg *and a partial signature $\Sigma_i$, the algorithm outputs either* 1 *or* 0.
- $\sigma/\perp \longleftarrow \mathtt{PPTS.Combine}(\mathsf{pp}, \mathsf{pw}, \{\Sigma_i\})_{i \in Q:|Q|=k})$*: On input the public parameters* pp *and the partial signatures $\{\Sigma_i\}_{i \in Q}$, the combining algorithm outputs a signature $\sigma$ of* msg.
- $1/0 \longleftarrow \mathtt{PPTS.Vrfy}(\mathsf{vk}, \mathsf{msg}, \sigma)$*: On input a message-signature pair* $(\mathsf{msg}, \sigma)$*, the algorithm outputs either* 1 *or* 0.

**Definition 24 (Compactness).** *We say that a $(k, n)$-*PPTS *scheme satisfies compactness if there exist polynomials $poly_1(\cdot)$, $poly_2(\cdot)$ such that for all $\lambda$, and any set $Q$ of servers with size at least $k$ the following holds.*
*For* $(\{\mathsf{sk}_i\}_{i=1}^N, \mathsf{vk}, \mathsf{pp}) \longleftarrow \mathsf{Init}(1^\lambda, (k, N), \mathbf{D})$*,* $\Sigma_i \longleftarrow \mathtt{PPTS.PartSign}(\mathsf{pp}, \mathsf{pw}, \mathsf{msg}, \mathsf{sk}_i)$ *for $i \in Q$,* $\sigma \longleftarrow \mathtt{PPTS.Combine}(\mathsf{pp}, \mathsf{pw}, \{\Sigma_i\})_{i \in Q:|Q|=k})$*, we have that* $|\sigma| \leq poly_1(\lambda)$ *and* $|\mathsf{vk}| \leq poly_2(\lambda)$.

**Definition 25 (Evaluation Correctness).** *If an honest user* $\mathbf{U}$ *interacts with a set of $k$ or more uncorrupted servers, say, $Q$ then it must output a valid signature $\sigma$ on a message* msg*; i.e., for any* msg*, any* $\mathsf{pw} \in \mathbf{D}$ *with* $(\{\mathsf{sk}_i\}_{i=1}^N, \mathsf{vk}, \mathsf{pp}) \longleftarrow \mathsf{Init}(1^\lambda, (k, N), \mathbf{D})$ *and*
$\sigma/\perp \longleftarrow \mathtt{PPTS.Combine}(\mathsf{pp}, \mathsf{pw}, \{\mathtt{PPTS.PartSign}(\mathsf{pp}, \mathsf{pw}, \mathsf{msg}, \mathsf{sk_i})\}_{\mathtt{i} \in \mathtt{Q}})$*, it holds that* $\Pr[\mathtt{PPTS.Vrfy}(\mathsf{vk}, \mathsf{msg}, \sigma) = 1] = 1 - \mathsf{negl}(\lambda)$.

**Definition 26 (Partial Verification Correctness).** *We say that a $(k, n)$-*PPTS *scheme satisfies partial verification correctness if for all $\lambda$ and any set $Q$ of servers with size at least $k$, the following holds. for any* msg*, any* pw $\in$

[6] Technically, we should write $\Sigma_i(\mathsf{pw})$ but for brevity we omit the pw.

$\mathbf{D}$ *with* $(\{\mathsf{sk}_i\}_{i=1}^N, \mathsf{vk}, \mathsf{pp}) \longleftarrow \mathsf{Init}(1^\lambda, (k, N), \mathbf{D})$*, it holds that for every* $i \in Q$*,* $\Pr[\texttt{PPTS.PartSignVerify}(\mathsf{pp}, \mathsf{pw}, \mathsf{msg}, \Sigma_i) = 1] = 1 - \mathsf{negl}(\lambda)$*.*

**Definition 27 (Unforgeability).** *A* $(k, N)$-PPTS *scheme on dictionary* $\mathbf{D}$ *is said to be unforgeable if for any PPT adversary* $\mathcal{A}_{\mathsf{PPTS}}$*, as described in the following game* $\mathsf{Expt}_{\mathtt{uf},\mathcal{A}_{\mathsf{PPTS}}}(1^\lambda, k, N, \mathbf{D})$*, outputs* 1 *with probability at most* $\frac{|\mathcal{O}|}{|\mathbf{D}|} + \mathsf{negl}(\lambda)$ *for a negligible function* $\mathsf{negl}(\lambda)$*, where* $|\mathcal{O}|$ *denotes the number of total queries* $\mathcal{A}_{\mathsf{PPTS}}$ *makes to the challenger for* PPTS.PartSign*,* PPTS.PartSignVerify *and* PPTS.Combine :

$\mathsf{Expt}_{\mathtt{uf},\mathcal{A}_{\mathsf{PPTS}}}(1^\lambda, k, N, \mathbf{D})$ :

- *On input the security parameter* $1^\lambda$*, the adversary* $\mathcal{A}$ *outputs an threshold access structure* $(k, N)$*.*
- *The challenger samples* $(\{\mathsf{sk}_i\}_{i=1}^N, \mathsf{vk}, \mathsf{pp}) \longleftarrow \mathsf{Init}(1^\lambda, (k, N), \mathbf{D})$ *and provides* $\mathsf{pp}$ *and* $\mathsf{vk}$ *to* $\mathcal{A}$*.*
- $\mathcal{A}$ *outputs a maximal unauthorized set* $S^* \subset P$ *of size* $k - 1$*.*
- *The challenger provides the set of keys* $\{\mathsf{sk}_i\}_{i \in S^*}$ *to* $\mathcal{A}$*.*
- $\mathcal{A}$ *issues a polynomial number of following adaptive queries:*
  - $\mathcal{A}$ *issues a message and server index* $(\mathsf{msg}, i)$*. For each query, the challenger computes* $\texttt{PPTS.PartSign}(\mathsf{pp}, \mathsf{pw}, \mathsf{msg}, \mathsf{sk}_i)$ *and returns* $\Sigma_i$ *to* $\mathcal{A}$*.*
  - $\mathcal{A}$ *issues a partial signature* $\Sigma_i$ *corresponding to server* $\mathbf{S}_i$*. For each query, the challenger computes* $\texttt{PPTS.PartSignVerify}(\mathsf{pp}, \mathsf{pw}, \Sigma_i)$ *and returns* 0 *or* 1*.*
  - $\mathcal{A}$ *issues a set of partial signatures* $\{\Sigma_i\}_{i \in T}$*, where* $|T| \geq k$*. For such a query, the challenger computes* $\texttt{PPTS.Combine}(\mathsf{pp}, T, \mathsf{pw}, \{\Sigma_i\}_{i \in T}, T)$ *and outputs* $\sigma$*.*
- *At the end of the experiment,* $\mathcal{A}$ *outputs a forgery* $(\mathsf{msg}^*, \sigma^*)$*. The experiment outputs* 1 *if* $\texttt{PPTS.Vrfy}(\mathsf{vk}, \mathsf{msg}^*, \sigma^*) = 1$ *and* $\mathsf{msg}^*$ *was not previously queried as a partial signing query.*

**Definition 28 (Robustness of PPTS).** *We say that a* PPTS *scheme for threshold access structure satisfies robustness if for all* $\lambda$*, the following holds. For any PPT adversary* $\mathcal{A}$*, the following experiment* $\mathtt{Expt}_{\mathcal{A},PPTS,Robust}(1^\lambda, 1^d)$ *outputs* 1 *with negligible probability:*

$\mathtt{Expt}_{\mathcal{A},PPTS,Robust}(1^\lambda, 1^d)$ :

1. *On input the security parameter* $1^\lambda$*, the adversary* $\mathcal{A}$ *outputs* $(k, N)$*-threshold access structure.*
2. *The challenger runs* $(\{sk_i\}_{i=1}^N, \mathsf{vk}, \mathsf{pp}) \leftarrow \mathtt{Init}(1^\lambda, (k, N), \mathbf{D})$ *and provides* $(\mathsf{pw}, \{sk_i\}_{i=1}^N, \mathsf{vk}, \mathsf{pp})$ *to* $\mathcal{A}$*.*
3. $\mathcal{A}$ *outputs a partial signature forgery* $(\mathsf{msg}^*, \Sigma_i^*, i)$*.*
4. *The challenger returns* 1 *if* $\texttt{PPTS.PartSign}(\mathsf{pp}, \mathsf{pw}, \mathsf{msg}^*, \mathsf{sk}_i) \neq \Sigma_i^*$ *and* $\texttt{PPTS.PartSignVerify}(\mathsf{pp}, \mathsf{pw}, \mathsf{msg}^*, \Sigma_i^*) = 1$*.*

- Initialization $(\{\mathsf{sk}_i\}_{i=1}^N, \mathsf{vk}, \mathsf{pp}) \longleftarrow \mathsf{Init}(1^\lambda, (k, N), \mathbf{D})$:
  - User chooses password $\mathsf{pw} \leftarrow \mathbf{D}$.
  - Set $\mathsf{PPUT}_{(k,N)} = (\mathtt{PPUT.Setup}, \mathtt{PPUT.Eval}, \mathtt{PPUT.Verify}, \mathtt{PPUT.Combine})$
  - Set $\mathtt{Sig} = (\mathsf{S.KeyGen}, \mathsf{S.Sign}, \mathsf{S.Verify})$. For our construction, we assume that the signing algorithm $\mathsf{S.Sign}$ is a deterministic algorithm. This is without loss of generality since any randomized signature scheme can be derandomized (i.e. using PRFs).
  - Execute $\mathsf{S.KeyGen}(1^\lambda)$ to obtain $(\mathsf{svk}, \mathsf{ssk})$
  - Obtain $(\mathsf{pp}_{pput}, S_1, \cdots, S_N) \leftarrow \mathtt{PPUT.Setup}(1^\lambda, 1^d, (k, N), \mathsf{pw}, \mathsf{ssk})$ for $N$ servers $\mathbf{S}_1, \ldots, \mathbf{S}_N$.
  - Sets: $\mathsf{pp} = \mathsf{pp}_{pput}$; $\mathsf{vk} = \mathsf{svk}$; $\mathsf{sk}_i = S_i$ for all $1 \leq i \leq N$.
- $\mathtt{PPTS.PartSign}(\mathsf{pp}, \mathsf{pw}, \mathsf{msg}, \mathsf{sk}_i)$:
  User contacts an available set $\{\mathbf{S}_{i_1}, \ldots, \mathbf{S}_{i_k}\}$ of servers indexed by $Q = \{i_1, \ldots, i_k\}$, and runs $\Sigma_i \leftarrow \mathtt{PPUT.Eval}(\mathsf{pp}_{pput}, \mathsf{sk}_i, \mathsf{pw}, C)$, where the circuit $C$ defined as
  $$C(\mathsf{ssk}) = \mathsf{S.Sign}(\mathsf{ssk}, \mathsf{msg}).$$
- $\mathtt{PPTS.PartSignVerify}(\mathsf{pp}, \mathsf{pw}, \mathsf{msg}, \Sigma_i)$:
  User runs $\mathtt{PPUT.Verify}(\mathsf{pp}_{pput}, \mathsf{pw}, \Sigma_i, C)$ and outputs 0 or 1.
- $\mathtt{PPTS.Combine}(\mathsf{pp}, \mathsf{pw}, \{\Sigma_i\}_{i \in Q:|Q|=k})$
  - User runs $\mathtt{PPUT.Combine}(\mathsf{pp}_{pput}, \mathsf{pw}, \{\Sigma_i\}_{i \in Q}, Q)$ to compute $\sigma$
  - outputs $(\mathsf{msg}, \sigma)$
- $\mathtt{PPTS.Vrfy}(\mathsf{vk}, \mathsf{msg}, \sigma)$ :
  - Run $\mathsf{S.Verify}(\mathsf{svk}, \mathsf{msg}, \sigma)$ to output 1/0.

**Fig. 4. Construction for Password Protected Threshold Signature**

### 5.1 A Construction for PPTS

In this section, we describe a generic construction (Fig. 4) of secure PPTS from PPUT and a deterministic unforgeable signature scheme Sig.

### 5.2 Security Analysis

The compactness property (Definition 24), the evaluation correctness (Def. 25), verification correctness (Definition 26) and the robustness property (Definition 28) of our PPTS construction follow from the corresponding properties of the underlying PPUT. In the following, we prove the unforgeability.

**Theorem 3.** *Suppose the underlying signature scheme is an unforgeable signature scheme. Then the* PPTS *obtained using the compiler construction as described in Fig. 4 is unforgeable satisfying Definition 27.*

*Proof.* The adversary can always try guessing a password and attack the system which gives him a probability of $\frac{|\mathcal{O}|}{\mathbf{D}}$ to forge a signature.

The proof proceeds via a sequence of hybrid experiments between $\mathcal{A}$ and the challenger.

- $\mathtt{H}_0$ : This is the PPTS real security experiment $\mathsf{Expt}_{\mathtt{uf},\mathcal{A}_{\mathsf{PPTS}}}(1^\lambda,(k,N),\mathbf{D})$ between an adversary $\mathcal{A}$ and a challenger.
- $\mathtt{H}_1$ : This experiment is the same as $\mathtt{H}_0$, except that, the challenger invokes the PPUT simulator Sim.PPUT for initialization, partial signature, partial signature verification and combine. For ease of the representation, we denote PPUT simulator as Sim.PPUT.Setup to simulate setup, Sim.PPUT.Eval to simulate partial evaluation, Sim.PPUT.Verify to simulate verification, and Sim.PPUT.Combine to simulate the combination of partial evaluations. The challenger first executes $\mathsf{S.KeyGen}(1^\lambda)$ to obtain $(\mathsf{svk},\mathsf{ssk})$. Then invokes $(\mathsf{pp}_{pput},S_1,\cdots,S_N)\leftarrow\mathtt{Sim.PPUT.Setup}(1^\lambda,1^d,(k,N),\bar{\mathsf{pw}})$ by choosing a random password $\bar{\mathsf{pw}}$, and sets $\mathsf{pp}=\mathsf{pp}_{pput}$, $\mathsf{vk}=\mathsf{svk}$, $\mathsf{sk}_i=S_i$ for all $1\le i\le N$, and sends $\{\mathsf{sk}_i\}_{i\in B}$ to $\mathcal{A}$. To reply partial signature against $(\mathsf{msg},i)$, the challenger first computes $\sigma=\mathsf{S.Sign}(\mathsf{ssk},\mathsf{msg})$ and sets $C(\mathsf{ssk})=\sigma$. Then invokes $\Sigma_i\leftarrow\mathtt{Sim.PPUT.Eval}(\mathsf{pp}_{pput},\mathsf{sk}_i,\bar{\mathsf{pw}},\sigma)$. Similarly, for partial signature verification and combine queries, the challenger invokes Sim.PPUT.Verify and Sim.PPUT.Combine, respectively.

The only difference between the experiments $\mathtt{H}_0$ and $\mathtt{H}_1$ is the way the challenger runs PPUT setup and the way it answers the queries. In particular, the challenger uses Sim.PPUT in $\mathtt{H}_1$. Hence,

$$|\Pr[\mathtt{H}_0(\mathcal{A})=1]-\Pr[\mathtt{H}_1(\mathcal{A})=1]|=$$

$$|\Pr[\mathtt{Expt}_{\mathcal{A},\mathsf{PPUT},Real}(1^\lambda,1^d)=1]-\Pr[\mathtt{Expt}_{\mathcal{A},\mathsf{PPUT},Ideal}(1^\lambda,1^d)=1]|\in\mathsf{negl}(\lambda).$$

**Reduction to** $\mathtt{Sig}=(\mathsf{S.KeyGen},\mathsf{S.Sign},\mathsf{S.Verify})$:
Let $\mathcal{A}$ be any adversary in $\mathtt{H}_1$. We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that wins the unforgeability experiment $\mathtt{Expt}_{\mathcal{A},\mathtt{Sig},uf}(1^\lambda)$. Algorithm $\mathcal{B}$ works as follows:

- At the beginning of the game, $\mathcal{B}$ receives $\mathsf{svk}$ from the unforgeability challenger. It instantiates PPUT setup $(\mathsf{pp}_{pput},S_1,\cdots,S_N)\leftarrow\mathtt{Sim.PPUT.Setup}(1^\lambda,1^d,(k,N),\bar{\mathsf{pw}})$ by choosing a random password $\bar{\mathsf{pw}}$, and sets $\mathsf{pp}=\mathsf{pp}_{pput}$, $\mathsf{vk}=\mathsf{svk}$, $\mathsf{sk}_i=S_i$ for all $1\le i\le N$, and provides $\mathsf{pp}=\mathsf{pp}_{pput}$, $\mathsf{vk}=\mathsf{svk}$ to $\mathcal{A}$. Note that by the definition of the unforgeability challenger, the view of $\mathcal{A}$ until this point is exactly the view in $\mathtt{H}_1$.
- When $\mathcal{A}$ outputs a maximal corrupted $S^*$, $\mathcal{B}$ provides $\{S_i\}_{i\in S^*}$. This is, also, a perfect simulation of $\mathtt{H}_1$.
- To reply partial signature against $(\mathsf{msg},i)$ that $\mathcal{A}$ makes, $\mathcal{B}$ submits $\mathsf{msg}$ as its own signing query to the unforgeability challenger and receives $\sigma=\mathsf{S.Sign}(\mathsf{ssk},\mathsf{msg})$ and sets $C(\mathsf{ssk})=\sigma$. Then invokes $\Sigma_i\leftarrow\mathtt{Sim.PPUT.Eval}(\mathsf{pp}_{pput},\mathsf{sk}_i,\bar{\mathsf{pw}},\sigma)$ by choosing a random password $\bar{\mathsf{pw}}$.

Similarly, for partial signature verification and combine queries, $\mathcal{B}$ invokes $\texttt{Sim.PPUT.Verify}$ and $\texttt{Sim.PPUT.Combine}$, respectively. Therefore, each partial signatures $\Sigma_i$ and signature $\sigma$ are simulated exactly as in $\texttt{H}_1$.

– At the end of the experiment, $\mathcal{A}$ outputs a forgery $(\mathsf{msg}, \sigma^*)$. Recall that $(\mathsf{msg}, \sigma^*)$ is a valid forgery if
  - $\texttt{PPTS.Vrfy}(\mathsf{vk}, \mathsf{msg}^*, \sigma^*) = 1$
  - $\mathsf{msg}^*$ was not previously queried as a partial signing query.

  By definition, $\texttt{PPTS.Vrfy}(\mathsf{vk}, \mathsf{msg}^*, \sigma^*) = \mathsf{S.Verify}(\mathsf{svk}, \mathsf{msg}^*, \sigma^*)$. Furthermore, $\mathcal{B}$ invokes the signing query to the unforgeability challenger only when $\mathcal{A}$ makes its partial signing queries. Therefore, the message, signature pair $(\mathsf{msg}, \sigma^*)$ is a valid forgery for $\mathcal{B}$. The algorithm $\mathcal{B}$ outputs $(\mathsf{msg}, \sigma^*)$ as its valid forgery.

Finally, by the correctness of $\mathcal{B}$, we have

$$\Pr[\texttt{H}_1(\mathcal{A}) = 1] = \Pr[\texttt{Expt}_{\mathcal{A},\texttt{Sig},uf}(1^\lambda) = 1] \in \mathsf{negl}(\lambda),$$

which concludes the proof. ■

**Lattice-based Instantiation in the Standard Model.** We instantiate using the lattice-based PPUT from section 4 and lattice-based adaptive EUF-CMA signature scheme by [3].

## 6 Conclusion

We presented a password protected universal thresholdizer scheme which can be seen as a general framework for introducing threshold functionality to a large class of non-threshold cryptographic schemes assuring user authentication. Our construction is based on the universal thresholdizer of [9] and a threshold password authenticated key exchange protocol and provides simulation-based security. As a relevant application, we presented a password protected threshold signature scheme. All constructions can be instantiated using the hardness assumptions from lattices in the standard model making the primitives quantum-safe. Modeling the security in the UC framework and exhibiting other possible applications are left as open problems.

## References

1. Michel Abdalla, Olivier Chevassut, Pierre-Alain Fouque, and David Pointcheval. A simple threshold authenticated key exchange from short secrets. In *ASIACRYPT*, pages 566–584. Springer, 2005.
2. Michel Abdalla, Mario Cornejo, Anca Nitulescu, and David Pointcheval. Robust password-protected secret sharing. In *ESORICS*, pages 61–79. Springer, 2016.
3. Jacob Alperin-Sheriff. Short signatures with short public keys from homomorphic trapdoor functions. In *PKC*, pages 236–255. Springer, 2015.

4. Ali Bagherzandi, Stanislaw Jarecki, Nitesh Saxena, and Yanbin Lu. Password-protected secret sharing. In *CCS*, pages 433–444, 2011.
5. Abhishek Banerjee and Chris Peikert. New and improved key-homomorphic pseudorandom functions. In *CRYPTO*, pages 353–370. Springer, 2014.
6. Steven Michael Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. 1992.
7. G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference*, pages 313–317, 1979.
8. Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *PKC*, pages 31–46, 2003.
9. Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter MR Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *CRYPTO*, pages 565–596. Springer, 2018.
10. Jan Camenisch, Anja Lehmann, Anna Lysyanskaya, and Gregory Neven. Memento: How to reconstruct your secrets from a single password in a hostile environment. In *CRYPTO*, pages 256–275. Springer, 2014.
11. Jan Camenisch, Anja Lehmann, and Gregory Neven. Optimal distributed password verification. In *CCS*, pages 182–194, 2015.
12. Ran Canetti and Shafi Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In *EUROCRYPT*, pages 90–106. Springer, 1999.
13. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Conference on the Theory and Application of Cryptology*, pages 307–315. Springer, 1989.
14. Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *CCS*, pages 1179–1194, 2018.
15. Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In *ACNS*, pages 156–174. Springer, 2016.
16. Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold dss signatures. In *EUROCRYPT*, pages 354–371. Springer, 1996.
17. Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. In *CRYPTO*, pages 408–432. Springer, 2001.
18. Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and t-pake in the password-only model. In *ASIACRYPT*, pages 233–253. Springer, 2014.
19. Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In *EuroS&P*, pages 276–291. IEEE, 2016.
20. Philip MacKenzie, Thomas Shrimpton, and Markus Jakobsson. Threshold password-authenticated key exchange. In *CRYPTO*, pages 385–400. Springer, 2002.
21. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, pages 700–718. Springer, 2012.
22. Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for np from (plain) learning with errors. In *CRYPTO*, pages 89–114. Springer, 2019.
23. Mario Di Raimondo and Rosario Gennaro. Provably secure threshold password-authenticated key exchange. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 507–523. Springer, 2003.
24. Partha Sarathi Roy, Sabyasachi Dutta, Willy Susilo, and Reihaneh Safavi-Naini. Password protected secret sharing from lattices. In *ACNS*, pages 442–459. Springer, 2021.

25. Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, pages 543–553. IEEE, 1999.
26. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
27. Victor Shoup. Practical threshold signatures. In *EUROCRYPT*, pages 207–220. Springer, 2000.