Smooth NIZK Arguments

Charanjit S. Jutla[1(✉)] and Arnab Roy[2]

[1] IBM T. J. Watson Research Center, Yorktown Heights, NY, USA
csjutla@us.ibm.com
[2] Fujitsu Laboratories of America, Sunnyvale, CA, USA
aroy@us.fujitsu.com

Abstract. We introduce a novel notion of smooth (-verifier) non- interactive zero-knowledge proofs (NIZK) which parallels the familiar notion of smooth projective hash functions (SPHF). We also show that the single group element quasi-adaptive NIZK (QA-NIZK) of Jutla and Roy (CRYPTO 2014) and Kiltz and Wee (EuroCrypt 2015) for linear subspaces can be easily extended to be computationally smooth. One important distinction of the new notion from SPHFs is that in a smooth NIZK the public evaluation of the hash on a language member using the projection key does not require the witness of the language member, but instead just requires its NIZK proof.

This has the remarkable consequence that if one replaces the traditionally employed SPHFs with the novel smooth QA-NIZK in the Gennaro-Lindell paradigm of designing universally-composable password- authenticated key-exchange (UC-PAKE) protocols, one gets highly efficient UC-PAKE protocols that are secure even under adaptive corruption. This simpler and modular design methodology allows us to give the first single-round *asymmetric* UC-PAKE protocol, which is also secure under adaptive corruption in the erasure model. Previously, all asymmetric UC-PAKE protocols required at least two rounds. In fact, our protocol just requires each party to send a single message asynchronously. In addition, the protocol has short messages, with each party sending only four group elements. Moreover, the server password file needs to store only one group element per client. The protocol employs asymmetric bilinear pairing groups and is proven secure in the (limited programmability) random oracle model and under the standard bilinear pairing assumption SXDH.

Keywords: QA-NIZK · Bilinear pairings · SXDH · MDDH
UC-PAKE · Online attack · Server compromise · Dual-system

1 Introduction

Ever since the remarkably efficient non-interactive zero knowledge (NIZK) proofs [BFM88] for algebraic statements were developed by Groth and Sahai (GS-NIZK) [GS12], there have been significant efficiency improvements and innovations in the construction of cryptographic protocols. Jutla and Roy [JR13, JR14]

A. Beimel and S. Dziembowski (Eds.): TCC 2018, LNCS 11239, pp. 235–262, 2018.
https://doi.org/10.1007/978-3-030-03807-6_9

and Libert, Peters, Joye and Yung [LPJY14] further improved the efficiency of algebraic NIZK proofs, culminating in *constant* size NIZK proofs for linear subspaces, independent of the number of equations and witnesses. This efficiency improvement came in the weaker Quasi-Adaptive setting [JR13], which nevertheless proved sufficient for many applications.

Quasi-adaptive NIZK (QA-NIZK) proofs were further extended to provide simulation soundness [LPJY14,KW15] and dual-system simulation soundness [JR15], thus lending applicability to many more applications, such as structure preserving signatures, password authenticated key exchange in the UC model, and keyed homomorphic CCA-secure encryption.

In this paper, we further extend (QA-)NIZK proofs to provide an additional property called *smooth soundness*. The idea is to force the verification step to consist of computing hashes in two different ways and comparing the result. To this end, the verifier is split into three algorithms: a randomized hash-key generation algorithm, a public hashing algorithm and a private hashing algorithm. The verification step starts off by generating two hash-keys, the private key and the projection key. Next, the setting allows computation of a private hash given the private hash-key and the word, and computation of the public hash using the projection key and just a QA-NIZK proof for the word - the witness for the word is *not* required. Completeness states that the private hash is equal to the public hash for a language member and correct (QA-)NIZK proof. Computational soundness states that it is hard to come up with a proof such that a non-language word passes the same equality check. The new *smoothness* property states that for any non-language word, the private hash algorithm outputs a value (computationally) indistinguishable from uniformly random, even when the projection key is given to the adversary.

Comparison with SPHFs. The new primitive is modeled after smooth projective hash functions (SPHF [CS02]). An SPHF also generates private and projection hash-keys and defines a private hash and a public hash. Further, similar properties hold where (1) for a member word, private hash equals public hash, (2) for a non-member word, private hash is uniformly random (even given projection hash-key). The crucial difference is that[1], whereas the SPHF public hash computation requires a witness of the member word, the smooth (QA-)NIZK public hash requires only a NIZK proof of the word. This allows for hiding of the witness, even when computing using the projection hash-key. In contrast, trapdoor-SPHFs as introduced by [BBC+13] allow a simulation world to have a trapdoor to evaluate a hash over a word without a witness and using only projection hash-key. As shown in Fig. 1, where trapdoor-SPHFs are compared with smooth QA-NIZK, their notion does not allow “erasure” of the witness w in the real world (if only projection hash-key is available).

While Fig. 1 is self-explanatory, it brings up an interesting alternative interpretation of smooth (QA)-NIZK. In the common reference string (CRS) setting, one can have a *composite*-SPHF, which is composed of two SPHFs: the first

[1] On the other hand, our constructions only allow computational smooth-soundness, while for SPHFs these properties hold information-theoretically.

SPHF's projection hash-key is published in the CRS, that enables one to compute an intermediate-hash using the witness w of a language member x. At this point, the witness can be erased. Next, the real projection hash-key hp is revealed (possibly, generated by another party along with the private hash-key hk). Then, a final hash on input x can be computed using hp and the intermediate-hash. If x is a language member then the final hash is same as that computed from x using hk. The question then arises as to why the intermediate-hash is depicted as a (QA)-NIZK in Fig. 1. However, we note that this first SPHF is already publicly verifiable, as the private hash-key of the first SPHF is generated by the CRS generator, and if it has to be used in any form in a private hash evaluation (and in the real world) it must be publicly available in the CRS. Indeed, in our construction the private hash-key k is given in the CRS as a commitment to k (this interpretation of QA-NIZK as a publicly-verifiable SPHF was given in [KW15]).

We remark that this interpretation of smooth (QA)-NIZK is similar to constructions of structure-preserving SPHF in [BC16a]. In that work, the intermediate-hash is a GS-NIZK proof in the commit and prove framework [GS12], and hence the (second) private-hash takes the commitments also as input. Although the first hash is not a SPHF, their construction can still be viewed as a smooth NIZK (as per our definition[2]). Since their construction also works only for linear subspaces, our smooth QA-NIZK construction turns out to be more efficient, namely that no commitments need to be given for private hash computation.

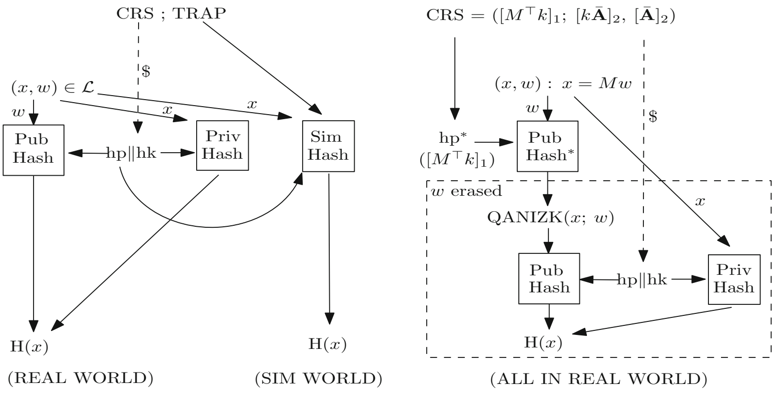

Fig. 1. Trapdoor-SPHF [BBC+13] vs Smooth QANIZK

Our Construction. In this work, we show that the single group element QA-NIZK arguments of [JR14,KW15] can be easily extended to be smooth. As a

[2] Since their construction uses the commit and prove paradigm of GS-NIZK proofs, their (composite-) construction is a smooth NIZK with a small tweak: they obtain information-theoretic smoothness, but computational zero-knowledge.

first application, we show that in the Gennaro-Lindell paradigm of designing universally-composable password-authenticated key exchange (UC-PAKE) protocols, if one replaces the traditional SPHFs with the novel smooth QA-NIZK, then one gets highly efficient single-round UC-PAKE protocols which are adaptively secure in the erasure model. At a high level, the UC simulator must emulate each party's outgoing commitment to the password, without knowing the password. This is not difficult, as one can use ElGamal encryption to achieve a hiding commitment. However, if the party is corrupted after its message has been sent, the simulator is at a loss to produce a witness which each party must retain to eventually compute the SPHF public hash. In our new protocol, the parties need only save the QA-NIZK and not the witness, as that suffices to compute the public hash. This nicely captures the main idea behind the single-round adaptively secure UC-PAKE of [JR15]. In this work, the novel abstraction further allows us to obtain a single-round adaptively secure *asymmetric* UC-PAKE, which is a much more difficult notion to understand even from a definitional perspective, let alone constructing one.

(Asymmetric) Password-Authenticated Key-Exchange. The problem of setting up a secure channel between two parties that only share a human-memorizable password (or a low-entropy secret) was first studied by Bellovin and Merritt [BM92], and later by Jiang and Gong [JG04]. Since then, this problem has been extensively studied and is called the *password-authenticated key-exchange* (PAKE) problem. One of the main challenges in designing such protocols is the intricacy in the natural security definition which requires that the protocol transcripts cannot be used to launch *offline dictionary attacks*. While an adversary can clearly try to guess the (low-entropy) password and impersonate one of the parties, its advantage from the fact that the password is of low entropy should be limited to such *online* impersonation attacks.

In a subsequent paper, Bellovin and Merritt [BM93] also considered a stronger model of server compromise such that if a server's password file is revealed to the adversary it cannot directly impersonate a client (cf. if the password was stored in the raw at the server). The adversary should be able to impersonate the client only if it succeeds in an offline dictionary attack on the revealed server password file. Clearly, this requires that the server does not store the password as it is (or in some reversibly-encrypted form), and protocols satisfying this stronger security requirement are referred to as *asymmetric PAKE* protocols.

Canetti et al. [CHK+05] also considered designing (symmetric) UC-PAKE protocols in the universally-composable (UC) framework [Can01]. One of their main contributions was the definition of a natural UC-PAKE ideal functionality ($\mathcal{F}_{\text{PAKE}}$). Gentry et al. [GMR06] extended the functionality of symmetric UC-PAKE [CHK+05] to the asymmetric setting ($\mathcal{F}_{\mathsf{apwKE}}$) and gave a general method of extending any symmetric UC-PAKE protocol to an asymmetric UC-PAKE protocol (from now on referred to as UC-APAKE). Their general method adds an additional round to the UC-PAKE protocol.

Our Contributions. In this paper, we give the first single-round UC-APAKE protocol (realizing $\mathcal{F}_{\mathsf{apwKE}}$). In fact, both parties just send a single message asynchronously. The protocol is realized in the (limited programmability[3]) random-oracle (RO) [BR93] hybrid-model under standard static assumptions for bilinear groups, namely SXDH [BBS04] and the general MDDH [EHK+13] assumption. Our protocol is also secure against adaptive corruption (in the erasure model) and is very succinct, with each message consisting of only four group elements. Moreover, for each client the server need store only *one* group elements as a "password hash". Many non-UC asymmetric PAKE protocols are at least two rounds [HK98,BPR00,BMP00,Mac01,Boy09]. Benhamouda and Pointcheval [BP13] proposed the first single round asymmetric PAKE protocol, but in a game-based model built on the BPR model [BPR00].

The first single-round UC-secure *symmetric* PAKE protocol was given in [KV11] (using bilinear pairings), which was then further improved (in the number of group elements) in subsequent papers [JR12,BBC+13]. Recently in [JR15], a single round UC-PAKE protocol (in the standard model and using bilinear pairings) was also proven secure against adaptive corruption using ideas from the dual-system IBE construction of Waters [Wat09]. However, the [JR15] construction did not employ their dual-system simulation-sound QA-NIZK proofs (DSS-QA-NIZK) in a black box manner. Instead, it used ideas from the DSS-QA-NIZK construction and properties as the underlying intuition for the proof.

In this paper, we show that the UC-PAKE of [JR15] can be built in a black-box manner using smooth QA-NIZK arguments. Next, we build on the verifier-based PAKE (VPAKE) construction of [BP13], to construct the first adaptively-secure UC-APAKE protocol, which in addition has a single (asynchronous) round. Since, in the UC framework, the simulator has to detect offline password guesses by an adversary that steals the server password file, for provable security this seems to inevitably require the RO model, and indeed our security proof is in the (limited programmability) RO model.

In our protocol, each party sends an ElGamal style encryption of the (hash of) the password *pw* to the other party, along with an SPHF of the underlying language and a projection verification hash-key of a smooth QA-NIZK of the underlying language (ElGamal augmented with the SPHF). If such a message is adversarially inserted, the simulator must have the capability to extract password pw' from it, so that it can feed the ideal functionality $\mathcal{F}_{\mathsf{apwKE}}$ to test this guess of the password. Thus, the NIZK proof must have simulation-sound extractability. It was shown in [JR15] that dual-system simulation soundness suffices for this purpose (and that makes the protocol very simple). When using smooth QA-NIZK, this dual-system simulation-soundness can be attained by simply sending an SPHF.

Detailed explanations can be found in Sect. 5 with proof details in the full version [JR16], where we also explain how the random oracle is used to extract the password efficiently from the exponent. This leads to a security reduction

[3] Basically, the output values of the random oracle are all randomly chosen, but different inputs can be assigned dynamically to these outputs [FLR+10].

which has an additive computational overhead of $n * m * \text{poly}(q)$, where n is the number of random oracle calls, m is the number of online attacks and q is the security parameter.

Recent Related Work. Recently, [JKX18] formulated a stronger UC-APAKE functionality that disallows use of pre-computation to attack a stolen password file. Using an ideal functionality for oblivious pseudo-random functions (OPRF), they give a compiler that converts any standard UC-APAKE realization (such as in this paper) into one that satisfies their stronger definition (in the OPRF-hybrid model). Thus, if the OPRF can be realized with adaptive corruption, then we obtain a adaptive corruption secure (strong) UC-APAKE realization. However, to the best of our knowledge, no adaptive corruption secure (UC-) OPRF realization is known, and hence the problem of realizing adaptive corruption secure strong UC-APAKE remains open.

Organization. The rest of the paper is organized as follows. In Sect. 2, we introduce the new notion of smooth QA-NIZK proofs. In Sect. 3, we recall the MDDH assumptions and establish a useful *boosting* theorem relating the assumptions. In Sect. 4, we give the single group element smooth QA-NIZK construction for linear subspaces. In Sect. 5, we describe the ideal functionality $\mathcal{F}_{\mathsf{apwKE}}$ for asymmetric password-authenticated key-exchange and construct the new single-round UC-APAKE protocol. Preliminaries and proofs of many of the theorems are relegated to the Appendix.

2 Smooth Quasi-Adaptive NIZK Proofs

We start by reviewing the definition of Quasi-Adaptive computationally-sound NIZK proofs (QA-NIZK) [JR13]. A witness relation is a binary relation on pairs of inputs, the first called a (potential) language member and the second called a witness. Note that each witness relation R defines a corresponding language L which is the set of all x for which there exists a witness w, such that $R(x, w)$ holds.

We will consider QA-NIZK proofs for a probability distribution $\mathcal{D}$ on a collection of (witness-) relations $\mathcal{R} = \{R_\rho\}$ (with corresponding languages L_ρ). Recall that in a QA-NIZK, the CRS can be set after the language parameter has been chosen according to $\mathcal{D}$. We recall the formal definition of Quasi-Adaptive NIZK below from [JR13].

Definition 1 (QA-NIZK [JR13]). *We call a tuple of efficient algorithms* (pargen, crsgen, prover, ver) *a* quasi-adaptive non-interactive zero-knowledge *(QA-NIZK) proof system for witness-relations $\mathcal{R}_\lambda = \{R_\rho\}$ with parameters sampled from a distribution $\mathcal{D}$ over associated parameter language* Lpar*, if there exist simulators* crssim *and* sim *such that for all non-uniform PPT adversaries $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$, we have (in all of the following probabilistic experiments, the experiment starts by setting λ as $\lambda \leftarrow \mathsf{pargen}(1^m)$, and choosing ρ as $\rho \leftarrow \mathcal{D}_\lambda$):*

Quasi-Adaptive Completeness:

$$\Pr\left[\begin{array}{l}\textsc{crs} \leftarrow \mathsf{crsgen}(\lambda,\rho)\\ (x,w) \leftarrow \mathcal{A}_1(\textsc{crs},\rho)\\ \pi \leftarrow \mathsf{prover}(\textsc{crs},x,w)\end{array} : \begin{array}{c}\mathit{ver}(\textsc{crs},x,\pi)=1 \ \textbf{\textit{if}}\\ R_\rho(x,w)\end{array}\right] = 1$$

Quasi-Adaptive Soundness:

$$\Pr\left[\begin{array}{l}\textsc{crs} \leftarrow \mathsf{crsgen}(\lambda,\rho)\\ (x,\pi) \leftarrow \mathcal{A}_2(\textsc{crs},\rho)\end{array} : \begin{array}{c}x \notin L_\rho \ \textbf{\textit{and}}\\ \mathit{ver}(\textsc{crs},x,\pi)=1]\end{array}\right] \approx 0$$

Quasi-Adaptive Zero-Knowledge:

$$\Pr\left[\textsc{crs} \leftarrow \mathsf{crsgen}(\lambda,\rho) \ : \ \mathcal{A}_3^{\mathsf{prover}(\textsc{crs},\cdot,\cdot)}(\textsc{crs},\rho)=1\right]$$
$$\approx$$
$$\Pr\left[(\textsc{crs},\mathsf{trap}) \leftarrow \mathsf{crssim}(\lambda,\rho) \ : \ \mathcal{A}_3^{\mathit{sim}^*(\textsc{crs},\mathsf{trap},\cdot,\cdot)}(\textsc{crs},\rho)=1\right],$$

where $\mathit{sim}^*(\textsc{crs},\mathsf{trap},x,w) = \mathsf{sim}(\textsc{crs},\mathsf{trap},x)$ for $(x,w) \in R_\rho$ and both oracles (i.e. prover *and sim^*) output failure if $(x,w) \notin R_\rho$.*

We call a QA-NIZK **smooth (-verifier)** if the verifier ver consists of three efficient algorithms ver = (hkgen, pubH, privH), and it satisfies the following modified completeness and soundness conditions. Here, hkgen is a probabilistic algorithm that takes a CRS as input and outputs two keys, hp, a projection hash key, and hk, a private hash key. The algorithm privH takes as input a word (e.g. a potential language member), and a (private hash) key, and outputs a string. Similarly, the algorithm pubH takes as input a word, a proof (for instance generated by prover), and a (projection hash) key hp, and outputs a string.

The **completeness** property is now defined as:

$$\Pr\left[\begin{array}{l}\textsc{crs} \leftarrow \mathsf{crsgen}(\lambda,\rho)\\ (x,w) \leftarrow \mathcal{A}_1(\textsc{crs},\rho)\\ \pi \leftarrow \mathsf{prover}(\textsc{crs},x,w)\\ (\mathsf{hp},\mathsf{hk}) \leftarrow \mathsf{hkgen}(\textsc{crs})\end{array} : \begin{array}{c}\mathsf{privH}(\mathsf{hk},x) = \mathsf{pubH}(\mathsf{hp},x,\pi)\\ \textbf{if } R_\rho(x,w)\end{array}\right] = 1$$

The QA-NIZK is said to satisfy **smooth-soundness** if for all words $x \notin L_\rho$, $\mathsf{privH}(\mathsf{hk},x)$ is computationally indistinguishable to the Adversary from uniformly random, even when the Adversary is given hp, and even if it produces x after receiving hp.

More precisely, **Quasi-Adaptive Smooth-Soundness** is the following property (let $\mathcal{U}$ be the uniform distribution on the range of privH, which is assumed to be of cardinality exponential in m): for every two-stage efficient oracle adversary $\mathcal{A}$

$$\Pr\left[\begin{array}{l}\textsc{crs} \leftarrow \mathsf{crsgen}(\lambda,\rho),\ (\mathsf{hp},\mathsf{hk}) \leftarrow \mathsf{hkgen}(\textsc{crs})\\ (x^*,\sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(\textsc{crs},\rho,\mathsf{hp}),\ u \leftarrow \mathcal{U}\end{array} : \mathcal{A}^{\mathcal{O}}(\mathsf{privH}(\mathsf{hk},x^*),\sigma) = 1 \mid Q\right]$$
$$\approx$$
$$\Pr\left[\begin{array}{l}\textsc{crs} \leftarrow \mathsf{crsgen}(\lambda,\rho),\ (\mathsf{hp},\mathsf{hk}) \leftarrow \mathsf{hkgen}(\textsc{crs})\\ (x^*,\sigma) \leftarrow \mathcal{A}^{\mathcal{O}}(\textsc{crs},\rho,\mathsf{hp}),\ u \leftarrow \mathcal{U}\end{array} : \mathcal{A}^{\mathcal{O}}(u,\sigma) = 1 \mid Q\right]$$

where the oracle $\mathcal{O}$ is instantiated with $\mathsf{privH}(\mathsf{hk}, \cdot)$, and Q is the condition that x^* is not in the language L_ρ **and** all oracle calls by the adversary in both stages are with L_ρ-language members. Here, σ is a local state of $\mathcal{A}$.

Note that as opposed to the information-theoretic smoothness property of projective hash functions, one cannot argue here that $\mathsf{privH}(\mathsf{hk}, x)$ for $x \in L_\rho$ can instead just be computed using hp, as that would also require efficiently computing a witness for x. Hence, the need to provide oracle access to $\mathsf{privH}(\mathsf{hk}, \cdot)$ for language members.

Also, note that smooth-soundness implies the earlier definition of soundness [JR13] if verification of (x, π) is defined as $\mathsf{privH}(\mathsf{hk}, x) = \mathsf{pubH}(\mathsf{hp}, x, \pi)$.

To differentiate the functionalities of the verifier of a QA-NIZK from similar functionalities of an SPHF, we will prepend the SPHF functionalities with keyword sphf and the QA-NIZK verifier functionalities with the keyword ver.

3 Matrix Decisional Assumptions

We will consider bilinear groups that consist of three cyclic groups of prime order q, $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ with an efficient bilinear map $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Group elements $\mathbf{g}_1$ and $\mathbf{g}_2$ will typically denote generators of the group $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. Following [EHK+13], in this section and the next we will use the notations $[a]_1, [a]_2$ and $[a]_T$ to denote $a\mathbf{g}_1, a\mathbf{g}_2$, and $a \cdot \mathsf{e}(\mathbf{g}_1, \mathbf{g}_2)$ respectively and use additive notations for group operations. When talking about a general group $\mathbb{G}$ with generator $\mathbf{g}$, we will just use the notation $[a]$ to denote $a\mathbf{g}$. However, in the UC-APAKE constructions, we will switch to multiplicative notation for easy readability.

For two vector or matrices A and B, we will denote the product $A^\top B$ as $A \cdot B$. The pairing product $\mathsf{e}([A]_1, [B]_2)$ evaluates to the matrix product $[AB]_T$ in the target group with pairing as multiplication and target group operation as addition.

We recall the *Matrix Decisional Diffie Hellman* or MDDH assumptions from [EHK+13]. A matrix distribution $\mathcal{D}_{l,k}$, where $l > k$, is defined to be an efficiently samplable distribution on $\mathbb{Z}_q^{l \times k}$ which is full-ranked with overwhelming probability. The $\mathcal{D}_{l,k}-$MDDH *assumption* in group $\mathbb{G}$ states that with samples $\mathbf{A} \leftarrow \mathcal{D}_{l,k}$ and $(\mathbf{s},\ \mathbf{s}') \leftarrow \mathbb{Z}_q^k \times \mathbb{Z}_q^l$, the tuple $([\mathbf{A}], [\mathbf{As}])$ is computationally indistinguishable from $([\mathbf{A}], [\mathbf{s}'])$. A matrix distribution $\mathcal{D}_{k+1,k}$ is simply denoted by $\mathcal{D}_k$.

Intuitively, a $\mathcal{D}_{l,k}-$MDDH assumption allows us to generate l (computationally) independently random group elements from an initial k independently random exponents. A $\mathcal{D}_k-$MDDH assumption allows us to generate one extra random group element. In this section, we will establish that, in fact, a $\mathcal{D}_k-$MDDH assumption can be *boosted* to generate additional (computationally) independently random elements. This will be useful to us in the next section to prove the smoothness property of our construction.

We remark that boosting is different from the random self-reducibility of $\mathcal{D}_{l,k}-$MDDH assumptions, as described by [EHK+13]. While the former aims to generate extra randomness from the same initial sample of vector of random exponents, the latter talks about results from several independent samples of vector of random exponents. Boosting can be seen as an abstraction of the *switching lemma* of [JR14] and follows the same blueprint for the proof.

For an $l \times k$ matrix $\mathbf{A}$, we denote $\bar{\mathbf{A}}$ to be the top $k \times k$ square sub-matrix of $\mathbf{A}$ and $\underline{\mathbf{A}}$ to be the bottom $(l-k) \times k$ sub-matrix of $\mathbf{A}$.

Theorem 1. *Let $\mathcal{D}_k$ be a matrix distribution on $\mathbb{Z}_q^{(k+1)\times k}$. Define another matrix distribution $\mathcal{D}_{l,k}$ on $\mathbb{Z}_q^{l\times k}$ as follows: First sample matrices $\mathbf{A} \leftarrow \mathcal{D}_k$ and $\mathbf{R} \leftarrow \mathbb{Z}_q^{(l-k)\times k}$ and then output $\begin{pmatrix} \bar{\mathbf{A}} \\ \mathbf{R} \end{pmatrix}$. Then the $\mathcal{D}_k-$MDDH assumption implies the $\mathcal{D}_{l,k}-$MDDH assumption.*

We will call *boosting* to be the process of stretching $\mathcal{D}_k$ to $\mathcal{D}_{l,k}$ as above. This theorem is proved as a corollary of an even more general theorem which we will describe after defining the notion of 'boostable'-ity as follows.

Definition 2. *We say that a matrix distribution $\mathcal{D}_k$ on $\mathbb{Z}_q^{(k+1)\times k}$ is* boostable *to a matrix distribution $\mathcal{D}_{l,k}$ on $\mathbb{Z}_q^{l\times k}$, where $l > k$, if there are efficiently samplable distributions $\mathcal{E}$ on $\mathbb{Z}_q^{(l-k)\times k}$ and $\mathcal{F}$ on $\mathbb{Z}_q^{(l-k)\times(k+1)}$, such that the following hold:*

- *For $\mathbf{A} \leftarrow \mathcal{D}_k, \mathbf{B} \leftarrow \mathcal{D}_{l,k}, \mathbf{E} \leftarrow \mathcal{E}, \mathbf{F} \leftarrow \mathcal{F}$, we have:*
$$\bar{\mathbf{B}} \approx \bar{\mathbf{A}}, \quad \underline{\mathbf{B}} \approx \mathbf{E}\bar{\mathbf{A}} \approx \mathbf{F}\mathbf{A}.$$
- *For $\mathbf{F} \leftarrow \mathcal{F}$, with overwhelming probability, all entries of the rightmost column $\mathbf{F}_r$ of $\mathbf{F}$ are non-zero.*

Theorem 2. *If a matrix distribution $\mathcal{D}_k$ on $\mathbb{Z}_q^{(k+1)\times k}$ is boostable to a matrix distribution $\mathcal{D}_{l,k}$ on $\mathbb{Z}_q^{l\times k}$ then the $\mathcal{D}_k$-MDDH assumption implies the $\mathcal{D}_{l,k}$-MDDH assumption.*

Proof. We prove this by a sequence of hybrids, where in the i-th hybrid we transform row $k+i$ from that of $[\mathbf{B}\mathbf{s}]$ to uniformly random. We start off with $i=0$, where we have the real output $[\mathbf{B}\mathbf{s}]$ and end with $i = l-k$ where we have the fake output which is uniformly random in $\mathbb{Z}_q^l$.

The i-th hybrid $([\mathbf{B}], [\mathbf{b}])$ is computed as follows. We sample $[\mathbf{A}]$ from $\mathcal{D}_k$ and $\mathbf{s}$ from $\mathbb{Z}_q^k$. We set $[\bar{\mathbf{B}}]$ as $[\bar{\mathbf{A}}]$ and, if $i \neq 0$, the row i of $[\underline{\mathbf{B}}]$ as the row i of $\mathbf{F}[\mathbf{A}]$. All other rows $j \neq i$ of $[\underline{\mathbf{B}}]$ are set to the j-th row of $\mathbf{E}[\bar{\mathbf{A}}]$. We set the top k elements of $[\mathbf{b}]$ to be $[\bar{\mathbf{A}}\mathbf{s}]$ and choose all the $(k+j)$-th elements, where $j < i$, of $[\mathbf{b}]$ uniformly at random from $\mathbb{Z}_q$. If $i \neq 0$, we set the $(k+i)$-th element of $[\mathbf{b}]$ to be the i-th element of $\mathbf{F}[\mathbf{A}\mathbf{s}]$. For all $j > i$, we set the $(k+j)$-th element of $[\mathbf{b}]$ to be the j-th element of $\mathbf{E}[\bar{\mathbf{A}}\mathbf{s}]$. To summarize, $[\mathbf{b}]$ is computed as:

$$\begin{bmatrix} [\bar{\mathbf{A}}\mathbf{s}] \\ \$ \\ \vdots \\ \$ \\ (\mathbf{F}[\mathbf{A}\mathbf{s}])_i \\ (\mathbf{E}[\bar{\mathbf{A}}\mathbf{s}])_{j=(i+1) \text{ to } (l-k)} \end{bmatrix}$$

We observe that the 0-th hybrid has the distribution of $([\mathbf{B}], [\mathbf{B}\mathbf{s}])$ and the $(l-k)$-th hybrid has the distribution of $([\mathbf{B}], [\mathbf{s}'])$, with $\mathbf{s}'$ uniform in $\mathbb{Z}_q^l$.

Now, $(\mathbf{F}[\mathbf{A}\mathbf{s}])_i = (\mathbf{F}_l)_i[\bar{\mathbf{A}}\mathbf{s}] + (\mathbf{F}_r)_i[\underline{\mathbf{A}}\mathbf{s}]$, where $\mathbf{F}_l$ is the first k-column submatrix of $\mathbf{F}$ and $\mathbf{F}_r$ is the last column of $\mathbf{F}$. Suppose we are given a $\mathcal{D}_k$-MDDH challenge $([\mathbf{A}], \chi = [\mathbf{A}\mathbf{s}]$ **or** $[\mathbf{s}'])$. If $\chi = [\mathbf{A}\mathbf{s}]$, then $(\mathbf{F}\chi)_i$ is distributed as $(\mathbf{F}[\mathbf{A}\mathbf{s}])_i$. Else, if $\chi = [\mathbf{s}']$, then $(\mathbf{F}\chi)_i$ is distributed uniformly randomly in $\mathbb{Z}_q$, since $(\mathbf{F}_r)_i$ is overwhelmingly non-zero by design. Next we transition to an intermediate hybrid i' where $[\mathbf{b}]$ is computed as:

$$\begin{bmatrix} [\bar{\mathbf{A}}\mathbf{s}] \\ \$ \\ \vdots \\ \$ \\ \$ \\ (\mathbf{E}[\bar{\mathbf{A}}\mathbf{s}])_{j=(i+1) \text{ to } (l-k)} \end{bmatrix}$$

As shown above, the hybrid i' is indistinguishable from hybrid i by the $\mathcal{D}_k$-MDDH assumption. Next we transition to the hybrid $i+1$ where $[\mathbf{b}]$ is computed as:

$$\begin{bmatrix} [\bar{\mathbf{A}}\mathbf{s}] \\ \$ \\ \vdots \\ \$ \\ \$ \\ (\mathbf{F}[\mathbf{A}\mathbf{s}])_{(i+1)} \\ (\mathbf{E}[\bar{\mathbf{A}}\mathbf{s}])_{j=(i+2) \text{ to } (l-k)} \end{bmatrix}$$

The hybrid $i+1$ is indistinguishable from hybrid i', as $\mathbf{E}\bar{\mathbf{A}}$ is identically distributed as $\mathbf{F}\mathbf{A}$. The theorem is thus established by chaining all the hybrids.

Corollary 1. *Any $\mathcal{D}_k$ distribution can be boosted to a $\mathcal{D}_{l,k}$ distribution which inherits the distribution of the top $k \times k$ matrix of the samples.*

This can be seen by setting the top $k \times k$ matrix of a $\mathcal{D}_{l,k}$ sample to be the top $k \times k$ matrix of a $\mathcal{D}_k$ sample and setting the bottom $(l-k) \times k$ sub-matrix of the $\mathcal{D}_{l,k}$ sample to be uniformly random in $\mathbb{Z}_q^{(l-k)\times k}$. The required distributions $\mathcal{E}$ and $\mathcal{F}$ are just the uniform distributions on their respective domains.

This corollary allows us to retain the *representation size* of the top square matrix of a $\mathcal{D}_k$ distribution sample, while boosting it to an assumption required for security proofs. In particular, in applications such as this paper, this can lead to shorter public keys.

Finally, observe that Theorem 2 and the justification of Corollary 1 establishes Theorem 1.

4 Smooth Quasi-Adaptive NIZK Constructions

In this section we show that the single element QA-NIZK [JR14,KW15] for witness-samplable linear subspaces can easily be extended to be smooth QA-NIZK. Particularly, under SXDH, the public hash key hp generated by $\mathsf{ver.hkgen}$ consists of a single group element. We follow the construction of Kiltz and Wee [KW15] and prove the result under the more general MDDH assumption in bilinear groups.

We follow additive notation for group operations in this section. In later sections we will use product notation.

Linear Subspace Languages. We first consider languages that are linear subspaces of vectors of $\mathbb{G}_1$ elements. In other words, the languages we are interested in can be characterized as languages parametrized by $[\mathbf{M}]_1$ as below:

$L_{[\mathbf{M}]_1} = \{[\mathbf{M}]_1\mathbf{x} \in \mathbb{G}_1^n \mid \mathbf{x} \in \mathbb{Z}_q^t\}$, where $[\mathbf{M}]_1$ is an $n \times t$ matrix of $\mathbb{G}_1$ elements.

Here $[\mathbf{M}]_1$ is an element of the associated *parameter language* Lpar, which is all $n \times t$ matrices of $\mathbb{G}_1$ elements. The parameter language Lpar also has a corresponding witness relation $\mathcal{R}_{\mathrm{par}}$, where the witness is a matrix of $\mathbb{Z}_q$ elements: $\mathcal{R}_{\mathrm{par}}([\mathbf{M}]_\infty, \mathbf{M}')$ iff $\mathbf{M} = \mathbf{M}'$.

Robust and Efficiently Witness-Samplable Distributions. Let the $t \times n$ dimensional matrix $[\mathbf{M}]_1$ be chosen according to a distribution $\mathcal{D}$ on Lpar. The distribution $\mathcal{D}$ is called *robust* if with probability close to one the left-most t columns of $[\mathbf{M}]_1$ are full-ranked. A distribution $\mathcal{D}$ on Lpar is called *efficiently witness-samplable* if there is a probabilistic polynomial time algorithm such that it outputs a pair of matrices $([\mathbf{M}]_1, \mathbf{M}')$ that satisfy the relation $\mathcal{R}_{\mathrm{par}}$ (i.e., $\mathcal{R}_{\mathrm{par}}([\mathbf{M}]_1, \mathbf{M}')$ holds), and further the resulting distribution of the output $[\mathbf{M}]_1$ is same as $\mathcal{D}$. For example, the uniform distribution on Lpar is efficiently witness-samplable, by first picking $\mathbf{M}$ at random, and then computing $[\mathbf{M}]_1$.

Smooth QA-NIZK Construction. We now describe a smooth computationally-sound Quasi-Adaptive NIZK ($\mathsf{pargen}, \mathsf{crsgen}, \mathsf{prover}, \mathsf{ver}$) for linear subspace languages $\{L_{[\mathbf{M}]_1}\}$ with parameters sampled from a robust and efficiently witness-samplable distribution $\mathcal{D}$ over the associated parameter language Lpar and given a $\mathcal{D}_k$-MDDH assumption.

crsgen: The crsgen algorithm generates the CRS as follows. Let $[\mathbf{M}^{n\times t}]_1$ be the parameter supplied to crsgen. It generates an $n \times k$ matrix $\mathbf{K}$ with all elements chosen randomly from $\mathbb{Z}_q$ and a $(k+1)\times k$ matrix $\mathbf{A}$ from the MDDH distribution $\mathcal{D}_k$. Let $\bar{\mathbf{A}}$ be the top $k \times k$ square matrix of $\mathbf{A}$.

The common reference string (CRS) has two parts CRS_p and CRS_v which are to be used by the prover and the verifier respectively.

$$\mathsf{CRS}_p^{t\times k} := ([\mathbf{P}]_1 = [\mathbf{M}^\top \mathbf{K}]_1) \qquad\qquad \mathsf{CRS}_v := ([\mathbf{C}]_2^{n\times k} = [\mathbf{K}\bar{\mathbf{A}}]_2, \quad [\bar{\mathbf{A}}]_2^{k\times k})$$

prover: Given candidate $[\mathbf{y}]_1 = [\mathbf{M}]_1\mathbf{x}$ with witness vector $\mathbf{x}^{t\times 1}$, the prover generates the following proof consisting of k elements in $\mathbb{G}_1$:

$$\pi := \mathbf{x}^\top \mathsf{CRS}_p$$

ver: The algorithm hkgen is as follows: Sample $\mathbf{s} \leftarrow \mathbb{Z}_q^k$. Given CRS_v as above, compute hk and hp as follows:

$$\mathsf{hk} := [\mathbf{C}]_2\ \mathbf{s}, \qquad \mathsf{hp} := [\bar{\mathbf{A}}]_2\ \mathbf{s}$$

The algorithms pubH and privH are as follows: Given candidate $[\mathbf{y}]_1$, and proof π, compute:

$$\mathsf{privH}(\mathsf{hk}, [\mathbf{y}]_1) := \mathsf{e}([\mathbf{y}^\top]_1, \mathsf{hk}) \qquad \mathsf{pubH}(\mathsf{hp}, \pi) := \mathsf{e}(\pi, \mathsf{hp})$$

Theorem 3. *The above algorithms* ($\mathsf{pargen}, \mathsf{crsgen}, \mathsf{prover}, \mathsf{ver}$) *constitute a smooth computationally -sound Quasi-Adaptive NIZK proof system for linear subspace languages* $\{L_{[\mathbf{M}]_1}\}$ *with parameters* $[\mathbf{M}]_1$ *sampled from a robust and efficiently witness-samplable distribution* $\mathcal{D}$ *over the associated parameter language* Lpar*, given any group generation algorithm for which the* $\mathcal{D}_k$−MDDH *assumption holds for group* $\mathbb{G}_2$*.*

The proofs of completeness, zero knowledge and soundness are same as [KW15]. The proof of smooth soundness follows.

Proof (Smooth Soundness). First, note that the range of privH is exponential in the security parameter, for otherwise an adversarial circuit can compute discrete logarithms with non-negligible probability. We prove smoothness by transforming the system over a sequence of games. Game $\mathbf{G_0}$ just replicates the construction, but samples $\mathbf{A}$ from a distribution $\mathcal{D}_{k+n-t,k}$ obtained by boosting the given distribution $\mathcal{D}_k$ by Theorem 1. The construction only uses the top $k \times k$ sub-matrix $\bar{\mathbf{A}}$ of the sample which is distributed identically for both $\mathcal{D}_k$ and $\mathcal{D}_{k+n-t,k}$. Let $\underline{\mathbf{A}}$ be the bottom $(n-t) \times k$ sub-matrix of $\mathbf{A}$.

In Game $\mathbf{G_1}$, the challenger efficiently samples $[\mathbf{M}]_1$ according to distribution $\mathcal{D}$, along with witness $\mathbf{M}$ (since $\mathcal{D}$ is an efficiently witness samplable distribution). Since $\mathbf{M}$ is an $n \times t$ dimensional rank t matrix, there is a rank $n-t$ matrix $\mathbf{M}^\perp$ of dimension $n \times (n-t)$ whose columns form a complete basis for the kernel of $\mathbf{M}^\top$, which means $\mathbf{M}^\top\mathbf{M}^\perp = \mathbf{0}^{t\times(n-t)}$. In this game, the NIZK CRS is computed as follows: Generate matrix $\mathbf{K}'^{\ n\times k}$ and compute the matrix $\mathbf{T}^{(n-t)\times k}$, such that $\mathbf{T}\bar{\mathbf{A}} = \underline{\mathbf{A}}$. Implicitly set: $\mathbf{K} = \mathbf{K}' + \mathbf{M}^\perp\mathbf{T}$. Therefore we have,

$$\begin{aligned} \mathsf{CRS}_p^{t\times k} &= [\mathbf{M}^\top\mathbf{K}]_1 = [\mathbf{M}^\top(\mathbf{K}' + \mathbf{M}^\perp\mathbf{T})]_1 = [\mathbf{M}^\top\mathbf{K}']_1 \\ [\mathbf{C}]_2^{n\times k} &= [(\mathbf{K}' + \mathbf{M}^\perp\mathbf{T})\bar{\mathbf{A}}]_2 = \mathbf{K}'[\bar{\mathbf{A}}]_2 + \mathbf{M}^\perp[\underline{\mathbf{A}}]_2, \\ \mathsf{hk} &= [\mathbf{C}]_2\ \mathsf{s}, \quad \mathsf{hp} = [\bar{\mathbf{A}}]_2\ \mathsf{s} \end{aligned}$$

In Game $\mathbf{G_2}$, we sample fresh random vectors $\mathbf{s}'$ in $\mathbb{Z}_q^k$ and $\mathbf{s}''$ in $\mathbb{Z}_q^{n-t}$ and modify the simulated computations as follows:

$$\begin{aligned} \mathsf{CRS}_p^{t\times k} &= [\mathbf{M}^\top\mathbf{K}']_1, \qquad [\mathbf{C}]_2^{n\times k} = \mathbf{K}'[\bar{\mathbf{A}}]_2 + \mathbf{M}^\perp[\underline{\mathbf{A}}]_2, \\ \mathsf{hk} &= \mathbf{K}'[\mathsf{s}']_2 + \mathbf{M}^\perp[\mathsf{s}'']_2, \quad \mathsf{hp} = [\mathsf{s}']_2 \end{aligned}$$

Given a $\mathcal{D}_{k+n-t,k}$ challenge which is either "real": $([\mathbf{A}]_2, [\bar{\mathbf{A}}\mathsf{s}]_2, [\underline{\mathbf{A}}\mathsf{s}]_2)$ or "fake": $([\mathbf{A}]_2, [\mathsf{s}']_2, [\mathsf{s}'']_2)$, we observe that the real tuple can be used to simulate Game $\mathbf{G_1}$, while the fake tuple can be used to simulate Game $\mathbf{G_2}$. Thus the games $\mathbf{G_1}$ and $\mathbf{G_2}$ are indistinguishable by the $\mathcal{D}_{k+n-t,k}$-MDDH assumption, which in turn is implied by the $\mathcal{D}_k$-MDDH assumption by Theorem 2.

Now in Game $\mathbf{G_2}$ we have,

$$\mathsf{privH}(\mathsf{hk}, [\mathbf{y}^*]_1) = \mathsf{e}\left([\mathbf{y}^{*\top}]_1, \mathbf{K}'[\mathbf{s}']_2 + \mathbf{M}^{\perp}[\mathbf{s}'']_2\right)$$

For the oracle queries where $[\mathbf{y}^*]_1 \in L_{[\mathbf{M}]_1}$, we have $\mathbf{y}^{*\top}\mathbf{M}^{\perp} = 0^{1\times(n-t)}$. Hence the simulator responds with $\mathsf{e}\left([\mathbf{y}^*]_1^{\top}, \mathbf{K}'[\mathbf{s}']_2\right)$. Note that $\mathbf{s}''$ does not appear in this response.

For the adversary supplied $[\mathbf{y}^*]_1 \notin L_{[\mathbf{M}]_1}$, we have $\mathbf{y}^{*\top}\mathbf{M}^{\perp} \neq 0^{1\times(n-t)}$. Therefore $\mathsf{privH}(\mathsf{hk}, \mathbf{y}^*)$ is uniformly random, as $\mathbf{s}''$ is independently random of everything else given to the adversary.

Smooth Split-CRS QA-NIZK for Tagged Affine Languages. QA-NIZKs for linear subspaces were also extended by [JR13] to integer tag-based languages as well as provided split-CRS[4] instantiation for affine languages. In [JR16], we combine all these extensions and describe a smooth computationally-sound Quasi-Adaptive NIZK $(\mathsf{pargen}, \mathsf{crsgen}, \mathsf{prover}, \mathsf{ver})$ for tagged affine linear subspace languages $\{L\}$, parametrized by $([\mathbf{M}_0]_1, [\mathbf{M}_1]_1, [\mathbf{M}_2]_1, [\mathbf{M}_3]_1, [\mathbf{a}]_1)$ and consisting of words of the form:

$$([\mathbf{M}_0\mathbf{x}]_1, [\mathbf{M}_1\mathbf{x} + \mathbf{a}]_1, [(\mathbf{M}_2 + \textsc{tag}.\mathbf{M}_3)\mathbf{x}]_1,\ \textsc{tag} \in \mathbb{Z}_q)\,,$$

with parameters sampled from a robust and efficiently witness-samplable distribution $\mathcal{D}$ over $(\mathbf{M}_0, \mathbf{M}_1, \mathbf{M}_2, \mathbf{M}_3, \mathbf{a})$ and given a $\mathcal{D}_k-$MDDH assumption. We assume that $\mathbf{M}_0$ is a square matrix and the robustness of $\mathcal{D}$ is defined by $\mathbf{M}_0$ being non-singular. The smooth QA-NIZK will be split-CRS [JR13], so that CRS_v is independent of the language parameters.

5 Asymmetric UC-PAKE: UC-APAKE

Based on the UC-PAKE functionality of [CHK+05], Gentry et al. [GMR06] gave another UC functionality for asymmetric PAKE (UC-APAKE). A salient feature of the UC-PAKE functionality [CHK+05] is that it models the security requirement that an adversary cannot perform efficient off-line computations on protocol transcripts to verifiably guess the low-entropy password. An adversary can only benefit from the low-entropy of the password by actually conducting an on-line attack (i.e. by impersonating one of the parties with a guessed password). This is modeled in the ideal world with a TestPwd capability available to the ideal world adversary: if TestPwd is called with the correct password, the ideal world adversary is allowed to set the session key. Moreover, in this functionality if any of the parties is corrupted, then the ideal world adversary is given the registered password.

[4] A split-CRS QA-NIZK allows the verifier CRS to be generated independent of the language.

5.1 The UC Ideal Functionality for Asymmetric PAKE

In asymmetric PAKE [GMR06], the ideal functionality also allows an adversary to steal the password file stored at the server (while not necessarily corrupting the server). However, this by itself does not directly provide the actual password to the adversary. However, after this point the adversary is allowed to perform $\mathsf{OfflineTestPwd}$ tests to mimic a similar capability in the real world (in fact, the ideal world adversary is even allowed to perform $\mathsf{OfflineTestPwd}$ tests before it steals the password file, but it does not get a confirmation of the guess being correct until after it steals the password file).

Moreover, after the "steal password file" event the adversary is also allowed to impersonate the server to a *correctly guessed* client, even without providing the actual password (as it can clearly do so in the real world). However, compromising impersonation of the client still requires providing a correct password. This differentiation in capabilities also becomes important when characterizing the complexity of a simulator in terms of the real world adversary, as we will see later.

The $\mathcal{F}_{\text{PAKE}}$ functionality for UC-PAKE was a single-session functionality. However, asymmetric PAKE requires that a password file be used across multiple sessions, so the $\mathcal{F}_{\mathsf{apwKE}}$ functionality for UC-APAKE is defined as a multiple-session functionality. Note that this cannot be accomplished simply using composition with joint state [CR03] because the functionality itself requires shared state that needs to be maintained between sessions. The complete UC-APAKE functionality $\mathcal{F}_{\mathsf{apwKE}}$ is described in detail in Fig. 2.

5.2 UC-APAKE Based on VPAKE and Smooth-NIZK

We now design an asymmetric UC-PAKE based on Verifier-based PAKE or VPAKE of Benhamouda and Pointcheval [BP13] and the novel Smooth NIZK proofs. The essential idea of [BP13] is that while the Client holds the actual password, the Server does not hold password in the clear. Instead the Server stores a hard to invert function called PHash (password hash) evaluated over the password and a random "salt" (PSalt) published in the CRS. While executing a session, the client sends encryptions of the password or another function called $\mathsf{PPreHash}$ (password pre-hash) evaluated on the password. Correspondingly, the server sends encryptions of the stored PHash.

Of course, some kind of zero-knowledge proof must accompany these encryptions, and to that end [BP13] can utilize the new smooth projective hash functions (SPHF) for CCA2-encryption [BBC+13] such as Cramer-Shoup encryption [CS02]. In each session, both parties generate fresh SPHF private and projection keys (to be employed on incoming messages). The projection key is sent (piggybacked) along with the encrypted message. If the encrypted messages use the correct password (meaning both parties have the same password or its PHash), then SPHF computed on the message by the receiving party using the SPHF hash key it generated equals the SPHF computed on the message by the sending party using the SPHF projection key it received. Thus, these SPHF hashes can be

Functionality $\mathcal{F}_{\mathsf{apwKE}}$

The functionality $\mathcal{F}_{\mathsf{apwKE}}$ is parameterized by a security parameter k. It interacts with an adversary S and a set of parties via the following queries:

Password Storage and Authentication Sessions

Upon receiving a query $(\mathsf{StorePwdFile}, \boldsymbol{sid}, P_i, pw)$ **from party** P_j**:**
If this is the first StorePwdFile query, store password data record $(\mathsf{file}, P_i, P_j, pw)$ and mark it uncompromised.

Upon receiving a query $(\mathsf{CltSession}, \boldsymbol{sid}, \mathsf{ssid}, P_i, P_j, pw)$ **from party** P_i**:**
Send $(\mathsf{CltSession}, sid, \mathsf{ssid}, P_i, P_j)$ to S. In addition, if this is the first CltSession query for ssid, then store session record $(\mathsf{Clt}, \mathsf{ssid}, P_i, P_j, pw)$ and mark this record fresh.

Upon receiving a query $(\mathsf{SvrSession}, \boldsymbol{sid}, \mathsf{ssid}, P_i)$ **from party** P_j**:**
If there is a password data record (file, P_i, P_j, pw), then send (SvrSession, sid, ssid, P_j, P_i) to S, and if this is the first SvrSession query for ssid, store session record (Svr, ssid, P_j, P_i, pw), and mark it fresh.

Stealing Password Data

Upon receiving a query $(\mathsf{StealPwdFile}, \boldsymbol{sid})$ **from adversary** S**:**
If there is no password data record reply to S with 'no password file'. Otherwise, do the following: If the password data record $(\mathsf{file}, P_i, P_j, pw)$ is marked uncompromised, mark it compromised. If there is a tuple $(\mathsf{offline}, pw')$ stored with $pw' = pw$ then send pw to S, otherwise reply to S with 'password file stolen'.

Upon receiving a query $(\mathsf{OfflineTestPwd}, \boldsymbol{sid}, pw')$ **from Adversary** S**:**
If there is no password data record, or if there is a password data record $(\mathsf{file}, P_i, P_j, pw)$ that is marked uncompromised, then store $(\mathsf{offline}, pw')$. Otherwise do: if $pw = pw'$, send pw back to S. If $pw \neq pw'$, reply with 'wrong guess'.

Active Session Attacks

Upon receiving a query $(\mathsf{TestPwd}, \boldsymbol{sid}, \mathsf{ssid}, P_i, pw')$ **from the adversary** S**:**
If there is a session record of the form $(\mathsf{role}, \mathsf{ssid}, P_i, P_j, pw)$ which is fresh, then do: If $pw = pw'$, mark the record compromised and reply to S with "correct guess". If $pw \neq pw'$, mark the record interrupted and reply with "wrong guess".

Upon receiving a query $(\mathsf{Impersonate}, \boldsymbol{sid}, \mathsf{ssid})$
If there is a session record of the form $(\mathsf{Clt}, \mathsf{ssid}, P_i, P_j, pw)$ which is fresh, then do: then if there is a password data record file $(\mathsf{file}, P_i, P_j, pw)$ that is marked compromised, mark the session record compromised and reply to S with 'correct guess', else mark the session record interrupted and reply with wrong guess.

Key Generation and Authentication

Upon receiving a query $(\mathsf{NewKey}, \boldsymbol{sid}, \mathsf{ssid}, P_i, sk)$ **from** S**, where** $|sk| = k$**:**
If there is a session record of the form $(\mathsf{role}, \mathsf{ssid}, P_i, P_j, pw)$ that is not marked completed,

- If this record is compromised, or either P_i or P_j is corrupted, then output (sid, ssid, sk) to player P_i.
- If this record is fresh, and there is a session record $(\mathsf{role}, \mathsf{ssid}, P_j, P_i, pw')$ with $pw' = pw$, and a key sk' was sent to P_j, and $(\mathsf{role}, \mathsf{ssid}, P_j, P_i, pw)$ was fresh at the time, then output $(sid, \mathsf{ssid}, sk')$ to P_i.
- In any other case, pick a new random key sk' of length k and send $(sid, \mathsf{ssid}, sk')$ to P_i.

Either way, mark the record (P_i, P_j, pw) as completed.

Upon receiving $(\mathsf{Corrupt}, \boldsymbol{sid}, P)$ **from** S**:** if there is a $(\mathsf{Clt}, sid, P, P', pw)$ recorded, return pw to S, and mark P_i corrupted. If there is a $(\mathsf{Svr}, sid, P, P', pw)$ recorded, then mark P corrupted and (internally) call $(\mathsf{StealPwdFile}, sid)$.

Fig. 2. The password-based key-exchange functionality $\mathcal{F}_{\mathsf{apwKE}}$

used to compute the session key. Smoothness property of the SPHF guarantees security of the VPAKE scheme.

Unfortunately, each party must retain the witness used in the CCA2 encryption, as computing the SPHF projection-hash of its outgoing encrypted message

Generate $\mathbf{g} \leftarrow \mathbb{G}_1$; $a_1, a_2, b_\mathsf{C}, b_\mathsf{S} \leftarrow \mathbb{Z}_q$ and let $\rho = \{\mathbf{a}_1 = \mathbf{g}^{a_1}, \mathbf{a}_2 = \mathbf{g}^{a_2}, \mathbf{b}_\mathsf{C} = \mathbf{g}^{b_\mathsf{C}}, \mathbf{b}_\mathsf{S} = \mathbf{g}^{b_\mathsf{S}}\}$.

Define languages $\begin{bmatrix} L_\mathsf{C} = \{(R, S, H) \mid \exists r, p : R = \mathbf{g}^r, S = \mathbf{a}_1^r \mathbf{b}_\mathsf{C}^p, H = \mathbf{b}_\mathsf{S}^p\} \\ L_\mathsf{S} = \{(R, S) \mid \exists r : R = \mathbf{g}^r, S = \mathbf{a}_2^r\} \end{bmatrix}$

Let $(\mathsf{hk}_\mathsf{C}, \mathsf{hp}_\mathsf{C}) \leftarrow \mathsf{sphf}(L_\mathsf{C}).\mathsf{hkgen}$ and $(\mathsf{hp}_\mathsf{S}, \mathsf{hk}_\mathsf{S}) \leftarrow \mathsf{sphf}(L_\mathsf{S}).\mathsf{hkgen}$.

Define languages:

$$\begin{bmatrix} L_\mathsf{C}^+ = \{(R, S, H, T, l) \mid \exists r, p : R = \mathbf{g}^r, S = \mathbf{a}_1^r \mathbf{b}_\mathsf{C}^p, H = \mathbf{b}_\mathsf{S}^p, T = \mathsf{sphf.pubH}(\mathsf{hp}_\mathsf{C}, \langle R, S, H\rangle, l; r, p)\} \\ L_\mathsf{S}^+ = \{(R, S, T, l) \mid \exists r : R = \mathbf{g}^r, S = \mathbf{a}_2^r, T = \mathsf{sphf.pubH}(\mathsf{hp}_\mathsf{S}, \langle R, S\rangle, l; r)\} \end{bmatrix}$$

Let $(\mathsf{pargen}_P, \mathsf{crsgen}_P, \mathsf{prover}_P, \mathsf{ver}_P)$ be Smooth QA-NIZKs for languages L_P^+, with $P \in \{C, S\}$.
Let $\textsc{crs}_P \leftarrow \mathsf{crsgen}_P(\rho)$ and $\mathcal{H}$ be a collision resistant hash function.
Let $\mathcal{RO}$ be a random oracle and let $\mathsf{phash} = \mathcal{RO}(sid, P_i, P_j, \mathsf{pwd})$.
Note that there are several sessions, designated by unique ssid-s, within the scope of a single *sid*. Thus, phash is the same across all these sessions.

$$\mathrm{CRS} := (\rho, \mathsf{hp}_\mathsf{C}, \mathsf{hp}_\mathsf{S}, \textsc{crs}_\mathsf{C}, \textsc{crs}_\mathsf{S}, \mathcal{H}).$$

$$\text{Server Persistent State} := \mathbf{b}_\mathsf{S}^{\mathsf{phash}}.$$

Client P_i	Network
Input $(\mathsf{CltSession}, sid, \mathsf{ssid}, P_i, P_j, \mathsf{pwd})$. Choose $r_1 \leftarrow \mathbb{Z}_q$ and $(\textsc{hk}_1, \textsc{hp}_1) \leftarrow \mathsf{ver}_\mathsf{S}.\mathsf{hkgen}(\textsc{crs}_\mathsf{S})$. Set $R_1 = \mathbf{g}^{r_1}$, $S_1 = \mathbf{a}_1^{r_1} \mathbf{b}_\mathsf{C}^{\mathsf{phash}}$, $T_1 = \mathsf{sphf}_\mathsf{C}.\mathsf{pubH}(\mathsf{hp}_\mathsf{C}, \langle R_1, S_1, \mathbf{b}_\mathsf{S}^{\mathsf{phash}}\rangle, i_1; r_1, \mathsf{phash})$, $W_1 = \mathsf{prover}_\mathsf{C}(\textsc{crs}_\mathsf{C}, \langle R_1, S_1, \mathbf{b}_\mathsf{S}^{\mathsf{phash}}, T_1, i_1\rangle; r_1, \mathsf{phash})$, where $i_1 = \mathcal{H}(sid, \mathsf{ssid}, P_i, P_j, R_1, S_1, \textsc{hp}_1)$. Erase r_1, send $(R_1, S_1, T_1, \textsc{hp}_1)$ and retain $(W_1, \textsc{hk}_1)$.	$\xrightarrow{R_1, S_1, T_1, \mathrm{HP}_1} P_j$
Receive $(R_2', S_2', T_2', \textsc{hp}_2')$. If any of $R_2', S_2', T_2', \textsc{hp}_2'$ is not in their respective group or is 1, set $\mathrm{sk}_1 \xleftarrow{\$} \mathbb{G}_T$, else compute $i_2' = \mathcal{H}(sid, \mathsf{ssid}, P_j, P_i, R_2', S_2', \textsc{hp}_2')$, and $\mathrm{sk}_1 = \mathsf{ver}_\mathsf{S}.\mathsf{privH}(\textsc{hk}_1, \langle R_2', S_2'/\mathbf{b}_\mathsf{S}^{\mathsf{phash}}, T_2', i_2'\rangle) \cdot \mathsf{ver}_\mathsf{C}.\mathsf{pubH}(\textsc{hp}_2', W_1)$. Output $(sid, \mathsf{ssid}, \mathrm{sk}_1)$.	$\xleftarrow{R_2', S_2', T_2', \mathrm{HP}_2'} P_j$
Server P_j	Network
Input $(\mathsf{SvrSession}, sid, \mathsf{ssid}, P_j, P_i, \text{Server Persistent State})$. Choose $r_2 \leftarrow \mathbb{Z}_q$ and $(\textsc{hk}_2, \textsc{hp}_2) \leftarrow \mathsf{ver}_\mathsf{C}.\mathsf{hkgen}(\textsc{crs}_\mathsf{C})$. Set $R_2 = \mathbf{g}^{r_2}$, $S_2 = \mathbf{a}_2^{r2} \mathbf{b}_\mathsf{S}^{\mathsf{phash}}$, $T_2 = \mathsf{sphf}_\mathsf{S}.\mathsf{pubH}(\mathsf{hp}_\mathsf{S}, \langle R_2, S_2/\mathbf{b}_\mathsf{S}^{\mathsf{phash}}\rangle, i_2; r_2)$, $W_2 = \mathsf{prover}_\mathsf{S}(\textsc{crs}_\mathsf{S}, \langle R_2, S_2/\mathbf{b}_\mathsf{S}^{\mathsf{phash}}, T_2, i_2\rangle; r_2)$, where $i_2 = \mathcal{H}(sid, \mathsf{ssid}, P_j, P_i, R_2, S_2, \textsc{hp}_2)$. Erase r_2, send $(R_2, S_2, T_2, \textsc{hp}_2)$ and retain $(W_2, \textsc{hk}_2)$.	$\xrightarrow{R_2, S_2, T_2, \mathrm{HP}_2} P_i$
Receive $(R_1', S_1', T_1', \textsc{hp}_1')$. If any of $R_1', S_1', T_1', \textsc{hp}_1'$ is not in their respective group or is 1, set $\mathrm{sk}_2 \xleftarrow{\$} \mathbb{G}_T$, else compute $i_1' = \mathcal{H}(sid, \mathsf{ssid}, P_i, P_j, R_1', S_1', \textsc{hp}_1')$, and $\mathrm{sk}_2 = \mathsf{ver}_\mathsf{C}.\mathsf{privH}(\textsc{hk}_2, \langle R_1', S_1', \mathbf{b}_\mathsf{S}^{\mathsf{phash}}, T_1', i_1'\rangle) \cdot \mathsf{ver}_\mathsf{S}.\mathsf{pubH}(\textsc{hp}_1', W_2)$. Output $(sid, \mathsf{ssid}, \mathrm{sk}_2)$.	$\xleftarrow{R_1', S_1', T_1', \mathrm{HP}_1'} P_i$

Fig. 3. Single round RO-hybrid UC-APAKE protocol under SXDH assumption.

using the received projection key requires this witness. In the strong simulation paradigm of universally composable security, this leads to a problem if an Adversary can corrupt a session dynamically after the outgoing message has been sent and the incoming message has not yet been received. Thus, this SPHF methodology can only handle static corruption. While Jutla and Roy [JR15] have recently given an efficient UC-PAKE protocol which can handle dynamic corruption, the construction uses ideas from dual-system simulation-sound QA-NIZK that they introduce there. These ideas are rather intricate and do not seem to allow a modular or generic design of such UC password-authenticated protocols.

In this paper, we show that the new notion of Smooth QA-NIZK allows easy to understand (and equally efficient) modular or generic design. Just as QA-NIZK proofs can be seen as generalization of projective hash proof systems to public verifiability (and also assuring zero-knowledge), the novel notion of Smooth QA-NIZK naturally generalizes the notion of smooth projective hash functions where instead of the witness, the publicly verifiable proof can be used to evaluate the projection-hash. The zero-knowledge property of this publicly verifiable proof assures that this proof and hence the projection-hash can be generated by a simulator with no access to the witness. In particular, each party in the UC-PAKE protocol can generate an encryption of the password and generate this publicly verifiable QA-NIZK proof, send the encryption to the other party, erase the witness and retain just the proof for later generation of session key.

The natural question that arises is whether one needs a notion of smooth-soundness under simulation. Indeed, one does need some form of unbounded simulation-soundness as the UC simulator generates QA-NIZK proofs on non-language members without access to the password. Unfortunately, the recent efficient unbounded simulation sound QA-NIZK construction of [KW15] does not extend to be smooth under unbounded simulation (or at least current techniques do not seem to allow one to prove so). The dual-system simulation sound QA-NIZK [JR15] does satisfy smoothness property, but it would need introduction of various new intricate definitions and complicated proofs. One may also ask whether CCA2 encryption by itself provides the required simulation soundness, but that is also not the case, as CCA2 encryption by itself does not give a privately-verifiable (say, via its underlying SPHF as in Cramer-Shoup encryption) proof that it is the password that is being encrypted.

In light of this, it turns out that the simplest way to design the UC-APAKE (or UC-PAKE) protocol is to use an ElGamal encryption of the password (or its $\mathsf{PPreHash}$ or PHash) and augment it with an SPHF proof of its consistency, and finally a Smooth QA-NIZK on this augmented ElGamal encryption. (If the reader is interested in the simpler UC-PAKE protocol secure under dynamic corruption in the new Smooth QA-NIZK framework, the UC-PAKE definition and protocol are provided in [JR16]).

We will also need the random oracle hybrid model to achieve the goal of a UC-APAKE protocol, as explained next. The focus of [BP13] was to design protocols which can be proven secure in the standard model. They formalized a security notion for APAKEs modifying the game-based BPR model [BPR00]. However, our focus is to construct an APAKE protocol in the UC model. In the UC model of [GMR06], the UC simulator must be able to detect offline password guess attempts of the adversary. This is not possible in the standard model as offline tests can be internally performed by the adversary. In order to intercept offline tests by the adversary, it thus becomes inevitable to use an idealized model, such as the random oracle model.

So in particular, we adapt the random oracle-based password hashing scheme of [BP13]. In the scheme, the public parameters are $\mathit{param} = \mathbf{b}_{\mathsf{c}}, \mathbf{b}_{\mathsf{s}}$ randomly sampled from $\mathbb{G}_1$ and a random oracle $\mathcal{RO}$. Define $\mathsf{phash} = \mathcal{RO}(sid,$ Client-id, Server-id, $\mathsf{pwd})$, where Client-id, Server-id are the ids of the participating parties, *sid* is the common session-id for all sessions between these parties and pwd is the password of the client. Note that there are several sessions designated by unique ssid-s within the scope of a single *sid*. Thus phash is the same across all these sessions. We set:

$$\begin{aligned}&\mathsf{PPreHash}(param, \mathsf{pwd}) = \mathbf{b}_{\mathsf{c}}^{\mathsf{phash}}\\&\mathsf{PSalt}(param) = \mathbf{b}_{\mathsf{s}}\\&\mathsf{PHash}(param, \mathsf{pwd}) = \mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}\end{aligned}$$

Corresponding to the asymmetric storages of the client and the server, we define the following languages, one for each party, which implicitly check the consistency of correct elements being used ($\mathbf{a}_1$ and $\mathbf{a}_2$ are essentially public keys for ElGamal encryption):

$$\begin{aligned}L_{\mathsf{c}} &= \{(R, S, H) \mid \exists r, p : R = \mathbf{g}^r, S = \mathbf{a}_1^r \mathbf{b}_{\mathsf{c}}^p, H = \mathbf{b}_{\mathsf{s}}^p\}\\L_{\mathsf{s}} &= \{(R, S) \mid \exists r : R = \mathbf{g}^r, S = \mathbf{a}_2^r\}\end{aligned}$$

We now plug these languages into UC-PAKE methodology described above. The client sends ElGamal encryption of $\mathbf{b}_{\mathsf{c}}^p$, as in (R, S) of L_{c}, while the server supplies the last element H for forming a word of L_{c}. The server sends ElGamal encryption of $\mathbf{b}_{\mathsf{s}}^p$, while the client divides out $\mathbf{b}_{\mathsf{s}}^p$ from the second component to form a word of L_{s}.

The CRS provides public smooth$_2$ SPHF keys for the languages L_{c} and L_{s}, which are used by the client and server respectively to compute T_1 and T_2 for their flows.

Lastly, we use Smooth QA-NIZK proofs for generating a public hash key and a private hash key over the above languages augmented with the SPHFs as below:

$$\begin{aligned}L_{\mathsf{c}}^{+} &= \left\{ (R, S, H, T, l) \mid \exists r, p : \begin{array}{c} R = \mathbf{g}^r, S = \mathbf{a}_1^r \mathbf{b}_{\mathsf{c}}^p, H = \mathbf{b}_{\mathsf{s}}^p, \\ T = \mathsf{sphf.pubH}(\mathsf{hp}_{\mathsf{c}}, \langle R, S, H \rangle, l; r, p) \end{array} \right\}\\L_{\mathsf{s}}^{+} &= \{(R, S, T, l) \mid \exists r : R = \mathbf{g}^r, S = \mathbf{a}_2^r, T = \mathsf{sphf.pubH}(\mathsf{hp}_{\mathsf{s}}, \langle R, S \rangle, l; r)\}\end{aligned}$$

The client generates a Smooth QA-NIZK verification key pair for the server language L_{s}^{+}, retains the private key HK_1 and sends the public key HP_1 along with the ElGamal encryption and the SPHF. The client computes a QA-NIZK proof W_1 of $(R_1, S_1, \mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}, T_1) \in L_{\mathsf{c}}^{+}$ with label $i_1 = \mathcal{H}(sid, \mathsf{ssid}, P_i, P_j, R_1, S_1, T_1, \text{HP}_1)$ and retains that for later key computation.

Similarly, the server generates a Smooth QA-NIZK verification key pair for the client language L_{c}^{+}, retains the private key HK_2 and sends the public key HP_2 along with the ElGamal encryption and the SPHF. The server computes a QA-NIZK proof W_2 of $(R_2, S_2/\mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}, T_2) \in L_{\mathsf{s}}^{+}$ with label $i_2 = \mathcal{H}(sid, \mathsf{ssid}, P_j, P_i, R_2, S_2, T_2, \text{HP}_2)$ and retains that for later key computation.

In the second part of the protocol, after receiving the peer flow, each party computes the final secret key as the product of the private Smooth QA-NIZK hash of the peer flow with own private Smooth QA-NIZK key and the public Smooth QA-NIZK hash of the (retained) QA-NIZK proof of own flow with the peer public Smooth QA-NIZK hash key. Formally the client computes:

$$\mathsf{ver}_\mathsf{s}.\mathsf{privH}(\textsc{hk}_1, \langle R'_2, S'_2/\mathbf{b}_\mathsf{s}^{\mathsf{phash}}, T'_2, i'_2\rangle) \cdot \mathsf{ver}_\mathsf{c}.\mathsf{pubH}(\textsc{hp}'_2, W_1).$$

Similarly, the server computes:

$$\mathsf{ver}_\mathsf{c}.\mathsf{privH}(\textsc{hk}_2, \langle R'_1, S'_1, \mathbf{b}_\mathsf{s}^{\mathsf{phash}}, T'_1, i'_1\rangle) \cdot \mathsf{ver}_\mathsf{s}.\mathsf{pubH}(\textsc{hp}'_1, W_2).$$

Given the completeness property of the Smooth QA-NIZK, it is not difficult to see that legitimately completed peer sessions end up with equal keys. In the next section, we prove that this protocol securely realizes $\mathcal{F}_{\mathsf{apwKE}}$, as stated in the theorem below.

The complete protocol is described in detail in Fig. 3. The SPHF sphf is required to be a smooth$_2$ projective hash function (see [JR16] for definitions). For simplicity, in this paper we focus on constructions based on $\mathcal{D}_1-$MDDH assumptions, and in particular the SXDH assumption.

Theorem 4. *Under the $\mathcal{D}_1-$MDDH assumption SXDH, the protocol in Fig. 3 securely realizes the $\mathcal{F}_{\mathsf{apwKE}}$ functionality in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{RO}})$-hybrid model, in the presence of adaptive corruption adversaries. The number of unique password arguments passed to* TestPwd *and* OfflineTestPwd *of $\mathcal{F}_{\mathsf{apwKE}}$ combined in the ideal world is at most the number of random oracle calls in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{RO}})$-hybrid world.*

We describe the intuition of the proof below and describe the UC simulator, while detailed formal steps proving indistinguishability of the real and the ideal world are relegated to the full version [JR16].

5.3 Main Idea of the UC Simulator

The UC simulator $\mathcal{S}$works as follows: It simulates the random oracle calls and records all the query response pairs. It will generate the CRS for $\widehat{\mathcal{F}}_{\text{PAKE}}$ using the real world algorithms, except for the Smooth QA-NIZK, for which it uses the simulated CRS generator. It also retains the private hash keys of the SPHF's. The next main difference is in the simulation of the outgoing message of the real world parties: $\mathcal{S}$ uses a dummy message μ instead of the real password which it does not have access to. Further, it postpones computation of W till the session-key generation time. Finally, another difference is in the processing of the incoming message, where $\mathcal{S}$ decrypts the incoming message R'_2, S'_2 and runs through the list of random oracle queries to search for a pwd', such that the decryption is $\mathbf{b}_\mathsf{s}^{\mathcal{RO}(sid, P_i, P_j, \mathsf{pwd}')}$, which it uses to call the ideal functionality's test function. It next generates an sk similar to how it is generated in the real-world. It sends sk to the ideal functionality to be output to the party concerned.

Since the (R_1, S_1) that it sends out is no longer such that $(R_1, S_1, \mathbf{b}_\mathsf{s}^{\mathsf{phash}})$ in the language L_c, it has to use the private key of the SPHF in order to compute T_1 on $(R_1, S_1, \mathbf{b}_\mathsf{s}^{\mathsf{phash}})$ and the QA-NIZK proof simulator to compute W_1.

There are other special steps designed to simulate stealing the password file and then impersonating the server to the client. Specifically, when the password file is stolen, the simulator still may not know pwd. It then preemptively sets phash to a random value and pretends that this is the random oracle response with the correct pwd query. Later on when there is a successful pwd query, which the simulator can find out by the online or offline testpwd ideal functionality calls, it sets the record accordingly.

In case of a stolen password file, the simulator includes a "Client Only Step" which lets it test (modified) server flows for consistency and call the Impersonate functionality if consistency checks out. The server simulation steps do not include such a step to model the security notion that even if the password file is stolen, the adversary should still not be able to impersonate the client.

5.4 Main Idea of the Proof of UC Realization

The proof that the simulator $\mathcal{S}$ described above simulates the Adversary in the real-world protocol, follows essentially from the properties of the Smooth QA-NIZK and smooth$_2$ SPHF, and we give a broad outline here. The proof will describe various experiments between a challenger $\mathcal{C}$ and the adversary, which we will just assume to be the environment $\mathcal{Z}$ (as the adversary $\mathcal{A}$ can be assumed to be just dummy and following $\mathcal{Z}$'s commands). In the first experiment the challenger $\mathcal{C}$ will just be the combination of the code of the simulator $\mathcal{S}$ above and $\widehat{\mathcal{F}}_{\text{PAKE}}$. In particular, after the environment issues a $\mathsf{CltSession}$ request with a password pwd, the challenger gets that password. So, while in the first experiment, the challenger (copying $\mathcal{S}$) does not use pwd directly, from the next experiment onwards, it can use pwd. Thus, the main goal of the ensuing experiments is to modify the fake tuples $\mathbf{g}^{r_1}, \mathbf{g}^{r'}$ by real tuples (as in real-world) $\mathbf{g}^{r_1}, \mathbf{a}_1^{r_1}\mathbf{b}_{\mathsf{c}}^{\mathsf{phash}}$, since the challenger has access to pwd, and hence phash. This is accomplished by a hybrid argument, modifying one instance at a time using DDH assumption in group $\mathbb{G}_1$.

The guarantee that the client cannot be impersonated by the adversary, even when the password file is stolen is established by noting that $\mathbf{b}_{\mathsf{c}}^{\mathsf{phash}}$, which is what the client encrypts in its flows, is hard to compute given the server persistent state $\mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}$. This is formally captured in the proof by using a DDH transition from $(\mathbf{b}_{\mathsf{s}}, \mathbf{b}_{\mathsf{c}}, \mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}, \mathbf{b}_{\mathsf{c}}^{\mathsf{phash}})$ to $(\mathbf{b}_{\mathsf{s}}, \mathbf{b}_{\mathsf{c}}, \mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}, \mathbf{b}_{\mathsf{c}}^{z})$, where z is independently random from phash.

Once all the instances are corrected, i.e. R_1, S_1 are generated as $\mathbf{g}^{r_1}, \mathbf{a}_1^{r_1}\mathbf{b}_{\mathsf{c}}^{\mathsf{phash}}$, the challenger can switch to the real-world because the tuples $R_1, S_1, \mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}$ are now in the language L_{c}. This implies that the session keys are generated exactly as in the real-world.

5.5 Adaptive Corruption

The UC protocol described above is also UC-secure against adaptive corruption of parties by the Adversary in the erasure model. In the real-world when the adversary corrupts a client (with a $\mathsf{Corrupt}$ command), it gets the internal state of the client. Clearly, if the party has already been invoked with a $\mathsf{CltSession}$ command then the password pwd is leaked at the minimum, and hence the ideal functionality $\mathcal{F}_{\text{PAKE}}$ leaks the password to the Adversary in the ideal world. In the protocol described above, the Adversary also gets W_1 and HK_1, as this is the only state maintained by each client between sending $R_1, S_1, T_1, \text{HP}_1$, and the final issuance of session-key. Simulation of HK_1 is easy for the simulator $\mathcal{S}$ since $\mathcal{S}$ generates HK_1 exactly as in the real world. For generating W_1, which $\mathcal{S}$ had postponed to computing till it received an incoming message from the adversary, it can now use the pwd which it gets from $\widehat{\mathcal{F}}_{\text{PAKE}}$ by issuing a $\mathsf{Corrupt}$ call to $\widehat{\mathcal{F}}_{\text{PAKE}}$. More precisely, it issues the $\mathsf{Corrupt}$ call, and gets pwd, and then calls the QA-NIZK simulator with the tuple $(R_1, S_1, \mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}, T_1, i_1)$ to get W_1. Note that this computation of W_1 is identical to the postponed computation of W_1 in the computation of client factor of sk_1 (which is really used in the output to the environment when $\mathsf{pwd}' = \mathsf{pwd}$).

In case of server corruption, the simulator does not get pwd, but is able to set phash which also enables it to compute W_2 using the QA-NIZK simulator on $(R_2, S_2/\mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}, T_2, i_2)$.

We first define a simulator which interfaces with the ideal functionality and the adversary and then through a series of experiments convert it to just the real world protocol interacting with the same adversary.

5.6 Simulator for the Protocol

We will assume that the adversary $\mathcal{A}$ in the UC protocol is dummy, and essentially passes back and forth commands and messages from the environment $\mathcal{Z}$. Thus, from now on we will use environment $\mathcal{Z}$ as the real adversary, which outputs a single bit. The simulator $\mathcal{S}$ will be the ideal world adversary for $\mathcal{F}_{\mathsf{apwKE}}$. It is a universal simulator that uses $\mathcal{A}$ as a black box. For each instance (session and a party), we will use a prime, to refer to variables received in the message from $\mathcal{Z}$ (i.e. $\mathcal{A}$). We will call a message *legitimate* if it was not altered by $\mathcal{Z}$, and delivered in the correct session and to the correct party.

Responding to Random Oracle Queries. Let the input be m. If there is a record of the form (m, r), that is, m was queried before and was responded with r, then just return r.

Otherwise, if m is of the form (sid, P_i, P_j, x), for some x and the password file has been stolen then call $\mathsf{OfflineTestPwd}$ with x. If the test succeeds then return phash, which must already have been set (see Stealing Password File below), and record (m, phash).

In all other cases, generate $r \leftarrow \mathbb{Z}_q$, record (m, r) and return r.

Setting the CRS. The simulator $\mathcal{S}$ picks the CRS just as in the real world, except the QA-NIZK CRS-es are generated using the crs-simulators, which also generate simulator trapdoors $\mathsf{trap}_{\mathsf{c}}, \mathsf{trap}_{\mathsf{s}}$. It retains $a_1, a_2, \mathsf{trap}_{\mathsf{c}}, \mathsf{trap}_{\mathsf{s}}, \mathsf{hk}_{\mathsf{c}}, \mathsf{hk}_{\mathsf{s}}$ as trapdoors.

New Client Session: Sending a Message to $\mathcal{Z}$. On message ($\mathsf{CltSession}$, *sid*, ssid, P_i, P_j) from $\mathcal{F}_{\mathsf{apwKE}}$, $\mathcal{S}$ starts simulating a new instance of the protocol for client P_i, server P_j, session identifier ssid, and CRS set as above. We will denote this instance by (P_i, ssid) and call it a *client instance*.

To simulate this instance, $\mathcal{S}$ chooses r_1, r_1', r_1'' at random, and sets $R_1 = \mathbf{g}^{r_1}$, $S_1 = \mathbf{g}^{r_1'}$ and $T_1 = \mathbf{g}^{r_1''}$. Next, $\mathcal{S}$ generates $(\text{HK}_1, \text{HP}_1) \leftarrow \mathsf{ver_s.hkgen}(\text{CRS}_\mathsf{s})$ and sets $i_1 = \mathcal{H}(sid, \mathsf{ssid}, P_i, P_j,\ R_1, S_1, \text{HP}_1)$. It retains (i_1, HK_1). It then hands $(R_1, S_1, T_1, \text{HP}_1)$ to $\mathcal{Z}$ on behalf of this instance.

New Server Session: Sending a Message to $\mathcal{Z}$. On message ($\mathsf{SvrSession}$, *sid*, ssid, P_j, P_i) from $\mathcal{F}_{\mathsf{apwKE}}$, $\mathcal{S}$ starts simulating a new instance of the protocol for client P_i, server P_j, session identifier ssid, and CRS set as above. We will denote this instance by (P_j, ssid) and call it a *server instance*.

To simulate this instance, $\mathcal{S}$ chooses r_2, r_2', r_2'' at random, and sets $R_2 = \mathbf{g}^{r_2}$, $S_2 = \mathbf{g}^{r_2'}$ and $T_2 = \mathbf{g}^{r_2''}$. Next, $\mathcal{S}$ generates $(\text{HK}_2, \text{HP}_2) \leftarrow \mathsf{ver_c.hkgen}(\text{CRS}_\mathsf{c})$ and sets $i_2 = \mathcal{H}(sid, \mathsf{ssid}, P_j, P_i,\ R_2, S_2, \text{HP}_2)$. It retains (i_2, HK_2). It then hands $(R_2, S_2, T_2, \text{HP}_2)$ to $\mathcal{Z}$ on behalf of this instance.

On Receiving a Message from $\mathcal{Z}$. On receiving a message $R_2', S_2', T_2', \text{HP}_2'$ from $\mathcal{Z}$ intended for a **client instance** (P, ssid), the simulator S does the following:

1. If any of the the real world protocol checks, namely group membership and non-triviality fail it goes to the step "Other Cases"below.
2. If the message received from $\mathcal{Z}$ is same as message sent by $\mathcal{S}$ on behalf of peer P' in session ssid, then $\mathcal{S}$ just issues a NewKey call for P.
3. ("Client Only Step"): If $\mathsf{StealPwdFile}$ has already taken place then do the following: If $S_2' = R_2'^{a_2} \mathbf{b}_\mathsf{s}^{\mathsf{phash}}$, then $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{apwKE}}$ with $(\mathsf{Impersonate}, P, sid, \mathsf{ssid})$ and skips to the "Key Setting" step below, and otherwise go to the step "Other Cases".
4. It searches its random oracle query response pairs $\{(m_k, h_k)\}_k$ and checks whether for some $k = x$ we have $S_2' = R_2'^{a_2} \mathbf{b}_\mathsf{s}^{h_x}$ and m_x is of the form $(sid, P_i, P_j, \mathsf{pwd}')$. If so, then $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{apwKE}}$ with ($\mathsf{TestPwd}$, ssid, P, pwd') else it goes to the step "Other Cases" below. If the test passes, it sets $\mathsf{phash} = h_x$ and goes to the "Key Setting" step below, else it goes to the step "Other Cases" below.
5. ("Key Setting Step"): Compute $i_2' = \mathcal{H}(sid, \mathsf{ssid}, P_j, P_i, R_2', S_2', \text{HP}_2')$. If $T_2' \neq \mathsf{sphf_s.privH}(\mathsf{hk_s}, \langle R_2', S_2'/\mathbf{b}_\mathsf{s}^{\mathsf{phash}} \rangle, i_2')$ then goto the step "Other Cases". Else, compute $W_1 = \mathsf{sim}(\text{CRS}_\mathsf{c}, \mathsf{trap_c}, \langle R_1, S_1, \mathbf{b}_\mathsf{s}^{\mathsf{phash}}, T_1, i_1 \rangle)$. Issue a NewKey call to $\widehat{\mathcal{F}}_{\text{PAKE}}$ with key
$$\mathsf{ver_s.privH}(\text{HK}_1, \langle R_2', S_2'/\mathbf{b}_\mathsf{s}^{\mathsf{phash}}, T_2', i_2' \rangle) \cdot \mathsf{ver_c.pubH}(\text{HP}_2', W_1)$$
6. ("Other Cases"): $\mathcal{S}$ issues a $\mathsf{TestPwd}$ call to $\widehat{\mathcal{F}}_{\text{PAKE}}$ with the dummy password μ, followed by a NewKey call with a random session key, which leads to the functionality issuing a random and independent session key to the party P.

On receiving a message $R_1', S_1', T_1', \text{HP}_1'$ from $\mathcal{Z}$ intended for a **server instance** (P, ssid), the response of the simulator $\mathcal{S}$ is symmetric to the response described above for client instances, except the above step "Client Only Step" is skipped.

Stealing Password File. If there was a successful online TestPwd call by the simulator, before this StealPwdFile call, the corresponding random oracle response h_k was already assigned to the variable phash. Otherwise, the simulator runs through the set of random oracle query response set of the adversary $\{(m_k, h_k)\}_k$, which were not used for an online TestPwd call. For all the m_k's of the form $(sid, P_i, P_j, \mathsf{pwd}')$, it calls $(\mathsf{OfflineTestPwd}, sid, \mathsf{pwd}')$. Next, $\mathcal{S}$ calls StealPwdFile. If StealPwdFile returns pwd then it must equal pwd$'$ in some m_k. Assign to the variable phash the value h_k from the earlier recorded random oracle response to m_k. Otherwise, phash is assigned a fresh random value. The Server Persistent State $\mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}$ is computed accordingly and given to the adversary.

Client Corruption. On receiving a Corrupt call from $\mathcal{Z}$ for client instance P_i in session ssid, the simulator $\mathcal{S}$ calls the Corrupt routine of $\mathcal{F}_{\mathsf{apwKE}}$ to obtain pwd. If $\mathcal{S}$ had already output a message to $\mathcal{Z}$, and not output sk_1 it computes

$$W_1 = \mathsf{sim}_{\mathsf{c}}(\text{CRS}_{\mathsf{c}}, \mathsf{trap}_{\mathsf{c}}, \langle R_1, S_1, \mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}, T_1, i_1\rangle).$$

and outputs this W_1 along with pwd, and HK_1 as internal state of P_i. Note that this computation of W_1 is identical to the computation of W_1 in the computation of sk_1 (which is really output to $\mathcal{Z}$ only when $\mathsf{pwd}' = \mathsf{pwd}$).

Without loss of generality, we can assume that in the real-world if the Adversary (or Environment $\mathcal{Z}$) corrupts an instance before the session key is output then the instance does not output any session key. This is so because the Adversary (or $\mathcal{Z}$) either sets the key for that session or can compute it from the internal state it broke into.

Server Corruption. On receiving a Corrupt call from $\mathcal{Z}$ for server instance P_j in session ssid, the simulator $\mathcal{S}$ first performs the steps in the section on Stealing Password File above. In particular this sets the value of phash. It then calls the Corrupt routine of $\mathcal{F}_{\mathsf{apwKE}}$. If $\mathcal{S}$ had already output a message to $\mathcal{Z}$, and not output sk_1 it computes

$$W_2 = \mathsf{sim}_{\mathsf{s}}(\text{CRS}_{\mathsf{s}}, \mathsf{trap}_{\mathsf{s}}, \langle R_2, S_2/\mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}, T_2, i_2\rangle).$$

and outputs this W_2 along with HK_2 as internal state of P_j. Note that pwd is not given out.

Complexity of the Simulator. Observe that on stealing the password file, the function OfflineTestPwd is only called once for each random oracle input, which was not already tested by calling TestPwd. Hence the number of unique password arguments passed to $\mathsf{TestPwd}$ and $\mathsf{OfflineTestPwd}$ of $\mathcal{F}_{\mathsf{apwKE}}$ combined in the ideal world is at most the number of random oracle calls in the hybrid model.

Time complexity-wise, most of the simulator steps are $\log q$-time, where q is the security parameter. Due to Step 4 of the simulator code, where for each of the m sessions, in the worst case, it might go through all the n random oracle calls, there is an additive component of $m * n * \log q$ time. So the simulator runs in $O(mn \log q)$-time.

5.7 Proof of Indistinguishability

We now describe a series of experiments between a probabilistic polynomial time challenger $\mathcal{C}$ and the environment $\mathcal{Z}$, starting with Expt_0 which we describe next. We will show that the view of $\mathcal{Z}$ in Expt_0 is same as its view in UC-APAKE ideal-world setting with $\mathcal{Z}$ interacting with $\mathcal{F}_{\mathsf{apwKE}}$ and the UC-APAKE simulator $\mathcal{S}$ described above in Sect. 5.6. We end with an experiment which is identical to the real world execution of the protocol in Fig. 3. We prove that environment has negligible advantage in distinguishing between these series of experiments, leading to a proof of realization of $\mathcal{F}_{\mathsf{apwKE}}$ by the protocol Π. Due to space limitations, in this version we only describe Expt_0, and rest of the experiments and related proofs of indistinguishability can be found in the full version [JR16].

Here is the complete code in Expt_0 (stated as it's overall experiment with $\mathcal{Z}$):

1. Responding to a random oracle query on input m: If there is a record of the form (m, r), then just return r. Otherwise, generate $r \leftarrow \mathbb{Z}_q$, record (m, r) and return r.
2. The challenger $\mathcal{C}$ picks the CRS just as in the real world, except the QA-NIZK CRS-es are generated using the crs-simulators, which also generate simulator trapdoors $\mathsf{trap_c}, \mathsf{trap_s}$. It retains $a_1, a_2, \mathsf{trap_c}, \mathsf{trap_s}, \mathsf{hk_c}, \mathsf{hk_s}$ as trapdoors.
   Next, (on $\mathsf{StorePwdFile}$) the challenger calls the random oracle with query $(sid, P_i, P_j, \mathsf{pwd})$. It sets phash equal to the random oracle response and sets the server persistent state as $\mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}$.
   Define PHASHISSET to be true after either $\mathsf{StealPwdFile}$ has been called or the random oracle has been called with $(sid, P_i, P_j, \mathsf{pwd})$ by the adversary, and false before.
   Define PWDCALLED to be true after the random oracle has been called with $(sid, P_i, P_j, \mathsf{pwd})$ by the adversary, and false before.
3. On receiving $(\mathsf{CltSession}, sid, \mathsf{ssid}, P_i, P_j)$ from $\mathcal{Z}$, $\mathcal{C}$ generates $(\textsc{hk}_1, \textsc{hp}_1) \leftarrow \mathsf{ver_s.hkgen}(\textsc{crs}_\mathsf{s})$. Next, $\mathcal{C}$ chooses r_1, r_1', r_1'' at random, and sets $R_1 = \mathbf{g}^{r_1}$, $S_1 = \mathbf{g}^{r_1'}$ and $T_1 = \mathbf{g}^{r_1''}$. It then hands $(R_1, S_1, T_1, \textsc{hp}_1)$ to $\mathcal{Z}$ on behalf of this instance.
4. On receiving $(R_2', S_2', T_2', \textsc{hp}_2')$ from $\mathcal{Z}$, intended for client session (P_i, ssid) (and assuming no corruption of this instance):
   (a) If the received elements are either not in their respective groups, or are trivially 1, output $\mathrm{sk}_1 \leftarrow \mathbb{G}_T$.
   (b) If the message received is identical to message sent by $\mathcal{C}$ in the same session (i.e. same ssid) on behalf of the peer, then output $\mathrm{sk}_1 \leftarrow \mathbb{G}_T$

(unless the simulation of peer also received a legitimate message and its key has already been set, in which case the same key is used to output sk_1 here).

(c) If *PhashIsSet* is false, then output $\mathrm{sk}_1 \leftarrow \mathbb{G}_T$.

(d) Compute: $i_2' = \mathcal{H}(sid, \mathsf{ssid}, P_j, P_i, R_2', S_2', \textsc{hp}_2')$. If $S_2' = R_2'^{a_2} \mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}$ and $T_2' = \mathsf{sphf}_{\mathsf{s}}.\mathsf{privH}(\mathsf{hk}_{\mathsf{s}}, \langle R_2', S_2'/\mathbf{b}_{\mathsf{s}}^{\mathsf{phash}} \rangle, i_2')$, compute:

$$W_1 = \mathsf{sim}_{\mathsf{c}}(\textsc{crs}_{\mathsf{c}}, \mathsf{trap}_{\mathsf{c}}, \langle R_1, S_1, \mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}, T_1, i_1 \rangle).$$

Output:

$$\mathrm{sk}_1 = \mathsf{ver}_{\mathsf{s}}.\mathsf{privH}(\textsc{hk}_1, \langle R_2', S_2'/\mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}, T_2', i_2' \rangle) \cdot \mathsf{ver}_{\mathsf{c}}.\mathsf{pubH}(\textsc{hp}_2', W_1)$$

(e) If the above check failed then output $\mathrm{sk}_1 \leftarrow \mathbb{G}_T$.

5. On a Corrupt call for client P_i, output pwd. If Step 3 has already happened then also output $\textsc{hk}_1$ and $W_1 = \mathsf{sim}_{\mathsf{c}}(\textsc{crs}_{\mathsf{c}}, \mathsf{trap}_{\mathsf{c}}, \langle R_1, S_1, \mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}, T_1, i_1 \rangle)$.
6. On receiving (SrvSession, *sid*, ssid, P_j, P_i) from $\mathcal{Z}$, follow steps symmetric to Step 4, swapping subscripts and languages accordingly and replacing the condition *PhashIsSet* by *PwdCalled* in Step 4c. Also, in step 4d, the condition becomes: if $S_1' = R_1'^{a_1} \mathbf{b}_{\mathsf{c}}^{\mathsf{phash}}$ and $T_1' = \mathsf{sphf}_{\mathsf{c}}.\mathsf{privH}(\mathsf{hk}_{\mathsf{c}}, \langle R_1', S_1', \mathbf{b}_{\mathsf{s}}^{\mathsf{phash}} \rangle, i_1')$,
7. On a Corrupt call for server P_j, if Step 3 has already happened then output $\textsc{hk}_2$, and $W_2 = \mathsf{sim}_{\mathsf{s}}(\textsc{crs}_{\mathsf{s}}, \mathsf{trap}_{\mathsf{s}}, \langle R_2, S_2/\mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}, T_2, i_2 \rangle)$. Finally, execute a StealPwdFile call, as described below.
8. On a StealPwdFile call, return $\mathbf{b}_{\mathsf{s}}^{\mathsf{phash}}$ as the Server Persistent State to the adversary.

All outputs of sk_1 are also accompanied with *sid*, ssid (but are not mentioned above for ease of exposition).

Note that each instance has two asynchronous phases: a phase in which $\mathcal{C}$ outputs $R_1, S_1, ...$ to $\mathcal{Z}$, and a phase where it receives a message from $\mathcal{Z}$. However, $\mathcal{C}$ cannot output sk_1 until it has completed both phases. These orderings are dictated by $\mathcal{Z}$. We will consider two different kinds of temporal orderings. A temporal ordering of different instances based on the order in which $\mathcal{C}$ outputs sk_1 in an instance will be called **temporal ordering by key output**. A temporal ordering of different instances based on the order in which $\mathcal{C}$ outputs its first message (i.e. $R_1, S_1, ...$) will be called **temporal ordering by message output**. It is easy to see that $\mathcal{C}$ can dynamically compute both these orderings by maintaining a counter (for each ordering).

We now claim that the view of $\mathcal{Z}$ in Expt_0 is statistically indistinguishable from its view in its combined interaction with $\mathcal{F}_{\mathsf{apwKE}}$ and $\mathcal{S}$. The CRS is set identically by both $\mathcal{C}$ and $\mathcal{S}$. While $\mathcal{C}$ has access to pwd from the outset and sets up the random oracle output phash corresponding to $(sid, P_i, P_j, \mathsf{ssid})$ at the beginning, $\mathcal{S}$ doesn't have access to pwd at the beginning and hence defers this step till the point where either (1) a correct online guess has been made, (2) the password file was stolen and a correct offline guess was made, (3) the client was corrupted. In all these three cases the simulator gets to know pwd and has the chance to set phash. At the point when password file is stolen, the correct pwd may not have been guessed, but phash has to be set in order to output the server persistent state. In that case $\mathcal{S}$ generates a random phash, remembers it and assigns it to the correct input when the actual password is queried. At all points, although their algorithms differ, we can see that $\mathcal{C}$ and $\mathcal{S}$ respond to random oracle queries identically.

Both $\mathcal{C}$ and $\mathcal{S}$ generate the client and server flows identically. In particular, observe that the condition PhashIsSet exactly captures the state of $\mathcal{S}$ for a client session where it knows phash and can compute the relevant elements and keys. $\mathcal{C}$ uses the condition PhashIsSet to do the same computations. Similarly for the server sessions with the condition PwdCalled. The stronger condition for the server reflects the absence of the "Client Only Step" in the server sessions simulation. In the steps where a party receives a message from the adversary, both $\mathcal{C}$ and $\mathcal{S}$ end up computing keys identically. While $\mathcal{C}$ directly checks by exponentiation with phash in the case that pwd was guessed correctly, $\mathcal{S}$ goes through the list of random oracle calls to see which response was used for exponentiation as it may not know pwd or phash at this point.

Due to space limitations, the rest of the experiments and related proofs of indistinguishability are relegated to the full version [JR16].

References

[BBC+13] Benhamouda, F., Blazy, O., Chevalier, C., Pointcheval, D., Vergnaud, D.: New techniques for SPHFs and efficient one-round PAKE protocols. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 449–475. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_25

[BBS04] Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_3

[BC16a] Blazy, O., Chevalier, C.: Structure-preserving smooth projective hashing. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 339–369. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_12

[BFM88] Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications (extended abstract). In: 20th ACM STOC, pp. 103–112. ACM Press, May 1988

[BM92] Bellovin, S.M., Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy, pp. 72–84. IEEE Computer Society Press, May 1992

[BM93] Bellovin, S.M., Merritt, M.: Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password file compromise. In: Ashby, V. (ed.) ACM CCS 93, pp. 244–250. ACM Press, November 1993

[BMP00] Boyko, V., MacKenzie, P., Patel, S.: Provably secure password-authenticated key exchange using diffie-hellman. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_12

[Boy09] Boyen, X.: HPAKE: password authentication secure against cross-site user impersonation. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 09. LNCS, vol. 5888, pp. 279–298. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10433-6_19

[BP13] Benhamouda, F., Pointcheval, D.: Verifier-based password-authenticated key exchange: new models and constructions. Cryptology ePrint Archive, Report 2013/833 (2013). http://eprint.iacr.org/2013/833

[BPR00] Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_11

[BR93] Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Ashby, V. (ed.) ACM CCS 93, pp. 62–73. ACM Press, November 1993

[Can01] Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press, October 2001

[CHK+05] Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_24

[CR03] Canetti, R., Rabin, T.: Universal composition with joint state. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 265–281. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_16

[CS02] Cramer, R., Shoup, V.: Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 45–64. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_4

[EHK+13] Escala, A., Herold, G., Kiltz, E., Ràfols, C., Villar, J.: An algebraic framework for diffie-hellman assumptions. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 129–147. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_8

[FLR+10] Fischlin, M., Lehmann, A., Ristenpart, T., Shrimpton, T., Stam, M., Tessaro, S.: Random oracles with(out) programmability. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 303–320. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_18

[GMR06] Gentry, C., MacKenzie, P., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 142–159. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_9

[GS12] Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. SIAM J. Comput. **41**(5), 1193–1232 (2012)

[HK98] Halevi, S., Krawczyk, H.: Public-key cryptography and password protocols. In: ACM CCS 98, pp. 122–131. ACM Press, November 1998

[JG04] Jiang, S., Gong, G.: Password Based key exchange with mutual authentication. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 267–279. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30564-4_19

[JKX18] Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: an asymmetric pake protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 456–486. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_15

[JR12] Jutla, C., Roy, A.: Relatively-sound NIZKs and password-based key-exchange. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 485–503. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30057-8_29

[JR13] Jutla, C.S., Roy, A.: Shorter quasi-adaptive NIZK proofs for linear subspaces. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8269, pp. 1–20. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42033-7_1

[JR14] Jutla, C.S., Roy, A.: Switching lemma for bilinear tests and constant-size NIZK proofs for linear subspaces. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 295–312. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_17

[JR15] Jutla, C.S., Roy, A.: Dual-system simulation-soundness with applications to UC-PAKE and more. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9452, pp. 630–655. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48797-6_26

[JR16] Jutla, C., Roy, A.: Smooth NIZK arguments with applications to asymmetric UC-PAKE. Cryptology ePrint Archive, Report 2016/233 (2016). http://eprint.iacr.org/2016/233

[KV11] Katz, J., Vaikuntanathan, V.: Round-optimal password-based authenticated key exchange. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 293–310. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19571-6_18

[KW15] Kiltz, E., Wee, H.: Quasi-adaptive NIZK for linear subspaces revisited. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 101–128. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_4

[LPJY14] Libert, B., Peters, T., Joye, M., Yung, M.: Non-malleability from malleability: simulation-sound quasi-adaptive NIZK proofs and CCA2-secure encryption from homomorphic signatures. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 514–532. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_29

[Mac01] MacKenzie, P.: More efficient password-authenticated key exchange. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 361–377. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45353-9_27

[Wat09] Waters, B.: Dual system encryption: realizing fully secure IBE and HIBE under simple assumptions. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 619–636. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_36