SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

May 2017, Vol. 60 052107:1–052107:16
doi: 10.1007/s11432-016-5522-z

Universally composable anonymous password authenticated key exchange

Xuexian HU[1,2,3*], Jiang ZHANG[4,1], Zhenfeng ZHANG[1] & Jing XU[1]

[1] *Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing* 100190*, China;*
[2] *State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou* 450002*, China;*
[3] *Science and Technology on Information Assurance Laboratory, Beijing* 100072*, China;*
[4] *State Key Laboratory of Cryptology, Beijing* 100878*, China*

Received October 12, 2015; accepted November 24, 2015; published online September 28, 2016

Abstract Anonymous password authenticated key exchange (APAKE) is an important cryptographic primitive, through which a client holding a password can establish a session key with a server both authentically and anonymously. Although the server is guaranteed that the client in communication is from a pre-determined group, but the client's actual identity is protected. Because of their convenience, APAKE protocols have been widely studied and applied to the privacy protection research. However, all existing APAKE protocols are handled in stand-alone models and do not adequately settle the problem of protocol composition, which is a practical issue for protocol implementation. In this paper, we overcome this issue by formulating and realizing an ideal functionality for APAKE within the well-known universal composability (UC) framework, which thus guarantees security under the protocol composition operations. Our formulation captures the essential security requirements of APAKE such as off-line dictionary attack resistance, client anonymity and explicit mutual authentication. Moreover, it addresses the arbitrary probabilistic distribution of passwords. The construction of our protocol, which utilizes SPHF-friendly commitments and CCA2-secure encryption schemes, can be instantiated and proven secure in the standard model, i.e., without random oracle heuristics.

Keywords anonymous password authentication, key exchange, universal composability, provable security, standard model

Citation Hu X X, Zhang J, Zhang Z F, et al. Universally composable anonymous password authenticated key exchange. Sci China Inf Sci, 2017, 60(5): 052107, doi: 10.1007/s11432-016-5522-z

1 Introduction

Password authenticated key exchange. Key exchange (KE) is a process in which session keys are established between two or more parties, based on certain special cryptographic credentials held by each party, for subsequent secure communication. It is important tool for establishing secure communication, and plays a pivotal role in guaranteeing the security of high-level cryptographic algorithms (e.g., symmetric encryption schemes) [1, 2]. Password authenticated key exchange (PAKE) is an important type of key exchange, that allows parties to authenticate each other and establish a high-entropy key based

* Corresponding author (email: huxuexian@tca.iscas.ac.cn)

on a shared low-entropy credential called a password. Because passwords can be easily recalled by human users, PAKE protocols are much more convenient and practical, compared with those KE protocols that rely on comprehensive public key infrastructure (PKI) systems for distributing digital certificates, or dedicated hardware for storing high-entropy symmetric keys. Although repeated evidence has shown the vulnerability to off-line dictionary attacks, PAKE protocols, are in fact reasonably secure against on-line guessing attacks, because adversaries can be throttled or slowed down through various means, such as presenting CAPTCHAs or introducing delays after several failed attempts [3]. After the pioneering work by Bellovin and Merrit [4], PAKE protocols have been extensively explored [5–10] and widely standardized [11, 12]. Moreover, it is most likely that these protocols will remain in widespread use in the foreseeable future.

Anonymous password authenticated key exchange. Traditionally, a participant of a PAKE protocol manages to protect its password secretly while paying no attention to protecting its identity, because such information is usually transmitted explicitly to help its partner identify the participant and determine which password should be used. However, along with the growing concern regarding network privacy risks, users tend to avoid as much as possible those applications that they believe may not protect their privacy. Consequently, reinforcing conventional PAKE protocols with additional privacy protection property, while maintaining a high level of convenience, is a quite significant issue. Among many practical settings, one important scenario is a client from a legitimate group wanting to communicate with a server securely, while still wanting to hide its actual identity from the server [13]. To meet these needs, (password-only) anonymous password authenticated key exchange (APAKE) was proposed in which a client establishes a session key with the server both authentically and anonymously.

In 2005, Viet et al. [14] proposed the first APAKE protocol, as well as an extended threshold version, by subtly integrating an oblivious transfer (OT) protocol within a two-party PAKE protocol. However, it was later pointed out by Shin et al. [15] that Viet et al.'s threshold APAKE protocol is vulnerable to an off-line dictionary attack. Shin et al. [15] also put forward an enhanced threshold APAKE protocol aiming to overcome this type of attack. However, Yang and Zhang [16] found that Shin et al.'s improvement is still under the threats of impersonation attacks and off-line dictionary attacks. To resist such attacks and improve the efficiency, Yang and Zhang [16] presented a novel APAKE protocol based on the SPEKE protocol [17]. Shin et al. recently [18] put forward another APAKE protocol, whose computation cost is reduced if a public bulletin board is made available for all protocol participants.

Apart from the above APAKE protocols, another kind of anonymous authenticated key exchange is considered in [19–21], which employs, on top of a password, additional storage. More specifically, each client registers its identity to the server and takes back the authentication credential. The client then uses its password to wrap this credential and stores it in extra (possibly public) storage, such as a memory card, a smart phone, or a cloud. To log into the server, the client has to submit both its password and the password-wrapped credential. Although this technique can greatly reduce the storage and computation complexity on the server side, an auxiliary storage device needs to be available in time for any client, which is not suitable for all real world scenarios. Henceforth, we do not consider any extra-storage-aided anonymous protocols, and restrict ourselves to the more convenient password-only APAKE setting.

Security models. Many existing PAKE protocols [6–8, 22] and, to the best of our knowledge, almost all APAKE protocols [14–16, 18] with provable security are analyzed in variants of the BPR model [5]. These game-based models restrict attention to a "stand-alone" setting, in which only the instances of a single protocol are considered in isolation, whereas no other protocols are allowed to be executed in the same network at the same time. As pointed out by Canetti [23], when placing the protocols in a more complex context, security can no longer be guaranteed by these stand-alone models. Nevertheless, in the real world, (A)PAKE protocols are most likely to be executed concurrently, or even through a composable approach, along with other protocols, for instance, message transport or secure e-commerce protocols. Some progress has been made with respect to UC-secure PAKE protocols during the last decade [24–28]. However, little if any effort has been made in arming APAKE with UC security. As a consequence, existing APAKE protocols provide a security level that is significantly lower than that of UC-secure PAKE protocols. Furthermore, previously-proposed APAKE security models suppose that passwords are chosen independently from a certain pre-determined (usually uniform) distribution; in practice, however, a party of an APAKE protocol might choose its password at will and use it (or a related password) simultaneously for various cryptographic protocols. Therefore, it is necessary to formalize and realize a formal definition for APAKE that can adequately address the above problems.

Contributions. To design APAKE protocols suitable for ubiquitous applications, and that can provide stronger security guarantees, we formalize and realize a new definition for APAKE within the well-known universal composability (UC) framework. Specifically, we present a security definition for password-only APAKE in terms of an ideal functionality in the UC framework, which captures the expected security requirements of exchanging a session key both authentically and anonymously by utilizing only a password. Similar to the formulation in [24], our definition makes no assumption regarding the distribution of passwords used by the protocol parties, which guarantees security for all efficient, and even related, distributions.

Additionally, we propose a new password-only APAKE protocol and prove that it securely realizes the APAKE functionality in the UC framework, thus ensuring its UC security. The APAKE protocol described herein relies on standard number-theoretical assumptions, i.e., without the application of random oracle heuristics. Furthermore, since our protocol uses generic building blocks, it actually implies a series of APAKE protocols in the standard model, by instantiating these generic building blocks using different hard assumptions, e.g., DDH, Quadratic Residuosity, or N-Residuosity assumptions [29].

2 Definition of security

2.1 The UC framework

The universal composability (UC) framework [23] is a well-known approach to the formalization of protocol security, through which protocols can be analyzed only once and utilized universally. Roughly speaking, the security of a given cryptographic task in the UC framework is captured through an ideal functionality $\mathcal{F}$, which is essentially a trusted party that interacts with a set of participating players in a secure manner. We state that a protocol π securely realizes the ideal functionality $\mathcal{F}$ if for every $\mathcal{A}$ there exists an ideal adversary (called a simulator) $\mathcal{S}$, such that no environment can distinguish these two scenarios with a non-negligible probability. The universal composition theorem then asserts that any larger protocol that uses multi-instances of $\mathcal{F}$ behaves essentially the same when these ideal instances are replaced by instances of protocol π.

When the common reference string (CRS) model is used, where different sessions of the protocol share some of the same CRS, one might have to resort to the UC framework with joint state (JUC) proposed by Canetti and Rabin [30]. They put forth a new composition theorem that allows us to deduce the security of a multi-instance system from the security of a single instance, even when different instances share a certain amount of state and randomness. This is achieved by additionally defining sub-SIDs (SSIDs) for identifying different sub-sessions.

2.2 Formulation of anonymity

Anonymity is an important and desirable attribute for certain systems. Electronic voting schemes and electronic cash schemes, for instance, require anonymity by their nature. In addition, Internet users may also seek anonymity in sending a politically sensitive email, or engaging in an unfamiliar chat room. For these types of applications, it is necessary that an individual be indistinguishable from the other entities within a predetermined set, i.e., an anonymity set. To achieve this aim in a particular cryptographic system, a reasonably important but usually unwritten assumption is that an outside adversary, or an honest-but-curious server, cannot identify an individual through approaches other than those that have been considered in the applied schemes. This implies that, aside from those properties explicitly considered by anonymity schemes, all individuals in an anonymity set are assumed to have the same (or at least indistinguishable) characteristics in all other aspects. For example, if pseudonymity or a proxy server is used to attain the anonymity of certain users within a specific group, we implicitly require that all such users have the same IP address, or that they possess different IP addresses that would be difficult to obtain by an adversary or the server. Therefore, an individual user cannot be identified by tracing its IP address. This assumption is reasonable in practice because various techniques can be employed to realize it, such as a Mix network or the onion router (Tor).

However, when resorting to the UC framework, we cannot formulate anonymity simply by making the above assumption implicitly. Since that, according to the basic computation model of the UC framework (Section 3 of [23]), an external-write instruction of an interactive Turing machine (ITM) should specify the identity of the target instance of the ITM explicitly. This requirement is useful in making clear which instance each message is intended for. As an aside, however, the identity of every client user is also revealed to an adversary if we simply set the party identity to be its unique identifier in the real world. To overcome the defects of the conventional UC framework in formalizing security properties that involve identity anonymity, we make some minor changes to the interpretation and utilization of the party identity field.

More specifically, as usual we allow each identity to be composed of two fields, the session identity, SID, and the party identity, PID. The former is used to identify a protocol instance, and the latter is used to differentiate between instances of an ITM within a specific protocol instance. We add an extra structure to the party identity PID by interpreting it as two separate sub-fields: a (public) address field, $\mathrm{PID_A}$, and an (optionally secret) identity party, $\mathrm{PID_I}$. Both of these sub-fields need to be unique during the entire execution, and either of them can be used to identify an instance of ITM. Once an instance of ITM is activated, it is required that its $\mathrm{SID}, \mathrm{PID_A}$ be determined, but not the field of $\mathrm{PID_I}$, which can be specified during the execution. When a message is asked to send through an external-write instruction, we are only required to specify the $\mathrm{PID_A}$ of the intended instance. Note that our modified formalization is compatible with the traditional UC frameworkin the sense that, when such a complicated PID is not needed, we can simply set $\mathrm{PID_I} = \mathrm{Null}, \mathrm{PID_A} = \mathrm{PID}$ and use it as usual. Nevertheless, we show that the revised UC framework is sufficiently powerful to model an anonymous authentication by presenting and realizing an APAKE functionality within this framework.

Remark 1. Informally speaking, to achieve anonymity, the address field $\mathrm{PID_A}$ of the party identity will serve as a pseudonymity for a legitimate client. Once a message is sent to a client within a pre-determined group, this pseudonymity is used to identify the destination, whereas the actual identification of the client (i.e., $\mathrm{PID_I}$) is kept secret. In practice, we can construct such a $\mathrm{PID_A}$ by simply concatenating the group name and a random string chosen by the client itself.

2.3 APAKE functionality

In this section, we present the ideal functionality for anonymous password authenticated key exchange in the UC framework, which is denoted by $\mathcal{F}_{\mathrm{APAKE}}$. Our starting points are the password authenticated key exchange functionalities defined in [24–26], and the security models for APAKE presented in [14,16]. The aim here is to formulate a functionality that captures the essential security requirements for anonymous password authenticated key exchange, including session key security, off-line dictionary attacks resistance, client anonymity and explicitly mutual authentication.

2.3.1 *The functionality* $\mathcal{F}_{\mathrm{APAKE}}$

The functionality $\mathcal{F}_{\mathrm{APAKE}}$ is parameterized by a security parameter k. It interacts with an adversary $\mathcal{S}$ and a set of parties $\boldsymbol{P} = \{P_l\}_{l \in I}$ through the following queries:

Initialization:

- Upon receiving $(\mathrm{NewSession}, \mathrm{sid}, \boldsymbol{Pid}, P_i, pw_i, \mathrm{client})$ from party P_i:

  - Send $(\mathrm{NewSession}, \mathrm{sid}, \boldsymbol{Pid}, \mathrm{client})$ to $\mathcal{S}$. If this is the first NewSession query, or the second NewSession query while a record $(\mathrm{sid}, \boldsymbol{Pid}, *, *, \mathrm{server})$ exists, $\boldsymbol{Pid}$ is an ordered set in the form of $\boldsymbol{Pid} = \boldsymbol{\Gamma} \cup \{P_s\} \subset \boldsymbol{P}$ and $P_i \in \boldsymbol{\Gamma}$, record $(\mathrm{sid}, \boldsymbol{Pid}, P_i, pw_i, \mathrm{client})$ and mark it as fresh.

- Upon receiving $(\mathrm{NewSession}, \mathrm{sid}, \boldsymbol{Pid}, P_s, \boldsymbol{pw}_s, \mathrm{server})$ from party P_s:

- Send $(\text{NewSession}, \text{sid}, \boldsymbol{Pid}, \text{server})$ to $\mathcal{S}$. If this is the first NewSession query, or the second NewSession query while a record $(\text{sid}, \boldsymbol{Pid}, *, *, \text{client})$ exists, $\boldsymbol{Pid}$ is an ordered set in the form of $\boldsymbol{Pid} = \boldsymbol{\Gamma} \cup \{P_s\} \subset \boldsymbol{P}$ and $\boldsymbol{pw}_s$ is an ordered set of the same size as $\boldsymbol{\Gamma}$, record $(\text{sid}, \boldsymbol{Pid}, P_s, \boldsymbol{pw}_s, \text{server})$ and mark it as fresh.

Test password:

- Upon receiving $(\text{TestPwd}, \text{sid}, \text{role}, P_i', pw_i')$ from the adversary $\mathcal{S}$:

- If role = client and a fresh record in the form of $(\text{sid}, \boldsymbol{Pid}, P_i, pw_i, \text{client})$ exists, do the following. If $P_i = P_i'$ and $pw_i = pw_i'$, mark the record as compromised and reply to $\mathcal{S}$ with "correct guess"; otherwise, mark the record as interrupted and reply with "wrong guess".

- If role = server and a fresh record in the form of $(\text{sid}, \boldsymbol{Pid}, P_s, \boldsymbol{pw}_s, \text{server})$ exists, do the following. If $\boldsymbol{pw}_s[P_i'] = pw_i'$, mark the record as compromised and reply to $\mathcal{S}$ with "correct guess"; otherwise, mark the record as interrupted and reply with "wrong guess".

Key generation:

- Upon receiving $(\text{ServerReady}, \text{sid}, \boldsymbol{Pid})$ from the adversary $\mathcal{S}$, if a fresh record in the form of $(\text{sid}, \boldsymbol{Pid}, P_s, \boldsymbol{pw}_s, \text{server})$ exists, then re-label it as ready.

- Upon receiving $(\text{NewKey}, \text{sid}, \boldsymbol{Pid}, \text{client}, sk)$ from the adversary $\mathcal{S}$, if there exists a record in the form of $(\text{sid}, \boldsymbol{Pid}, P_i, pw_i, \text{client})$ that is not marked as completed, then conduct:

- If the record is compromised, or P_i is corrupted, set $sk_c \leftarrow sk$.

- Else, if the record is interrupted, set $sk_c \leftarrow \text{error}$.

- Else, if a record $(\text{sid}, \boldsymbol{Pid}, P_s, \boldsymbol{pw}_s, \text{server})$ such that $\boldsymbol{pw}_s[P_i] = pw_i$ exists, then do:

$\star$ If P_s is corrupted, set $sk_c \leftarrow sk$.

$\star$ Else, if the record $(\text{sid}, \boldsymbol{Pid}, P_s, \boldsymbol{pw}_s, \text{server})$ is labeled ready, choose sk' uniformly at random from $\{0,1\}^k$, set $sk_c \leftarrow sk'$, and store the tuple $(\text{sid}, \boldsymbol{Pid}, sk')$.

- Otherwise, if none of the above rules apply, set $sk_c \leftarrow \text{error}$.

Then, send to P_i the tuple $(\text{sid}, \boldsymbol{Pid}, sk_c)$, mark the record $(\text{sid}, \boldsymbol{Pid}, P_i, pw_i, \text{client})$ as completed, and report the final result (which is completed if $sk_c \in \{0,1\}^k$, or error if $sk_c = \text{error}$) to $\mathcal{S}$.

- Upon receiving $(\text{NewKey}, \text{sid}, \boldsymbol{Pid}, \text{server}, sk)$ from the adversary $\mathcal{S}$, if there exists a record in the form of $(\text{sid}, \boldsymbol{Pid}, P_s, \boldsymbol{pw}_s, \text{server})$ that is not marked as completed, then conduct:

- If the record is compromised, or P_s is corrupted, set $sk_s \leftarrow sk$.

- Else, if the record is interrupted, set $sk_s \leftarrow \text{error}$.

- Else, if a record $(\text{sid}, \boldsymbol{Pid}, P_i, pw_i, \text{client})$ such that $\boldsymbol{pw}_s[P_i] = pw_i$ exists, then do:

$\star$ If P_i is corrupted, set $sk_s \leftarrow sk$.

$\star$ Else, if these exists a record $(\text{sid}, \boldsymbol{Pid}, sk_c)$, set $sk_s \leftarrow sk_c$.

- Otherwise, if none of the above rules apply, set $sk_s \leftarrow \text{error}$.

Then, send to P_s the tuple $(\text{sid}, \boldsymbol{Pid}, sk_s)$, mark the record $(\text{sid}, \boldsymbol{Pid}, P_s, \boldsymbol{pw}_s, \text{server})$ as completed, and report the final result (which is completed if $sk_s \in \{0,1\}^k$, or error if $sk_s = \text{error}$) to $\mathcal{S}$.

Remark 2. In the above formulation, the party identity PID of a client is set to $\text{PID}_\text{A} = \boldsymbol{Pid}||\text{client}$ and $\text{PID}_\text{I} = P_i$. Strictly speaking, some random strings should also be attached after PID_A to guarantee the uniqueness. Herein we omit these strings for simplicity and clarity of the description.

Remark 3. Mutual authentication is guaranteed through the following techniques: (1) Strict order is required in the generation of the session keys. The client will generate a random session key when server is fresh and the passwords match; in addition, the server will obtain the same session key only when the client has generated a session key and the passwords match. (2) Both parties report error explicitly when a failed password guess is detected or when their passwords do not match.

2.3.2 *Design explanations*

In the definition of $\mathcal{F}_{\text{APAKE}}$, similar to the approach adopted by [24] and [31], we let the environment $\mathcal{Z}$ choose the passwords for all participants. As indicated in [24], this approach allows a scenario in which the same password is used for many different protocols. Furthermore, since the definition quantifies over all PPT environment, $\mathcal{F}_{\mathrm{APAKE}}$ also captures the case where passwords are chosen according to some arbitrary, and even related, distributions.

The APAKE functionality $\mathcal{F}_{\mathrm{APAKE}}$ starts with an initialization phase, in which an anonymous client or a server provides an input to inform the functionality its interest in taking part in the protocol. The input is in the form of $(\mathrm{NewSession}, \mathrm{sid}, \boldsymbol{Pid}, P_i, pw_i, \mathrm{client})$ or $(\mathrm{NewSession}, \mathrm{sid}, \boldsymbol{Pid}, P_s, \boldsymbol{pw}_s, \mathrm{server})$, where $\boldsymbol{Pid}$ is an ordered set $\boldsymbol{Pid} = \boldsymbol{\Gamma} \cup \{P_s\}$ representing the identifiers of the clients $\boldsymbol{\Gamma} = \{P_1, P_2, \ldots, P_n\}$ and the server P_s. In the following, we also use $\boldsymbol{pw}_s[P_i]$ to denote the password in the ordered set $\boldsymbol{pw}_s$ whose index is the same as the index of P_i in the ordered set $\boldsymbol{\Gamma}$.

Upon initialization, a session is declared fresh. A TestPwd query from the adversary, which models the active on-line password guessing attacks, will alter the status of this session. As usual, when the adversary makes a correct guess, we change the status of the session from fresh to compromised; otherwise, we relabel the session as interrupted. However, because APAKE is an asymmetric protocol specifying different behaviors for the client and server, we treat a guess as being correct differently according to the role of the corresponding session. Specifically, when a query is aimed at a client session, we require the adversary to provide the correct identity and password, i.e., $P_i = P_i'$ and $pw_i = pw_i'$; when a query is aimed at a server session, we require the password provided by the adversary to be a legal part of the password vector held by the server, i.e., $\boldsymbol{pw}_s[P_i'] = pw_i'$. Next, the functionality informs the adversary of whether its guess is correct or incorrect through a "correct guess" or "wrong guess", respectively.

During the key generation phase, the power to determine the session key at will is given to the adversary when the session is compromised, and when the corresponding party or its partner is corrupted. For a normal session key generation, several techniques are adopted to achieve explicit mutual authentication. First, similar to that in [26], we introduce a ready state for the server session. Subsequently, a client session will generate a random session key if and only if there is a partnered server session in a ready state. Second, we stipulate the order of key generation for our purposes. In particular, we require the client session to choose its session key before the server instance. Next, the server session will generate its session key if and only if a client session that has already outputted a session key exists.

3 Cryptographic tools

In this section, we review definitions for cryptographic tools used in our construction, such as smooth projective hash functions and SPHF-friendly equivocable and extractable commitments.

3.1 Smooth projective hash functions

The smooth projective hash function family was first introduced by Cramer and Shoup [32] for constructing a CCA2 secure public key encryption scheme, and was later developed [8, 28, 29, 31, 33] as one of the main tools used in the construction of PAKE protocols. In particular, it is a family of hash functions that admits two keys. One key can be used to efficiently compute the hash values for all messages in the hash domain, and the other key can be used to properly compute the hash values on a specified subset, while giving almost no information regarding the hash values for messages that are not in the subset.

Let X be a finite, non-empty set (i.e., the domain), and $L \subset X$ be a certain subset of this domain, which is usually a formal NP language such that it is difficult to distinguish a random element selected from L from a random element chosen from $X \setminus L$. Note that, for any word $C \in L$, there must be a witness w for this fact. A family of smooth projective hash functions $\mathcal{H}$, from domain X to a set (or group) G, $|G| = O(2^k)$, consists of the following algorithms $\mathcal{H} = (\mathrm{HashKG}, \mathrm{ProjKG}, \mathrm{Hash}, \mathrm{ProjH})$.

- $\mathrm{HashKG}(L)$: The key generation algorithm HashKG takes as input language L and produces a hash key hk for language L.
- $\mathrm{ProjKG}(hk, L, C)$: The key projection algorithm takes the hash key hk, language L and word $C \in L$ as input, and generates a projected hash key as hp.
- $\mathrm{Hash}(hk, L, C)$: The hash algorithm takes a hash key hk and a word $C \in L$ as input, and outputs a hash value $\mathrm{Hash}(hk, L, C)$.

• $\mathrm{ProjH}(hp, L, C, w)$: The projected hash algorithm takes the projected hash key hp, word $C \in L$ and witness w of the fact as input, and returns a hash value as $\mathrm{ProjH}(hp, L, C, w)$.

The smooth projective hash functions should satisfy both the correctness and smoothness properties. The correctness property guarantees that if $C \in L$ and w is the corresponding witness, then it holds that $\mathrm{Hash}(hk, L, C) = \mathrm{ProjH}(hp, L, C, w)$. The smoothness property, which captures the security of the family, assures that for any $C \in X \backslash L$, $\mathrm{Hash}(hk, L, C)$ is statistically close to a uniform random element from the range of the hash function, and even the projected key hp is given. In other words, the following two probabilistic distribution ensembles are statistically indistinguishable:

$$\{C, hp, \mathrm{Hash}(hk, L, C)\}_{hk \overset{\$}{\leftarrow} \mathrm{HashKG}(L)} \overset{s}{\equiv} \{C, hp, g\}_{hk \overset{\$}{\leftarrow} \mathrm{HashKG}(L), g \overset{\$}{\leftarrow} G}.$$

3.2 Equivocable and extractable commitments

A commitment scheme is a fundamental cryptography primitive, allowing a committer to commit to a specific value while keeping it hidden from the receiver. It usually consists of two phases. In the commit phase, the committer generates and sends to the receiver a commitment C associated with a message m. Next, in the opening phase, the committer reveals m by releasing the opening data δ to the receiver, who will be able to verify whether C is a commitment of m. Basically, a commitment should fulfill two security requirements, hiding and binding properties. The hiding property guarantees that an adversary cannot obtain any information from the commitment value C, whereas the binding property ensures that a commitment generated by the committer can not be opened in two different ways.

Definition 1 (Non-interactive labeled commitment). A non-interactive labeled commitment scheme is defined by three algorithms $(\mathrm{SetupCom}, \mathrm{Com}, \mathrm{VerCom})$:

• $\mathrm{SetupCom}(1^k)$ takes as input the security parameter k, and outputs a common reference string ρ.

• $\mathrm{Com}^l(m)$ takes as input an arbitrary label string l and a message m, and outputs a commitment C and the corresponding opening information δ.

• $\mathrm{VerCom}^l(C, \delta, m)$ takes as input a commitment C, a opening data δ and a message m, and outputs 1 if C is indeed a valid commitment of message m with opening data δ, and outputs 0, otherwise.

However, for some applications of commitments, such as the construction of (A)PAKE protocols with provable security in the standard model, it is necessary for the commitments to additionally satisfy the equivocability and extractability properties. The equivocability guarantees that a simulator who knows a certain equivocable trapdoor can generate a commitment that can be opened in two or more ways. The extractability property allows one to extract a corresponding message m from a commitment C. For our purpose, herein we review the definition of non-interactive equivocable and extractable (i.e., E^2-) commitments with the associated smooth projective hash functions [28].

Definition 2 (E^2-commitment). A non-interactive labeled E^2-commitment scheme is a commitment specified in Definition 1, but extended with the following algorithms:

• $\mathrm{SetupComT}(1^k)$ takes as input the security parameter k, and outputs a common reference string ρ as well as a trapdoor τ.

• $\mathrm{SimCom}^l(\tau)$ takes as input an arbitrary label string l and the trapdoor τ, and outputs a pair (C, eqk) consisting of a commitment C and an equivocable key eqk.

• $\mathrm{OpenCom}^l(C, \mathrm{eqk}, m)$ takes as input a pair (C, eqk) outputted by the SimCom algorithm together with a message m, and then outputs an opening data δ for C and m.

• $\mathrm{ExtCom}^l(\tau, C)$ takes as input an arbitrary label string l, the trapdoor τ and a commitment C, and then outputs the corresponding commit message m, or $\perp$ if the extraction fails.

Denote by $(C, \delta) \leftarrow \mathrm{SCom}^l(\tau, m)$ the probabilistic algorithm computed as $(C, \mathrm{eqk}) \leftarrow \mathrm{SimCom}^l(\tau)$ and $\delta \leftarrow \mathrm{OpenCom}^l(C, \mathrm{eqk}, m)$. To achieve E^2-commitment security, great care should be taken because an adversary might exploit the weakness of additional equivocation and extraction queries. Consequently, the following properties are then associated with the commiment:

• Correctness. For all common reference strings ρ generated correctly, all commitments and opening data generated honestly will be valid with respect to the verification test, i.e., for all m and l, it holds

```
$\mathrm{Exp}_{\mathcal{A}}^{\text{s-sim-ind-b}}(1^k)$:
    $(\rho,\tau) \overset{\$}{\leftarrow} \mathrm{SetupComT}(1^k)$
    $(l, x, \mathrm{state}) \overset{\$}{\leftarrow} \mathcal{A}^{\mathrm{SCom}(\tau,\cdot),\mathrm{ExtCom}(\tau,\cdot)}(1^k)$
    if $b = 0$, set $(C,\delta) \overset{\$}{\leftarrow} \mathrm{Com}^l(x)$;
    otherwise, set $(C,\delta) \overset{\$}{\leftarrow} \mathrm{SCom}^l(\tau, x)$
    return $\mathcal{A}^{\mathrm{SCom}(\tau,\cdot),\mathrm{ExtCom}(\tau,\cdot)}(\mathrm{state}, C, \delta)$
```

```
$\mathrm{Exp}_{\mathcal{A}}^{\text{r-bin-ext-b}}(1^k)$:
    $(\rho,\tau) \overset{\$}{\leftarrow} \mathrm{SetupComT}(1^k)$
    $(C, l) \overset{\$}{\leftarrow} \mathcal{A}^{\mathrm{SCom}(\tau,\cdot),\mathrm{ExtCom}(\tau,\cdot)}(1^k)$
    $x' \leftarrow \mathrm{ExtCom}^l(\tau, C)$
    if $(l, x', C)$ is not outputted by SCom,
    $\exists x \neq x', \exists \delta$, s.t. $\mathrm{VerCom}^l(C, \delta, x) = 1$
    then return 1; otherwise, return 0
```

Figure 1 Security experiments with respect to E^2-commitments.

that $\mathrm{VerCom}^l(\mathrm{Com}^l(m), m) = 1$.

• Trapdoor correctness. All simulated commitments generated by the SimCom algorithm can be opened to any message, and all commitments generated by the Com algorithm can be extracted correctly: for all l and m, $\mathrm{VerCom}^l(\mathrm{SimCom}^l(\tau, m), m) = 1$, and if $(C, \delta) \leftarrow \mathrm{Com}^l(m)$, then $\mathrm{ExtCom}^l(\tau, C) = m$.

• Setup indistinguishability. The common reference string ρ generated by SetupComT is computationally indistinguishable from that generated by SetupCom.

• Strong simulation indistinguishability. No PPT adversary can distinguish the real commitments generated by the Com algorithm from fake commitments generated by the SCom algorithm, even with oracle access to the ExtCom and SCom algorithms. We restrict queries on ExtCom with commitment generated by SCom to be replied with the associated SCom inputs. The corresponding formal security is defined by two experiments $\mathrm{Exp}_{\mathcal{A}}^{\text{s-sim-ind-b}}(1^k)$, $b \in \{0, 1\}$ (see Figure 1). A commitment scheme $\mathcal{C}$ is said to be (t, ϵ)-strongly-simulation-indistinguishable if the advantage function $\mathrm{Adv}_{\mathcal{C}}^{\text{s-sim-ind}}(t) = \max_{\mathcal{A}} |\Pr[\mathrm{Exp}_{\mathcal{A}}^{\text{s-sim-ind-0}}(1^k) = 1] - \Pr[\mathrm{Exp}_{\mathcal{A}}^{\text{s-sim-ind-1}}(1^k) = 1]| < \epsilon$, where the maximum is taken over all $\mathcal{A}$ running in time at most t;

• Robust binding extractability. No PPT adversary can make a fool of the extractor by generating a commitment C that extracts to m' whereas C can be opened to another value m, even with oracle access to the ExtCom and SCom algorithms. Formal security is defined through $\mathrm{Exp}_{\mathcal{A}}^{\text{r-bin-ext-b}}$, $b \in \{0, 1\}$ experiments (see Figure 1). A commitment scheme $\mathcal{C}$ is said to be (t, ϵ)-robustly-binding-extractable if $\mathrm{Adv}_{\mathcal{C}}^{\text{r-bin-ext}}(t) = \max_{\mathcal{A}} |\Pr[\mathrm{Exp}_{\mathcal{A}}^{\text{r-bin-ext-0}} = 1] - \Pr[\mathrm{Exp}_{\mathcal{A}}^{\text{r-bin-ext-1}} = 1]| < \epsilon$.

Note that strong simulation indistinguishability, together with setup indistinguishability, implies the basic hiding property; whereas the robust binding extractability and setup indistinguishability imply the basic binding property. It is important to note that the robust binding extractability also guarantees that the underlying NP language $L_x = \{(l, C) | \exists \delta, \mathrm{VerCom}^l(C, x, \delta) = 1\}$ associated with SPHF is well defined. Thus, a commitment is said to be SPHF-friendly if it satisfies both strong simulation indistinguishability and robust binding extractability, and admits an SPHF associated with the language L_x.

In [28], Abdalla et al. also provided an instantiation of an SPHF-friendly E^2-commitment, by combining Haralambiev's commitment [34] and a variant of the Cramer-Shoup encryption scheme [35]. Haralambiev's commitment, which takes a group element as the opening data, provides the scheme with a perfect hiding property, whereas the variation of Cramer-Shoup encryption guarantees that the proposed scheme is extractable.

4 UC-secure APAKE protocol

In this section, we describe our APAKE protocol in the UC framework (called UC-APAKE), which is constructed from an SPHF-friendly E^2-commitment scheme, a CCA2-secure public key encryption scheme, and a one-time signature scheme.

4.1 Description of our protocol

Denote by k the security parameter. Assume that $\mathcal{C} = (\mathrm{SetupCom}, \mathrm{Com}, \mathrm{VerCom}, \mathrm{SetupComT}, \mathrm{SimCom}, \mathrm{OpenCom}, \mathrm{ExtCom})$ is an SPHF-friendly E^2-commitment, and that $\mathcal{H} = (\mathrm{HashKG}, \mathrm{ProjKG}, \mathrm{Hash}, \mathrm{ProjH})$

$$\begin{array}{rcl}
 & \text{CRS: } \rho, pk' & \\
P_i(pw_i) & & S\left(\begin{array}{l}\boldsymbol{\Gamma}=\{P_1,\ldots,P_n\}\\ \boldsymbol{pw}_s=\{pw_j\}_{1\leqslant j\leqslant n}\end{array}\right)\\
l_c{:=}\mathrm{sid}||\boldsymbol{Pid} & & \\
(C_i,\delta_i)\leftarrow \mathrm{Com}^{l_c}(pw_i||i) & \xrightarrow{\quad C_i \quad} & \\
 & & \boldsymbol{hk}=(hk_1,hk_2,\ldots,hk_n)\\
 & & \text{for every } j=1,\ldots,n, \text{ compute}\\
 & & hp_j=\mathrm{ProjKG}(hk_j;l_c,C_i,pw_j||j)\\
 & & tk_j||tp_j=\mathrm{Hash}(hk_j;l_c,C_i,pw_j||j)\\
 & & \delta_j=tp_1\oplus tp_j\\
 & & \boldsymbol{hp}=(hp_1,\ldots,hp_n,\delta_2,\ldots,\delta_n)\\
 & & \tau_s||sk_s=tp_1\\
 & & (VK,SK)\leftarrow \mathrm{SignKG}(1^k)\\
 & & l_s=\mathrm{sid}||\boldsymbol{Pid}||C_i||\boldsymbol{hp}||VK\\
 & & C'_j=\mathrm{Enc}'_{pk'}(pw_j;l_s;tk_j)\\
 & & \boldsymbol{C}'=(C'_1,C'_2,\ldots,C'_n)\\
 & \xleftarrow{\boldsymbol{hp}, VK, \boldsymbol{C}', \sigma} & \sigma\leftarrow \mathrm{Sign}_{SK}(\boldsymbol{C}')\\
\text{if } \mathrm{Verify}_{VK}(\boldsymbol{C}';\sigma)=1 & & \\
tk_i||tp_i=\mathrm{ProjH}(hp_i;l_c,C_i,pw_i||i;\delta_i) & & \\
\text{if } i=1, \text{ then } tp=tp_i & & \\
\text{if } i\geqslant 2, \text{ then } tp=tp_i\oplus\delta_i & & \\
\tau_c||sk_c=tp & & \\
l_s=\mathrm{sid}||\boldsymbol{Pid}||C_i||\boldsymbol{hp}||VK & & \\
\text{verify } C'_i=\mathrm{Enc}'_{pk'}(pw_i;l_s;tk_i) & \xrightarrow{\quad \tau_c \quad} & \\
 & & \text{verify } \tau_c=\tau_s\\
\text{output } (\mathrm{sid},\boldsymbol{Pid},sk_c) & & \text{output } (\mathrm{sid},\boldsymbol{Pid},sk_s)
\end{array}$$

Figure 2 The UC-secure APAKE protocol UC-APAKE.

is its associated smooth projective hash function family. Denote by $L_m=\{(l,C,m):\exists\delta,\mathrm{VerCom}^l(C,\delta,m)=1\}$ and $L=\cup_{m\in\mathcal{D}\times[n]}L_m$ the underlying languages, where $\mathcal{D}$ is the dictionary and n is the maximum number of users. For the sake of simplicity, we generally omit the languages in the SPHF functions in the following description. Assume that $\mathcal{E}'=(\mathrm{Gen}',\mathrm{Enc}',\mathrm{Dec}')$ is a CCA2-secure labeled public key encryption scheme, and that $\Sigma=(\mathrm{SignKG},\mathrm{Sign},\mathrm{Verfy})$ is a secure one-time signature scheme. For simplicity, we assume that the output of the hash functions in the SPHF family is of an appropriate length, i.e., $l_h=3k$, such that it can be divided into three k-bit substrings. The common reference string of our protocol consists of a common reference string ρ used for the commitment $\mathcal{C}$, and a public key pk' for the CCA-secure public key encryption scheme $\mathcal{E}'$. Note that no protocol participant knows the corresponding secret key with respect to pk'.

Suppose that, in an execution of the protocol, a client P_i is activated by an input $(\mathrm{NewSession},\mathrm{sid},\boldsymbol{Pid},P_i,pw_i,\mathrm{client})$ from the environment, where $\boldsymbol{Pid}$ is an ordered set $\boldsymbol{Pid}=\boldsymbol{\Gamma}\cup\{P_s\}$ and $P_i\in\boldsymbol{\Gamma}$. Alternatively, a server P_s is activated by an input $(\mathrm{NewSession},\mathrm{sid},\boldsymbol{Pid},P_s,\boldsymbol{pw}_s,\mathrm{server})$, where $\boldsymbol{Pid}=\boldsymbol{\Gamma}\cup\{P_s\}$ and the cardinality $|\boldsymbol{pw}_s|$ equals $|\boldsymbol{\Gamma}|$. After the activation, the client first begins to compute and send a commitment, and the server initially enters a waiting stage for a message coming from a client. More precisely, they proceed with the following concrete steps (see also in Figure 2):

(1) The client P_i sets $l_c{:=}\mathrm{sid}||\boldsymbol{Pid}$ and computes a commitment through $(C_i,\delta_i)\leftarrow\mathrm{Com}^{l_c}(pw_i||i)$. Then, it sends the commitment C_i to the trusted server P_s, and stores the opening data δ_i.

(2) Upon receiving message C_i, the sever first chooses independently n hash keys $\boldsymbol{hk}=(hk_1,\ldots,hk_n)$ according to the key generation algorithm HashKG. Next, for every $j=1,2,\ldots,n$, it computes $hp_j=$

Table 1 Comparison of password-only APAKE protocols

Protocol	Assumption	Model	Mutual authentication	Round
APAKE	Random Oracle	Game-based	Yes	3
NAPAKE	Random Oracle	Game-based	No	2
VEAP	Random Oracle	Game-based	Yes	3
Our protocol	Standard	The UC framework	Yes	3

ProjKG$(hk_j; l_c, C_i, pw_j||j)$, $tk_j||tp_j = \text{Hash}(hk_j; l_c, C_i, pw_j||j)$, $\delta_j = tp_1 \oplus tp_j$, where $tk_j \in \{0,1\}^k$ and $tp_j \in \{0,1\}^{2k}$. The server sets $\boldsymbol{hp} = (hp_1, \ldots, hp_n, \delta_2, \ldots, \delta_n)$ and $\tau_s||sk_s = tp_1$. Then, it generates a key pair $(VK, SK) \leftarrow \text{SignKG}(1^k)$ for a one-time signature scheme, sets $l_s = \text{sid}||\boldsymbol{Pid}||C_i||\boldsymbol{hp}||VK$, computes a tuple of ciphertext $\boldsymbol{C}' = (C'_1, C'_2, \ldots, C'_n)$ such that $C'_j = \text{Enc}'_{pk'}(pw_j; l_s; tk_j)$, and computes $\sigma = \text{Sign}_{SK}(\boldsymbol{C}')$. Finally, the server sends the message $(\boldsymbol{hp}, VK, \boldsymbol{C}', \sigma)$ to the client.

(3) When the message $(\boldsymbol{hp}, VK, \boldsymbol{C}', \sigma)$ is received by the client C_i, it first verifies that $\text{Verfy}_{VK}(\sigma, \boldsymbol{C}') = 1$. Then, it selects from $\boldsymbol{hp}$ the projected key hp_i corresponding to the index of P_i in $\boldsymbol{\Gamma}$. By utilizing the password pw_i and opening data δ_i associated with ciphertext C_i, the client computes the hash value through the projected hash algorithm $tk_i||tp_i = \text{ProjH}(hp_i; l_c, C_i, pw_i||i; \delta_i)$. If client P_i is the first element in the ordered set $\boldsymbol{\Gamma}$, it sets $tp = tp_1$; if P_i is not the first client in set $\boldsymbol{\Gamma}$, then it sets $tp = tp_i \oplus \delta_i$. After that, the client P_i sets $\tau_c||sk_c = tp$ and $l_s = \text{sid}||\boldsymbol{Pid}||C_i||\boldsymbol{hp}||VK$, and verifies that for the i-th ciphetext in $\boldsymbol{C}'$, it holds $C'_i = \text{Enc}'_{pk'}(pw_i; l_s; tk_i)$. If the verification fails, the client aborts this session by outputting $(\text{sid}, \boldsymbol{Pid}, \text{error})$; otherwise, it sends the message τ_c to the server, and sets the state as accepted and outputs $(\text{sid}, \boldsymbol{Pid}, sk_c)$.

(4) Upon receiving message τ_c from the client, the server verifies whether $\tau_c = \tau_s$. If the verification fails, the server simply aborts the session and outputs $(\text{sid}, \boldsymbol{Pid}, \text{error})$; otherwise, it sets the state as accepted and outputs $(\text{sid}, \boldsymbol{Pid}, sk_s)$.

Correctness. If the protocol is honestly executed, the client P_i holding a legitimate password $pw_i = \boldsymbol{pw}_s[P_i]$ will definitely compute the same hash value as the server, $\text{ProjH}(hp_i; l_c, C_i, pw_i||i; \delta_i) = \text{Hash}(hk_i; l_c, C_i, pw_i||i)$, according to the correctness of the smooth projective hash function family. Hence, all verifications will succeed, and the client and server will compute equivalent session key $sk_c = sk_s$.

Remark 4. Note that index i is taken into account explicitly in the computation of each commitment, and hence in the computation of every smooth projective hash function. This technique is employed to restrict the power of the adversary to impersonate a particular client once each time, rather than to imitate all clients through only a single on-line query. To explain this more clearly, let us consider an adversary against a protocol in which pw_i is used instead of $pw_i||i$. A commitment is then computed as $\text{Com}^l(pw^*)$ using pw^* and sent to the server. After receiving the response from the server, the adversary can compute the projective hash value through impersonating the i-th client, for all $i = 1, 2, \ldots, n$. If $pw^* = pw_i$ for some $i \in \{1, 2, \ldots, n\}$, the adversary will successfully recompute the corresponding ciphertext C'_i. Under this scenario, the adversary has an advantage n times larger than that of an adversary against our protocol.

4.2 Protocol comparison

In this section, we compare the UC-APAKE protocol with previously proposed APAKE protocols [14, 16, 18], in terms of security and efficiency, as summarized in Table 1.

Security comparison. The most arresting feature of our UC-APAKE protocol is that it provides security guarantees stronger than all previous APAKE protocols [14, 16, 18] in the password-only setting. On the one hand, while previous APAKE protocols were analyzed in game-based models formulating only stand-alone security, our UC-APAKE protocol is provably secure in the UC framework. Recall that stand-alone game-based models cannot guarantee security when the protocol is concurrently executed or composed with other protocols. The security definition and proof in the UC framework guarantee that the UC-APAKE protocol will be universally composable and can thus be used in more realistic environments. Furthermore, our security definition supports arbitrary password distributions. On the other hand, while the security proofs of previous APAKE protocols rely on the random oralce (RO) model, the UC-APAKE protocol has been proven to be secure in the standard model, i.e., without random oracle heuristics. Recall that the RO model is a kind of ideal assumption and might not be instantiable in the real world [36].

Efficiency comparisons. Similar to those PAKE protocols [6–8, 22] proven secure in the standard model, the UC-APAKE protocol is not as efficient as its analogues in the RO model. However, we claim that our protocol is still very practical in terms of efficiency. First, based on the most efficient PAKE protocol [7, 26] in the standard model, the UC-APAKE protocol needs only three rounds (i.e., message flows) while providing mutual authentication. Second, although the computational overhead on the server side is linear in the number of clients, the computation on the client side is very similar to that in [7, 26] and is thus relatively efficient.

5 Proof of security

Denote by $\widehat{\mathcal{F}}_{\rm APAKE}$ the multi-session extension of the functionality $\mathcal{F}_{\rm APAKE}$. Let $\mathcal{F}_{\rm CRS}$ be an ideal functionality that provides a CRS to all parties. The following theorem asserts that our APAKE protocol presented in Subsection 4.1 securely realizes $\widehat{\mathcal{F}}_{\rm APAKE}$ in the UC framework with joint state.

Theorem 1. Assume that $\mathcal{C}$ is an SPHF-friendly E^2- commitment scheme, $\mathcal{H} = ({\rm HashKG}, {\rm ProjKG}, {\rm Hash}, {\rm ProjH})$ is an associated smooth projective hash function family, $\mathcal{E}' = ({\rm Gen}', {\rm Enc}', {\rm Dec}')$ is a CCA2-secure labeled public key encryption scheme, and $\Sigma = ({\rm SignKG}, {\rm Sign}, {\rm Verfy})$ is a secure one-time signature scheme. Then, the APAKE protocol UC-PAKE described in Subsection 4.1 realizes $\widehat{\mathcal{F}}_{\rm APAKE}$ in the $\mathcal{F}_{\rm CRS}$ hybrid model, in the presence of static adversaries.

Proof. To prove that the protocol securely realizes the functionality $\widehat{\mathcal{F}}_{\rm APAKE}$, we have to show that for any real-world PPT adversary $\mathcal{A}$, there exists an ideal-world adversary (i.e., the simulator) $\mathcal{S}$ such that no PPT environment can distinguish an execution with the protocol UC-PAKE and the adversary $\mathcal{A}$ from an execution with the ideal protocol of $\widehat{\mathcal{F}}_{\rm APAKE}$ and the simulator $\mathcal{S}$. Note that we prove the theorem in the context of the UC framework with joint state [30], and henceforth use $({\rm sid}, {\rm ssid})$ in place of sid when the latter is needed. We also use ssid occasionly for simplicity. In the following, we first describe the construction of the simulator, and then prove the indistinguishability of two executions.

5.1 Description of the simulator

The simulator $\mathcal{S}$ is essentially an ideal-world adversary, that interacts with the functionality $\widehat{\mathcal{F}}_{\rm APAKE}$ and the environment. To guarantee that the view of the environment in the ideal-world is indistinguishable from its view in the real-world, $\mathcal{S}$ has to invoke the real-world adversary $\mathcal{A}$, by simulating all other entities for $\mathcal{A}$. Then, for the most part, the simulator $\mathcal{A}$ simply follows the action of adversary $\mathcal{A}$ appropriately.

When the simulator $\mathcal{S}$ is initialized using security parameter k, it first runs the trapdoor commitment setup algorithm $(\rho, \tau) \leftarrow {\rm SetupComT}(1^k)$ and $(pk', sk') \leftarrow {\rm Gen}'(1^k)$. Next, $\mathcal{S}$ initializes the real-world adversary $\mathcal{A}$ and gives it (ρ, pk') as the common reference string. In the simulated world, we say that a message is oracle-generated if it was computed and sent by an honest session and is delivered to its destination without any modifications; otherwise, we deem it as non-oracle-generated.

C1. When receiving a message $({\rm NewSession}, {\rm sid}, {\rm ssid}, \boldsymbol{Pid}, {\rm client})$ from the functionality $\widehat{\mathcal{F}}_{\rm APAKE}$, the simulator $\mathcal{S}$ activates a new session of the APAKE protocol UC-PAKE for party identity ${\rm PID}^{\langle c\rangle} = ({\rm PID}_{\rm A}^{\langle c\rangle} = \boldsymbol{Pid}||{\rm client}, {\rm PID}_{\rm I}^{\langle c\rangle} = {\rm Null})$, with session identifier ssid. We denote the session by $({\rm PID}_{\rm A}^{\langle c\rangle}, {\rm ssid})$. Next, $\mathcal{S}$ computes $(C, {\rm eqk}) \leftarrow {\rm SimCom}^{l_c}(\tau)$, where $l_c = ({\rm sid}||\boldsymbol{Pid})$. After that, it gives C to $\mathcal{A}$ as if it is a message from the client session $({\rm PID}_{\rm A}^{\langle c\rangle}, {\rm ssid})$ to a server session.

S1. When receiving a message $({\rm NewSession}, {\rm sid}, {\rm ssid}, \boldsymbol{Pid}, {\rm server})$, the simulator $\mathcal{S}$ activates a new session for party identity ${\rm PID}^{\langle s\rangle} = ({\rm PID}_{\rm A}^{\langle s\rangle} = \boldsymbol{Pid}||{\rm server}, {\rm PID}_{\rm I}^{\langle s\rangle} = P_s)$, with session identifier ssid. Denote the session by $({\rm PID}_{\rm A}^{\langle s\rangle}, {\rm ssid})$.

S2. When a server session $({\rm PID}_{\rm A}^{\langle s\rangle}, {\rm ssid})$ receives C from a client session $({\rm PID}_{\rm A}^{\langle c\rangle}, {\rm ssid})$, do:

• If C is non-oracle-generated, let the simulator $\mathcal{S}$ extract the string $pw'_i||i$ and obtain $P'_i = \boldsymbol{\Gamma}[i]$. Then, $\mathcal{S}$ asks $(\text{TestPwd}, \text{sid}, \text{ssid}, \text{server}, P'_i, pw'_i)$ to the functionality $\widehat{\mathcal{F}}_{\text{APAKE}}$. If $\mathcal{S}$ receives a "correct guess" message, it sets $\boldsymbol{pw}_s[P'_i] = pw'_i$, and $\boldsymbol{pw}_s[P] = pw_0$ for all $P \in \boldsymbol{\Gamma} \setminus P'_i$, where pw_0 is a predetermined password (called a dummy password). Next, $\mathcal{S}$ simulates an honest server session $(\text{PID}_{\text{A}}^{\langle s\rangle}, \text{ssid})$ by selecting random hash keys $\boldsymbol{hk}$, generating (VK, SK) and computing $\boldsymbol{hp}, \tau_s||sk_s, \sigma$ from $\boldsymbol{pw}_s$. It then sends the message $(\boldsymbol{hp}, VK, \boldsymbol{C}', \sigma)$ to the adversary $\mathcal{A}$. Note that the corresponding server session in the ideal world is compromised in this case. If $\mathcal{S}$ receives a "wrong guess" message, it chooses $tk_j||tp_j, j = 1, \ldots, n$ as uniformly random strings, rather than computing them as hash values. The server $\mathcal{S}$ then computes $C'_j = \text{Enc}'_{pk'}(pw_0; l_s; tk_j), j = 1, \ldots, n$. The other computations are conducted as specified. Note that the corresponding server session in the ideal world is interrupted in this case.

• If C is oracle-generated, the server session processes similar to that when $\mathcal{S}$ receives a "wrong guess" message. That is, $tk_j||tp_j, j = 1, \ldots, n$ are selected randomly and $C'_j = \text{Enc}'_{pk'}(pw_0; l_s; tk_j), j = 1, \ldots, n$. The remaining messages are computed as specified. Next, the simulator $\mathcal{S}$ sends a query $(\text{ServerReady}, \text{sid}, \text{ssid}, \boldsymbol{Pid})$ to the functionality. Note that the server session is fresh initially and its status will be changed from fresh to ready in this situation.

C2. When a client session $(\text{PID}_{\text{A}}^{\langle c\rangle}, \text{ssid})$ receives a message $(\boldsymbol{hp}, VK, \boldsymbol{C}', \sigma)$ from a server session $(\text{PID}_{\text{A}}^{\langle s\rangle}, \text{ssid})$, do:

• If this message is oracle-generated, the sever session $(\text{PID}_{\text{A}}^{\langle s\rangle}, \text{ssid})$ must have received a message C that is claimed to be from the client session $(\text{PID}_{\text{A}}^{\langle c\rangle}, \text{ssid})$ sometimes before. If C is not oracle-generated by the client session $(\text{PID}_{\text{A}}^{\langle c\rangle}, \text{ssid})$, the simulator $\mathcal{S}$ simply lets the client session reject by outputting $(\text{sid}, \text{ssid}, \boldsymbol{Pid}, \text{error})$. Next, it sends a query $(\text{NewKey}, \text{sid}, \text{ssid}, \boldsymbol{Pid}, \text{client}, sk)$ to the functionality. Note that the client session in the ideal world is fresh, but with no partnered server session that is ready. Therefore, an error message will be outputted to the corresponding client session. If C is exactly the same message oracle-generated by the client session $(\text{PID}_{\text{A}}^{\langle c\rangle}, \text{ssid})$ previously, the simulator sets $\tau_c||sk_c = tp_1$, which is equivalent to the value in the server session $(\text{PID}_{\text{A}}^{\langle s\rangle}, \text{ssid})$. Next, $\mathcal{S}$ sends the query $(\text{NewKey}, \text{sid}, \text{ssid}, \boldsymbol{Pid}, \text{client}, sk_c)$ to the functionality. Note that the corresponding client session in the ideal world is fresh while there is a partnered server session that is ready. Therefore, if both sessions have matching passwords, a random session key sk'_c will be selected for the client session. The simulator $\mathcal{S}$ then sends the message τ_c as if it is from $(\text{PID}_{\text{A}}^{\langle c\rangle}, \text{ssid})$ to $(\text{PID}_{\text{A}}^{\langle s\rangle}, \text{ssid})$. However, if they have distinct passwords, an error message will be responded for the NewKey query. Then, $\mathcal{S}$ sends a randomly selected string τ_c.

• If this message is non-oracle-generated, the simulator first checks whether the signature is valid. Otherwise, it simply terminates this session by outputting $(\text{sid}, \text{ssid}, \boldsymbol{Pid}, \text{error})$. Next, $\mathcal{S}$ decrypts the ciphertexts $\boldsymbol{C}' = (C'_1, C'_2, \ldots, C'_n)$ via the secret key sk' generated by Gen' at the beginning of the simulation, with label $l_s = \text{sid}||\boldsymbol{Pid}||C||\boldsymbol{hp}||Vk$, where C is the message previously sent by the client session. If one of the decryption algorithm outputs $\perp$, the simulator terminates this session by outputting an error message. If all decryptions are successful, the simulator will obtain a vector of passwords $\boldsymbol{pw}_s = (pw_1, \ldots, pw_n)$. Next, it decommits C to the message $pw_j||j$ for each $j \in \{1, \ldots, n\}$ as $\delta_j = \text{OpenCom}^{l_c}(C, \text{eqk}, pw_j||j)$. Then, it simulates the remaining computation in the client session as if the client is $P_j = \boldsymbol{\Gamma}[j]$. If there exists one j such that the verification of ciphertext C'_j is valid, the simulator sets $P_j = \boldsymbol{\Gamma}[j]$ and queries $(\text{TestPwd}, \text{sid}, \text{ssid}, \text{client}, P_j, pw_j)$ to the functionality. If the response is "correct guess", the simulator $\mathcal{S}$ asks a query $(\text{NewKey}, \text{sid}, \text{ssid}, \boldsymbol{Pid}, \text{client}, sk_c)$ and sends the third flow of message τ_c using the values $\tau_c||sk_c$ computed above. Note that the corresponding client session in the ideal world is compromised and the session key sk_c will be transmitted from the simulated-world session to the ideal-world session. If the response is "wrong guess", the simulator $\mathcal{S}$ still asks the query $(\text{NewKey}, \text{sid}, \text{ssid}, \boldsymbol{Pid}, \text{client}, sk_c)$, but lets the simulated client session terminate and output $(\text{sid}, \text{ssid}, \boldsymbol{Pid}, \text{error})$, with no message delivered to the server session any more. Note that the corresponding client session in the ideal world is interrupted and a new random key sk'_c will be selected, regardless of the value sk_c included in the NewKey query.

S3. When a server session $(\text{PID}_{\text{A}}^{\langle s\rangle}, \text{ssid})$ receives a message τ_c from a client session $(\text{PID}_{\text{A}}^{\langle c\rangle}, \text{ssid})$, it compares whether $\tau_s = \tau_c$ and outputs sk_s or error depending on whether the verification successes and fails respectively. Next, $\mathcal{S}$ sends the query $(\text{NewKey}, \text{sid}, \text{ssid}, \boldsymbol{Pid}, \text{server}, sk_s)$ to the functionality. Recall that this server session might be in one of three states, i.e., compromised, interrupted, or ready, after the simulation of step S2. The session key generated for the corresponding server session in the ideal world will then differ according to these different states.

5.2 Proof of indistinguishability

In this section, we prove the indistinguishability between the real world execution and the ideal world execution. To achieve this aim, we incrementally define a sequence of hybrid games. It starts from the real world game, in which an adversary interacts with real parties and the environment, and ends with the simulated games, in which we construct a simulator (as an adversary in the ideal world) that interacts with the functionality and the environment, by invoking the real world adversary as a subroutine. In each game, we modify certain aspects of the simulation and prove that these modifications incur a negligible change in the view of the environment.

Note that, the simulator in the ideal world cannot obtain the passwords of honest parties from the functionality, unless it correctly guesses the password for an honest session. Therefore, the key point in the construction of the simulator $\mathcal{S}$ is to simulate the protocol participants without achieving their passwords. In the following, we first assume a simulator that is given access to all passwords for honest parties. Then, we gradually weaken this assumption in the game sequence. Finally, we restrict the ability of the simulator in obtaining the knowledge of these passwords only by querying TestPwd and NewKey, meeting the specifications of an ideal world adversary.

Game G_0. This is the real world game. In this game, the simulator $\mathcal{S}$ is assumed to have access to all passwords sent by the environment to all parties. Then, it computes $\rho \leftarrow \text{SetupCom}(1^k)$ and $(pk', sk') \leftarrow \text{Gen}'(1^k)$, sets (ρ, pk') as the common reference string, and simulates all honest parties according to the specifications of the protocol UC-PAKE for a real world adversary $\mathcal{A}$.

Game G_1. In this game, we change the way in which the common reference string is generated. Specifically, the simulator uses the trapdoor setup algorithm $\text{SetupComT}(1^k)$ to replace the original setup algorithm $\text{SetupCom}(1^k)$, obtaining $(\rho, \tau) \leftarrow \text{SetupComT}(1^k)$ and recording the trapdoor τ. Moreover, when a key pair is generated as $(pk', sk') \leftarrow \text{Gen}'(1^k)$, the simulator records the secret key sk' as well. Then, $\mathcal{S}$ provides the real world adversary $\mathcal{A}$ with (ρ, pk') as the common reference string.

Recall that the commitment scheme satisfies the setup indistinguishability property, which guarantees that the common reference string generated by SetupComT be computationally indistinguishable from that by SetupCom. It is easy to conclude that the modification in this game changes the view of the environment $\mathcal{Z}$ at most negligibly.

Game G_2. In this game we modify the way in which client sessions are simulated. Consider a case in which a client session $(\text{PID}_{\text{A}}^{\langle c \rangle}, \text{ssid})$ receives a message $(\boldsymbol{hp}, VK, \boldsymbol{C}', \sigma)$ that is oracle-generated by certain server session $(\text{PID}_{\text{A}}^{\langle s \rangle}, \text{ssid})$. We classify this message into two different subcases. Recall that the corresponding server session must be honest, hold the same ssid and $\boldsymbol{Pid}$ with the client session and, moreover, receive a message C that is claimed to have been sent from $(\text{PID}_{\text{A}}^{\langle c \rangle}, \text{ssid})$. If message C received by $(\text{PID}_{\text{A}}^{\langle s \rangle}, \text{ssid})$ is exactly the message oracle-generated by $(\text{PID}_{\text{A}}^{\langle c \rangle}, \text{ssid})$, we say that the message $(\boldsymbol{hp}, VK, \boldsymbol{C}', \sigma)$ is matching-server-generated; otherwise, we say it is not matching-server-generated.

If $(\boldsymbol{hp}, VK, \boldsymbol{C}', \sigma)$ is matching-server-generated by $(\text{PID}_{\text{A}}^{\langle s \rangle}, \text{ssid})$, and if the client session possesses a password consistent with that of the server session, $\mathcal{S}$ simulates the client session as follows. It omits the verification procedure and simply sets $\tau_c || sk_c = tp_1$, where tp_1 is the internal value computed by the server session. In other cases, either the incoming message is not matching-server-generated or they have inconsistent passwords, and the simulation is treated the same as in the previous game.

Note that when a matching-server-generated message is received, and both sessions have consistent passwords, the verification procedure on the client side will certainly be successful. As a result, this game is indistinguishable from the prior game owing to the correctness of the smooth projective hash functions.

Game G_3. In this game, we continue to deal with the client sessions that receive matching-server-generated messages $(\boldsymbol{hp}, VK, \boldsymbol{C}', \sigma)$. Recall that in game G_2 the simulator is assumed to have access to the passwords held by all parties. Here, we make no such assumptions, and let the simulator request a NewKey query instead. If the response indicates that the client session in the ideal world is completed, we treat the simulated client session as it has an password equal to the corresponding server session. If the response indicates that the client session in the ideal world reaches an error state, we treat the simulated client session as having an inconsistent password with the corresponding server session.

Game G_4. In this game, we again change the simulation of the client sessions. More concretely, when an honest client session is activated, the simulator now computes $(C_i, \delta_i) \leftarrow \mathrm{SCom}^{l_c}(pw_i||i)$ via the trapdoor τ instead of $(C_i, \delta_i) \leftarrow \mathrm{Com}^{l_c}(pw_i||i)$. Recall that the simulated commitment algorithm $(C_i, \delta_i) \leftarrow \mathrm{SCom}^{l_c}(pw_i||i)$ is composed of two steps, $(C_i, \mathrm{eqk}_i) \leftarrow \mathrm{SimCom}^{l_c}(\tau)$ and $\delta_i \leftarrow \mathrm{OpenCom}^{l_c}(C_i, \mathrm{eqk}_i, pw_i||i)$.

One can verify that the difference between this game and the former one is negligible through a standard hybrid argument technique, which reduces the difference to the strong simulation indistinguishability of the underlying commitment scheme. In each hybrid game, we can apply the $\mathrm{Exp}_{\mathcal{A}}^{\text{s-sim-ind-b}}$ security experiment to one honest client session. Recall that there are at most polynomial many (denoted by poly(k)) honest client sessions. Then, the difference is bounded by $\mathrm{poly}(k) \cdot \mathrm{Adv}_{\mathcal{C}}^{\text{s-sim-ind}}(t)$, which is negligible when $\mathrm{Adv}_{\mathcal{C}}^{\text{s-sim-ind}}(t)$ is negligible for security parameter k.

Game G_5. In this game, we deal with honest server sessions receiving oracle-generated messages in the first flow. In detail, if an honest server session $(\mathrm{PID}_{\mathrm{A}}^{\langle s \rangle}, \mathrm{ssid})$ receives a message C oracle-generated by a client session $(\mathrm{PID}_{\mathrm{A}}^{\langle c \rangle}, \mathrm{ssid})$, we choose uniformly random values $tk_j||tp_j$ instead of computing them as hash values, as specified in the protocol. Recall that the session keys in honest client sessions that receive matching-server-generated messages have been replaced in game G_2, and thus no inconsistencies will be incurred in this game.

To prove the indistinguishability between this game and the previous one, two subcases should be considered, according to whether these two sessions possess consistent passwords. If their passwords are inconsistent, we can easily reduce the difference to the smooth property of the underlying smooth projective hash functions with invalid inputs. On the other hand, if they have consistent passwords, the indistinguishability then reduces to the case in which the output of a smooth projective hash function is pseudorandom when the witness δ and the hash key $\boldsymbol{hk}$ are kept secret.

Game G_6. In this game we continue to deal with honest server sessions receiving oracle-generated messages in the first flow, via modifying the approach that ciphertexts $\{C'_j\}$ are generated. Now, we do not compute C'_j as a ciphertext of pw_j but as a ciphertext of the dummy password pw_0. Recall that all $\{tk_j\}$ were replaced by random values in the previous game. Through a hybrid proof, the CPA security of the encryption scheme Enc' then guarantees that the difference between this game and the prior one will be negligible.

Game G_7. In this game we modify the simulation of honest server sessions receiving non-oracle-generated messages in the first flow as follows. If an honest server session $(\mathrm{PID}_{\mathrm{A}}^{\langle s \rangle}, \mathrm{ssid})$ receives a message C that is non-oracle-generated, we extract the underlying message through the trapdoor τ to obtain $pw_i||i = \mathrm{ExtCom}^{l_c}(\tau, C)$. We then let the simulator ask the functionality $\widehat{\mathcal{F}}_{\mathrm{APAKE}}$ a NewKey query with $P_i = \boldsymbol{\Gamma}[i]$ and pw_i. If the response is "correct guess", nothing changes. Otherwise, if the response is "wrong guess", we change all $\{tk_j||tp_j\}$ values in the corresponding server session to uniformly random.

First, a non-oracle-generated message is always extractable due to robust binding extractability of the commitment scheme. Hence, because of the smooth property of the underlying smooth projective hash functions, the above modification does not lead to non-negligible difference.

Game G_8. In this game, we again handle client sessions receiving oracle-generated messages in the form of $(\boldsymbol{hp}, VK, \boldsymbol{C}', \sigma)$. Recall that matching-server-generated messages have been treated in game G_2, G_3. Thus, if a message $(\boldsymbol{hp}, VK, \boldsymbol{C}', \sigma)$ received by a client session $(\mathrm{PID}_{\mathrm{A}}^{\langle c \rangle}, \mathrm{ssid})$ is not matching-server-generated, we simply let the client session output an error message and abort this session.

When the oracle-generated message $(\boldsymbol{hp}, VK, \boldsymbol{C}', \sigma)$ received by $(\mathrm{PID}_{\mathrm{A}}^{\langle c \rangle}, \mathrm{ssid})$ from $(\mathrm{PID}_{\mathrm{A}}^{\langle s \rangle}, \mathrm{ssid})$ is not matching-server-generated, we claim that the label l_s used in $\boldsymbol{C}'$ includes a message C that is not equal to the one held by the corresponding client session. Subsequently, the CCA2 non-malleable security (which is equivalent to CCA2 indistinguishable security) of the labled encryption scheme $\mathcal{E}'$ implies that the modification made in this game alters the view of the environment with at most negligible probability.

Game G_9**.** In this game, we handle client sessions each of which receives a non-oracle-generated message $(\boldsymbol{hp}, VK, \boldsymbol{C}', \sigma)$, by letting the simulator $\mathcal{S}$ behave in the same way as that of a non-oracle-generated message is received in step C2 of Subsection 5.1. It can be easily verified that this step is perfectly simulated without any access to the client's password, by using the de-commitment trapdoor τ and the decryption secret key sk' of $\mathcal{E}'$.

Until now, we have constructed a simulator $\mathcal{S}$ that interfaces with a real-world adversary $\mathcal{A}$ and the functionality $\widehat{\mathcal{F}}_{\mathrm{APAKE}}$ in the ideal world, without access to the passwords of the parties directly. However, $\mathcal{S}$ can still check that whether a tested password is correct through TestPwd queries, and can check whether two sessions possess consistent passwords through NewKey queries. We can conclude that, the real-world protocol execution is indistinguishable from the execution in the simulated world, and is hence indistinguishable from the ideal-world execution.

6 Conclusion

In this paper, we formulated an ideal functionality for APAKE within the UC framework, which captures the expected security requirements such as off-line dictionary attack resistance, client anonymity and explicit mutual authentication under protocol composition. An APAKE protocol was also proposed using generic cryptographic primitives, i.e., equivocable and extractable commitments and smooth projective hash functions. It was proven that the proposed protocol securely realizes the APAKE functionality in the standard model. This is the first APAKE protocol secure within the UC framework.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61502527, 61170278, 61170279, 61379150), National Basic Research Program of China (Grant Nos. 2013CB338003, 2012CB315905), National High Technology Research and Development Program of China (Grant No. 2012AA01A 403), China Postdoctoral Science Foundation (Grant No. 2014M552524), and Foundation of Science and Technology on Information Assurance Laboratory (Grant No. KJ-14-004).

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Wang S B, Zhu Y, Ma D, et al. Lattice-based key exchange on small integer solution problem. Sci China Inf Sci, 2014, 57: 112111

2 Zhang J, Zhang Z F, Ding J D, et al. Authenticated key exchange from ideal lattices. In: Oswald E, Fischlin M, eds. Advances in Cryptology–EUROCRYPT 2015, LNCS 9057. Berlin: Springer, 2015. 719–751

3 Camenisch J, Lehmann A, Lysyanskaya A, et al. Memento: how to reconstruct your secrets from a single password in a hostile environment. In: Garay J, Gennaro R, eds. Advances in Cryptology–CRYPTO 2014, LNCS 8617. Berlin: Springer, 2014. 256–275

4 Bellovin S M, Merritt M. Encrypted key exchange: password-based protocols secure against dictionary attacks. In: Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy, Los Alamitos, 1992. 72–84

5 Bellare M, Pointcheval D, Rogaway P. Authenticated key exchange secure against dictionary attacks. In: Preneel B, ed. Advances in Cryptology–EUROCRYPT 2000, LNCS 1807. Berlin: Springer, 2000. 139–155

6 Katz J, Ostrovsky R, Yung M. Efficient password-authenticated key exchange using human-memorable passwords. In: Pfitzmann B, ed. Advances in Cryptology–EUROCRYPT 2001, LNCS 2045. Berlin: Springer, 2001. 475–494

7 Jiang S Q, Gong G. Password based key exchange with mutual authentication. In: Handschuh H, Hasan M, eds. Selected Areas in Cryptography, LNCS 3357. Berlin: Springer, 2005. 267–279

8 Benhamouda F, Blazy O, Chevalier C, et al. New techniques for SPHFs and efficient one-round PAKE protocols. In: Canetti R, Garay J, eds. Advances in Cryptology–CRYPTO 2013, LNCS 8042. Berlin: Springer, 2013. 449–475

9 Chien H Y, Wu T C, Yeh M K. Provably secure gateway-oriented password-based authenticated key exchange protocol resistant to password guessing attacks. J Inf Sci Eng, 2013, 29: 249–265

10 Li W M, Wen Q Y, Su Q, et al. Password-authenticated multiple key exchange protocol for mobile applications. China Commun, 2012, 9: 64–72

11 IEEE. IEEE standard specifications for password-based public-key cryptographic techniques. IEEE Std 1363.2-2008. doi: 10.1109/IEEESTD.2009.4773330

12 Sheffer Y, Zorn G, Tschofenig H, et al. An EAP authentication method based on the encrypted key exchange (EKE) protocol. RFC 6124. https://www.rfc-editor.org/info/rfc6124

13 Lindell Y. Anonymous authentication. J Priv Confidentiality, 2007, 2: 35–63

14 Viet D, Yamamura A, Tanaka H. Anonymous password-based authenticated key exchange. In: Maitra S, Veni M C, Venkatesan R, eds. Progress in Cryptology–INDOCRYPT 2005, LNCS 3797. Berlin: Springer, 2005. 244–257

15 Shin S, Kobara K, Imai H. A secure threshold anonymous password-authenticated key exchange protocol. In: Miyaji A, Kikuchi H, Rannenberg K, eds. Advances in Information and Computer Security, LNCS 4752. Berlin: Springer, 2007. 444–458

16 Yang J, Zhang Z F. A new anonymous password-based authenticated key exchange protocol. In: Chowdhury D, Rijmen V, Das A, eds. Progress in Cryptology–INDOCRYPT 2008, LNCS 5365. Berlin: Springer, 2008. 200–212

17 Jablon D P. Strong password-only authenticated key exchange. ACM SIGCOMM Comput Commun Rev, 1996, 26: 5–26

18 Shin S, Kobara K, Imai H. Anonymous password-authenticated key exchange: new construction and its extensions. IEICE Trans Fund Electron Commun Comput Sci, 2010, 93: 102–115

19 Yang Y J, Zhou J Y, Weng J, et al. A new approach for anonymous password authentication. In: Proceedings of the 25th Annual Computer Security Applications Conference, Honolulu, 2009. 199–208

20 Yang Y J, Zhou J Y, Wong J W, et al. Towards practical anonymous password authentication. In: Proceedings of the 26th Annual Computer Security Applications Conference. New York: ACM, 2010. 59–68

21 Qian H F, Gong J Q, Zhou Y. Anonymous password-based key exchange with low resources consumption and better user-friendliness. Secur Commun Netw, 2012, 5: 1379–1393

22 Abdalla M, Benhamouda F, Pointcheval D, et al. SPOKE: simple password-only key exchange in the standard model. Cryptology ePrint Archive, Report 2014/609. https://eprint.iacr.org/eprint-bin/versions.pl?entry=2014/609

23 Canetti R. Universally composable security: a new paradigm for cryptographic protocols. In: Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, Washington, 2001. 136–145

24 Canetti R, Halevi S, Katz J, et al. Universally composable password-based key exchange. In: Cramer R, ed. Advances in Cryptology–EUROCRYPT 2005, LNCS 3494. Berlin: Springer, 2005. 404–421

25 Abdalla M, Catalano D, Chevalier C, et al. Efficient two-party password-based key exchange protocols in the UC framework. In: Malkin T, ed. Topics in Cryptology–CT-RSA 2008, LNCS 4964. Berlin: Springer, 2008. 335–351

26 Groce A, Katz J. A new framework for efficient password-based authenticated key exchange. In: Proceedings of the 17th ACM Conference on Computer and Communications Security–CCS'10. New York: ACM, 2010. 516–525

27 Hu X X, Zhang Z F, Liu W F. Universal composable password authenticated key exchange protocol in the standard model (in Chinese). J Softw, 2011, 22: 2820–2832

28 Abdalla M, Benhamouda F, Blazy O, et al. SPHF-friendly non-interactive commitments. In: Sako K, Sarkar P, eds. Advances in Cryptology–ASIACRYPT 2013, LNCS 8269. Berlin: Springer, 2013. 214–234

29 Gennaro R, Lindell Y. A framework for password-based authenticated key exchange. In: Biham E, ed. Advances in Cryptology–EUROCRYPT 2003, LNCS 2656. Berlin: Springer, 2003. 524–543

30 Canetti R, Rabin T. Universal composition with joint state. In: Boneh D, ed. Advances in Cryptology-CRYPTO 2003, LNCS 2729. Berlin: Springer, 2003. 265–281

31 Abdalla M, Chevalier C, Pointcheval D. Smooth projective hashing for conditionally extractable commitments. In: Halevi S, ed. Advances in Cryptology–CRYPTO 2009, LNCS 5677. Berlin: Springer, 2009. 671–689

32 Cramer R, Shoup V. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In: Knudsen L, ed. Advances in Cryptology–EUROCRYPT 2002, LNCS 2332. Berlin: Springer, 2002. 45–64

33 Katz J, Vaikuntanathan V. Round-optimal password-based authenticated key exchange. J Cryptol, 2013, 26: 714–743

34 Haralambiev K. Efficient cryptographic primitives for non-interactive zero-knowledge proofs and applications. Dissertation for Ph.D. Degree. New York: New York University, 2011

35 Cramer R, Shoup V. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk H, ed. Advances in Cryptology–CRYPTO'98, LNCS 1462. Berlin: Springer, 1998. 13–25

36 Bellare M, Boldyreva A, Palacio A. An uninstantiable random oracle model scheme for a hybrid-encryption problem. In: Cachin C, Camenisch J, eds. Advances in Cryptology–EUROCRYPT 2004, LNCS 3027. Berlin: Springer, 2004. 171–188