Password Authenticated Keyword Search

Kaibin Huang
Department of Computer Science
National Chengchi University
Taiwan
Email: 100753504@nccu.edu.tw

Mark Manulis
Surrey Centre for Cyber Security
University of Surrey
United Kingdom
Email: mark@manulis.eu

Liqun Chen
Surrey Centre for Cyber Security
University of Surrey
United Kingdom
Email: liqun.chen@surrey.ac.uk

***Abstract*—In this paper we introduce *Password Authenticated Keyword Search (PAKS)*, a cryptographic scheme where any user can use a single human-memorizable password to outsource encrypted data with associated keywords to a group of servers and later retrieve this data through the encrypted keyword search procedure. PAKS ensures that only the legitimate user who knows the initially registered password can perform outsourcing and retrieval of the encrypted data. In particular, PAKS guarantees that no single server can mount an offline attack on the user's password or learn any information about the encrypted keywords. The concept behind PAKS protocols extends previous concepts behind searchable encryption by removing the requirement on the client to store high-entropy keys, thus making the protocol device-agnostic on the user side. In this paper we model three security requirements for PAKS schemes (indistinguishability against chosen keyword attacks, authentication and consistency) and propose an efficient direct construction in a two-server setting those security we prove in the standard model under the Decisional Diffie-Hellman assumption. Our efficiency comparison shows that the proposed scheme is practical and offers high performance in relation to computations and communications on the user side.**

I. Introduction

Searchable encryption. Using protocols for *Searchable Encryption* [1]–[4] clients with limited computing and storage resources can outsource encrypted data to a server or a collection of servers, perform search over the encrypted data (typically using encrypted keywords) and eventually retrieve searched data while preserving its privacy against the servers. Existing searchable encryption schemes can be broadly split into those where the keyword search procedure requires either high-entropy shared keys such as *Symmetric Searchable Encryption (SSE)* schemes from [3], [5], [6] or a private-public key pair such as *Public Key Encryption with Keyword Search (PEKS)* schemes from [1], [2], [6]–[10] on the user side.

In practice, the requirement to maintain high-entropy keys on the user side results in less flexibility when it comes to the use of multiple, different devices for outsourcing and retrieval of data. The user is effectively prevented from using different devices unless the private key is made available to every such device.

Password Authenticated Keyword Search (PAKS). The idea of basing searchable encryption solely on passwords, proposed in this paper, helps to avoid costly and risky key management on the user side and enables the whole process to be device-agnostic. This, however, comes with challenges considering that both passwords and keywords typically have low entropy. Amongst the core security properties of PAKS there is a need to guarantee that only the legitimate user, who knows the password, can outsource, search and retrieve data. Hence, basing security of searchable encryption schemes on passwords introduces the need for a distributed server environment where trust is spread across at least two non-colluding servers, as is also the case in many password-based protocols for authentication and secret sharing, e.g. [11]–[19]. While a more general secret sharing architecture with t-out-of-n servers would be applicable as well, the use of two servers can be seen as the most practical scenario and the minimum requirement to achieve protection against offline dictionary attacks. The use of two servers is required not only to protect passwords but also to prevent keyword guessing attacks, like those demonstrated in [10] for (public key-based) PEKS schemes.

We model PAKS as a cryptographic scheme where users can register their passwords with the servers and then re-use these passwords for multiple sessions of the outsource and retrieval protocols. In each outsource session the user can outsource encrypted keywords along with some (encrypted) document to both servers. The retrieval protocol realises the search procedure based on the keyword that the user inputs to the protocol and provides the user with all documents associated with that keyword allowing the user to also verify the integrity of the retrieved documents. We define security of the PAKS scheme using BPR-like models [20], [21] that have been widely used for password-based protocols. We define privacy of PAKS keywords through *indistinguishability against chosen keyword attacks (IND-CKA)* while considering active adversaries, possibly in control of at most one server, who can also register own passwords in the system. While IND-CKA security protects against the adversary who does not know the password from successfully retrieving outsourced data, we additionally require *authentication* to protect the outsourcing operation itself, thus preventing the adversary from outsourcing data on behalf of the user; this requirement must also hold even if the adversary controls one of the servers. Our third PAKS requirement, *consistency*, akin to [1], ensures the correctness of the retrieval process, in particular prevents cases where one keyword is used to outsource data that can then be retrieved using a different keyword.

Our direct PAKS construction follows conceptually the

978-1-5386-1027-5/17 $31.00 © 2017 IEEE
DOI 10.1109/PAC.2017.11

following more general approach that combines ideas behind *Password Authenticated Secret Sharing (PASS)* [11]–[17] and SSE [3], [5], [6]. In the registration phase, the user picks a password π and a high-entropy symmetric key K that will be used to encrypt keywords and secret-shares K protected with π across both servers. In order to outsource keywords the user engages into the PASS reconstruction protocol to obtain K and then into the SSE outsource protocol to outsource the keywords. In order to search for keywords and retrieve data the user again reconstructs K using PASS and performs the keyword search using SSE. We stress however that our construction is direct and does not use PASS and SSE as generic building blocks. A generic construction from these two primitives remains currently out of reach due to significant differences in the syntax, functionality and security amongst the existing PASS protocols. First, PASS protocols do not separate registration from secret sharing phase and therefore do not enforce user authentication upon secret sharing which would be required for the outsourcing protocol in PAKS. Existing PASS protocols were proven in different security models, e.g. BPR-like in [11], [15] and UC-based in [12], [16], [17], and do not necessarily follow the same functionality and syntax, which makes it hard to use PASS as a generic building block in PAKS without revising the syntax and security models of those PASS protocols. While we could update the syntax of PASS protocols to allow for a generic usage in PAKS such update would introduce changes to the original PASS protocols and require new security proofs. Moreover, generic constructions often lead to less efficient instantiations than directly constructed schemes. For all the aforementioned reasons we are not formally proposing a generic PAKS construction in this paper and opt for a direct and efficient scheme (cf. Section IV) based on well-known assumptions in the standard model.

Paper organization. We introduce preliminaries and building blocks in Section II. The syntax and security of the PAKS scheme are formalized in Section III. Our direct PAKS construction (along with efficiency considerations and comparison with prior work) is proposed in Section IV and its security is analysed in Section V.

II. Preliminaries and Building Blocks

Pedersen commitments [22]. Let g, h be two generators in a multiplicative cyclic group $\mathbb{G}$ with order q, and the discrete logarithm between h and g is unknown. For a message $m \in \mathbb{Z}_q^*$, the Pedersen commitment is computed as $\mathtt{c} \leftarrow g^r h^m$ where $r \xleftarrow{\$} \mathbb{Z}_q^*$ and is opened by providing (r, m). We recall that Pedersen commitments offer computational binding based the discrete logarithm problem, i.e. assuming $Adv^{\mathtt{DL}}_{\mathcal{A}}(\kappa)$ is negligible, and provide perfect hiding.

Pseudorandom function (PRF) [23], [24]. Let $k \in \mathcal{K}_{\mathtt{PRF}}$ be a high min-entropy key in the PRF key space. A pseudorandom function PRF is called $(t, q, \epsilon(\kappa))$-secure if for any PPT algorithm $\mathcal{A}$ running in time t with at most q oracle queries the probability $Adv^{\mathtt{PRF}}_{\mathcal{A}}(\kappa) \leq \epsilon(\kappa)$ for distinguishing the outputs of $\mathtt{PRF}(k, m)$ from the outputs of a truly random function f of the same length, assuming that $\mathcal{A}$ has oracle access to $\mathcal{O}_{\mathtt{PRF}}(\cdot)$ which contains either $\mathtt{PRF}(k, \cdot)$ or $f(\cdot)$ and which cannot be queried on m.

Key derivation function (KDF) [25]. Let Σ be a source of key material. A key derivation function KDF is called $(t, q, \epsilon(\kappa))$-secure with respect to Σ if for any PPT algorithm $\mathcal{A}$ running in time t with at most q oracle queries the probability $Adv^{\mathtt{KDF}}_{\mathcal{A}}(\kappa) \leq \epsilon(\kappa)$ for distinguishing the output of $\mathtt{KDF}(k, c)$ from uniformly drawn random strings of the same length, assuming that $(k, \alpha) \leftarrow \Sigma$ where k is the secret key material and α is some side information. It is assumed that $\mathcal{A}$ knows α, has control over the context information c and has oracle access to $\mathtt{KDF}(k, \cdot)$ which cannot be queried on c.

Message authentication code [26]. A message authentication code $(\mathtt{KGen}, \mathtt{Tag}, \mathtt{Vrfy})$ is comprised of the algorithms

- $\mathtt{KGen}(\kappa)$: on input security parameter κ output key $\mathtt{mk} \leftarrow \{0,1\}^\kappa$.
- $\mathtt{Tag}(\mathtt{mk}, m)$: on input a key $\mathtt{mk}$ and a message m, output tag $\mu \leftarrow \mathtt{Tag}(\mathtt{mk}, m)$.
- $\mathtt{Vrfy}(\mathtt{mk}, m, \mu)$: on input a key $\mathtt{mk}$, a message m and a tag μ outputs 1 if μ is valid or 0 otherwise.

A MAC is secure if any PPT algorithm $\mathcal{A}$ without knowledge of $\mathtt{mk}$ has only negligible probability $Adv^{\mathtt{MAC}}_{\mathcal{A}}(\kappa)$ to forge a tag μ^* for some message m^*. $\mathcal{A}$ has access to the tag oracle $\mathcal{O}_{\mathtt{Tag}}(\cdot)$ which returns $\mu \leftarrow \mathtt{Tag}(\mathtt{mk}, m)$ on input m. The only restriction is that m^* is never queried to $\mathcal{O}_{\mathtt{Tag}}(\cdot)$.

III. Password Authenticated Keyword Search: Syntax and Definitions

In this section we provide definitions for the functionality of PAKS and its security properties.

A. Syntax of PAKS

After initialization, the functionality of PAKS is defined by three protocols. These protocols are themselves defined as interactive algorithms executed by the protocol participants.

Functionality of PAKS. Our PAKS functionality allows any user U to perform initial registration procedure with any two servers $\mathtt{S}_0$ and $\mathtt{S}_1$ in the system and then use the registered password π (from some dictionary $\mathcal{D}$) to outsource and retrieve data based on associated keywords $w \in \mathcal{W}$. Each server $\mathtt{S}_d$, $d \in \{0,1\}$ maintains its own database where for each user it records the associated secret information $\mathtt{info}_d$ obtained during the registration procedure and the outsourced data $(C, \mathtt{ix})$ obtained from multiple executions of the outsource protocol; C represents ciphertexts for the keywords whereas ix stands for the outsourced (and typically encrypted) document that is associated with the encrypted keywords. Similar to other searchable encryption schemes (e.g. [1]) we do not explicitly model the encryption of outsourced documents and use indices $\mathtt{ix} \in \boldsymbol{I}$ as placeholders instead.

- $\mathtt{Setup}(1^\kappa)$ is an initialisation algorithm that on input a security parameter $\kappa \in \mathbb{N}$ generates public parameters par of the scheme.
- Register is a registration protocol executed between some user U (running interactive algorithm RegisterU) and two servers $\mathtt{S}_0$ and $\mathtt{S}_1$ (running interactive algorithms

$\texttt{RegisterS}_d$, $d \in \{0,1\}$) according to the following specification:

- $\texttt{RegisterU}(\texttt{par}, \pi, \texttt{S}_0, \texttt{S}_1)$: on input par and some password $\pi \leftarrow \mathcal{D}$, this algorithm interacts with $\texttt{RegisterS}_d$, $d \in \{0,1\}$ and outputs a flag $\texttt{s} \in \{\texttt{succ}, \texttt{fail}\}$. If $(\texttt{s} = \texttt{succ})$, the user remembers π and forgets all other informations.
- $\texttt{RegisterS}_d(\texttt{par}, \texttt{U}, \texttt{S}_{1\text{-}d})$: on input par this algorithm interacts with RegisterU (and possibly $\texttt{RegisterS}_{1\text{-}d}$) and at the end of successful interaction stores some secret information $\texttt{info}_d$ associated with U at $\texttt{S}_d$.

- Outsource is an outsourcing protocol executed between some user U (running interactive algorithm OutsourceU) and two servers $\texttt{S}_0$ and $\texttt{S}_1$ (running interactive algorithms $\texttt{OutsourceS}_d$, $d \in \{0,1\}$) according to the following specification:
  - $\texttt{OutsourceU}(\texttt{par}, \pi, w, \texttt{ix}, \texttt{S}_0, \texttt{S}_1)$: on input π, a keyword w, and some index ix this algorithms interacts with $\texttt{OutsourceS}_d$, $d \in \{0,1\}$ and outputs a flag $\texttt{s} \in \{\texttt{succ}, \texttt{fail}\}$.
  - $\texttt{OutsourceS}_d(\texttt{par}, \texttt{U}, \texttt{info}_d)$: on input $\texttt{info}_d$ this algorithm upon successful interaction with OutsourceU (and possibly $\texttt{OutsourceS}_{1\text{-}d}$) stores a record $(C, \texttt{ix})$ in its database $\boldsymbol{C}_d$.
- Retrieve is a retrieval protocol executed between some user U (running interactive algorithm RetrieveU) and two servers $\texttt{S}_0$ and $\texttt{S}_1$ (running interactive algorithms $\texttt{RetrieveS}_d$, $d \in \{0,1\}$) according to the following specification:
  - $\texttt{RetrieveU}(\texttt{par}, \pi, w, \texttt{S}_0, \texttt{S}_1)$: on input π and a keyword w this algorithm upon successful interaction with $\texttt{RetrieveS}_d$, $d \in \{0,1\}$ outputs set $\boldsymbol{I}$ containing all ix associated with w.
  - $\texttt{RetrieveS}_d(\texttt{par}, \texttt{U}, \texttt{info}_d)$: on input $\texttt{info}_d$ this algorithm interacts with RetrieveU (and possibly $\texttt{RetrieveS}_{1\text{-}d}$) and outputs a flag $\texttt{s} \in \{\texttt{succ}, \texttt{fail}\}$.

Correctness. The PAKS scheme is *correct* if for all $\kappa \in \mathbb{N}, \texttt{ix} \in \mathcal{I}, w \in \mathcal{W}, \pi \in \mathcal{D}$, $\texttt{par} \leftarrow \texttt{Setup}(1^\kappa)$ the probability $\Pr[\texttt{ix} \in \boldsymbol{I}] = 1$ iff

$$\begin{aligned}
\langle \texttt{succ}, \texttt{info}_0, \texttt{info}_1 \rangle \leftarrow \langle &\texttt{RegisterU}(\texttt{par}, \pi, \texttt{S}_0, \texttt{S}_1), \\
&\texttt{RegisterS}_0(\texttt{par}, \texttt{U}, \texttt{S}_1), \texttt{RegisterS}_1(\texttt{par}, \texttt{U}, \texttt{S}_0) \rangle; \\
\langle \texttt{succ}, (C, \texttt{ix}), (C, \texttt{ix}) \rangle \leftarrow \langle &\texttt{OutsourceU}(\texttt{par}, \pi, w, \texttt{ix}, \texttt{S}_0, \texttt{S}_1), \\
&\texttt{OutsourceS}_0(\texttt{par}, \texttt{U}, \texttt{info}_0), \texttt{OutsourceS}_1(\texttt{par}, \texttt{U}, \texttt{info}_1) \rangle; \\
\langle \boldsymbol{I}, \texttt{succ}, \texttt{succ} \rangle \leftarrow \langle &\texttt{RetrieveU}(\texttt{par}, \pi, w, \texttt{S}_0, \texttt{S}_1), \\
&\texttt{RetrieveS}_0(\texttt{par}, \texttt{U}, \texttt{info}_0), \texttt{RetrieveS}_1(\texttt{par}, \texttt{U}, \texttt{info}_1) \rangle;
\end{aligned}$$

In other words, the user should always be able to retrieve *all* indices ix that were previously outsourced under some keyword w as long as this user is registered and has used its registered password π in those outsourcing and in the retrieval protocol sessions.

$Exp^{\texttt{IND-CKA-}b}_{\texttt{PAKS},\mathcal{A}}(\kappa)$
$\boldsymbol{\tau} \leftarrow \emptyset$; $i^* \leftarrow (-1)$; $j \leftarrow 0$;
$\texttt{Set} \leftarrow \emptyset$; $\texttt{par} \leftarrow \texttt{Setup}(1^\kappa)$;
$b' \leftarrow \mathcal{A}^{\texttt{Ch}_{\texttt{ind}}(b,\cdot,\cdot,\cdot,\cdot),\texttt{Reg}(\cdot),\texttt{Out}(\cdot,\cdot,\cdot),\texttt{Ret}(\cdot,\cdot),\texttt{RetS}(\cdot)}(\texttt{par})$;
return b'

$Exp^{\texttt{Auth}}_{\texttt{PAKS},\mathcal{A}}(\kappa)$
$\boldsymbol{\tau} \leftarrow \emptyset$; $j \leftarrow 0$; $\texttt{Set} \leftarrow \emptyset$; $\texttt{par} \leftarrow \texttt{Setup}(1^\kappa)$;
$(i^*, w^*, \texttt{ix}^*) \leftarrow \mathcal{A}^{\texttt{Reg}(\cdot),\texttt{Out}(\cdot,\cdot,\cdot),\texttt{OutS}(\cdot),\texttt{Ret}(\cdot,\cdot)}(\texttt{par})$;
$\langle \boldsymbol{I}, \texttt{succ}, \texttt{succ} \rangle \leftarrow \texttt{Ret}(i^*, w^*)$;
if $(((i^*, w^*, \texttt{ix}^*) \notin \texttt{Set}) \wedge (\texttt{ix}^* \in \boldsymbol{I}))$ return 1
else return 0

$Exp^{\texttt{Con}}_{\texttt{PAKS},\mathcal{A}}(\kappa)$
$\boldsymbol{\tau} \leftarrow \emptyset$; $i^* \leftarrow (-1)$; $j \leftarrow 0$;
$\texttt{Set} \leftarrow \emptyset$; $\texttt{par} \leftarrow \texttt{Setup}(1^\kappa)$;
$w_1 \leftarrow \mathcal{A}^{\texttt{Ch}_{\texttt{con}}(\cdot,\cdot,\cdot),\texttt{Reg}(\cdot),\texttt{Out}(\cdot,\cdot,\cdot),\texttt{Ret}(\cdot,\cdot),\texttt{RetS}(\cdot)}(\texttt{par})$;
$\langle \boldsymbol{I}, \texttt{succ}, \texttt{succ} \rangle \leftarrow \texttt{Ret}(i^*, w_1)$;
if $(((i^*, w_1, \texttt{ix}^*) \notin \texttt{Set}) \wedge (\texttt{ix}^* \in \boldsymbol{I}))$ return 1
else return 0

Fig. 1. PAKS security experiments. The oracles are defined in Section III-B.

B. Security definitions of PAKS

We define security of PAKS using three main requirements: indistinguishability against chosen keyword attacks (IND-CKA), authentication and consistency. We adopt a BPR-like modeling approach [21] for password-based cryptographic protocols and define security through experiments (cf. Figure 1) where a PPT adversary $\mathcal{A}$ has full control over the communication channels and can interact with parties (controlled by a simulator) through the set of oracles defined in the following.

Adversary model and oracles. For each user U we allow $\mathcal{A}$ to take full control over at most *one* of the two servers $\texttt{S}_0$ and $\texttt{S}_1$ that were chosen by U during the registration phase to capture the required distributed trust relationship. We mostly use $\texttt{S}_d$ to denote the uncorrupted server and $\texttt{S}_{1\text{-}d}$ to denote the server controlled by the adversary. The oracles allow $\mathcal{A}$ to invoke interactive algorithms for all protocols of PAKS which will be executed (honestly) by the simulator. $\mathcal{A}$ can interact with these algorithms and by this participate in the protocol. In particular, we allow $\mathcal{A}$ to participate in outsourcing and retrieval protocols on behalf of some corrupted server and also as some (illegitimate) user who tries to guess the registered password during the execution of the protocol.

Let $\boldsymbol{\tau}$ be an initially empty array that will be populated with tuples of the form $\boldsymbol{\tau}[j] \leftarrow (d, \pi, \texttt{info}_d)$ at the end of each successful j-th registration session such that π is the registered password and $\texttt{info}_d$ is the secret data stored at the server $\texttt{S}_d$ at the end of that session. We also use variables $i^* \in \mathbb{Z}$, $\texttt{ix}^* \in \mathcal{I}$ and a set Set that are maintained by the experiments. The adversary $\mathcal{A}$ can access the following oracles.

- Challenge oracle $\texttt{Ch}_{\texttt{ind}}(b, \cdot, \cdot, \cdot, \cdot)$: on input $(i, w_0, w_1, \texttt{ix}^*)$, the oracle aborts if $((i^* \geq 0) \vee (i \geq j) \vee ((i, w_0) \in$

$\mathtt{Set}) \vee ((i, w_1) \in \mathtt{Set}))$. Otherwise, it sets $i^* \leftarrow i$ and invokes oracle $\mathtt{Out}(i^*, w_b, \mathtt{ix}^*)$. Note that this oracle will be used to model IND-CKA security of PAKS.

- Challenge oracle $\mathtt{Ch}_{\mathtt{con}}(\cdot,\cdot,\cdot)$: on input $(i, w_0, \mathtt{ix}^*)$, the oracle aborts if $((i^* \geq 0) \vee (i \geq j))$. Otherwise, it sets $i^* \leftarrow i$ and invokes oracle $\mathtt{Out}(i^*, w_0, \mathtt{ix}^*)$. Note that this oracle will be used to model consistency of PAKS.
- Registration oracle $\mathtt{Reg}(\cdot)$: on input $d \in \{0,1\}$, the experiment first initializes $\boldsymbol{C}_{d,j} \leftarrow \emptyset$ as a database for session j. Then, it randomly picks fresh $(\pi \xleftarrow{\$} \mathcal{D}) \wedge ((i, \pi, \cdot) \notin \boldsymbol{\tau})$ for all $i \in [0, j-1]$. The Register protocol is executed with $\mathcal{A}$ where the oracle plays the roles of honest U and $\mathtt{S}_d$ executing algorithms $\mathtt{RegisterU}(\mathtt{par}, \pi, \mathtt{S}_0, \mathtt{S}_1)$ and $\mathtt{RegisterS}_d(\mathtt{par}, \mathtt{U}, \mathtt{S}_{1\text{-}d})$, respectively, and $\mathcal{A}$ plays the role of corrupted $\mathtt{S}_{1\text{-}d}$. After interactions, the experiment records $\boldsymbol{\tau}[j] \leftarrow (d, \pi, \mathtt{info}_d)$, delivers j to the adversary and increases $j \leftarrow j+1$.
- Outsource oracle $\mathtt{Out}(\cdot,\cdot,\cdot)$: on input $(i, w, \mathtt{ix})$, the oracle aborts if $i \geq j$; or otherwise, it obtains $(d, \pi, \mathtt{info}_d) \leftarrow \boldsymbol{\tau}[i]$. The Outsource protocol is then executed with $\mathcal{A}$ where the oracle plays the roles of honest U and $\mathtt{S}_d$ executing algorithms $\mathtt{OutsourceU}(\mathtt{par}, \pi, w, \mathtt{ix}, \mathtt{S}_0, \mathtt{S}_1)$ and $\mathtt{OutsourceS}_d(\mathtt{par}, \mathtt{U}, \mathtt{info}_d)$, respectively, and $\mathcal{A}$ plays the role of malicious $\mathtt{S}_{1\text{-}d}$. In both Auth and Con experiments, the oracle additionally computes $\mathtt{Set} \leftarrow \mathtt{Set} \cup (i, w, \mathtt{ix})$.
- Outsource oracle (server only) $\mathtt{OutS}(\cdot)$: on input i, the oracle aborts if $i \geq j$; otherwise, it obtains $(d, \pi, \mathtt{info}_d) \leftarrow \boldsymbol{\tau}[i]$. The Outsource protocol is then executed with $\mathcal{A}$ where the oracle plays the role of honest $\mathtt{S}_d$ executing algorithm $\mathtt{OutsourceS}_d(\mathtt{par}, \mathtt{U}, \mathtt{info}_d)$ and $\mathcal{A}$ plays the roles of (illegitimate) U and corrupted $\mathtt{S}_{1\text{-}d}$. Note that this oracle will be used to model authentication of PAKS.
- Retrieve oracle $\mathtt{Ret}(\cdot,\cdot)$: on input (i, w), the oracle aborts if $i \geq j$. In the IND-CKA experiment, the oracle also aborts if $((i = i^*) \wedge (w \in \{w_0, w_1\}))$. Otherwise, it obtains the parameters $(d, \pi, \mathtt{info}_d) \leftarrow \boldsymbol{\tau}[i]$. The Retrieve protocol is then executed with $\mathcal{A}$ where the oracle plays the roles of honest U and $\mathtt{S}_d$ executing algorithms $\mathtt{RetrieveU}(\mathtt{par}, \pi, w, \mathtt{S}_0, \mathtt{S}_1)$ and $\mathtt{RetrieveS}_d(\mathtt{par}, \mathtt{U}, \mathtt{info}_d)$, respectively, and $\mathcal{A}$ plays the role of corrupted $\mathtt{S}_{1\text{-}d}$. In the IND-CKA experiment, if $(i^* = -1)$ the oracle additionally computes $\mathtt{Set} \leftarrow \mathtt{Set} \cup (i, w)$.
- Retrieve oracle (server only) $\mathtt{RetS}(\cdot)$: on input i, the oracle aborts if $i \geq j$; otherwise, it obtains $(d, \pi, \mathtt{info}_d) \leftarrow \boldsymbol{\tau}[i]$. The Retrieve protocol is then executed with $\mathcal{A}$ where the oracle plays the role of honest $\mathtt{S}_d$ executing algorithm $\mathtt{RetrieveS}_d(\mathtt{par}, \mathtt{U}, \mathtt{info}_d)$ and $\mathcal{A}$ plays the roles of (illegitimate) U and corrupted $\mathtt{S}_{1\text{-}d}$. Note that this oracle will be used to models IND-CKA and Con-security of PAKS.

Indistinguishability against Chosen Keyword Attacks (IND-CKA). The IND-CKA property for PAKS is defined through the experiment $Exp^{\mathtt{IND\text{-}CKA}\text{-}b}_{\mathtt{PAKS},\mathcal{A}}(\kappa)$ (cf. Figure 1) and is closely related to [5] except that our setting is based on passwords. $\mathcal{A}$ is given the public parameters par and permitted to adaptively access oracles $\mathtt{Ch}_{\mathtt{ind}}(b,\cdot,\cdot,\cdot,\cdot)$, $\mathtt{Reg}(\cdot)$, $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{Ret}(\cdot,\cdot)$ and $\mathtt{RetS}(\cdot)$ at most 1, q_r, q_o, q_t and q_s times, respectively. In particular, our IND-CKA experiment captures the following ways that $\mathcal{A}$ may try to retrieve data: (i) from interaction with an honest user U and the honest server $\mathtt{S}_d$ playing the role of corrupted $\mathtt{S}_{1\text{-}d}$ (which is captured through the oracle $\mathtt{Ret}(\cdot,\cdot)$), or (ii) from interaction with the honest server $\mathtt{S}_d$ playing the role of illegitimate user, e.g. trying to guess the registered password, and the corrupted server $\mathtt{S}_{1\text{-}d}$ (which is captured through the oracle $\mathtt{RetS}(\cdot)$).

Let $Adv^{\mathtt{IND\text{-}CKA}}_{\mathtt{PAKS},\mathcal{A}}(\kappa) \stackrel{\text{def}}{=} \Pr[b' = b : b' \leftarrow Exp^{\mathtt{IND\text{-}CKA}\text{-}b}_{\mathtt{PAKS},\mathcal{A}}(\kappa)] - \frac{1}{2}$ denote the advantage of $\mathcal{A}$ in the IND-CKA security experiment. A PAKS scheme is called IND-CKA-*secure* if the probability $Adv^{\mathtt{IND\text{-}CKA}}_{\mathtt{PAKS},\mathcal{A}}(\kappa) \leq \frac{q_s}{|\mathcal{D}|} + \epsilon(\kappa)$ where $|\mathcal{D}|$ is the dictionary size and $\epsilon(\kappa)$ is negligible in the security parameter κ. Note that probability $\frac{q_s}{|\mathcal{D}|}$ relates to the use of oracle $\mathtt{RetS}(\cdot)$ that models on-line dictionary attacks and assumes uniform distribution of passwords within $\mathcal{D}$, as is also common in BPR-like models.

Authentication (Auth). The property of authentication for PAKS is defined using experiment $Exp^{\mathtt{Auth}}_{\mathtt{PAKS},\mathcal{A}}(\kappa)$ in Figure 1. $\mathcal{A}$ is given the public parameters par and permitted to access oracles $\mathtt{Reg}(\cdot)$, $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{OutS}(\cdot)$ and $\mathtt{Ret}(\cdot,\cdot)$ with at most q_r, q_o, q_s and q_t times, respectively. Our experiment effectively captures attacks where $\mathcal{A}$ tries to outsource some data $\mathtt{ix}^*$ on behalf of some user U without knowing the registered password (via $\mathtt{OutS}(\cdot)$ oracle), possibly after having interacted with U and the honest server $\mathtt{S}_d$. In its attack on authentication $\mathcal{A}$ can play the role of a corrupted server $\mathtt{S}_{1\text{-}d}$ and also mount man-in-the-middle attacks on sessions of Outsource and Retrieve protocols involving user U.

A PAKS scheme provides *authentication* if for all PPT $\mathcal{A}$ the probability $Adv^{\mathtt{Auth}}_{\mathtt{PAKS},\mathcal{A}}(\kappa) = \Pr[1 \leftarrow Exp^{\mathtt{Auth}}_{\mathtt{PAKS},\mathcal{A}}(\kappa)] \leq \frac{q_s}{|\mathcal{D}|} + \epsilon(\kappa)$. As in the IND-CKA case, we again need to account for the possibility of online guessing attacks via the oracle $\mathtt{OutS}(\cdot)$.

Consistency (Con). We define consistency of PAKS using experiment $Exp^{\mathtt{Con}}_{\mathtt{PAKS},\mathcal{A}}(\kappa)$ in Figure 1, adapting this notion from [1] to the PAKS setting. The property aims to rule out false positives when performing the keyword search. $\mathcal{A}$ is given the public parameters par and permitted to access oracles $\mathtt{Ch}_{\mathtt{con}}(\cdot,\cdot,\cdot)$, $\mathtt{Reg}(\cdot)$, $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{Ret}(\cdot,\cdot)$ and $\mathtt{RetS}(\cdot)$ with at most 1, q_r, q_o, q_t and q_s times, respectively. In particular, $\mathcal{A}$ should not be able to come up with different keywords w_0 and w_1 from which w_0 will be used by an honest user to outsource some data $\mathtt{ix}^*$ (via $\mathtt{Ch}_{\mathtt{con}}(\cdot,\cdot,\cdot)$ oracle) and w_1 used to retrieve this data later. A PAKS scheme is called *consistent* if the probability $Adv^{\mathtt{Con}}_{\mathtt{PAKS},\mathcal{A}}(\kappa) = \Pr[1 \leftarrow Exp^{\mathtt{Con}}_{\mathtt{PAKS},\mathcal{A}}(\kappa)] \leq \frac{q_s}{|\mathcal{D}|} + \epsilon(\kappa)$.

IV. Our direct PAKS construction

In this section we propose a direct and efficient construction of PAKS. It follows our general idea of combining suitable

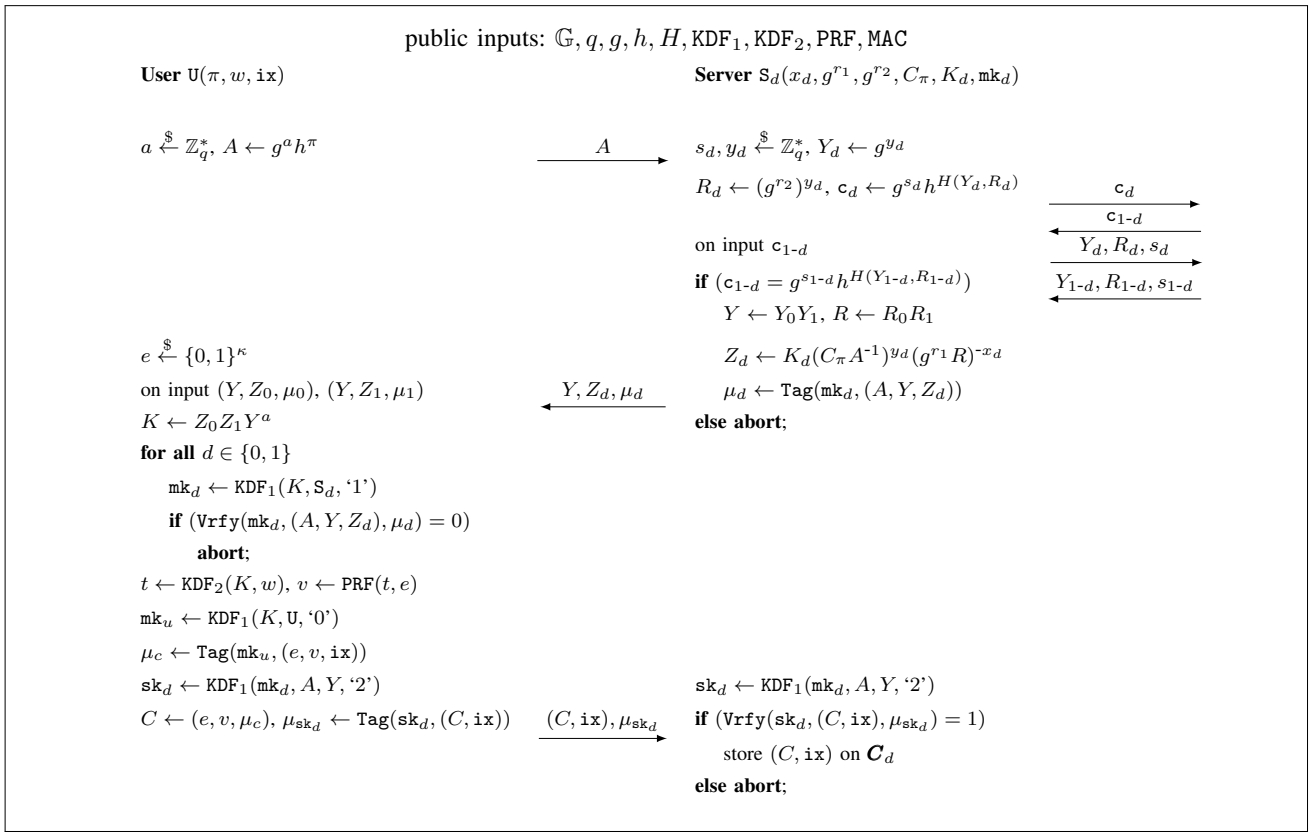

Fig. 2. The `Outsource` protocol between `U` and $\mathtt{S}_d$. The server-side algorithm includes communication between servers $\mathtt{S}_d$ and $\mathtt{S}_{1\text{-}d}$.

password-authenticated secret sharing with symmetric searchable encryption techniques. In the introduction we explained the difficulties behind an attempt to construct PAKS generically using PASS and SSE schemes and motivated our choice for a direct construction.

High-level idea. Our PAKS protocol is inspired by the techniques used in the recent password-authenticated secret sharing protocol from [15] which we modified to address the functionality and requirements of PAKS and extended with a suitable mechanism for symmetric searchable encryption of keywords. In particular, we define a new registration protocol `Register` upon which the user registers its password π encrypted in C_π with both servers and also picks a symmetric key K for which it computes appropriate shares K_0 and K_1 which are then sent to the corresponding servers. The reconstruction of K is protected by π and MAC codes μ_d are used to ensure the validity of K upon its reconstruction. The protocols `Outsource` and `Retrieve` proceed according to the similar pattern. First, the user reconstructs K using its password π after communication with both servers. Then, in `Outsource` protocol `U` uses K in combination with its keyword w to derive a trapdoor $t \leftarrow \mathtt{KDF}_2(K, w)$ and a fresh randomness e to derive verifier $v \leftarrow \mathtt{PRF}(t, e)$. The pair (e, v) becomes part of the outsourced ciphertext C which is bound to some data `ix`. During the `Retrieve` protocol the user can recompute the trapdoor t for a given keyword w and then send it to the servers who can the find all outsourced ciphertexts C for which $v \leftarrow \mathtt{PRF}(t, e)$ holds and hence identify which data `ix` needs to be returned. In order to prevent servers from creating their own pairs (e, v) for a given t the outsourced ciphertext C additionally includes a MAC tag μ_c which authenticates (e, v) and also `ix` and which can only be computed and verified using K. During the `Retrieve` protocol the user will ensure that it final search result contains only data that passes this integrity and authenticity check. In addition both protocols make use of MACs to ensure authenticity of messages, where the MAC keys are derived from K on the user side. We emphasis that our PAKS construction is in the password-only setting where servers are not required to possess any public keys for the security of the PAKS scheme. However, if the registration protocol `Register` is performed remotely over a public network then this protocol needs to be executed over server-authenticated secure-channels (e.g. TLS). In order to enable reconstruction of K by the user and to protect this phase with the password both servers communicate with each other as part of the `Outsource` and `Retrieve` protocols. While in practice this communication between the two servers will likely be protected using a secure channel (e.g. TLS) we

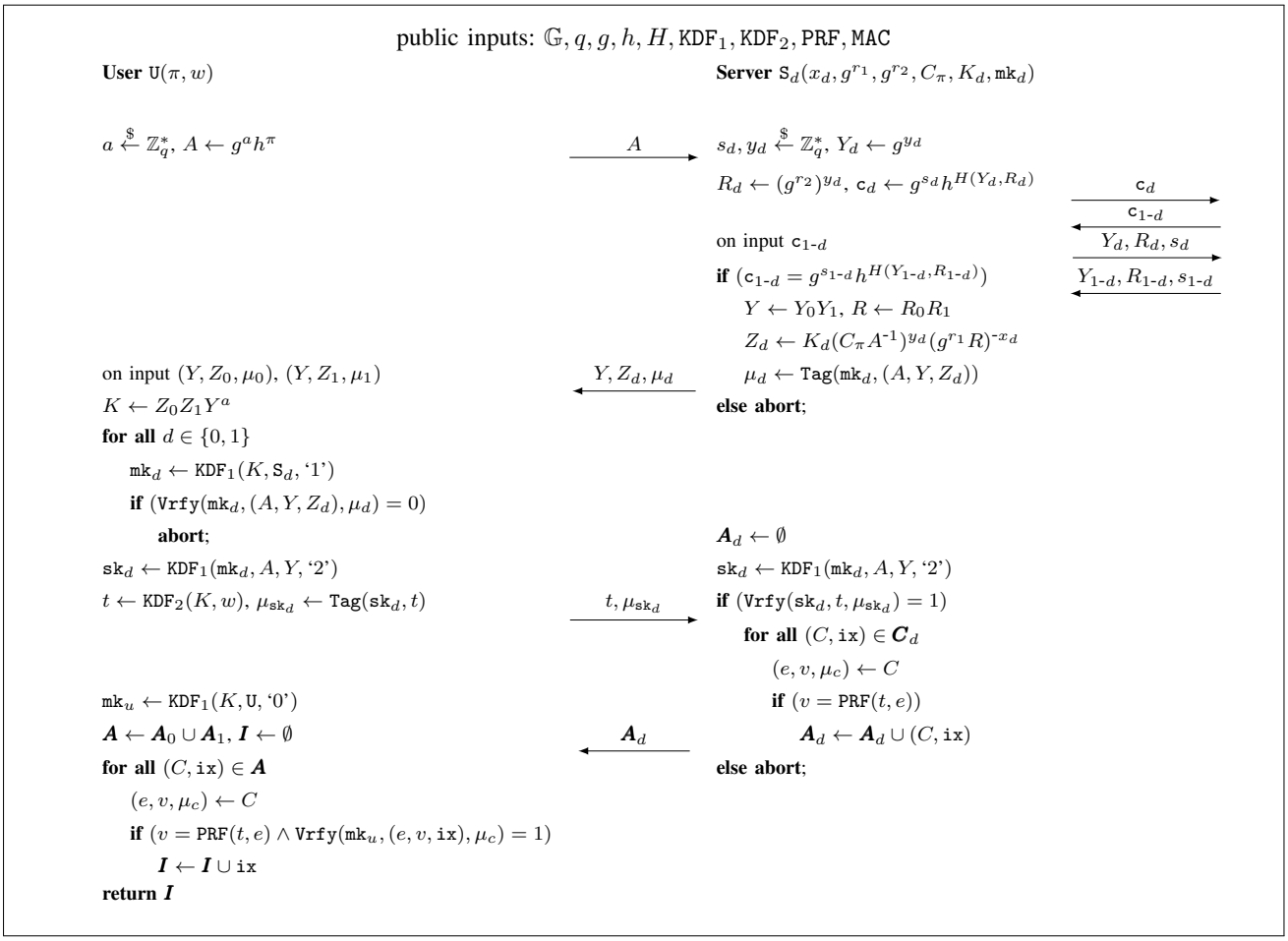

Fig. 3. The `Retrieve` protocol between $\mathtt{U}$ and $\mathtt{S}_d$. The server-side algorithm includes communication between servers $\mathtt{S}_d$ and $\mathtt{S}_{1\text{-}d}$.

stress that in our protocols this communication can take place over an insecure channel.

Detailed description. In the following we provide a detailed description of all algorithms and protocols underlying our direct PAKS scheme, along with Figures 2 and 3 that illustrate the protocols `Outsource` and `Retrieve`, respectively.

- $\mathtt{Setup}(1^\kappa)$: Generated public parameters $\mathtt{par}$ contain $\{\mathbb{G}, q, g, h, \mathtt{KDF}_1, \mathtt{KDF}_2, \mathtt{PRF}, \mathtt{MAC}\}$, where $(\mathbb{G}, q, g, h)$ represents a multiplicative cyclic group $\mathbb{G}$ with a prime order q and generators $g, h \xleftarrow{\$} \mathbb{G}$ where the discrete logarithm of h with respect to base g is unknown. $H : \mathbb{G} \times \mathbb{G} \to \mathbb{Z}_q^*$ is a collision-resistant hash function. $\mathtt{KDF}_1 : \{0,1\}^* \to \mathcal{K}_{\mathtt{MAC}}$ and $\mathtt{KDF}_2 : \mathbb{G} \times \mathcal{W} \to \mathcal{K}_{\mathtt{PRF}}$ are two key derivation functions. $\mathtt{PRF} : \mathcal{K}_{\mathtt{PRF}} \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ is a pseudo-random function. $\mathtt{MAC} = (\mathtt{KGen}, \mathtt{Tag}, \mathtt{Vrfy})$ is a message authentication code with $\mathtt{Tag} : \mathcal{K}_{\mathtt{MAC}} \times \{0,1\}^* \to \{0,1\}^\kappa$ and $\mathtt{Vrfy} : \mathcal{K}_{\mathtt{MAC}} \times \{0,1\}^* \times \{0,1\}^\kappa \to \{0,1\}$ where $\mathcal{K}_{\mathtt{PRF}}$ and $\mathcal{K}_{\mathtt{MAC}}$ are $\mathtt{PRF}$ and $\mathtt{MAC}$ key spaces, respectively. We assume that passwords from $\mathcal{D}$ are represented as elements of $\mathbb{Z}_q^*$.
- Protocol `Register`: User $\mathtt{U}$ picks $r_1, r_2, x_0, x_1 \xleftarrow{\$} \mathbb{Z}_q^*$ and $K, K_0 \xleftarrow{\$} \mathbb{G}$; computes $X \leftarrow g^{x_0 + x_1}$, $K_1 \leftarrow X^{r_1} K (K_0)^{-1}$ and $C_\pi \leftarrow X^{r_2} h^\pi$. Then, for $d \in \{0,1\}$, the user computes $\mathtt{mk}_d \leftarrow \mathtt{KDF}_1(K, \mathtt{S}_d, \text{'1'})$, sets $\mathtt{info}_d \leftarrow (x_d, g^{r_1}, g^{r_2}, C_\pi, K_d, \mathtt{mk}_d)$ and sends $\mathtt{info}_d$ to server $\mathtt{S}_d$, $d \in \{0,1\}$ over server-authenticated secure channels. Finally, $\mathtt{U}$ memorizes π.
- Protocol `Outsource`: For $d \in \{0,1\}$, the `Outsource` protocol between $\mathtt{U}$ and $\mathtt{S}_d$ is illustrated in Figure 2 and detailed in the following. Note that as part of the `Outsource` protocol both $\mathtt{S}_0$ and $\mathtt{S}_1$ communicate with each other, possibly over an insecure channel.
  1) User $\mathtt{U}$ randomly selects $a \xleftarrow{\$} \mathbb{Z}_q^*$, $e \xleftarrow{\$} \{0,1\}^\kappa$ and sends $A \leftarrow g^a h^\pi$ to both servers.
  2) On input A, server $\mathtt{S}_d$ executes following steps:
     a) Pick $s_d, y_d \xleftarrow{\$} \mathbb{Z}_q^*$, compute $Y_d \leftarrow g^{y_d}, R_d \leftarrow (g^{r_2})^{y_d}$.
     b) Send Pedersen commitment $\mathtt{c}_d \leftarrow g^{s_d} h^{H(Y_d, R_d)}$ to server $\mathtt{S}_{1\text{-}d}$ and wait for its response $\mathtt{c}_{1\text{-}d}$.

TABLE I
EFFICIENCY COMPARISON WITH PASSWORD-ONLY PASS SCHEMES.

	Computation cost (unit: exp)				Communication cost (unit: bits)			Rounds	
	Sharing		Retrieval		Sharing	Retrieval		Sharing	Retrieval
	user	server	user	server	user-server	user-server	server-server		
BJSL11 [11]	6	0	33	16	$24q + 4\kappa$	$22q + 2\kappa$	0	1	2
JKK14 [14]	4	1	11	4	$8q + 4\kappa$	$8q + 4\kappa$	0	1	1
YCHL15 [15]	1	0	7	12	6κ	$10q$	$8q$	1	1
JKKX16 [17]	3	1	3	1	$8q + 4\kappa$	$8q + 4\kappa$	0	2	1
In our PAKS scheme	6	0	3	8	$8q + 2\kappa$	$6q$	$6q + 2\kappa$	1	1

c) Send the opening (Y_d, R_d, s_d) to server $\mathtt{S}_{1\text{-}d}$ and wait for its response $(Y_{1\text{-}d}, R_{1\text{-}d}, s_{1\text{-}d})$. If $\mathtt{c}_{1\text{-}d} \neq g^{s_{1\text{-}d}} h^{H(Y_{1\text{-}d}, R_{1\text{-}d})}$ then abort.
d) Send (Y, Z_d, μ_d) to $\mathtt{U}$ where $Y \leftarrow Y_0 Y_1$, $R \leftarrow R_0 R_1$, $\mu_d \leftarrow \mathtt{Tag}(\mathtt{mk}_d, (A, Y, Z_d))$ and $Z_d \leftarrow K_d (C_\pi A^{-1})^{y_d} (g^{r_1} R)^{-x_d}$.

3) Upon receiving (Y, Z_0, μ_0) and (Y, Z_1, μ_1) from both servers, user $\mathtt{U}$ executes following steps:
a) If $\mathtt{Vrfy}(\mathtt{mk}_d, (A, Y, Z_d), \mu_d) = 0$ for any $d \in \{0, 1\}$ then abort, else compute $K \leftarrow Z_0 Z_1 Y^a$.
b) Compute $t \leftarrow \mathtt{KDF}_2(K, w)$, $v \leftarrow \mathtt{PRF}(t, e)$, $\mathtt{mk}_u \leftarrow \mathtt{KDF}_1(K, \mathtt{U}, \text{'0'})$, $\mu_c \leftarrow \mathtt{Tag}(\mathtt{mk}_u, (e, v, \mathtt{ix}))$ and $C \leftarrow (e, v, \mu_c)$.
c) Send $((C, \mathtt{ix}), \mu_{\mathtt{sk}_d})$ to server $\mathtt{S}_d$, $d \in \{0, 1\}$ where $\mu_{\mathtt{sk}_d} \leftarrow \mathtt{Tag}(\mathtt{sk}_d, (C, \mathtt{ix}))$ using $\mathtt{sk}_d \leftarrow \mathtt{KDF}_1(\mathtt{mk}_d, A, Y, \text{'2'})$.

4) On input $((C, \mathtt{ix}), \mu_{\mathtt{sk}_d})$, server $\mathtt{S}_d$ stores $(C, \mathtt{ix})$ in its database $\boldsymbol{C}_d$ if $\mathtt{Vrfy}(\mathtt{sk}_d, (C, \mathtt{ix}), \mu_{\mathtt{sk}_d}) = 1$ for $\mathtt{sk}_d \leftarrow \mathtt{KDF}_1(\mathtt{mk}_d, A, Y, \text{'2'})$, else $\mathtt{S}_d$ aborts.

- Protocol `Retrieve`: For $d \in \{0, 1\}$, the `Retrieve` protocol between $\mathtt{U}$ and $\mathtt{S}_d$ is illustrated in Figure 3 and detailed in the following. Note that as part of the `Outsource` protocol both $\mathtt{S}_0$ and $\mathtt{S}_1$ communicate with each other, possibly over an insecure channel.

1) User $\mathtt{U}$ randomly selects $a \xleftarrow{\$} \mathbb{Z}_q^*$ and sends $A \leftarrow g^a h^\pi$ to both servers.
2) On input A, server $\mathtt{S}_d$ executes following steps:
a) Pick $s_d, y_d \xleftarrow{\$} \mathbb{Z}_q^*$, compute $Y_d \leftarrow g^{y_d}$, $R_d \leftarrow (g^{r_2})^{y_d}$.
b) Send Pedersen commitment $\mathtt{c}_d \leftarrow g^{s_d} h^{H(Y_d, R_d)}$ to server $\mathtt{S}_{1\text{-}d}$ and wait for its response $\mathtt{c}_{1\text{-}d}$.
c) Send opening (Y_d, R_d, s_d) to server $\mathtt{S}_{1\text{-}d}$ and waits for its response $(Y_{1\text{-}d}, R_{1\text{-}d}, s_{1\text{-}d})$. If $\mathtt{c}_{1\text{-}d} \neq g^{s_{1\text{-}d}} h^{H(Y_{1\text{-}d}, R_{1\text{-}d})}$ then abort.
d) Send (Y, Z_d, μ_d) to $\mathtt{U}$ where $Y \leftarrow Y_0 Y_1$, $R \leftarrow R_0 R_1$, $\mu_d \leftarrow \mathtt{Tag}(\mathtt{mk}_d, (A, Y, Z_d))$ and $Z_d \leftarrow K_d (C_\pi A^{-1})^{y_d} (g^{r_1} R)^{-x_d}$.

3) Upon receiving (Y, Z_0, μ_0) and (Y, Z_1, μ_1) from both servers, $\mathtt{U}$ executes following steps:
a) If $\mathtt{Vrfy}(\mathtt{mk}_d, (A, Y, Z_d), \mu_d) = 0$ for any $d \in \{0, 1\}$ then abort, else compute $K \leftarrow Z_0 Z_1 Y^a$.
b) Compute $t \leftarrow \mathtt{KDF}_2(K, w)$ and $\mu_{\mathtt{sk}_d} \leftarrow \mathtt{Tag}(\mathtt{sk}_d, t)$ using $\mathtt{sk}_d \leftarrow \mathtt{KDF}_1(\mathtt{mk}_d, A, Y, \text{'2'})$. Send $(t, \mu_{\mathtt{sk}_d})$ to $\mathtt{S}_d$, $d \in \{0, 1\}$.

4) On input $(t, \mu_{\mathtt{sk}_d})$, server $\mathtt{S}_d$ executes following steps:
a) If $\mathtt{Vrfy}(\mathtt{sk}_d, t, \mu_{\mathtt{sk}_d}) = 0$ then abort, else compute $\mathtt{sk}_d \leftarrow \mathtt{KDF}_1(\mathtt{mk}_d, A, Y, \text{'2'})$.
b) Initialize set $\boldsymbol{A}_d \leftarrow \emptyset$. For all $(C, \mathtt{ix}) \in \boldsymbol{C}_d$, parse $(e, v, \mu_c) \leftarrow C$ and add $(C, \mathtt{ix})$ to $\boldsymbol{A}_d$ if $v = \mathtt{PRF}(t, e)$. Finally, send $\boldsymbol{A}_d$ to $\mathtt{U}$.

5) Upon receiving $\boldsymbol{A}_0$ and $\boldsymbol{A}_1$, user $\mathtt{U}$ initializes an empty set $\boldsymbol{I} \leftarrow \emptyset$. Then, for all $(C, \mathtt{ix}) \in (\boldsymbol{A}_0 \cup \boldsymbol{A}_1)$, parses $(e, v, \mu_c) \leftarrow C$ and adds $\mathtt{ix}$ to $\boldsymbol{I}$ if $v = \mathtt{PRF}(t, e)$ and $\mathtt{Vrfy}(\mathtt{mk}_u, (e, v, \mathtt{ix}), \mu_c) = 1$. This step guarantees that only outsourced data for which the integrity check was performed successfully will be added to the output set $\boldsymbol{I}$.

The correctness of the protocol can be easily verified once we illustrate here that if the correct password π is used by the user in the `Outsource` and `Retrieve` protocols then the reconstruction through $Z_0 Z_1 Y^a$ results in a correct key K:

$$\begin{aligned} Z_0 Z_1 Y^a =& K_0 (C_\pi A^{-1})^{y_0} (g^{r_1} R)^{-x_0} \cdot \\ & K_1 (C_\pi A^{-1})^{y_1} (g^{r_1} R)^{-x_1} \cdot g^{a(y_0+y_1)} \\ =& X^{r_1} K (X^{r_2} g^{-a})^{y_0+y_1} (g^{r_1} R)^{-(x_0+x_1)} g^{a(y_0+y_1)} \\ =& X^{r_1} K X^{r_2(y_0+y_1)} X^{-(r_1)} X^{-r_2(y_0+y_1)} = K \end{aligned}$$

A. Efficiency comparison

Given that our direct PAKS construction follows the general idea of building PAKS protocols based on the techniques used for password-authenticated secret sharing, we compare performance with existing PASS protocols. Since our PAKS scheme assumes password-only setting (except for the registration) we restrict our comparison to password-only PASS schemes [11], [14], [15], [17] and compare only the costs that arise from the sharing and retrieval of the symmetric key K — note that in our PAKS scheme sharing of K is performed as part of the `Register` protocol whereas retrieval of K is part of both `Outsource` and `Retrieve` protocols and is accomplished in step 3 of these protocols. Since our PAKS scheme adopts a two-server architecture but the aforementioned PASS schemes were designed for a more general t-out-of-n threshold setting we consider their costs for the special case of $t = n = 2$ to ease the comparison. The results of the comparison are presented in Table I. We compare computation costs through the number of modular exponentiations for the user and each

of the servers during the sharing and retrieval phases of the symmetric key K. We also compare communication costs in the number of bits communicated in both phases, while considering user-server and server-server communications. For the lengths of elements in $\mathbb{G}$ and $\mathbb{Z}_q^*$ we use $|\mathbb{G}| = q$ and $|q| = \kappa$ bits, respectively. We also compare the number of rounds needed for the sharing and retrieval of K.

We observe that in terms of computation and communication costs key sharing and reconstruction phases in our PAKS scheme compare fairly well with those of existing PASS protocols. In particular, only [17] which is the most computationally efficient PASS protocol today offers better overall computation and communication performance. We stress however that for PAKS protocols the efficiency of the retrieval phase is of greater importance than of the sharing phase. This is because in PAKS sharing of K is performed only once as part of the registration procedure, but retrieval of K occurs each time the user wants to outsource data or search for keywords. Furthermore, due to the simplified key management (i.e. reliance on passwords only) PAKS offers device-agnostic use of the functionality to the user and can possibly be executed on different client devices (ranging from desktops over to smartphones). In this case it becomes important to keep the costs associated with computations on the user side and the user-server communication low. Considering this we observe that in comparison to [17] our PAKS scheme achieves similar and even partly better performance for computations and communication involving the user device.

As a result of our comparison we conclude that our PAKS scheme is sufficiently practical since the additional costs arising from the encrypted keyword search functionality within our PAKS protocols are negligible (due to the nature of computations involved) in comparison to the costly key sharing and retrieval steps.

B. Extensions with multiple keywords

In the given specification of our PAKS construction users can use only one keyword w in each execution of `Outsource` and `Retrieve` protocols at a time. Often, users may want to be able to outsource or search for documents associated with multiple keywords. Our PAKS scheme can be extended to provide efficient support for multiple keywords. Let $\boldsymbol{w} = (w_1, \ldots, w_n)$ be a set of outsourced keywords for some document $\mathtt{ix}$ and let $\boldsymbol{w'} = (w'_1, \ldots, w'_m)$ be a set of searched keywords. In the following we show how to support (i) outsourcing of $\mathtt{ix}$ with $\boldsymbol{w}$ through a single session of the `Outsource` protocol and (ii) search for all suitable documents $\mathtt{ix}$ using $\boldsymbol{w'}$ through a single session of the `Retrieve` protocol, based on three different types of search queries [8]: conjunctive queries ($\boldsymbol{w} = \boldsymbol{w'}$), disjunctive queries ($|\boldsymbol{w} \cap \boldsymbol{w'}| > 0$), and those for a subset of keywords ($\boldsymbol{w'} \subseteq \boldsymbol{w}$).

Outsourcing documents with multiple keywords. In order to outsource some document $\mathtt{ix}$ associated with multiple keywords $\boldsymbol{w} = (w_1, \ldots, w_n)$, user $\mathtt{U}$ can compute $\boldsymbol{v} = (v_1, \ldots, v_n)$, $t_i \leftarrow \mathtt{KDF}_2(K, w_i)$ and $v_i \leftarrow \mathtt{PRF}(t_i, e)$ for $i = 1, \cdots, n$, and $\mu_c \leftarrow \mathtt{Tag}(\mathtt{mk}_u, (e, \boldsymbol{v}))$ as part of the same `Outsource` execution and outsource $C \leftarrow (e, \boldsymbol{v}, \mu_c)$ as the resulting ciphertext to both servers.

Search queries with multiple keywords. In order to search for documents using multiple keywords, i.e. $w'_1, \ldots, w'_m$, $m \leq n$, within a single execution of the `Retrieve` protocol, user $\mathtt{U}$ can send a set of authenticated trapdoors $t_i = \mathtt{KDF}_2(K, w_i)$ for all searched keywords w'_i, $i = 1, \cdots, m$ to both servers. Then, for all $(C, \mathtt{ix}) = (e, \boldsymbol{v}, \mu_c, \mathtt{ix})$ stored in the database $\boldsymbol{C}_d$, server $\mathtt{S}_d$ can initialize an empty output set $\boldsymbol{A}_d$, compute $\boldsymbol{v'} = (v'_1, \cdots, v'_m)$ where $v'_i = \mathtt{PRF}(t_i, e)$, $i = 1, \ldots, m$, and update the output set $\boldsymbol{A}_d \leftarrow \boldsymbol{A}_d \cup (C, \mathtt{ix})$ according to the following conditions, depending on the type of search search query, i.e.

- for conjunctive queries $w'_1 \wedge \ldots \wedge w'_m$: if $\boldsymbol{v} = \boldsymbol{v'}$
- for disjunctive queries $w'_1 \vee \ldots \vee w'_m$: if $|\boldsymbol{v} \cap \boldsymbol{v'}| > 0$
- for subset queries $(w'_1, \ldots, w'_m) \subseteq \boldsymbol{w}$: if $\boldsymbol{v'} \subseteq \boldsymbol{v}$.

C. Password change

Our PAKS scheme allows users to change their passwords without changing the encryption keys K, avoiding re-encryption of outsourced keywords. A new password π^* can be registered with the knowledge of the current π as follows:

1) User $\mathtt{U}$ sends $A \leftarrow g^a h^\pi$ to both servers (as in `Outsource` and `Retrieve`). Each server $\mathtt{S}_d$, $d \in \{0, 1\}$ uses its $\mathtt{info}_d$ to respond with $(Y, Z_d, g^{r_2}, C_\pi, \mu_d)$ where $\mu_d \leftarrow \mathtt{Tag}(\mathtt{mk}_d, (Y, Z_d, g^{r_2}, C_\pi))$.
2) Upon reconstructing $\mathtt{mk}_d$ as in `Outsource` and `Retrieve` protocols and verifying μ_d, the user picks random $r^* \xleftarrow{\$} \mathbb{Z}_q^*$, computes $C_{\pi^*} \leftarrow (C_\pi h^{-\pi})^{r^*} h^{\pi^*}$ and $\mu_d^* \leftarrow \mathtt{Tag}(\mathtt{mk}_d, (g^{r_2})^{r^*}, C_{\pi^*})$, and sends $(g^{r_2 r^*}, C_{\pi^*}, \mu_d^*)$ to both servers.
3) If $\mathtt{Vrfy}(\mathtt{mk}_d, (g^{r_2 r^*}, C_{\pi^*}), \mu_d^*) = 1$ then each $\mathtt{S}_d$ replaces (g^{r_2}, C_π) in its $\mathtt{info}_d$ with $(g^{r_2 r^*}, C_{\pi^*})$.

Note that current π is used to authenticate the user towards both servers. If the user no longer remembers π then changing the password while keeping the encryption key K would require additional authentication mechanisms based on which $\mathtt{U}$ would be able to retrieve $\mathtt{info}_d$ from $\mathtt{S}_d$ to reconstruct and re-share K with the new π^*, as in the registration phase.

V. Security analysis of PAKS

In the following we prove the security of our direct PAKS scheme using our definitions from Section III-B. In the proofs we adopt the standard game-hopping technique. Let $\mathtt{succ}_n$ denote the event that the adversary wins in the experiment n.

A. IND-CKA-security of our PAKS scheme

Theorem 1: Our direct PAKS construction is IND-CKA-secure assuming the hardness of the DDH problem and security of $\mathtt{KDF}_1$, $\mathtt{KDF}_2$, PRF and MAC.

Proof. **Experiment** $Exp_0^{\mathtt{IND}}$. The simulator initializes $\boldsymbol{\tau}, i^*, j, \mathtt{Set}$ and $\mathtt{par} \leftarrow \{\mathbb{G}, q, g, h, H, \mathtt{KDF}_1, \mathtt{KDF}_2, \mathtt{PRF}, \mathtt{MAC}\}$ as defined in the real security experiment $Exp^{\mathtt{IND\text{-}CKA}\text{-}b}_{\mathtt{PAKS}, \mathcal{A}}(\kappa)$. The oracles $\mathtt{Ch}_{\mathtt{ind}}(b, \cdot, \cdot, \cdot, \cdot)$, $\mathtt{Reg}(\cdot)$, $\mathtt{Out}(\cdot, \cdot, \cdot)$, $\mathtt{Ret}(\cdot, \cdot)$ and $\mathtt{RetS}(\cdot)$ are implemented as follows.

- $\mathtt{Ch}_{\mathtt{ind}}(b,\cdot,\cdot,\cdot,\cdot)$: on input $(i,w_0,w_1,\mathtt{ix}^*)$ the oracle aborts if $((i^*\geq 0)\vee(i\geq j)\vee((i,w_0)\in\mathtt{Set})\vee((i,w_1)\in\mathtt{Set}))$; otherwise, it sets $i^*\leftarrow i$ and invokes oracle $\mathtt{Out}(i^*,w_b,\mathtt{ix}^*)$.
- $\mathtt{Reg}(\cdot)$: on input $d\in\{0,1\}$ the simulator randomly selects fresh $\pi\xleftarrow{\$}\mathcal{D}$ and $K\xleftarrow{\$}\mathbb{G}$ and initialises an empty database $\boldsymbol{C}_{d,j}$. The simulator and $\mathcal{A}$ complete the `Register` protocol, where the simulator plays the roles of $\mathtt{U}$ and $\mathtt{S}_d$, and $\mathcal{A}$ plays the role of $\mathtt{S}_{1\text{-}d}$. The oracle sends j to $\mathcal{A}$ as a session identifier. Finally, it records $\boldsymbol{\tau}[j]\leftarrow(d,\pi,\mathtt{info}_d,r_2,x_{1\text{-}d})$, $\mathtt{info}_d\leftarrow(\mathtt{S}_{1\text{-}d},x_d,g^{r_1},g^{r_2},C_\pi,K_d,\mathtt{mk}_d)$, increments $j\leftarrow j+1$, and stores r_2 and $x_{1\text{-}d}$ for later use in the proof.
- $\mathtt{Out}(\cdot,\cdot,\cdot)$: on input $(i,w,\mathtt{ix})$, the simulator aborts if $(i\geq j)$; otherwise, it obtains $(d,\pi,\mathtt{info}_d,r_2,x_{1\text{-}d})\leftarrow\boldsymbol{\tau}[i]$. Then, the simulator plays the roles of $\mathtt{U}$ and $\mathtt{S}_d$ and interacts with $\mathcal{A}$ who plays the role of $\mathtt{S}_{1\text{-}d}$ in the `Outsource` protocol.
- $\mathtt{Ret}(\cdot,\cdot)$: on input (i,w), the simulator aborts if $(i\geq j)\vee((i=i^*)\wedge(w\in\{w_0,w_1\}))$; or otherwise, it obtains $(d,\pi,\mathtt{info}_d,r_2,x_{1\text{-}d})\leftarrow\boldsymbol{\tau}[i]$. Then, it plays the roles of $\mathtt{U}$ and $\mathtt{S}_d$ and interacts with $\mathcal{A}$ who plays the role of $\mathtt{S}_{1\text{-}d}$ party in the `Retrieve` protocol. Finally, the simulator computes $\mathtt{Set}\leftarrow\mathtt{Set}\cup(i,w)$ if $(i^*=-1)$.
- $\mathtt{RetS}(\cdot)$: on input i, the simulator aborts if $(i\geq j)$; otherwise, it obtains parameters $(d,\pi,\mathtt{info}_d,r_2,x_{1\text{-}d})\leftarrow\boldsymbol{\tau}[i]$ and executes $\mathtt{Retrieve}\mathtt{S}_d(\mathtt{par},\mathtt{U},\mathtt{info}_d)$.

Lemma 1: $Adv^{\mathtt{IND\text{-}CKA}}_{\mathtt{PAKS},\mathcal{A}}(\kappa)=\Pr[\mathtt{succ}^{\mathtt{IND}}_0]-1/2$

Experiment $Exp^{\mathtt{IND}}_1$. This experiment is similar to $Exp^{\mathtt{IND}}_0$ except that the simulator aborts if some value for y_d used on behalf of honest server $\mathtt{S}_d$ appears in two different protocol sessions through oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{Ret}(\cdot,\cdot)$ and $\mathtt{RetS}(\cdot)$.

Lemma 2: $\Pr[\mathtt{succ}^{\mathtt{IND}}_0]=\Pr[\mathtt{succ}^{\mathtt{IND}}_1]$

Experiment $Exp^{\mathtt{IND}}_2$. This experiment is similar to $Exp^{\mathtt{IND}}_1$ except that the simulator aborts if some value for Y appears in two different protocol sessions executed through oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{Ret}(\cdot,\cdot)$ and $\mathtt{RetS}(\cdot)$.

1) By the perfect hiding property of Pedersen commitments, value $Y_{1\text{-}d}$ is guaranteed to be independent from Y_d because the adversary acquires nothing from $\mathtt{c}_d$.
2) Due to the binding property of Pedersen commitments, which is based on the hardness of the DL problem, it is hard to open $\mathtt{c}_{1\text{-}d}$ to a different $Y'_{1\text{-}d}\neq Y_{1\text{-}d}$.

Since $Y_{1\text{-}d}$ is guaranteed to be independent from Y_d; and Y_d is fresh, we can follow that Y is fresh based on the hardness of the DL problem.

Lemma 3: $|\Pr[\mathtt{succ}^{\mathtt{IND}}_1]-\Pr[\mathtt{succ}^{\mathtt{IND}}_2]|\leq Adv^{\mathtt{DL}}_{\mathcal{A}}(\kappa)$

Experiment $Exp^{\mathtt{IND}}_3$. This experiment is similar to $Exp^{\mathtt{IND}}_2$ except that in oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{Ret}(\cdot,\cdot)$ and $\mathtt{RetS}(\cdot)$, the message (Z_d,μ_d) from the honest server $\mathtt{S}_d$ to the user is replaced with (E,μ'_d) where $E\xleftarrow{\$}\mathbb{G}$ and $\mu'_d\leftarrow\mathtt{Tag}(\mathtt{mk}_d,A,Y,E)$. We discuss the following two cases:

1) For the oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$ and $\mathtt{Ret}(\cdot,\cdot)$, let (g,g^α,g^β,Q) be an instance of the `DDH` problem, the simulator aims to output 1 if $Q=g^{\alpha\beta}$; or 0 otherwise. The simulator sets $A\leftarrow g^\alpha h^\pi$, $Y_d\leftarrow g^\beta$, $R_d\leftarrow(g^\beta)^{r_2}$ and
$$Z_d\leftarrow K_d(g^\beta)^{r_2(x_0+x_1)}Q^{\text{-}1}(g^{r_1}\cdot(g^\beta)^{r_2}\cdot R_{1\text{-}d})^{\text{-}x_d}$$
If $Q=g^{\alpha\beta}$, this experiment is identical to $Exp^{\mathtt{IND}}_2$; otherwise, to $Exp^{\mathtt{IND}}_3$. The hardness of the DDH problem implies the indistinguishability of $Exp^{\mathtt{IND}}_2$ from $Exp^{\mathtt{IND}}_3$.
2) For oracle $\mathtt{RetS}(\cdot)$, assume π' is the password tried by $\mathcal{A}$, the key K (in $Exp^{\mathtt{IND}}_2$) is equal to $Z_0Z_1Y^ah^{(\pi-\pi')(y_0+y_1)}$; under the `DDH` assumption, the adversary cannot distinguish $h^{(\pi-\pi')(y_0+y_1)}$ (in $Exp^{\mathtt{IND}}_2$) from a random number in $\mathbb{G}$ (in $Exp^{\mathtt{IND}}_3$) unless $\pi'=\pi$ which denotes a successful on-line dictionary attack. By the uniform distribution of passwords, its probability is estimated as $q_s\cdot Adv^{\mathtt{DDH}}_{\mathcal{A}}(\kappa)+\frac{q_s}{|\mathcal{D}|}$.

Lemma 4: $|\Pr[\mathtt{succ}^{\mathtt{IND}}_2]-\Pr[\mathtt{succ}^{\mathtt{IND}}_3]|\leq(q_s+1)Adv^{\mathtt{DDH}}_{\mathcal{A}}(\kappa)+\frac{q_s}{|\mathcal{D}|}$

Experiment $Exp^{\mathtt{IND}}_4$. This experiment is similar to $Exp^{\mathtt{IND}}_3$ except that in each session i, values $\mathtt{mk}_u\leftarrow\mathtt{KDF}_1(K,\mathtt{U},\text{`0'})$, $\mathtt{mk}_d\leftarrow\mathtt{KDF}_1(K,\mathtt{S}_d,\text{`1'})$, $\mathtt{mk}_{1\text{-}d}\leftarrow\mathtt{KDF}_1(K,\mathtt{S}_{1\text{-}d},\text{`1'})$ are replaced with $\mathtt{mk}_u\leftarrow F_1(i,\mathtt{U},\text{`0'})$, $\mathtt{mk}_d\leftarrow F_1(i,\mathtt{S}_d,\text{`1'})$ and $\mathtt{mk}_{1\text{-}d}\leftarrow F_1(i,\mathtt{S}_{1\text{-}d},\text{`1'})$, respectively. A table T_1 is initialized to be empty in the beginning of $Exp^{\mathtt{IND}}_4$. The deterministic function $F_1:\{0,1\}^*\rightarrow\mathcal{K}_{\mathtt{MAC}}$ is defined as follows: if $\exists(i,id,k,\mathtt{mk})\in T_1$ then $F_1(i,id,k)$ returns $\mathtt{mk}$; otherwise, the simulator randomly picks a fresh $\mathtt{mk}\xleftarrow{\$}\mathcal{K}_{\mathtt{MAC}}$, stores $(i,id,k,\mathtt{mk})$ in T_1 and returns $\mathtt{mk}$ where fresh means that no record of the form $(\cdot,\cdot,\cdot,\mathtt{mk})\in T_1$ exists so far. Since $\mathcal{A}$ only acquires $\mathtt{mk}_{1\text{-}d}$, by the uniform distribution of K and the security of $\mathtt{KDF}_1$, we obtain

Lemma 5: $|\Pr[\mathtt{succ}^{\mathtt{IND}}_3]-\Pr[\mathtt{succ}^{\mathtt{IND}}_4]|\leq q_r\cdot Adv^{\mathtt{KDF}}_{\mathcal{A}}(\kappa)$

Experiment $Exp^{\mathtt{IND}}_5$. This experiment is similar to $Exp^{\mathtt{IND}}_4$ except that in each session i of oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$ and $\mathtt{Ret}(\cdot,\cdot)$, value $t\leftarrow\mathtt{KDF}_2(K,w)$ is replaced with $t\leftarrow F_2(i,w)$. T_2 is initialized as an empty table in the beginning of $Exp^{\mathtt{IND}}_5$. F_2 returns t if $\exists(i,w,t)\in T_2$; otherwise, F_2 picks a fresh $t\xleftarrow{\$}\mathcal{K}_{\mathtt{PRF}}$, stores (i,w,t) in T_2 and returns t where fresh means that no record of the form $(\cdot,\cdot,t)$ exists in T_2. By the uniform distribution of K and the security of $\mathtt{KDF}_2$, we have

Lemma 6: $|\Pr[\mathtt{succ}^{\mathtt{IND}}_4]-\Pr[\mathtt{succ}^{\mathtt{IND}}_5]|\leq(q_o+q_t)Adv^{\mathtt{KDF}}_{\mathcal{A}}(\kappa)$

Experiment $Exp^{\mathtt{IND}}_6$. This experiment is similar to $Exp^{\mathtt{IND}}_5$ except for one of the following cases:

1) For the oracle $\mathtt{Out}(\cdot,\cdot,\cdot)$, the adversary successfully forges $((C,\mathtt{ix}),\mu_{\mathtt{sk}_d})$ which satisfies $\mathtt{Vrfy}(\mathtt{sk}_d,(C,\mathtt{ix}),\mu_{\mathtt{sk}_d})=1$.
2) For the oracles $\mathtt{Ret}(\cdot,\cdot)$ or $\mathtt{RetS}(\cdot)$, the adversary successfully forges $(t,\mu_{\mathtt{sk}_d})$ which satisfies $\mathtt{Vrfy}(\mathtt{sk}_d,t,\mu_{\mathtt{sk}_d})=1$.

By the unforgeability of `MAC`, we have

Lemma 7: $|\Pr[\mathtt{succ}^{\mathtt{IND}}_5]-\Pr[\mathtt{succ}^{\mathtt{IND}}_6]|\leq(q_o+q_t+q_s)Adv^{\mathtt{MAC}}_{\mathcal{A}}(\kappa)$

Experiment $Exp^{\mathtt{IND}}_7$. This experiment is similar to $Exp^{\mathtt{IND}}_6$ except that in oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$ and $\mathtt{Ret}(\cdot,\cdot)$, the value v is set in a different way. Let $\mathcal{O}_{\mathtt{PRF}}(\cdot)$ be the oracle from the security

experiment of the pseudorandom function PRF; and let T_v be initialized as an empty table in the beginning of $Exp_7^{\mathtt{IND}}$. When the simulator needs to compute $v \leftarrow \mathtt{PRF}(t,e)$ in session i, it obtains v using table T_v. If $\exists(i,t,e,r_v,v) \in T_v$, the simulator uses v from T_v; otherwise, it randomly picks $r_v \xleftarrow{\$} \{0,1\}^\kappa$, stores $(i,t,e,r_v,\mathcal{O}_{\mathtt{PRF}}(r_v))$ in T_v and obtains $v \leftarrow \mathcal{O}_{\mathtt{PRF}}(r_v)$. Assuming the pseudorandomness of PRF, we have

***Lemma** 8:* $\Pr[\mathtt{succ}_7^{\mathtt{IND}}] \leq 1/2 + (q_o + q_t)Adv_{\mathcal{A}}^{\mathtt{PRF}}(\kappa)$

As a consequence, based on Lemmas 1 to 8 we can conclude that our proposed PAKS construction is IND-CKA-secure assuming the intractability of the DDH problem and security of $\mathtt{KDF}_1$, $\mathtt{KDF}_2$, PRF and MAC.

B. Authentication property of our PAKS scheme

***Theorem** 2:* Our proposed PAKS construction provides authentication based on the hardness of the DDH problem and security of $\mathtt{KDF}_1$, $\mathtt{KDF}_2$ and MAC.

Proof. **Experiment** $Exp_0^{\mathtt{Auth}}$. The simulator initializes $\boldsymbol{\tau}, j, \mathtt{Set}$ and $\mathtt{par} \leftarrow \{\mathbb{G}, q, g, h, H, \mathtt{KDF}_1, \mathtt{KDF}_2, \mathtt{PRF}, \mathtt{MAC}\}$ as defined in the real security experiment $Exp_{\mathtt{PAKS},\mathcal{A}}^{\mathtt{Auth}}(\kappa)$. The oracles $\mathtt{Reg}(\cdot)$, $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{OutS}(\cdot)$ and $\mathtt{Ret}(\cdot,\cdot)$ are executed by the simulator as follows.

- $\mathtt{Reg}(\cdot)$: on input $d \in \{0,1\}$ the simulator randomly selects a fresh $\pi \xleftarrow{\$} \mathcal{D}$ and $K \xleftarrow{\$} \mathbb{G}$, and initializes an empty database $\boldsymbol{C}_{d,j}$. Then, the simulator and $\mathcal{A}$ execute the Register protocol, where the simulator plays the role of $\mathtt{U}, \mathtt{S}_d$ and $\mathcal{A}$ plays the role of $\mathtt{S}_{1\text{-}d}$. The simulator then sends j to $\mathcal{A}$ as a session identifier. Finally, the simulator records $\boldsymbol{\tau}[j] \leftarrow (d, \pi, \mathtt{info}_d, r_2, x_{1\text{-}d})$, $\mathtt{info}_d \leftarrow (\mathtt{S}_{1\text{-}d}, x_d, g^{r_1}, g^{r_2}, C_\pi, K_d, \mathtt{mk}_d)$, increments $j \leftarrow j+1$, and stores r_2 and $x_{1\text{-}d}$ for later use in the proof.
- $\mathtt{Out}(\cdot,\cdot,\cdot)$: on input $(i, w, \mathtt{ix})$, the simulator aborts if $(i \geq j)$; otherwise, it obtains $(d, \pi, \mathtt{info}_d, r_2, x_{1\text{-}d}) \leftarrow \boldsymbol{\tau}[i]$. Then, it sets $\mathtt{Set} \leftarrow \mathtt{Set} \cup (i, w, \mathtt{ix})$. Finally, it plays the roles of U and $\mathtt{S}_d$, and interacts with $\mathcal{A}$ who plays the role of $\mathtt{S}_{1\text{-}d}$ party in the Outsource protocol.
- $\mathtt{OutS}(\cdot)$: on input i, the simulator aborts if $(i \geq j)$; otherwise, it obtains parameters $(d, \pi, \mathtt{info}_d, r_2, x_{1\text{-}d}) \leftarrow \boldsymbol{\tau}[i]$ and executes $\mathtt{Outsource}\mathtt{S}_d(\mathtt{par}, \mathtt{U}, \mathtt{info}_d)$.
- $\mathtt{Ret}(\cdot,\cdot)$: on input (i, w), the simulator aborts if $(i \geq j)$; otherwise, it obtains parameters $(d, \pi, \mathtt{info}_d, r_2, x_{1\text{-}d}) \leftarrow \boldsymbol{\tau}[i]$. Then, it plays the roles of U and $\mathtt{S}_d$, and interacts with $\mathcal{A}$ who plays the role of $\mathtt{S}_{1\text{-}d}$ in the Retrieve protocol.

***Lemma** 9:* $Adv_{\mathtt{PAKS},\mathcal{A}}^{\mathtt{Auth}}(\kappa) = \Pr[\mathtt{succ}_0^{\mathtt{Auth}}]$

Experiment $Exp_1^{\mathtt{Auth}}$. This experiment is similar to $Exp_0^{\mathtt{Auth}}$ except that the value y_d is ensured to be fresh in every session executed by the simulator through the oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{OutS}(\cdot)$ and $\mathtt{Ret}(\cdot,\cdot)$.

***Lemma** 10:* $\Pr[\mathtt{succ}_0^{\mathtt{Auth}}] = \Pr[\mathtt{succ}_1^{\mathtt{Auth}}]$

Experiment $Exp_2^{\mathtt{Auth}}$. This experiment is similar to $Exp_1^{\mathtt{Auth}}$ except that the simulator aborts if a value for Y repeats in two different sessions of the protocol executed by the simulator through oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{OutS}(\cdot)$, and $\mathtt{Ret}(\cdot,\cdot)$.

1) By the perfect hiding of Pedersen commitments, values of $Y_{1\text{-}d}$ are guaranteed to be independent from Y_d because the adversary acquires nothing from $\mathtt{c}_d$.
2) Because of the binding property of Pedersen commitments, which is based on the hardness of the DL problem, it is hard to open $\mathtt{c}_{1\text{-}d}$ to a different value $Y'_{1\text{-}d} \neq Y_{1\text{-}d}$.

Since $Y_{1\text{-}d}$ is guaranteed to be independent from Y_d and Y_d is fresh, the freshness of Y is implied by the hardness of the DL problem.

***Lemma** 11:* $|\Pr[\mathtt{succ}_1^{\mathtt{Auth}}] - \Pr[\mathtt{succ}_2^{\mathtt{Auth}}]| \leq Adv_{\mathcal{A}}^{\mathtt{DL}}(\kappa)$

Experiment $Exp_3^{\mathtt{Auth}}$. This experiment is similar to $Exp_2^{\mathtt{Auth}}$ except that in oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{Ret}(\cdot,\cdot)$ and $\mathtt{OutS}(\cdot)$, the message (Z_d, μ_d) from the honest server $\mathtt{S}_d$ to the user is replaced with (E, μ'_d) where $E \xleftarrow{\$} \mathbb{G}$ and $\mu'_d \leftarrow \mathtt{Tag}(\mathtt{mk}_d, A, Y, E)$. We consider the following two case:

1) For oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$ and $\mathtt{Ret}(\cdot,\cdot)$, let $(g, g^\alpha, g^\beta, Q)$ be an instance of the DDH problem, the simulator aims to output 1 if $Q = g^{\alpha\beta}$; or 0 otherwise. The simulator sets $A \leftarrow g^\alpha h^\pi$, $Y_d \leftarrow g^\beta$, $R_d \leftarrow (g^\beta)^{r_2}$ and
$$Z_d \leftarrow K_d (g^\beta)^{r_2(x_0+x_1)} Q^{\mathtt{-1}} (g^{r_1} \cdot (g^\beta)^{r_2} \cdot R_{1\text{-}d})^{\mathtt{-}x_d}$$
If $Q = g^{\alpha\beta}$, then this experiment is identical to $Exp_2^{\mathtt{Auth}}$; otherwise, it is identical to $Exp_3^{\mathtt{Auth}}$. The hardness of the DDH problem directly implies the indistinguishability of $Exp_2^{\mathtt{Auth}}$ from $Exp_3^{\mathtt{Auth}}$.
2) For the oracle $\mathtt{OutS}(\cdot)$, assume π' is a password used by the adversary, the key K (in $Exp_2^{\mathtt{Auth}}$) is equal to $Z_0 Z_1 Y^a h^{(\pi-\pi')(y_0+y_1)}$; under the DDH assumption, the adversary cannot distinguish $h^{(\pi-\pi')(y_0+y_1)}$ (in $Exp_2^{\mathtt{Auth}}$) from a random number in $\mathbb{G}$ (in $Exp_3^{\mathtt{Auth}}$) unless $\pi' = \pi$ which denotes a successful on-line dictionary attack. By the uniform distribution of passwords, its probability is estimated as $q_s \cdot Adv_{\mathcal{A}}^{\mathtt{DDH}}(\kappa) + \frac{q_s}{|\mathcal{D}|}$.

***Lemma** 12:* $|\Pr[\mathtt{succ}_2^{\mathtt{Auth}}] - \Pr[\mathtt{succ}_3^{\mathtt{Auth}}]| \leq (q_s + 1)Adv_{\mathcal{A}}^{\mathtt{DDH}}(\kappa) + \frac{q_s}{|\mathcal{D}|}$

Experiment $Exp_4^{\mathtt{Auth}}$. This experiment is similar to $Exp_3^{\mathtt{Auth}}$ except that in each session i, values for $\mathtt{mk}_u \leftarrow \mathtt{KDF}_1(K, \mathtt{U}, \text{`0'})$, $\mathtt{mk}_d \leftarrow \mathtt{KDF}_1(K, \mathtt{S}_d, \text{`1'})$, $\mathtt{mk}_{1\text{-}d} \leftarrow \mathtt{KDF}_1(K, \mathtt{S}_{1\text{-}d}, \text{`1'})$ are replaced with $\mathtt{mk}_u \leftarrow F_1(i, \mathtt{U}, \text{`0'})$, $\mathtt{mk}_d \leftarrow F_1(i, \mathtt{S}_d, \text{`1'})$ and $\mathtt{mk}_{1\text{-}d} \leftarrow F_1(i, \mathtt{S}_{1\text{-}d}, \text{`1'})$, respectively. A table T_1 is initialized to be empty in the beginning of $Exp_4^{\mathtt{Auth}}$. A deterministic function $F_1 : \{0,1\}^* \rightarrow \mathcal{K}_{\mathtt{MAC}}$ is defined as follows: if $\exists(i, id, k, \mathtt{mk}) \in T_1$, $F_1(i, id, k)$ then return mk; otherwise, the simulator randomly picks a fresh $\mathtt{mk} \xleftarrow{\$} \mathcal{K}_{\mathtt{MAC}}$, stores $(i, id, k, \mathtt{mk})$ on T_1 and returns $\mathtt{mk} \leftarrow F_1(i, id, k)$ where fresh means that no record of the form $(\cdot,\cdot,\cdot,\mathtt{mk}) \in T_1$ exists so far. Since the adversary only acquires $\mathtt{mk}_{1\text{-}d}$, by the uniform distribution of K as well as the security of $\mathtt{KDF}_1$, we obtain

***Lemma** 13:* $|\Pr[\mathtt{succ}_3^{\mathtt{Auth}}] - \Pr[\mathtt{succ}_4^{\mathtt{Auth}}]| \leq q_r \cdot Adv_{\mathcal{A}}^{\mathtt{KDF}}(\kappa)$

Experiment $Exp_5^{\mathtt{Auth}}$. This experiment is similar to $Exp_4^{\mathtt{Auth}}$ except that in each session i for the oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$ and $\mathtt{Ret}(\cdot,\cdot)$, the value $t \leftarrow \mathtt{KDF}_2(K, w)$ is replaced with $t \leftarrow F_2(i, w)$. T_2 is initialized as an empty table in the beginning

of $Exp_5^{\mathtt{Auth}}$. Function F_2 returns t if $\exists(i,w,t) \in T_2$; otherwise, the simulator randomly picks a fresh $t \xleftarrow{\$} \mathcal{K}_{\mathtt{PRF}}$, stores (i,w,t) on table T_2 and returns t where fresh means that no record of the form $(\cdot,\cdot,t)$ exists so far in T_2 . By the uniform distribution of K and the security of $\mathtt{KDF}_2$, we obtain

Lemma 14: $|\Pr[\mathtt{succ}_4^{\mathtt{Auth}}] - \Pr[\mathtt{succ}_5^{\mathtt{Auth}}]| \leq (q_o + q_t)Adv_{\mathcal{A}}^{\mathtt{KDF}}(\kappa)$

We observe that $Exp_5^{\mathtt{Auth}}$ is simulated independent the key K. The only probability of winning $Exp_5^{\mathtt{Auth}}$ comes from the adversary successfully forging μ_c for $(e,v,\mathtt{ix})$ such that $\mathtt{Vrfy}(\mathtt{mk}_u,(e,v,\mathtt{ix}),\mu_c)=1$. Assuming that MAC is unforgeable, we obtain

Lemma 15: $\Pr[\mathtt{succ}_5^{\mathtt{Auth}}] = Adv_{\mathcal{A}}^{\mathtt{MAC}}(\kappa)$

To sum, by Lemmas 9 to 15, we can conclude that our direct PAKS scheme provides authentication based on the hardness of the DDH problem and security of $\mathtt{KDF}_1$, $\mathtt{KDF}_2$ and MAC.

C. Consistency property of our PAKS scheme

Theorem 3: Our direct PAKS construction offers consistency based on the hardness of the DDH problem and security of $\mathtt{KDF}_1$, $\mathtt{KDF}_2$, PRF and MAC.

Proof. **Experiment** $Exp_0^{\mathtt{Con}}$. The simulator initializes $\boldsymbol{\tau}, i^*, j, \mathtt{Set}$ and $\mathtt{par} \leftarrow \{\mathbb{G}, q, g, h, H, \mathtt{KDF}_1, \mathtt{KDF}_2, \mathtt{PRF}, \mathtt{MAC}\}$ as defined in the real security experiment $Exp_{\mathtt{PAKS},\mathcal{A}}^{\mathtt{Con}}(\kappa)$. The oracles $\mathtt{Ch}_{\mathtt{con}}(\cdot,\cdot,\cdot)$, $\mathtt{Reg}(\cdot)$, $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{Ret}(\cdot,\cdot)$ and $\mathtt{RetS}(\cdot)$ are answered as follows.

- $\mathtt{Ch}_{\mathtt{con}}(\cdot,\cdot,\cdot)$: on input $(i,w_0,\mathtt{ix}^*)$, the simulator aborts if $((i^* \geq 0) \vee (i \geq j))$. Otherwise, it sets $i^* \leftarrow i$ and invokes oracle $\mathtt{Out}(i^*,w_0,\mathtt{ix}^*)$.
- $\mathtt{Reg}(\cdot)$: on input $d \in \{0,1\}$ the simulator randomly selects fresh $\pi \xleftarrow{\$} \mathcal{D}$ and $K \xleftarrow{\$} \mathbb{G}$, and initializes an empty database $\boldsymbol{C}_{d,j}$. Then, the simulator and $\mathcal{A}$ execute the Register protocol, where the simulator plays the roles of U and $\mathtt{S}_d$, and interacts with $\mathcal{A}$ that plays the role of $\mathtt{S}_{1\text{-}d}$. The simulator then sends j to $\mathcal{A}$ as a session identifier. Finally, it records $\boldsymbol{\tau}[j] \leftarrow (d,\pi,\mathtt{info}_d,r_2,x_{1\text{-}d})$, $\mathtt{info}_d \leftarrow (\mathtt{S}_{1\text{-}d},x_d,g^{r_1},g^{r_2},C_\pi,K_d,\mathtt{mk}_d)$, increments $j \leftarrow j+1$, and stores variables r_2 and $x_{1\text{-}d}$ for later use in the proof.
- $\mathtt{Out}(\cdot,\cdot,\cdot)$: on input $(i,w,\mathtt{ix})$, it aborts if $(i \geq j)$; otherwise, it obtains $(d,\pi,\mathtt{info}_d,r_2,x_{1\text{-}d}) \leftarrow \boldsymbol{\tau}[i]$. Then, it sets $\mathtt{Set} \leftarrow \mathtt{Set} \cup (i,w,\mathtt{ix})$. The simulator and $\mathcal{A}$ then execute the Outsource protocol, where the simulator plays the roles of U and $\mathtt{S}_d$, and interacts with $\mathcal{A}$ that plays the role of $\mathtt{S}_{1\text{-}d}$.
- $\mathtt{Ret}(\cdot,\cdot)$: on input (i,w), it aborts if $(i \geq j)$; or otherwise, it obtains parameters $(d,\pi,\mathtt{info}_d,r_2,x_{1\text{-}d}) \leftarrow \boldsymbol{\tau}[i]$. The simulator and $\mathcal{A}$ then execute the Retrieve protocol where the simulator plays the roles of U and $\mathtt{S}_d$, and interacts with $\mathcal{A}$ that plays the role of $\mathtt{S}_{1\text{-}d}$.
- $\mathtt{RetS}(\cdot)$: on input i, the simulator aborts if $(i \geq j)$; otherwise, it obtains parameters $(d,\pi,\mathtt{info}_d,r_2,x_{1\text{-}d}) \leftarrow \boldsymbol{\tau}[i]$ and executes $\mathtt{Retrieve}\mathtt{S}_d(\mathtt{par},\mathtt{U},\mathtt{info}_d)$.

Lemma 16: $Adv_{\mathtt{PAKS},\mathcal{A}}^{\mathtt{Con}}(\kappa) = \Pr[\mathtt{succ}_0^{\mathtt{Con}}]$

Experiment $Exp_1^{\mathtt{Con}}$. This experiment is similar to $Exp_0^{\mathtt{Con}}$ except that the value y_d is ensured to be fresh in every session executed by the simulator through the oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{Ret}(\cdot,\cdot)$ and $\mathtt{RetS}(\cdot)$.

Lemma 17: $\Pr[\mathtt{succ}_0^{\mathtt{Con}}] = \Pr[\mathtt{succ}_1^{\mathtt{Con}}]$

Experiment $Exp_2^{\mathtt{Con}}$. This experiment is similar to $Exp_1^{\mathtt{Con}}$ except that the simulator aborts if a value for Y repeats in two different sessions of the protocol executed by the simulator through oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{Ret}(\cdot,\cdot)$ and $\mathtt{RetS}(\cdot)$.

1) By the perfect hiding of Pedersen commitments, values of $Y_{1\text{-}d}$ are guaranteed to be independent from Y_d because the adversary acquires nothing from $\mathtt{c}_d$.
2) Because of the binding property of Pedersen commitments, which is based on the hardness of the DL problem, it is hard to open $\mathtt{c}_{1\text{-}d}$ to a different value $Y'_{1\text{-}d} \neq Y_{1\text{-}d}$.

Since $Y_{1\text{-}d}$ is guaranteed to be independent from Y_d and Y_d is fresh, this implies that the freshness of Y is based on the hardness of the DL problem.

Lemma 18: $|\Pr[\mathtt{succ}_1^{\mathtt{Con}}] - \Pr[\mathtt{succ}_2^{\mathtt{Con}}]| \leq Adv_{\mathcal{A}}^{\mathtt{DL}}(\kappa)$

Experiment $Exp_3^{\mathtt{Con}}$. This experiment is similar to $Exp_2^{\mathtt{Con}}$ except that in oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{Ret}(\cdot,\cdot)$ and $\mathtt{RetS}(\cdot)$, the message (Z_d,μ_d) from the honest server $\mathtt{S}_d$ to the user is replaced with (E,μ'_d) where $E \xleftarrow{\$} \mathbb{G}$ and $\mu'_d \leftarrow \mathtt{Tag}(\mathtt{mk}_d,A,Y,E)$. We discuss the following two cases:

1) For the oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$ and $\mathtt{Ret}(\cdot,\cdot)$, let (g,g^α,g^β,Q) be an instance of the DDH problem, the simulator aims to output 1 if $Q = g^{\alpha\beta}$; or 0 otherwise. The simulator sets $A \leftarrow g^\alpha h^\pi$, $Y_d \leftarrow g^\beta$, $R_d \leftarrow (g^\beta)^{r_2}$ and
$$Z_d \leftarrow K_d (g^\beta)^{r_2(x_0+x_1)} Q^{-1} (g^{r_1} \cdot (g^\beta)^{r_2} \cdot R_{1\text{-}d})^{-x_d}$$
If $Q = g^{\alpha\beta}$, this experiment is identical to $Exp_2^{\mathtt{Con}}$; otherwise, identical to $Exp_3^{\mathtt{Con}}$. The hardness of the DDH problem thus implies the indistinguishability of $Exp_2^{\mathtt{Con}}$ from $Exp_3^{\mathtt{Con}}$.
2) For oracle $\mathtt{RetS}(\cdot)$, assume π' is the password tried by $\mathcal{A}$, the key K (in $Exp_2^{\mathtt{Con}}$) is equal to $Z_0 Z_1 Y^a h^{(\pi-\pi')(y_0+y_1)}$; under the DDH assumption, the adversary cannot distinguish $h^{(\pi-\pi')(y_0+y_1)}$ (in $Exp_2^{\mathtt{Con}}$) from a random number in $\mathbb{G}$ (in $Exp_3^{\mathtt{Con}}$) unless $\pi' = \pi$ which denotes a successful on-line dictionary attack. By the uniform distribution of passwords, its probability is estimated as $q_s \cdot Adv_{\mathcal{A}}^{\mathtt{DDH}}(\kappa) + \frac{q_s}{|\mathcal{D}|}$.

Lemma 19: $|\Pr[\mathtt{succ}_2^{\mathtt{Con}}] - \Pr[\mathtt{succ}_3^{\mathtt{Con}}]| \leq (q_s + 1)Adv_{\mathcal{A}}^{\mathtt{DDH}}(\kappa) + \frac{q_s}{|\mathcal{D}|}$

Experiment $Exp_4^{\mathtt{Con}}$. This experiment is similar to $Exp_3^{\mathtt{Con}}$ except that in each session i, values for $\mathtt{mk}_u \leftarrow \mathtt{KDF}_1(K,\mathtt{U},\text{`0'})$, $\mathtt{mk}_d \leftarrow \mathtt{KDF}_1(K,\mathtt{S}_d,\text{`1'})$, $\mathtt{mk}_{1\text{-}d} \leftarrow \mathtt{KDF}_1(K,\mathtt{S}_{1\text{-}d},\text{`1'})$ are replaced with $\mathtt{mk}_u \leftarrow F_1(i,\mathtt{U},\text{`0'})$, $\mathtt{mk}_d \leftarrow F_1(i,\mathtt{S}_d,\text{`1'})$ and $\mathtt{mk}_{1\text{-}d} \leftarrow F_1(i,\mathtt{S}_{1\text{-}d},\text{`1'})$, respectively. An empty table T_1 is initialized in the beginning of $Exp_4^{\mathtt{Con}}$. The function $F_1(i,id,k)$ returns mk if $\exists(i,id,k,\mathtt{mk}) \in T_1$; otherwise, it picks a fresh $\mathtt{mk} \xleftarrow{\$} \mathcal{K}_{\mathtt{MAC}}$, stores $(i,id,k,\mathtt{mk})$ in T_1, and returns $\mathtt{mk} \leftarrow F_1(i,id,k)$. Since the adversary only obtains $\mathtt{mk}_{1\text{-}d}$, by the uniform distribution of K and the security of $\mathtt{KDF}_1$, we obtain

Lemma 20: $|\Pr[\mathtt{succ}_3^{\mathtt{Con}}] - \Pr[\mathtt{succ}_4^{\mathtt{Con}}]| \leq q_r \cdot Adv_{\mathcal{A}}^{\mathtt{KDF}}(\kappa)$

Experiment $Exp_5^{\mathtt{Con}}$. Let $C_0 = (e_0, v_0, \mu_c)$ be the ciphertext that has been outsourced in response to the query of $\mathcal{A}$ to the oracle $\mathtt{Ch}_{\mathtt{con}}(\cdot,\cdot,\cdot)$ on input $(i, w_0, \mathtt{ix}^*)$. Note that $v_0 = \mathtt{PRF}(t_0, e_0)$ where $t_0 \leftarrow \mathtt{KDF}_2(K, w_0)$. This experiment is similar to $Exp_4^{\mathtt{Con}}$ except that while processing $\mathtt{Ret}(i^*, w_1)$ on behalf of honest $\mathtt{U}$ and $\mathtt{S}_d$ the simulator aborts if $t_0 \leftarrow \mathtt{KDF}_2(K, w_1)$. In this case we obtain a $\mathtt{KDF}_2$ collision, i.e. $\mathtt{KDF}_2(K, w_0) = \mathtt{KDF}_2(K, w_1)$ for $w_0 \neq w_1$, and hence

Lemma 21: $|\Pr[\mathtt{succ}_4^{\mathtt{Con}}] - \Pr[\mathtt{succ}_5^{\mathtt{Con}}]| \leq q_o \cdot Adv_{\mathcal{A}}^{\mathtt{KDF}}(\kappa)$

Experiment $Exp_6^{\mathtt{Con}}$. This experiment is similar to $Exp_5^{\mathtt{Con}}$ except that while processing $\mathtt{Ret}(i^*, w_1)$ on behalf of honest $\mathtt{U}$ and $\mathtt{S}_d$ the simulator aborts if $v_0 = \mathtt{PRF}(t_1, e_0)$ for some $t_1 \leftarrow \mathtt{KDF}_2(K, w_1)$. In this case we obtain a $\mathtt{PRF}$ collision, i.e. $\mathtt{PRF}(t_0, e_0) = \mathtt{PRF}(t_1, e_0)$ for $t_0 \neq t_1$, and hence

Lemma 22: $|\Pr[\mathtt{succ}_5^{\mathtt{Con}}] - \Pr[\mathtt{succ}_6^{\mathtt{Con}}]| \leq q_o \cdot Adv_{\mathcal{A}}^{\mathtt{PRF}}(\kappa)$

$Exp_6^{\mathtt{Con}}$ ensures that the original ciphertext $C_0 = (e_0, v_0, \mu_c)$ that has been outsourced in response to the query $(i, w_0, \mathtt{ix}^*)$ will never pass the verification performed by honest $\mathtt{U}$ and $\mathtt{S}_d$. Hence, in order to win in $Exp_6^{\mathtt{Con}}$ the adversary needs to come up with $C^* = (e^*, v^*, \mu_c^*)$ where $v^* = \mathtt{PRF}(t_1, e^*)$ for $t_1 \leftarrow \mathtt{KDF}_2(K, w_1)$ and μ_c^* is a valid authentication tag on the message $(e^*, v^*, \mathtt{ix}^*)$, which would constitute a MAC forgery. Assuming that $\mathtt{MAC}$ is unforgeable, we conclude

Lemma 23: $\Pr[\mathtt{succ}_6^{\mathtt{Con}}] = Adv_{\mathcal{A}}^{\mathtt{MAC}}(\kappa)$

As a result, based on Lemmas 16 to 23, our PAKS construction offers consistency based on the assumed hardness of the $\mathtt{DL}$, $\mathtt{DDH}$ problems and the security of $\mathtt{KDF}_1$, $\mathtt{KDF}_2$, $\mathtt{PRF}$ and $\mathtt{MAC}$.

VI. Conclusion

We introduced Password Authenticated Keyword Search (PAKS) as a new concept where search over encrypted keywords can be performed solely with the help of a human-memorizable password. In comparison to earlier formats of searchable encryption the use of passwords simplifies key management and by removing the need for storing and managing high-entropy keys on the user side makes the whole process device-agnostic. The use of passwords introduces however new challenges to the design of PAKS protocols; in particular, creating the need for a distributed server architecture to achieve security against offline dictionary attacks.

In this paper we modeled the functionality and security properties of PAKS, incl. $\mathtt{IND\text{-}CKA}$-security for keyword privacy, authentication for outsourcing, and consistency for the search procedure, and proposed a direct PAKS construction those security and privacy has been proven under standard assumptions. Our direct PAKS construction is an optimised version of a more general concept for building PAKS protocols based on techniques underlying password-authenticated secret sharing and symmetric searchable encryption. The proposed PAKS scheme is practical and offers high performance in relation to computations and communications on the user side.

References

[1] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi, "Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions," *J. Cryptology*, vol. 21, no. 3, pp. 350–391, 2008.

[2] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public Key Encryption with Keyword Search," in *EUROCRYPT'04*, 2004, pp. 506–522.

[3] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *Journal of Computer Security*, vol. 19, no. 5, pp. 895–934, 2011.

[4] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *CCS'12*, ACM, 2012, pp. 965–976.

[5] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *ICICS'05*, ser. LNCS, vol. 3783. Springer, 2005, pp. 414–426.

[6] C. Örencik, A. Selcuk, E. Savas, and M. Kantarcioglu, "Multi-keyword search over encrypted data with scoring and search pattern obfuscation," *Int. J. Inf. Sec.*, vol. 15, no. 3, pp. 251–269, 2016.

[7] D. J. Park, K. Kim, and P. J. Lee, "Public key encryption with conjunctive field keyword search," in *WISA'04*, ser. LNCS, vol. 3325. Springer, 2004, pp. 73–86.

[8] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *TCC'07*, ser. LNCS, vol. 4392. Springer, 2007, pp. 535–554.

[9] V. Kuchta and M. Manulis, "Public Key Encryption with Distributed Keyword Search," in *INTRUST'15*, ser. LNCS, vol. 9565. Springer, 2016, pp. 62–83.

[10] R. Chen, Y. Mu, G. Yang, F. Guo, and X. Wang, "Dual-server public-key encryption with keyword search for secure cloud storage," *IEEE Trans. Information Forensics and Security*, vol. 11, no. 4, pp. 789–798, 2016.

[11] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu, "Password-protected secret sharing," in *CCS'11*. ACM, 2011, pp. 433–444.

[12] J. Camenisch, A. Lysyanskaya, and G. Neven, "Practical yet universally composable two-server password-authenticated secret sharing," in *CCS'12*, ACM, 2012, pp. 525–536.

[13] J. Camenisch, A. Lehmann, A. Lysyanskaya, and G. Neven, "Memento: How to reconstruct your secrets from a single password in a hostile environment," in *CRYPTO'14*, ser. LNCS, vol. 8617. Springer, 2014, pp. 256–275.

[14] S. Jarecki, A. Kiayias, and H. Krawczyk, "Round-optimal password-protected secret sharing and T-PAKE in the password-only model," in *ASIACRYPT'14*, ser. LNCS, vol. 8874. Springer, 2014, pp. 233–253.

[15] X. Yi, F. Hao, L. Chen, and J. K. Liu, "Practical threshold password-authenticated secret sharing protocol," in *ESORICS'15*, ser. LNCS, vol. 9326. Springer, 2015, pp. 347–365.

[16] J. Camenisch, R. R. Enderlein, and G. Neven, "Two-server password-authenticated secret sharing uc-secure against transient corruptions," in *PKC'15*, ser. LNCS, vol. 9020. Springer, 2015, pp. 283–307.

[17] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu, "Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online)," in *EuroS&P'16*. IEEE, 2016, pp. 276–291.

[18] F. Kiefer and M. Manulis, "Universally Composable Two-Server PAKE," in *ISC'16*, ser. LNCS, vol. 9866. Springer, 2016, pp. 147–166.

[19] ——, "Blind Password Registration for Two-Server Password Authenticated Key Exchange and Secret Sharing Protocols," in *ISC'16*, ser. LNCS, vol. 9866. Springer, 2016, pp. 95–114.

[20] M. Abdalla, P. Fouque, and D. Pointcheval, "Password-based authenticated key exchange in the three-party setting," in *PKC'05*, ser. LNCS, vol. 3386. Springer, 2005, pp. 65–84.

[21] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *EUROCRYPT'00*, ser. LNCS, vol. 1807. Springer, 2000, pp. 139–155.

[22] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO'91*, ser. LNCS, vol. 576. Springer, 1991, pp. 129–140.

[23] M. Luby and C. Rackoff, "How to construct pseudorandom permutations from pseudorandom functions," *SIAM J. Comput.*, vol. 17, no. 2, pp. 373–386, 1988.

[24] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, "A pseudorandom generator from any one-way function," *SIAM J. Comput.*, vol. 28, no. 4, pp. 1364–1396, 1999.

[25] H. Krawczyk, "Cryptographic extraction and key derivation: The HKDF scheme," in *CRYPTO'10*, ser. LNCS, vol. 6223. Springer, 2010, pp. 631–648.

[26] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *CRYPTO'96*, ser. LNCS, vol. 1109. Springer, 1996, pp. 1–15.