International Journal of Information Security (2021) 20:675–693
https://doi.org/10.1007/s10207-020-00524-5

REGULAR CONTRIBUTION


# Password-authenticated searchable encryption

**Liqun Chen**[1] · **Kaibin Huang**[1] · **Mark Manulis**[1] · **Venkkatesh Sekar**[1]

Published online: 22 November 2020


**Abstract**

We introduce *Password Authenticated Searchable Encryption (PASE)*, a novel searchable encryption scheme where a single human-memorizable password can be used to outsource (encrypted) data with associated keywords to a group of servers and later retrieve this data through the encrypted keyword search procedure. PASE ensures that only the legitimate user who knows the initially registered password can perform these operations. In particular, PASE 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 PASE 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 the functionality of PASE along with two security requirements (indistinguishability against chosen keyword attacks and authentication) 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 constructions support outsourcing and retrieval procedures based on multiple keywords and allow users to change their passwords without any need for the re-encryption of the outsourced data. Our theoretical efficiency comparisons and experimental performance and scalability measurements show that the proposed scheme is practical and offers high performance in relation to computations and communications on the user side. The practicality of our PASE scheme is further demonstrated through its implementation within a JavaScript-based web application that can readily be executed on any (mobile) browser and remains practical for commodity user devices such as laptops and smartphones.

**Keywords** Searchable encryption · Distributed password authentication

## 1 Introduction

### 1.1 Searchable encryption

Using protocols for *Searchable Encryption* [2,10,20,29] 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 or a private-public key pair such as *Public Key Encryption with Keyword Search (PEKS)* schemes on the user side.

✉ Mark Manulis
mark@manulis.eu

[1] Surrey Centre for Cyber Security, University of Surrey, Guildford, United Kingdom

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.

#### 1.1.1 Symmetric searchable encryption

Symmetric searchable encryption enables the user to encrypt the data, organizing it in an arbitrary way (before encryption) and includes additional data structures to allow for efficient access of relevant data. In this setting, the initial work for the user (i.e., for preprocessing the data) is at least as large as the data, but subsequent work (i.e., for accessing the data) is very small relative to the size of the data for both the user and the server. Ostrovsky demonstrated that symmetric searchable encryption can be achieved in its full generality and with optimal security using Oblivious RAM but with huge overhead [35]. Further works try to make the construction

efficient with more rounds and a weaker security model to reduce the overhead. Song et al. [21] approached SSE using a new two layered encryption, whose outer layer discloses whether a particular keyword is stored in an inner encryption using a trapdoor. Unfortunately, search requires computation linear in the size of each document and reveals statistical information about the distribution of the underlying plaintext. Both of these where limitations were addressed by Goh [23] through associating secure indexes to each document in a collection. It also introduced the notion of semantic security against chosen-keyword attacks (called IND-CKA), which is the first formal notion of security defined for searchable encryption.

In the context of complex search queries, the above schemes are restricted to single-keyword equality queries. Ballard *et al.* [5] provided an secure and efficient system to perform Boolean keyword searches using Shamir's secret sharing. Curtmola et al. [20] introduced two variants (adaptively secure and non adaptively secure) SSE with the use of lookup tables. Chase et al. [17] introduced the notion of structured encryption, where arbitrarily structured data are encrypted in such a way that it can be queried through the use of a query specific token that can only be generated with knowledge of the secret key. The scheme improves over non-adaptive variant of [20] achieving keyword search through generating dictionaries of each keyword which contain pointer-output for each document. Kamara et al. [29] further refine the model to a dynamic searchable encryption scheme based on the inverted indexes approach of [20].

Other variants of SSE include Message Lock Encryption by Bellare et al. [8], where the key under which encryption and decryption are performed is derived from the message itself and search pattern obfuscation by Orencik at al. [34] using preprocessed term frequency-inverse document frequency (tf-idf) weights of keyword-document pairs.

#### 1.1.2 Public key encryption with keyword search (PEKS)

The notion of Public Key Encryption with Keyword Search was introduced by Boneh et al. [10] using bilinear maps and trapdoor permutations. The mechanism provided an efficient way to check whether a keyword is associated with a given document without leaking anything else about the document. However, due to the computation cost of public key encryption, the constructions were applicable to searching on a small number of keywords rather than an entire file. Moving beyond just equality-based keyword search, Park et al. [36] and Boneh et al. [10] extended PEKS for conjunctive [10,36], subset and range [10] queries on encrypted data.

However, the PEKS construction does not allow the recipient to decrypt keywords, i.e., encryption is not invertible. This was addressed by Fuhr et al. [22], through introducing decryptable searchable encryption using identity-based key encapsulation mechanism (ID-KEM). The concept also paved way for management of encrypted data, since the decryption key and the trapdoor derivation key are generated independently from one another and hence data can be decrypted by an entity and trapdoors be generated by some other managing party. Abdalla et al. [2] defined the computational and statistical relaxations of the existing notion of perfect consistency, showing that [10] is computationally consistent, and providing a new scheme that is statistically consistent. Third party delegation was further studied by Ibraimi et al. [25], employing the notion of Public Key Encryption with Delegated Search (PKEDS) which enables a third party to search an document for a particular keyword encrypted by the user.

Other variants of public key encryption in the context of keyword search include Deterministic Searchable Encryption [6] and Plaintext-Checkable Encryption [15]. Bellare et al. [6] achieved deterministic searchable encryption using RSA-DOAEP, a length preserving deterministic encryption scheme. A plaintext-checkable encryption scheme is a probabilistic public-key encryption scheme with the additional functionality that anyone can test whether a ciphertext is the encryption of a given plaintext message under a public encryption key. Canard et al. [15] provided an efficient construction for plaintext checkable encryption using an ElGamal-based approach.

### 1.2 Password-authenticated searchable encryption (PASE)

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 PASE, 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., [4,12–14,26–28,30,31,40]. The use of two servers provides the most practical scenario and the minimum requirement to achieve protection against offline dictionary attacks, while a more general secret sharing architecture with $t$-out-of-$n$ servers would be applicable as well. Chen *et al.* [18] further demonstrated the resilience of two server model against keyword guessing attacks. Thus, the PASE's two server model offers best performance to protection trade-off for (public key-based) PEKS schemes, protecting against offline dictionary and keyword guessing attacks.

We model PASE as a searchable encryption 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 realizes 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 PASE scheme using BPR-like models [3,9] that have been widely used for password-based protocols. We define privacy of PASE 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 direct PASE construction follows conceptually the following more general approach that combines ideas behind *Password Authenticated Secret Sharing (PASS)* [4,12–14,26,27,40] and SSE [5,20,34]. In the registration phase, the user picks a password $\pi$ and a high-entropy symmetric key $K$ that will be used to encrypt keywords and secret-shares $K$ protected with $\pi$ 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 PASE. Existing PASS protocols were proven in different security models, e.g., BPR-like in [4,40] and UC-based in [12,14,27,28] and do not necessarily follow the same functionality and syntax, which makes it hard to use PASS as a generic building block in PASE 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 PASE 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 PASE construction in this paper and opt for a direct and efficient scheme (cf. Sect. 3) based on well-known assumptions in the standard model.

### 1.3 Paper organization

Section 2 formally models PASE functionality and defines its main security properties. Section 3 introduces our direct PASE construction. We recall the underlying cryptographic building blocks and present a high-level design rationale for the scheme. This section also compares the efficiency of the key reconstruction phase of the proposed PASE scheme with existing PASS protocols and highlights additional support for multi-keyword operations and password change. Section 4 contains formal security analysis of the proposed scheme. In Sect. 5, we present our browser-based demonstrator with complete implementation of the proposed PASE functionality. This section also contains experimental results on the evaluation of performance and scalability of our implementation on commodity user devices. Section 6 concludes this paper.

## 2 PASE model and definitions

In this section, we model the functionality of PASE and provide definitions of its security requirements.

### 2.1 PASE functionality

#### 2.1.1 Syntax of algorithms and protocols

In our PASE model, any user $\mathtt{U}$ can perform an initial registration procedure with any two servers $\mathtt{S}_0$ and $\mathtt{S}_1$ in the system and then use the registered password $\pi$ (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; $C$ is used to represent a ciphertext for the keywords, whereas index $\mathtt{ix}$ stands for the outsourced (and possibly encrypted) document that is associated with the encrypted keywords. Similar to other searchable encryption schemes (e.g., [2]) we do not explicitly model the encryption of outsourced documents and use indices $\mathtt{ix} \in \boldsymbol{I}$ as placeholders for these documents.

– $\mathtt{Setup}(1^\kappa)$ is an initialization algorithm that on input a security parameter $\kappa \in \mathbb{N}$ generates public parameters $\mathtt{par}$ of the scheme.

– Register is a registration protocol executed between some user U (running interactive algorithm RegisterU) and two servers $S_0$ and $S_1$ (running interactive algorithms $\text{RegisterS}_d$, $d \in \{0, 1\}$) according to the following specification:
  – RegisterU(par, $\pi$, $S_0$, $S_1$): on input par and some password $\pi \leftarrow \mathcal{D}$, this algorithm interacts with $\text{RegisterS}_d$, $d \in \{0, 1\}$ and outputs a flag $s \in \{\text{succ}, \text{fail}\}$. If $(s = \text{succ})$, the user remembers $\pi$ and forgets all other informations.
  – $\text{RegisterS}_d$(par, U, $S_{1\text{-}d}$): on input par, this algorithm interacts with RegisterU (and possibly $\text{RegisterS}_{1\text{-}d}$) and at the end of successful interaction stores some secret information $\text{info}_d$ associated with U at $S_d$.
– Outsource is an outsourcing protocol executed between some user U (running interactive algorithm OutsourceU) and two servers $S_0$ and $S_1$ (running interactive algorithms $\text{OutsourceS}_d$, $d \in \{0, 1\}$) according to the following specification:
  – OutsourceU(par, $\pi$, $w$, ix, $S_0$, $S_1$): on input $\pi$, a keyword $w$, and some index ix this algorithms interacts with $\text{OutsourceS}_d$, $d \in \{0, 1\}$ and outputs a flag $s \in \{\text{succ}, \text{fail}\}$.
  – $\text{OutsourceS}_d$(par, U, $\text{info}_d$): on input $\text{info}_d$, this algorithm upon successful interaction with OutsourceU (and possibly $\text{OutsourceS}_{1\text{-}d}$) stores a record $(C, \text{ix})$ in its database $\boldsymbol{C}_d$.
– Retrieve is a retrieval protocol executed between some user U (running interactive algorithm RetrieveU) and two servers $S_0$ and $S_1$ (running interactive algorithms $\text{RetrieveS}_d$, $d \in \{0, 1\}$) according to the following specification:
  – RetrieveU(par, $\pi$, $w$, $S_0$, $S_1$): on input $\pi$ and a keyword $w$, this algorithm upon successful interaction with $\text{RetrieveS}_d$, $d \in \{0, 1\}$ outputs set $\boldsymbol{I}$ containing all ix associated with $w$.
  – $\text{RetrieveS}_d$(par, U, $\text{info}_d$): on input $\text{info}_d$, this algorithm interacts with RetrieveU (and possibly $\text{RetrieveS}_{1\text{-}d}$) and outputs a flag $s \in \{\text{succ}, \text{fail}\}$.

#### 2.1.2 Correctness

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

$\langle \text{succ}, \text{info}_0, \text{info}_1 \rangle \leftarrow \langle \text{RegisterU}(\text{par}, \pi, S_0, S_1), \text{RegisterS}_0(\text{par}, \text{U}, S_1), \text{RegisterS}_1(\text{par}, \text{U}, S_0) \rangle$;

$\langle \text{succ}, (C, \text{ix}), (C, \text{ix}) \rangle \leftarrow \langle \text{OutsourceU}(\text{par}, \pi, w, \text{ix}, S_0, S_1), \text{OutsourceS}_0(\text{par}, \text{U}, \text{info}_0), \text{OutsourceS}_1(\text{par}, \text{U}, \text{info}_1) \rangle$;

$\langle \boldsymbol{I}, \text{succ}, \text{succ} \rangle \leftarrow \langle \text{RetrieveU}(\text{par}, \pi, w, S_0, S_1), \text{RetrieveS}_0(\text{par}, \text{U}, \text{info}_0), \text{RetrieveS}_1(\text{par}, \text{U}, \text{info}_1) \rangle$;

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 $\pi$ in these outsourcing and retrieval protocol sessions.

### 2.2 PASE security model

The security of PASE is defined based on two main security goals: indistinguishability against chosen keyword attacks (IND-CKA) and authentication. We adopt a BPR-like modeling approach [9] for password-based cryptographic protocols and define security through experiments (cf. Fig. 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.

#### 2.2.1 Adversarial model and oracles

For each user U, we allow $\mathcal{A}$ to take full control over at most *one* of the two servers $S_0$ and $S_1$ that were chosen by U during the registration phase to capture the required distributed trust relationship. We mostly use $S_d$ to denote the uncorrupted server and $S_{1\text{-}d}$ to denote the server controlled by the adversary. The oracles allow $\mathcal{A}$ to invoke interactive algorithms for all protocols of PASE 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 $\boldsymbol{\tau}[j] \leftarrow (d, \pi, \text{info}_d)$ at the end of each successful $j$-th registration session such that $\pi$ is the registered password and $\text{info}_d$ is the secret data stored at the server $S_d$ at the end of that session. We also use variables $i^* \in \mathbb{Z}$, $\text{ix}^* \in \mathcal{I}$ and a set Set that are maintained by the experiments. The adversary $\mathcal{A}$ can access the following oracles.

– Challenge oracle $\text{Ch}_{\text{ind}}(b, \cdot, \cdot, \cdot, \cdot)$: on input $(i, w_0, w_1, \text{ix}^*)$, the oracle aborts if $((i^* \geq 0) \vee (i \geq j) \vee ((i, w_0) \in \text{Set}) \vee ((i, w_1) \in \text{Set}))$. Otherwise, it sets $i^* \leftarrow i$ and invokes oracle $\text{Out}(i^*, w_b, \text{ix}^*)$. Note that this oracle will be used to model IND-CKA security of PASE.

– Registration oracle $\texttt{Reg}(\cdot)$: 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 $\texttt{Register}$ protocol is executed with $\mathcal{A}$ where the oracle plays the roles of honest $\texttt{U}$ and $\texttt{S}_d$ executing algorithms $\texttt{RegisterU}(\texttt{par}, \pi, \texttt{S}_0, \texttt{S}_1)$ and $\texttt{RegisterS}_d(\texttt{par}, \texttt{U}, \texttt{S}_{1\text{-}d})$, respectively, and $\mathcal{A}$ plays the role of corrupted $\texttt{S}_{1\text{-}d}$. After interactions, the experiment records $\boldsymbol{\tau}[j] \leftarrow (d, \pi, \texttt{info}_d)$, delivers $j$ to the adversary and increases $j \leftarrow j+1$.
– Outsource oracle $\texttt{Out}(\cdot,\cdot,\cdot)$: on input $(i, w, \texttt{ix})$, the oracle aborts if $i \geq j$; or otherwise, it obtains $(d, \pi, \texttt{info}_d) \leftarrow \boldsymbol{\tau}[i]$. The $\texttt{Outsource}$ protocol is then executed with $\mathcal{A}$ where the oracle plays the roles of honest $\texttt{U}$ and $\texttt{S}_d$ executing algorithms $\texttt{OutsourceU}(\texttt{par}, \pi, w, \texttt{ix}, \texttt{S}_0, \texttt{S}_1)$ and $\texttt{OutsourceS}_d(\texttt{par}, \texttt{U}, \texttt{info}_d)$, respectively, and $\mathcal{A}$ plays the role of malicious $\texttt{S}_{1\text{-}d}$. In $\texttt{Auth}$ experiment, the oracle additionally computes $\texttt{Set} \leftarrow \texttt{Set} \cup (i, w, \texttt{ix})$.
– Outsource oracle (server only) $\texttt{OutS}(\cdot)$: on input $i$, the oracle aborts if $i \geq j$; otherwise, it obtains $(d, \pi, \texttt{info}_d) \leftarrow \boldsymbol{\tau}[i]$. The $\texttt{Outsource}$ protocol is then executed with $\mathcal{A}$ where the oracle plays the role of honest $\texttt{S}_d$ executing algorithm $\texttt{OutsourceS}_d(\texttt{par}, \texttt{U}, \texttt{info}_d)$ and $\mathcal{A}$ plays the roles of (illegitimate) $\texttt{U}$ and corrupted $\texttt{S}_{1\text{-}d}$. Note that this oracle will be used to model authentication of PASE.
– Retrieve oracle $\texttt{Ret}(\cdot,\cdot)$: on input $(i, w)$, the oracle aborts if $i \geq j$. In the $\texttt{IND-CKA}$ experiment, the oracle also aborts if $((i = i^*) \wedge (w \in \{w_0, w_1\}))$. Otherwise, it obtains the parameters $(d, \pi, \texttt{info}_d) \leftarrow \boldsymbol{\tau}[i]$. The $\texttt{Retrieve}$ protocol is then executed with $\mathcal{A}$ where the oracle plays the roles of honest $\texttt{U}$ and $\texttt{S}_d$ executing algorithms $\texttt{RetrieveU}(\texttt{par}, \pi, w, \texttt{S}_0, \texttt{S}_1)$ and $\texttt{RetrieveS}_d(\texttt{par}, \texttt{U}, \texttt{info}_d)$, respectively, and $\mathcal{A}$ plays the role of corrupted $\texttt{S}_{1\text{-}d}$. In the $\texttt{IND-CKA}$ experiment, if $(i^* = -1)$ the oracle additionally computes $\texttt{Set} \leftarrow \texttt{Set} \cup (i, w)$.
– Retrieve oracle (server only) $\texttt{RetS}(\cdot)$: on input $i$, the oracle aborts if $i \geq j$; otherwise, it obtains $(d, \pi, \texttt{info}_d) \leftarrow \boldsymbol{\tau}[i]$. The $\texttt{Retrieve}$ protocol is then executed with $\mathcal{A}$ where the oracle plays the role of honest $\texttt{S}_d$ executing algorithm $\texttt{RetrieveS}_d(\texttt{par}, \texttt{U}, \texttt{info}_d)$ and $\mathcal{A}$ plays the roles of (illegitimate) $\texttt{U}$ and corrupted $\texttt{S}_{1\text{-}d}$. Note that this oracle will be used to model $\texttt{IND-CKA}$-security of PASE.

#### 2.2.2 Indistinguishability against chosen keyword attacks (IND-CKA)

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

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

#### 2.2.3 Authentication (Auth)

The property of authentication for PASE is defined using experiment $Exp^{\texttt{Auth}}_{\texttt{PASE},\mathcal{A}}(\kappa)$ in Fig. 1. $\mathcal{A}$ is given the public parameters $\texttt{par}$ and permitted to access oracles $\texttt{Reg}(\cdot)$, $\texttt{Out}(\cdot,\cdot,\cdot)$, $\texttt{OutS}(\cdot)$ and $\texttt{Ret}(\cdot,\cdot)$ 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 $\texttt{ix}^*$ on behalf of some user $\texttt{U}$ without knowing the registered password (via $\texttt{OutS}(\cdot)$ oracle), possibly after having interacted with $\texttt{U}$ and the honest server $\texttt{S}_d$. In its attack on authentication, $\mathcal{A}$

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

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

**Fig. 1** PASE security experiments. The oracles are defined in Sect. 2.2

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

## 3 Our direct PASE construction

In this section, we propose a direct and efficient construction of PASE. It follows our general idea of combining suitable password-authenticated secret sharing with symmetric searchable encryption techniques. In the introduction, we explained the difficulties behind an attempt to construct PASE generically using PASS and SSE schemes and motivated our choice for a direct construction.

### 3.1 Cryptographic building blocks

In our PASE construction, we rely on a number of well-known cryptographic primitives that we briefly introduce in the following.

#### 3.1.1 Pedersen commitments [37]

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 $Adv^{\mathrm{DL}}_{\mathcal{A}}(\kappa)$ is negligible and provide perfect hiding.

#### 3.1.2 Pseudorandom function (`PRF`) [24,33]

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 $m$.

#### 3.1.3 Key derivation function (`KDF`) [32]

Let $\Sigma$ be a source of key material. A key derivation function `KDF` is called $(t, q, \epsilon(\kappa))$-secure with respect to $\Sigma$ 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 $\alpha$ is some side information. It is assumed that $\mathcal{A}$ knows $\alpha$, has control over the context information $c$ and has oracle access to $\mathtt{KDF}(k, \cdot)$ which cannot be queried on $c$.

#### 3.1.4 Message authentication code [7]

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

- $\mathtt{KGen}(\kappa)$: on input security parameter $\kappa$ 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 $\mu$ outputs 1 if $\mu$ 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 $\mu^*$ 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 $\mathcal{O}_{\mathtt{Tag}}(\cdot)$.

### 3.2 High-level design rationale

Our PASE protocol is inspired by the techniques used in the recent password-authenticated secret sharing protocol from [40] which we modified to address the functionality and requirements of PASE 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 $\pi$ encrypted in $C_\pi$ 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 $\pi$, and MAC codes $\mu_d$ 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 $\pi$ 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 $\mu_c$ which authenticates $(e, v)$ and also $\mathtt{ix}$ and which can only be computed and verified using $K$. During the $\mathtt{Retrieve}$ protocol, the user will ensure that it final search result contains only data that pass 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 emphasize that our PASE construction is in the password-only setting where servers are not required to possess any public keys for the security of the PASE scheme. However, if the registration protocol $\mathtt{Register}$ 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 $\mathtt{Outsource}$ and $\mathtt{Retrieve}$ protocols. While in practice this communication between the two servers will likely be protected using a secure channel (e.g., TLS), we stress that in our protocols this communication can take place over an insecure channel.

## 3.3 Detailed description

In the following, we provide a detailed description of all algorithms and protocols underlying our direct PASE scheme, along with Figs. 2 and 3 that illustrate the protocols $\mathtt{Outsource}$ and $\mathtt{Retrieve}$, respectively.

### 3.3.1 Initialization procedure $\mathtt{Setup}(1^\kappa)$

The algorithm generates public parameters $\mathtt{par}$ containing $\{\mathbb{G}, q, g, h, \mathtt{KDF}_1, \mathtt{KDF}_2, \mathtt{PRF}, \mathtt{MAC}\}$, where $(\mathbb{G}, q, g, h)$ represents a multiplicative cyclic group $\mathbb{G}$ with a prime order $q$ and generators $g, h \xleftarrow{\$} \mathbb{G}$ such that the discrete logarithm of $h$ with respect to base $g$ remains 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 $\mathbb{Z}_q^*$.

### 3.3.2 Registration protocol $\mathtt{Register}$

In order to register, a 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 each server $\mathtt{S}_d$, $d \in \{0,1\}$ over a server-authenticated secure channel. Finally, $\mathtt{U}$ memorizes $\pi$.

### 3.3.3 Outsourcing protocol $\mathtt{Outsource}$

The $\mathtt{Outsource}$ protocol between the user $\mathtt{U}$ and each server $\mathtt{S}_d$, $d \in \{0,1\}$ is illustrated in Fig. 2, and its steps are detailed in the following. Note that as part of the $\mathtt{Outsource}$ 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 $\mathtt{c}_{1\text{-}d}$.
   (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 $(Y, Z_d, \mu_d)$ 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 $(Y, Z_0, \mu_0)$ and $(Y, Z_1, \mu_1)$ 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.

### 3.3.4 Retrieval protocol $\mathtt{Retrieve}$

The $\mathtt{Retrieve}$ protocol between the user $\mathtt{U}$ and each server $\mathtt{S}_d, d \in \{0,1\}$ is illustrated in Fig. 3, and its steps are detailed in the following. Note that as part of the $\mathtt{Retrieve}$ 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.

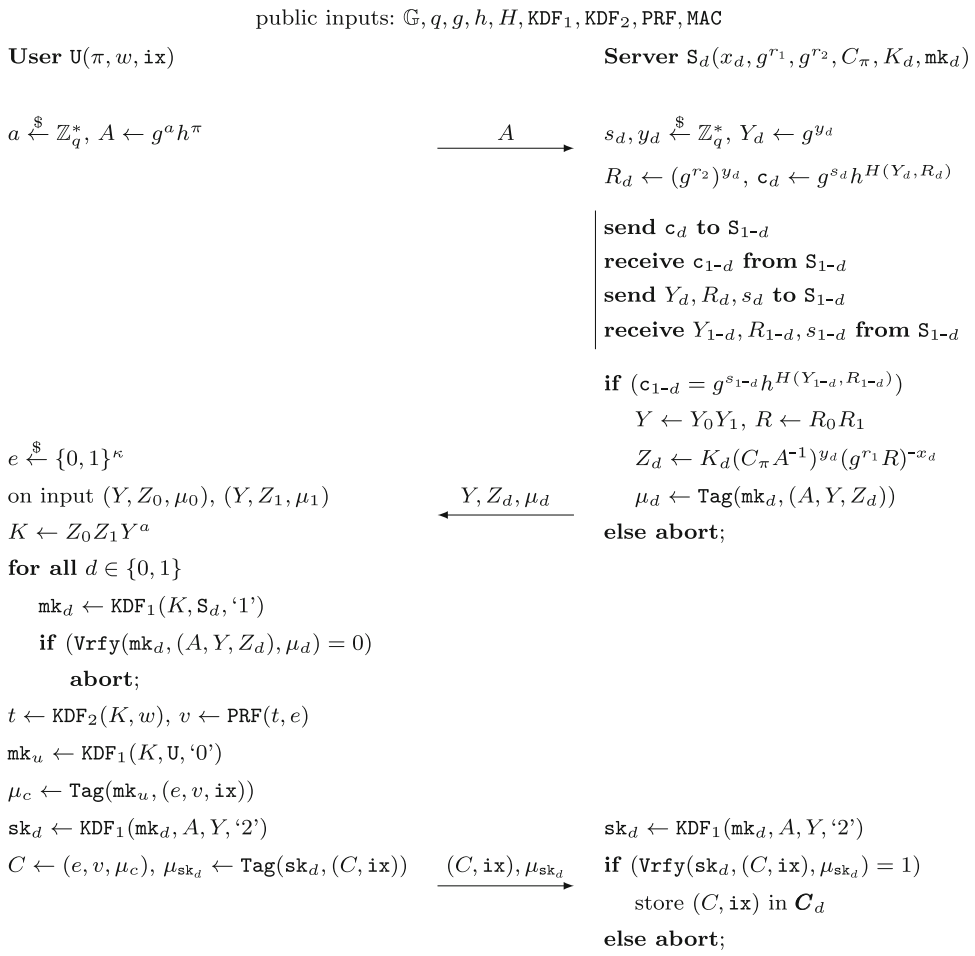


**Fig. 2** The `Outsource` protocol between user `U` and server $\mathtt{S}_d$. The server-side algorithm includes communication between servers $\mathtt{S}_d$ and $\mathtt{S}_{1\text{-}d}$

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 $(Y, Z_d, \mu_d)$ to `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 $(Y, Z_0, \mu_0)$ and $(Y, Z_1, \mu_1)$ from both servers, `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 `U`.
5. Upon receiving $\boldsymbol{A}_0$ and $\boldsymbol{A}_1$, user `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 $\boldsymbol{I}$.

#### 3.3.5 Correctness of our PASE scheme

In the following, we illustrate that if the initially registered password $\pi$ is used by the user in the executions of the `Outsource` and `Retrieve` protocols then computing $Z_0 Z_1 Y^a$ in the key reconstruction phase results in the original key $K$:

$$Z_0 Z_1 Y^a = K_0 (C_\pi A^{-1})^{y_0} (g^{r_1} R)^{-x_0}.$$

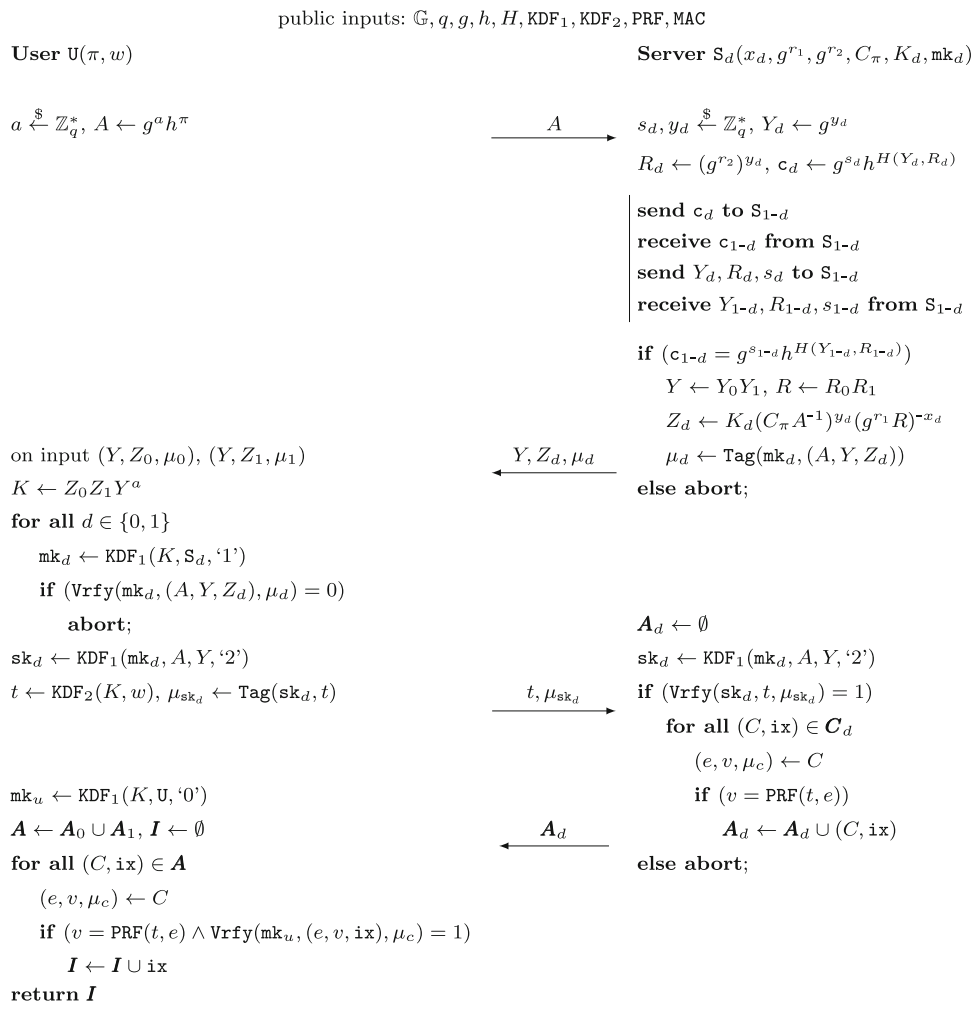


**Fig. 3** The `Retrieve` protocol between $\mathtt{U}$ and $\mathtt{S}_d$. The server-side algorithm includes communication between servers $\mathtt{S}_d$ and $\mathtt{S}_{1\text{-}d}$

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

## 3.4 Efficiency analysis and improvements

### 3.4.1 Efficiency comparison with existing PASS protocols

Given that our direct PASE construction follows the general idea of building PASE protocols based on the techniques used for password-authenticated secret sharing, we compare performance with existing PASS protocols. Since our PASE scheme assumes password-only setting (except for the registration), we restrict our comparison with password-only PASS schemes [4,26–28,40] and compare only the costs that arise from the sharing and retrieval of the symmetric key $K$—note that in our PASE 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 3a) of these protocols. Since our PASE 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 1.

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 PASE scheme compare fairly well with those of existing PASS protocols. In particular, only [27,28] which are the most computationally efficient PASS protocol today offers better overall computation and communication performance. We stress, however, that for PASE protocols the efficiency

**Table 1** Efficiency comparison of our PASE key reconstruction phase with password-only PASS schemes

| | Computation ($exp$) | | | | Communication (bits) | | | Rounds | |
|---|---|---|---|---|---|---|---|---|---|
| | Sharing | | Retrieval | | Sharing | Retrieval | | Sharing | Retrieval |
| | u | s | u | s | u-s | u-s | s-s | | |
| BJSL11 [4] | 6 | 0 | 33 | 16 | $24q + 4\kappa$ | $22q + 2\kappa$ | 0 | 1 | 2 |
| JKK14 [26] | 4 | 1 | 11 | 4 | $8q + 4\kappa$ | $8q + 4\kappa$ | 0 | 1 | 1 |
| YCHL15 [40] | 1 | 0 | 7 | 12 | $6\kappa$ | $10q$ | $8q$ | 1 | 1 |
| JKKX16 [27] | 4 | 1 | 4 | 1 | $8q + 4\kappa$ | $8q + 4\kappa$ | 0 | 2 | 1 |
| JKKX17 [28] | 2 | 1 | 2 | 1 | $2q + 2\kappa$ | $4q + 2\kappa$ | 0 | 2 | 1 |
| In our PASE | 6 | 0 | 3 | 8 | $8q + 2\kappa$ | $6q$ | $6q + 2\kappa$ | 1 | 1 |

The computation costs are measured in modular exponentiations; the communications costs are measured in bits. Both of these costs are provided separately for the user (u) and each server (s)

of the retrieval phase is of greater importance than of the sharing phase. This is because in PASE 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) PASE 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 with [27,28] our PASE 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 PASE scheme is sufficiently practical since the additional costs arising from the encrypted keyword search functionality within our PASE protocols are negligible (due to the nature of computations involved) in comparison with the costly key sharing and retrieval steps.

### 3.4.2 PASE with sublinear search complexities

Our PASE construction supports all CRUD operations required for a database, but its search complexity is $\mathcal{O}(D)$ for a database $DB$ of size $D$ whilst state-of-the-art schemes achieve a better bound of $\mathcal{O}(\log D)$. The search complexity of our PASE can be decreased using the techniques from [16,19,38], yet at the cost of some security and/or functionality limitations. For instance, using the techniques from [16,19] would require limiting PASE functionality to static databases loosing dynamic updates. The latter can be preserved with an ORAM but at a higher cost of $\mathcal{O}(D \log D)$ for periodic oblivious sorting [38].

The state-of-the-art approach in [19] uses dynamic databases with limited updates. We adopt it here because of the best trade-off between efficiency and functionality. Currently, within each `Outsource` round we outsource one keyword $w$ associated with some document `ix`. Using [19], within each `Outsource` round we can outsource a batch of documents by treating them as a static database $DB$. The optimization is achieved by constructing a look-up table $T$ in the setup phase which holds pointers to locations of the documents in $DB$ such that the table inputs depend on the document keyword $w$. This restricts the functionality to dynamic databases that allow only addition of documents but not their removal as the latter would require an update of the look-up table resulting in worse than linear complexity.

In order to extend the `Outsource` protocol (cf. Fig. 2) to outsource a database $DB = \{(w_i, \mathtt{ix}_i) | 1 \leq i \leq N\}$, for all unique keywords $w_i$ in $DB$, we compose a list $L_i = \{(w_i, \mathtt{ind}(\mathtt{ix}_{i_j}), \mathtt{ix}_{i_j}) | (w_i, \mathtt{ix}_{i_j}) \in DB, 1 \leq j \leq n_i\}$ where $\mathtt{ind}(\mathtt{ix}_{i_j})$ is the index of the file in $DB$ and $|L_i| = n_i$. The protocol is then executed for each tuple $(w_i, \mathtt{ind}(\mathtt{ix}_{i_j}), \mathtt{ix}_{i_j})$ followed by the computation of an additional key $o_{i_j} \leftarrow \mathtt{KDF}_2(t_i, j)$ and the look-up table entry $T[o_{i_j}] := \mathtt{ind}(\mathtt{ix}_{i_j})$. Once these values are calculated for all entries in $DB$, the entire encrypted database $\boldsymbol{C}_d$ is sent along with the look-up table $T$ to each server. Notice that $\boldsymbol{C}_d$ preserves the same order of elements from $DB$, and $\mathtt{ind}(\cdot)$ should give the same location for both $\boldsymbol{C}_d$ and $DB$. The `Retrieve` protocol is performed in the same way (cf. Fig. 3), except each server receives $(t, \mu_{\mathtt{sk}_d}, \{o_{i_j}\}_{j=1}^{n_i})$ for $|L_i| = n_i$. Servers use keys $\{o_{i_j}\}_{j=1}^{n_i}$ to identify entries $\{C_i, \mathtt{ix}_{i_j})\}_{j=1}^{n_i}$ in $\boldsymbol{C}_d$ based on $\mathtt{ind}(\mathtt{ix}_{i_j})$ from the look-up table $T$. To stop adversaries from trivially differentiating based on the list size, [19] extends $DB$ to $DB^*$ with dummy documents, such that all lists have the same size $|L_i| = n$, for $n = max_i\{n_i\}$. The resulting version of PASE would achieve the lower search bound of $\mathcal{O}(\log D)$ but have the aforementioned limitation on the removal of documents. It intuitively satisfies the same security guarantees as the original version based on the fact that each outsource operation can be seen as an outsource of a new independent static database.

### 3.5 Extensions with multiple keywords

In the given specification of our PASE 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 PASE 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 `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 `ix` with $\boldsymbol{w}$ through a single session of the `Outsource` protocol and (ii) search for all suitable documents `ix` using $\boldsymbol{w}'$ through a single session of the `Retrieve` protocol, based on three different types of search queries [11]: conjunctive queries ($\boldsymbol{w} = \boldsymbol{w}'$), disjunctive queries ($|\boldsymbol{w} \cap \boldsymbol{w}'| > 0$), and those for a subset of keywords ($\boldsymbol{w}' \subseteq \boldsymbol{w}$).

#### 3.5.1 Outsourcing documents with multiple keywords

In order to outsource some document `ix` associated with multiple keywords $\boldsymbol{w} = (w_1, \ldots, w_n)$, user `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.

#### 3.5.2 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 `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 $\boldsymbol{v}' \subseteq \boldsymbol{v}$.

### 3.6 Password change

Our PASE scheme allows users to change their passwords without changing the encryption keys $K$. The latter requirement is crucial since otherwise all outsourced keywords would need to be re-encrypted. In the following, we describe how a user can change current password $\pi$ to a new password $\pi^*$ depending on whether the user still knows $\pi$ or has forgotten it.

#### 3.6.1 Changing known passwords

A new password $\pi^*$ can be registered with the knowledge of the current $\pi$ as follows:

1. User `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 $\mu_d$, 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 $(g^{r_2}, C_\pi)$ in its $\mathtt{info}_d$ with $(g^{r_2 r^*}, C_{\pi^*})$.

Note that the current $\pi$ is used implicitly to authenticate the user toward both servers.

#### 3.6.2 Changing forgotten passwords

The above procedure cannot be executed if the user has forgotten her current password $\pi$. In this case, the user can no longer implicitly authenticate itself during the password change procedure. Since our PASE construction relies only on passwords, we naturally need to assume some alternative fall-back authentication mechanism (e.g., similar to those used on the web) that would be able to distinguish legitimate users from potential impersonators. We assume that a fall-back authentication mechanisms is in place which allows the user to independently set up secure channels with each of the two servers $\mathtt{S}_d$, $d \in \{0, 1\}$. The establishment of such channels still leaves us with a challenge to register a new password $\pi^*$ for that user without changing the previously registered encryption key $K$. We observe that upon the initial registration the encryption key $K$ satisfies the following equation $K_0 K_1 = X^{r_1} K$ where $X = g^{x_0 + x_1}$ and $(K_d, x_d)$ is known only to the corresponding $\mathtt{S}_d$. Moreover, the current password $\pi$ is encrypted in the ElGamal ciphertext $(g^{r_2}, C_\pi = X^{r_2} h^{\pi})$ stored on both servers. In the following password change protocol, this ElGamal ciphertext is replaced with $(g^{r_2^*}, C_{\pi^*} = X^{r_2^*} h^{\pi^*})$ for the new password $\pi^*$ such that the underlying base $X$ remains unchanged:

1. Each server $\mathtt{S}_d$, $d \in \{0, 1\}$, computes $X_d \leftarrow g^{x_d}$ using $x_d$ from $\mathtt{info}_d$ and sends $X_d$ to `U` over the previously established secure channel.

2. User U picks random $r_2^* \xleftarrow{\$} \mathbb{Z}_q^*$, computes $C_{\pi^*} \leftarrow (g^{x_0} g^{x_1})^{r_2^*} h^{\pi^*}$ and responds with $(g^{r_2^*}, C_{\pi^*})$ to $\mathtt{S}_d$.
3. Each server $\mathtt{S}_d$ replaces $(g^{r_2}, C_\pi)$ with $(g^{r_2^*}, C_{\pi^*})$ in its $\mathtt{info}_d$.

This password change protocol can be seen as a compressed version of the registration protocol. Jumping ahead of Sect. 4, we observe that the newly registered password $\pi^*$ remains protected against an adversary who can compromise at most one of the two servers under the same assumptions as the old password $\pi$.

## 4 Security analysis

In the following, we prove the security of our direct PASE scheme using our definitions from Sect. 2.2. In the proofs, we adopt the standard game-hopping technique. Let $\mathtt{succ}_n$ denote the event that the adversary wins in the experiment $n$.

### 4.1 IND-CKA-security of our PASE scheme

**Theorem 1** *Our direct PASE 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 $Exp^{\mathtt{IND\text{-}CKA}\text{-}b}_{\mathtt{PASE},\mathcal{A}}(\kappa)$. 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 initializes an empty database $\boldsymbol{C}_{d,j}$. The simulator and $\mathcal{A}$ complete the $\mathtt{Register}$ protocol, where the simulator plays the roles of 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 U and $\mathtt{S}_d$ and interacts with $\mathcal{A}$ who plays the role of $\mathtt{S}_{1\text{-}d}$ in the $\mathtt{Outsource}$ 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 U and $\mathtt{S}_d$ and interacts with $\mathcal{A}$ who plays the role of $\mathtt{S}_{1\text{-}d}$ party in the $\mathtt{Retrieve}$ 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 $\mathtt{RetrieveS}_d(\mathtt{par}, \mathtt{U}, \mathtt{info}_d)$.

□

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

**Experiment** $Exp_1^{\mathtt{IND}}$. This experiment is similar to $Exp_1^{\mathtt{IND}}$ 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** $\Pr[\mathtt{succ}_1^{\mathtt{IND}}] = \Pr[\mathtt{succ}_1^{\mathtt{IND}}]$

**Experiment** $Exp_2^{\mathtt{IND}}$. This experiment is similar to $Exp_2^{\mathtt{IND}}$ 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** $|\Pr[\mathtt{succ}_2^{\mathtt{IND}}] - \Pr[\mathtt{succ}_2^{\mathtt{IND}}]| \leq Adv^{\mathtt{DL}}_{\mathcal{A}}(\kappa)$

**Experiment** $Exp_3^{\mathtt{IND}}$. This experiment is similar to $Exp_3^{\mathtt{IND}}$ except that in oracles $\mathtt{Out}(\cdot, \cdot, \cdot)$, $\mathtt{Ret}(\cdot, \cdot)$ and $\mathtt{RetS}(\cdot)$, the message $(Z_d, \mu_d)$ from the honest server $\mathtt{S}_d$ to the user is replaced with $(E, \mu'_d)$ 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 $(g, g^\alpha, g^\beta, Q)$ be an instance of the $\mathtt{DDH}$ 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 $Exp_3^{\mathtt{IND}}$; otherwise, to $Exp_3^{\mathtt{IND}}$. The hardness of the DDH problem implies the indistinguishability of $Exp_3^{\mathtt{IND}}$ from $Exp_3^{\mathtt{IND}}$.

2. For oracle $\mathtt{RetS}(\cdot)$, assume $\pi'$ is the password tried by $\mathcal{A}$, the key $K$ (in $Exp_3^{\mathtt{IND}}$) 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 $Exp_3^{\mathtt{IND}}$) from a random number in $\mathbb{G}$ (in $Exp_3^{\mathtt{IND}}$) 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 4** $|\Pr[\mathtt{succ}_3^{\mathtt{IND}}] - \Pr[\mathtt{succ}_3^{\mathtt{IND}}]| \leq (q_s+1) Adv_{\mathcal{A}}^{\mathtt{DDH}}(\kappa) + \frac{q_s}{|\mathcal{D}|}$

**Experiment** $Exp_4^{\mathtt{IND}}$. This experiment is similar to $Exp_4^{\mathtt{IND}}$ 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 $Exp_4^{\mathtt{IND}}$. 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** $|\Pr[\mathtt{succ}_4^{\mathtt{IND}}] - \Pr[\mathtt{succ}_4^{\mathtt{IND}}]| \leq q_r \cdot Adv_{\mathcal{A}}^{\mathtt{KDF}}(\kappa)$

**Experiment** $Exp_5^{\mathtt{IND}}$. This experiment is similar to $Exp_5^{\mathtt{IND}}$ 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 $Exp_5^{\mathtt{IND}}$. $F_2$ 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** $|\Pr[\mathtt{succ}_5^{\mathtt{IND}}] - \Pr[\mathtt{succ}_5^{\mathtt{IND}}]| \leq (q_o+q_t) Adv_{\mathcal{A}}^{\mathtt{KDF}}(\kappa)$

**Experiment** $Exp_6^{\mathtt{IND}}$. This experiment is similar to $Exp_6^{\mathtt{IND}}$ 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** $|\Pr[\mathtt{succ}_6^{\mathtt{IND}}] - \Pr[\mathtt{succ}_6^{\mathtt{IND}}]| \leq (q_o + q_t + q_s) Adv_{\mathcal{A}}^{\mathtt{MAC}}(\kappa)$

**Experiment** $Exp_7^{\mathtt{IND}}$. This experiment is similar to $Exp_7^{\mathtt{IND}}$ 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 PASE construction is IND-CKA-secure assuming the intractability of the DDH problem and security of $\mathtt{KDF}_1$, $\mathtt{KDF}_2$, PRF and MAC.

## 4.2 Authentication property of our PASE scheme

**Theorem 2** *Our proposed PASE 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 $Exp_{\mathtt{PASE},\mathcal{A}}^{\mathtt{Auth}}(\kappa)$. 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 $\mathtt{Register}$ 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 $\mathtt{U}$ and $\mathtt{S}_d$ and interacts with $\mathcal{A}$ who plays the role of $\mathtt{S}_{1\text{-}d}$ party in the $\mathtt{Outsource}$ 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 $\mathtt{U}$ and $\mathtt{S}_d$ and interacts with $\mathcal{A}$ who plays the role of $\mathtt{S}_{1\text{-}d}$ in the $\mathtt{Retrieve}$ protocol. □

**Lemma 9** $Adv^{\mathtt{Auth}}_{\mathtt{PASE},\mathcal{A}}(\kappa) = \Pr[\mathtt{succ}^{\mathtt{Auth}}_0]$

**Experiment** $Exp^{\mathtt{Auth}}_1$. This experiment is similar to $Exp^{\mathtt{Auth}}_0$ 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** $\Pr[\mathtt{succ}^{\mathtt{Auth}}_0] = \Pr[\mathtt{succ}^{\mathtt{Auth}}_1]$

**Experiment** $Exp^{\mathtt{Auth}}_2$. This experiment is similar to $Exp^{\mathtt{Auth}}_1$ 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** $|\Pr[\mathtt{succ}^{\mathtt{Auth}}_1] - \Pr[\mathtt{succ}^{\mathtt{Auth}}_2]| \leq Adv^{\mathtt{DL}}_{\mathcal{A}}(\kappa)$

**Experiment** $Exp^{\mathtt{Auth}}_3$. This experiment is similar to $Exp^{\mathtt{Auth}}_2$ except that in oracles $\mathtt{Out}(\cdot,\cdot,\cdot)$, $\mathtt{Ret}(\cdot,\cdot)$ and $\mathtt{OutS}(\cdot)$, the message $(Z_d, \mu_d)$ from the honest server $\mathtt{S}_d$ to the user is replaced with $(E, \mu'_d)$ 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

   $$Z_d \leftarrow K_d (g^\beta)^{r_2(x_0+x_1)} Q^{-1} (g^{r_1} \cdot (g^\beta)^{r_2} \cdot R_{1\text{-}d})^{-x_d}$$

   If $Q = g^{\alpha\beta}$, then this experiment is identical to $Exp^{\mathtt{Auth}}_2$; otherwise, it is identical to $Exp^{\mathtt{Auth}}_3$. The hardness of the DDH problem directly implies the indistinguishability of $Exp^{\mathtt{Auth}}_2$ from $Exp^{\mathtt{Auth}}_3$.
2. For the oracle $\mathtt{OutS}(\cdot)$, assume $\pi'$ is a password used by the adversary, the key $K$ (in $Exp^{\mathtt{Auth}}_2$) 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 $Exp^{\mathtt{Auth}}_2$) from a random number in $\mathbb{G}$ (in $Exp^{\mathtt{Auth}}_3$) 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 12** $|\Pr[\mathtt{succ}^{\mathtt{Auth}}_2] - \Pr[\mathtt{succ}^{\mathtt{Auth}}_3]| \leq (q_s + 1)Adv^{\mathtt{DDH}}_{\mathcal{A}}(\kappa) + \frac{q_s}{|\mathcal{D}|}$

**Experiment** $Exp^{\mathtt{Auth}}_4$. This experiment is similar to $Exp^{\mathtt{Auth}}_3$ 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 $Exp^{\mathtt{Auth}}_4$. 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 $\mathtt{mk}$; 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** $|\Pr[\mathtt{succ}^{\mathtt{Auth}}_3] - \Pr[\mathtt{succ}^{\mathtt{Auth}}_4]| \leq q_r \cdot Adv^{\mathtt{KDF}}_{\mathcal{A}}(\kappa)$

**Experiment** $Exp^{\mathtt{Auth}}_5$. This experiment is similar to $Exp^{\mathtt{Auth}}_4$ 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 $Exp^{\mathtt{Auth}}_5$. 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** $|\Pr[\mathtt{succ}^{\mathtt{Auth}}_4] - \Pr[\mathtt{succ}^{\mathtt{Auth}}_5]| \leq (q_o + q_t)Adv^{\mathtt{KDF}}_{\mathcal{A}}(\kappa)$

We observe that $Exp^{\mathtt{Auth}}_5$ is simulated independent the key $K$. The only probability of winning $Exp^{\mathtt{Auth}}_5$ comes from the adversary successfully forging $\mu_c$ 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** $\Pr[\mathtt{succ}^{\mathtt{Auth}}_5] = Adv^{\mathtt{MAC}}_{\mathcal{A}}(\kappa)$

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

## 5 PASE in practice: browser-based implementation and performance evaluation

In order to demonstrate the functionality of our PASE scheme, we implemented a stateful web application that can be accessed from any web or mobile browser. Our PASE demonstrator implements the client and server sides of the

protocol and comes with a single portal (cf. Fig. 4) through which users can register, outsource/retrieve files based on multiple keywords and change their passwords. The source code is available from https://github.com/Spockuto/surrey-paks.

The entire PASE implementation is written in Javascript with the client side backed by browser's V8 engine[1] and the server side backed by NodeJS server[2]. By choosing JavaScript, we could use Stanford JavaScript Crypto library[3] in the implementation of both sides (client and server) whereby reusing some parts of the code. An alternative would be to use libsodium[4] or OpenSSL with a wrapper based on PHP. Since modern applications heavily adopt JavaScript, our implementation can in turn be used as a library to provide support for other applications that wish to use the functionality of PASE.

In the following, we provide a more detailed description of our PASE demonstrator and evaluate performance of its functionality.

### 5.1 Cryptographic implementation choices

The following choices of cryptographic parameters and algorithms have been made for our implementation. For the cyclic group $\mathbb{G}$ of prime order $q$ and its generator $g$, we use the parameters of the NISTP384 elliptic curve group[5]. The additional generator $h$ is chosen at random. For the hash function $H$, we adopt SHA256 (256 bits). Both key derivation functions $\mathtt{KDF}_1$ and $\mathtt{KDF}_2$ are implemented as PBKDF2 (256 bits)[6]. Although PBKDF2 might not be the most efficient on mobile devices, better alternatives such as ARGON2[7] have not been adopted yet in major cryptographic libraries. Our pseudorandom function `PRF` uses AES256 in GCM mode with the output truncated to 256 bits[8]. For the message authentication code `MAC`, we adopt the standard HMAC[9] construction.

### 5.2 PASE client and servers

The JavaScript code running on the client includes the `main.js` file (about 800 LoCs) to manage the requests and formatting and the `crypto.js` file (300 LoCs) to execute the protocols on the client side. The code running on each server is split into multiple files with the `protocol.js` (400 LoC) occupying the major part of the implementation. The demonstrator requires NodeJS environment to run and can be deployed instantly.

In our implementation, one server acts as a primary server in that it serves the PASE website and is also used to store all outsourced files. It is helped by the secondary server during the registration, outsourcing and retrieval protocols. The database adopted in our PASE implementation is MongoDB[10], which is particularly suited for storing and retrieving files. The MongoDB is also used to store all user related information from the registration process. Each server runs its own instance of the database.

Our browser-based demonstrator offers a registration interface where a user can provide its username (e.g., email address) and a chosen password to execute the registration protocol with both servers. Once registered, the user can outsource files and associate them with multiple keywords. Similarly, the user can retrieve outsourced files based on the keywords entered into the corresponding form. Note that for outsourcing and retrieval, the login form must contain the registered username and password. Our demonstrator supports outsourcing and retrieval using multiple keywords, which can be entered into the corresponding box separated by commas. If multiple keywords are used in the retrieval protocol, then the output produced currently will be based on the logic used to define subset queries (cf. Sect. 3.5).

#### 5.2.1 Communication

In our PASE protocols, there are two types of communication which are implemented using different techniques as discussed in the following:

- The client-server communication which is present in the registration, outsourcing and retrieval protocols is realized in our implementation using AJAX queries which are executed asynchronously to provide better functionality. This is only possible if the server accepts Cross Origin Resourse Sharing which can be easily setup through the NodeJS core library.
- The server-server communication in the outsourcing and retrieval protocols requires both serves to maintain a shortlived state information for the two communication rounds of the protocol session (cf. Fig. 2). This is realized in our implementation using NodeCache[11] functionality which provides a simple and fast internal caching for NodeJS servers.

[1] https://developers.google.com/v8/.

[2] https://nodejs.org/.

[3] https://crypto.stanford.edu/sjcl/.

[4] https://github.com/jedisct1/libsodium.

[5] http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf.

[6] https://www.ietf.org/rfc/rfc2898.txt.

[7] https://password-hashing.net/argon2-specs.pdf.

[8] https://csrc.nist.gov/publications/detail/sp/800-38d/final.

[9] https://tools.ietf.org/rfc/rfc2104.txt.

[10] https://www.mongodb.com/.

[11] https://www.npmjs.com/package/node-cache.

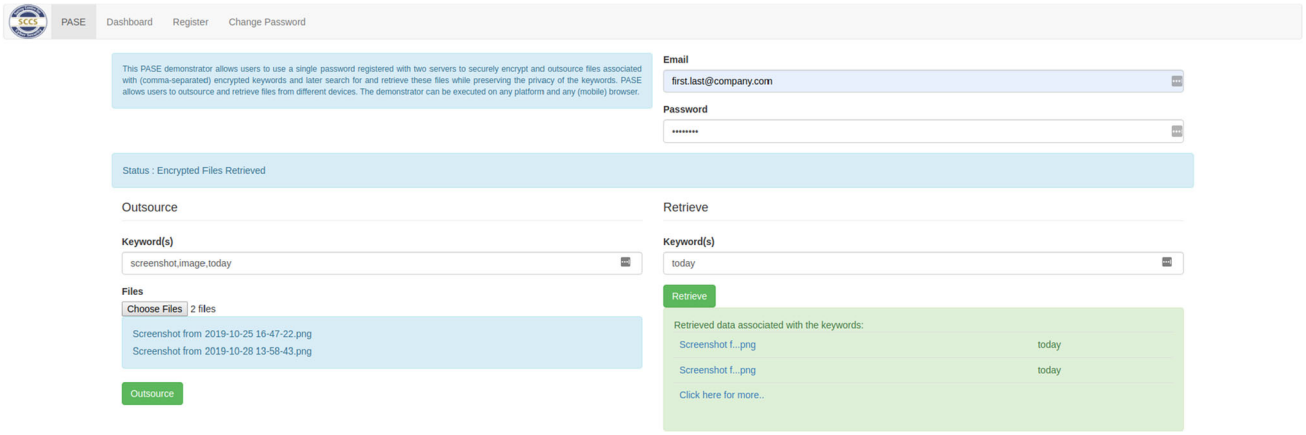


**Fig. 4** PASE portal including `Outsource` and `Retrieve` forms

## 5.3 Encryption of outsourced files

In our demonstrator, we expand the implemented PASE functionality with the encryption of outsourced files, in addition to encrypted keywords. For this purpose, the client could use the secret-shared key $K$ which it reconstructs on the client side during the execution of the outsourcing and retrieval protocols. More precisely, the client could use $K$ to derive another key and use it with some standard symmetric encryption scheme, e.g., AES, to encrypt outsourced files and decrypt them upon retrieval. In addition, to minimize information leakage and distribute the encrypted data among the two servers, we XOR the encrypted data `ENC` with a random stream of data `RND` generated to the length of the encrypted data using Fortuna[12] PRNG. The resulting files $\mathtt{F}_0 \leftarrow \mathtt{ENC} \oplus \mathtt{RND}$ and $\mathtt{F}_1 \leftarrow \mathtt{RND}$ are sent to the respective servers $\mathtt{S}_0$ and $\mathtt{S}_1$. The data `ix` (cf. Fig. 2), in this case, would be a concatenation of the encrypted file name with the random IV generated for symmertic encryption($ix \leftarrow \mathtt{Enc(Name)||IV}$). The encrypted file name acts as the encrypted file identifier in the server for querying. During `Retrieve`, from `ix`, the file name is decrypted using the reconstructed key $K$ and `IV`. The encrypted file name is used to retrieve files $\mathtt{F}_0$ and $\mathtt{F}_1$ from servers $\mathtt{S}_0$ and $\mathtt{S}_1$, respectively. The file is recovered by $\mathtt{Dec}(\mathtt{F}_0 \oplus \mathtt{F}_1)$ and made available to the user.

## 5.4 Evaluation of performance and scalability

### 5.4.1 Performance of PASE

Our performance evaluation focuses on the computational overheads of the PASE scheme and does not consider the varying network latency. All experiments were performed on a MacBook Pro laptop with 2.2GHz Intel Core i7 and 16GB RAM (server and client instances) and OnePlus 5 smartphone with Qualcomm Snapdragon 835 octa-core 2.45GHz and 8GB RAM (client instance).

The results of our measurements are summarized in Table 2 with separate timings provided for the client and server side computations. For the client side, the table contains measurements performed on both the laptop and smartphone. The registration procedure includes all steps of the `Register` protocol. In the table, the time needed to reconstruct the symmetric key $K$, which is accomplished in step 3a of our PASE `Outsource` and `Retrieve` protocols (cf. Sect. 3), is measured separately from the time needed to outsource keywords (steps 3b and 3c of `Outsource`) and retrieve files (steps 3b and 5 of `Retrieve`). We observe that key reconstruction time on the client side is more than twice as fast as on the server side (when measured on the same device). Note that the key reconstruction procedure is identical in both protocols and its time is independent of the used keywords. In constrast, the measurements provided for outsourcing and retrieval procedures in Table 2 cover only keyword-dependent steps. Table 2 provides average timings based on one keyword, which are computed from multiple executions of the protocol involving a set of 100 randomly generated keywords, with each keyword being between 5 and 10 characters long. For each execution, a random keyword was chosen from the set and the resulting average was computed over 1000 executions.

Based on the measurements, we can highlight that our PASE registration procedure remains well under 1s on both the laptop and the smartphone. The time for outsourcing and retrieval is clearly dominated by the time needed to reconstruct $K$, which also remains well under 1s. The keyword-dependent computations in both protocols are very efficient, taking less than 100ms per keyword. On the client side, the outsourcing procedure is slightly more efficient than

[12] https://www.schneier.com/academic/fortuna/.

**Table 2** Performance evaluation of our PASE implementation

| Procedure | Client | | Each server |
|---|---|---|---|
| | MacBook Pro 2.2GHz Intel Core i7 | OnePlus 5 Snapdragon 835 2.45GHz | MacBook Pro 2.2GHz Intel Core i7 |
| Registration | 180ms | 360ms | 15ms |
| Reconstruction of $K$ | 100ms | 195ms | 229ms |
| Outsourcing | 23ms | 68ms | 22ms |
| Retrieval | 26ms | 73ms | 28ms |

the retrieval procedure due to the additional integrity checks performed in step 5 of the `Retrieve` protocol.

### 5.4.2 Performance of file encryption

We evaluated the performance of our file encryption scheme by using test files of size 100 KB, 1 MB and 10 MB on our client instances (MacBook Pro and OnePlus 5). We limited the upper bound of file size to 10MB because of JavaScript's bottleneck on the encryption size. Each encryption round includes the time taken to encrypt the file using AES, generate the random stream of data and perform the XOR operation (cf. Sect. 5.3). On laptop, the encryption scheme during `Outsource` averaged at 15ms for 100 KB, 64 ms for 1 MB and 476 ms for 10 MB files. During the execution, the random stream generation accounted for 8ms for 100 KB, 52 ms for 1 MB and 380 ms for 10MB files. Decrypting the file in `Retrieve` took 15 ms for 100KB, 45 ms for 1MB and 270ms for 10MB files. On mobile, the encryption scheme during `Outsource` averaged at 25 ms for 100 KB, 105 ms for 1 MB and 540 ms for 10 MB files. During the execution, the random stream generation accounted for 16 ms for 100 KB, 65 ms for 1 MB and 310 ms for 10 MB files. Decrypting the file in `Retrieve` took 90 ms for 100 KB, 300 ms for 1 MB and 3s for 10MB files.

Based on our measurements, we can highlight that an encryption scheme (e.g., AES) can be practically adapted into our protocol with less computation overhead. The total time taken for encryption and decryption, at large file sizes, is clearly dominated by random stream generation. Hence, we propose a configuration setting where the random stream generation is available as an optional security enhancement for the user to choose. With this configuration, users can leverage their computational flexibility to securely encrypt and distribute files of their choice.

### 5.4.3 Scalability of PASE

In addition to the measurements involving one keyword per execution, we are interested in the scalability of our PASE implementation. For this purpose, we have extended our measurements to calculate an average time for keyword-dependent outsourcing and retrieval computations with up to 30 keywords (which is for example, the maximum number of hashtags allowed per image on Instagram[13]). In our experiments, for each execution multiple keywords were randomly chosen from the same set of 100 keywords that were used in the experiments behind Table 2. A linear regression model was then applied to the average discrete timings to derive a linear approximation.

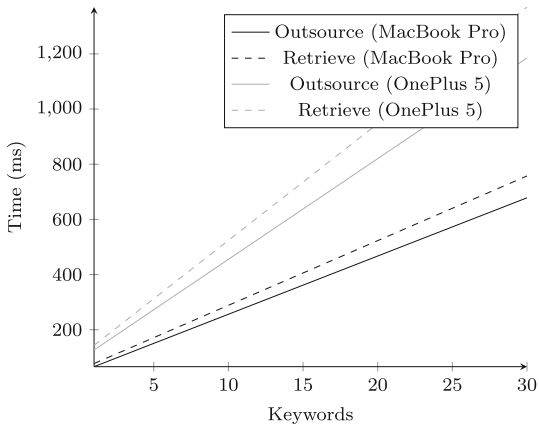


**Fig. 5** Scalability of keyword-dependent outsourcing and retrieval operations on the client side using MacBook Pro laptop and OnePlus 5 smartphone

Our experimental results for client-side keyword-dependent computations are plotted in Fig. 5. These timings suggest that our implementation remains scalable on commodity user devices such as laptops and smartphones. For example, client-side processing of 10 keywords in the outsourcing phase requires about 256 ms (laptop) and 455 ms (smartphone), whereas computations associated with a subset query of 10 keywords during the retrieval phase require about 289 ms (laptop) and 523 ms (smartphone). If we add constant key reconstruction costs from Table 2, then the overall time for client-side processing of 10 keywords would be about 356 ms (laptop) and 879 ms (smartphone) in outsourcing and about 389 ms (laptop) and 947 ms (smartphone) in retrieval phases.

[13] https://blog.hootsuite.com/instagram-hashtags.

#### 5.4.4 Strengthening password-based authentication

The PASE protocol is proven sound by rigorous mathematical analysis, but the usage of password for authentication indirectly inherits several issues associated with passwords and acts as a single point of failure for the entire architecture. Moving beyond brute force and online attacks, passwords are vulnerable to re-usage, leakage and social engineering attacks. A study [39] on password usage states 38% reused the same password for two different online services, and 21% of them slightly modified an old one to sign up for a new service. Have I been pwned (HIBP)[14], a popular website which reports data breaches provides records over 500 million actual unique passwords leaked from various data breaches through a variety of attacks including credential stuffing and phishing. The study also shows that users with more passwords are more likely to reuse them, or use variations. The 2020 Verizon Data Breach Investigations Report (DBIR) [1] reports over 80% of breaches within hacking involve brute force or the use of lost or stolen credentials. To protect against such password weakness, the PASE protocol can be extended modularly with 2FA, a secondary authentication mechanism which provides a one-time password (OTP) or code generated or received by an authenticator (e.g., a security token or smartphone) that only the user possesses to complement the primary password used for authentication. The PASE scheme allows inclusion of additional complimentary authentication scheme without comprising the integrity of the internal PASE protocol which relies on high entropy keys generated from the primary password.

## 6 Conclusion

Password-Authenticated Searchable Encryption (PASE) introduced in this paper is a new concept for searchable encryption where the search over encrypted keywords can be performed solely with the help of a human-memorizable password. The main advantage over previous concepts is a simplified key management which removes the need for storing and managing high-entropy keys on the user side and makes the whole process device-agnostic. Basing searchable encryption on passwords introduces major design challenges; in particular, creating the need for a distributed server architecture to achieve security against offline dictionary attacks.

We modeled the functionality and security properties of PASE, incl. `IND-CKA`-security for keyword privacy and authentication for outsourcing for the search procedure and proposed a direct PASE construction those security and privacy has been proven under standard assumptions. Our direct PASE construction is an optimized version of a more general concept for building PASE protocols based on techniques underlying password-authenticated secret sharing and symmetric searchable encryption.

We evaluated the practicality of our PASE scheme through implementation of a JavaScript-based web application that can readily be executed on any (mobile) browser. The conducted performance and scalability evaluation of our implementation shows that the proposed PASE approach remains practical on commodity user devices such as laptops and smartphones.

[14] https://haveibeenpwned.com/Passwords.

## Compliance with ethical standards

**Conflict of interest** All authors declare that they have no conflict of interest.

**Ethical approval** This article does not contain any studies with human participants performed by any of the authors.



## References

1. 2020 data breach investigations report, https://enterprise.verizon.com/en-gb/resources/reports/dbir/. Accessed 31 Aug 2020
2. Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., Shi, H.: Searchable encryption revisited: consistency properties, relation to anonymous IBE, and Extensions. J. Cryptol. **21**(3), 350–391 (2008)
3. Abdalla, M., Fouque, P., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) PKC'05. LNCS, vol. 3386, pp. 65–84. Springer, Berlin (2005)
4. Bagherzandi, A., Jarecki, S., Saxena, N., Lu, Y.: Password-protected secret sharing. In: CCS'11. ACM, pp. 433–444 (2011)
5. Ballard, L., Kamara, S., Monrose, F.: Achieving efficient conjunctive keyword searches over encrypted data. In: ICICS'05. LNCS, vol. 3783, Springer, pp. 414–426 (2005)
6. Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Menezes, A. (ed.) Advances in Cryptology—CRYPTO 2007, In: 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings. LNCS, vol. 4622, pp. 535–552. Springer (2007).https://doi.org/10.1007/978-3-540-74143-5_30
7. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: CRYPTO'96. LNCS, vol. 1109, Springer, pp. 1–15 (1996)

8. Bellare, M., Keelveedhi, S., Ristenpart, T.: Message-locked encryption and secure deduplication. In: Advances in Cryptology – EUROCRYPT 2013, pp. 296–312. Springer, Berlin, Heidelberg (2013), https://doi.org/10.1007/978-3-642-38348-9_18
9. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: EUROCRYPT'00. LNCS, vol. 1807, Springer, pp. 139–155 (2000)
10. Boneh, D., Crescenzo, G.D., Ostrovsky, R., Persiano, G.: Public Key Encryption with Keyword Search. In: EUROCRYPT'04. pp. 506–522 (2004)
11. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: TCC'07. LNCS, vol. 4392, Springer, pp. 535–554 (2007)
12. Camenisch, J., Enderlein, R.R., Neven, G.: Two-server password-authenticated secret sharing uc-secure against transient corruptions. In: PKC'15. LNCS, vol. 9020, Springer, pp. 283–307 (2015)
13. Camenisch, J., Lehmann, A., Lysyanskaya, A., Neven, G.: Memento: How to reconstruct your secrets from a single password in a hostile environment. In: CRYPTO'14. LNCS, vol. 8617, Springer, pp. 256–275 (2014)
14. Camenisch, J., Lysyanskaya, A., Neven, G.: Practical yet universally composable two-server password-authenticated secret sharing. In: CCS'12. pp. 525–536 (ACM, 2012)
15. Canard, S., Fuchsbauer, G., Gouget, A., Laguillaumie, F.: Plaintext-checkable encryption. In: Dunkelman, O. (ed.) Topics in Cryptology—CT-RSA 2012—The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27—March 2, 2012. Proceedings. LNCS, vol. 7178, pp. 332–348. Springer (2012), https://doi.org/10.1007/978-3-642-27954-6-21
16. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: Annual Cryptology Conference. Springer, pp. 353–373 (2013)
17. Chase, M., Kamara, S.: Structured encryption and controlled disclosure. In: Advances in Cryptology —ASIACRYPT 2010, pp. 577–594. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_33
18. Chen, R., Mu, Y., Yang, G., Guo, F., Wang, X.: Dual-server public-key encryption with keyword search for secure cloud storage. IEEE Trans. Inf. For. Sec. **11**(4), 789–798 (2016)
19. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. J. Comput. Secur. **19**(5), 895–934 (2011)
20. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. J. Comp. Secur. **19**(5), 895–934 (2011)
21. Dawn Xiaoding Song, Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Proceeding 2000 IEEE Symposium on Security and Privacy. S P 2000. pp. 44–55 (2000)
22. Fuhr, T., Paillier, P.: Decryptable searchable encryption. In: Provable Security, pp. 228–236. Springer, Berlin, Heidelberg, https://doi.org/10.1007/978-3-540-75670-5_17
23. Goh, E.J.: Secure indexes. Cryptology ePrint Archive, Report 2003/216 (2003), http://eprint.iacr.org/2003/216/. Accessed 31 Aug 2020
24. Håstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A pseudorandom generator from any one-way function. SIAM J. Comput. **28**(4), 1364–1396 (1999)
25. Ibraimi, L., Nikova, S., Hartel, P., Jonker, W.: Public-key encryption with delegated search. In: International Conference on Applied Cryptography and Network Security. Springer, pp. 532–549 (2011)
26. Jarecki, S., Kiayias, A., Krawczyk, H.: Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In: ASIACRYPT'14. LNCS, vol. 8874, Springer, pp. 233–253 (2014)
27. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In: EuroS&P'16. pp. 276–291. IEEE (2016)
28. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: TOPPSS: cost-minimal password-protected secret sharing based on threshold OPRF. IACR Cryptology ePrint Archive 2017, 363 (2017), http://eprint.iacr.org/2017/363. Accessed 31 Aug 2020
29. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: CCS'12. pp. 965–976 (ACM, 2012)
30. Kiefer, F., Manulis, M.: Blind Password Registration for Two-Server Password Authenticated Key Exchange and Secret Sharing Protocols. In: ISC'16. LNCS, vol. 9866, Springer, pp. 95–114 (2016)
31. Kiefer, F., Manulis, M.: Universally Composable Two-Server PAKE. In: ISC'16. LNCS, vol. 9866, Springer, pp. 147–166 (2016)
32. Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: CRYPTO'10. LNCS, vol. 6223, Springer, pp. 631–648 (2010)
33. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM J. Comput. **17**(2), 373–386 (1988)
34. Örencik, C., Selcuk, A., Savas, E., Kantarcioglu, M.: Multi-keyword search over encrypted data with scoring and search pattern obfuscation. Int. J. Inf. Sec. **15**(3), 251–269 (2016)
35. Ostrovsky, R.: Software protection and simulation on oblivious RAMs. Ph.D. thesis, Massachusetts Institute of Technology (1992)
36. Park, D.J., Kim, K., Lee, P.J.: Public key encryption with conjunctive field keyword search. In: WISA'04. LNCS, vol. 3325, Springer, pp. 73–86 (2004)
37. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO'91. LNCS, vol. 576, Springer, pp. 129–140 (1991)
38. Stefanov, E., Papamanthou, C., Shi, E.: Practical dynamic searchable encryption with small leakage. IACR Cryptology ePrint Archive 2013, 832 (2013), http://eprint.iacr.org/2013/832. Accessed 31 Aug 2020
39. Wang, C., Jan, S.T., Hu, H., Bossart, D., Wang, G.: The next domino to fall: Empirical analysis of user passwords across online services. In: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy. p. 196-203. CODASPY '18, Association for Computing Machinery, New York, NY, USA (2018), https://doi.org/10.1145/3176258.3176332
40. Yi, X., Hao, F., Chen, L., Liu, J.K.: Practical threshold password-authenticated secret sharing protocol. In: ESORICS'15. LNCS, vol. 9326, Springer, pp. 347–365 (2015)