Privacy Preserving Delegated Word Search in the Cloud

Kaoutar Elkhiyaoui, Melek Önen and and Refik Molva
EURECOM, Sophia-Antipolis, France
$\{$*elkhiyao, onen, molva*$\}$*@eurecom.fr*

Keywords: Privacy preserving keyword search, delegation, cloud

Abstract: In this paper, we address the problem of privacy preserving delegated word search in the cloud. We consider a scenario where a data owner outsources its data to a cloud server and delegates the search capabilities to a set of third party users. In the face of *semi-honest* cloud servers, the data owner does not want to disclose any information about the outsourced data; yet it still wants to benefit from the highly parallel cloud environment. In addition, the data owner wants to ensure that delegating the search functionality to third parties does not allow these third parties to jeopardize the confidentiality of the outsourced data, neither does it prevent the data owner from *efficiently* revoking the access of these authorized parties. To these ends, we propose a word search protocol that builds upon techniques of keyed hash functions, oblivious pseudo-random functions and Cuckoo hashing to construct a searchable index for the outsourced data, and uses private information retrieval of short information to guarantee that word search queries do not reveal any information about the data to the cloud server. Moreover, we combine attribute-based encryption and oblivious pseudo-random functions to achieve an efficient revocation of authorized third parties. The proposed scheme is suitable for the cloud as it can be easily parallelized.

1 INTRODUCTION

The cloud computing paradigm offers clients the ease of outsourcing the storage of their massive data with the advantage of reducing cost and assuring availability. Large-scale cloud infrastructures bring up severe security and privacy issues: Apart from traditional security challenges, the outsourced storage of ”big data” raises the challenge of processing it at the cloud in a secure and privacy preserving manner while considering the cloud provider itself as a potential adversary.

While data owners (i.e. clients) can simply encrypt their data before outsourcing it to the cloud, traditional confidentiality mechanisms fall short when it comes to mining/processing the data. Recently, several solutions have been proposed to allow the search of words over encrypted data. In this paper however, we address the problem of delegated word search whereby in addition to the data owner itself, some authorized third-parties can perform search operations over private data. In addition to security and privacy properties that classical search solutions assure under a semi-honest (i.e., honest-but-curious) security model, a privacy preserving delegated word search mechanism includes the delegation and revocation operations: The data owner should be able to remove the search capability of a third party at any point in time through an efficient revocation mechanism.

We propose a new privacy preserving word search solution whereby as in [5], the data owner constructs a searchable index with all words listed in its files and similarly to [3], it applies a private information retrieval to guarantee that the adversary including the cloud itself does not discover any information about the search query and its result. The newly proposed solution outperforms existing ones thanks to a combination of Cuckoo hashing with private information retrieval for the search operation. The use of Cuckoo hashing helps in assigning one word to a unique position in the index, thus removing the probability of collisions within the index: The data owner first constructs a confidential index where each particular element corresponds to a unique word and fills it in with some private information derived from the actual word. The search operation consists of the computation of the position corresponding to the queried word using Cuckoo hashing, and building the corresponding PIR query to be sent to the cloud provider.

Moreover, the delegation operation is assured thanks to the use of attribute based encryption (ABE) which only allows users holding certain "attributes" to search over the data. For example, when companies outsource their logs over the cloud, they can allow some data protection commissioner to search over them under an audit operation. Whereas efficient revocation is achieved by a combination of ABE and oblivious pseudo random functions. The revocation operation does not imply the re-encryption of the outsourced data and only requires an update of the access policy by the data owner which can be considered as a negligible cost.

The major contributions of the paper can be summarized as follows:

- We propose a new word search protocol which is based on an efficient word-index construction thanks to the use of Cuckoo hashing and the transformation of PIR into privacy preserving word search.
- The newly proposed solution also includes delegation and revocation capabilities thanks to the use of Attribute Based Encryption and Oblivious Pseudo Random Functions. The revocation operation does not incur any cost except for the update of the access policy by the data owner.
- We define the main privacy requirements and further provide a formal analysis of these properties.

Section 2 introduces the generic problem of privacy preserving delegated word search and the application scenario. The different privacy requirements are formally defined in section 3. The first version of the privacy preserving word search solution is described in section 4. The entire solution including the delegation and revocation operations is presented in section 5. We analyze the new solution in terms of security and performance in Sections 6 and 7. Finally, Section 8 reviews the state of the art.

2 BACKGROUND

We consider a scenario where a data owner outsources some privacy sensitive data to a cloud server and wishes to later on perform some operations over it without revealing any details about the data. The operation we are focusing on is word search over encrypted data and in our scenario the data owner may wish to delegate part of the search operations to authorized third parties. An illustrative example of such a requirement can be a scenario wherein due to regulatory matters, some data (such as logs) still need to be searchable by third parties such as data protection commissioners. The three entities involved in a privacy preserving delegated word search and the main algorithms are formally defined in the following sections.

2.1 Entities

A privacy preserving delegated word search involves the following entities:

- **Data owner** $\mathcal{O}$: It possesses a large file F that it outsources to the cloud server $\mathcal{S}$. Without loss of generality, we assume that the number of distinct words in F is n and the corresponding set is defined as $\mathcal{L}_\omega = \{\omega_1, \omega_2, ..., \omega_n\}$. Similarly to previous work such as [3, 6], we assume that once $\mathcal{O}$ outsources a file F, it will no longer modify it.
- **Cloud server** $\mathcal{S}$: It stores an *encrypted* version of the outsourced file F and a searchable index I of the set $\mathcal{L}_\omega$ of "distinct" words present in F.
- **Authorized user** $\mathcal{U}$: It has access to a set of credentials that enable it to perform search queries on F. This authorized user could be an auditor which as part of its auditing task has to search the activity logs of $\mathcal{O}$. We also note that in some cases an authorized user could correspond to the data owner that wants to perform word search on its outsourced data.

2.2 Privacy Preserving Delegated Word-Search

In accordance with the work of Curtmola et al. [6], a privacy preserving delegated word-search comprises the following algorithms:

- $\mathsf{Setup}(\zeta) \rightarrow (\mathsf{MK}, \mathcal{P})$: It is a randomized algorithm that is executed by the data owner $\mathcal{O}$. It takes as input the security parameter ζ, and outputs a master key MK and a set of public parameters $\mathcal{P}$ that will be used by subsequent algorithms to perform the word-search.
- $\mathsf{Encrypt}(\mathsf{MK}, F) \rightarrow C$: This algorithm is run by $\mathcal{O}$. It has as input the master key MK and the file F, and outputs an encryption C of file F.
- $\mathsf{BuildIndex}(\mathsf{MK}, F) \rightarrow I$: This algorithm has as input the master key MK and a file F and outputs an index I of distinct words ω_i present in F. This algorithm is generally run by the data owner $\mathcal{O}$.
- $\mathsf{Delegate}(\mathsf{MK}, \mathsf{St}_o, \mathsf{id}_u) \rightarrow K_u$: This algorithm is executed by $\mathcal{O}$ to delegate the search capabilities on its files to some third party user. On input of the master key MK, the current state St_o of $\mathcal{O}$ and the identifier id_u of some user $\mathcal{U}$, $\mathsf{Delegate}$ outputs a secret key K_u that will be provided to $\mathcal{U}$.

- $\mathsf{Token}(\omega, \mathsf{St}_u, K_u) \rightarrow \tau$: This algorithm is executed by authorized users or the data owner O to generate a search token for some word ω. It takes as input the word ω, the current state St_u of authorized user $\mathcal{U}$ and the key K_u and outputs a search token τ.
- $\mathsf{Query}(\tau) \rightarrow \mathcal{Q}$: It is a randomized algorithm that is run by authorized users to generate word search queries. On input of a token τ, Query outputs a word search query $\mathcal{Q}$ that will be forwarded to cloud server $\mathcal{S}$.
- $\mathsf{Response}(\mathcal{Q}, I) \rightarrow \mathcal{R}$: This algorithm is invoked by $\mathcal{S}$ whenever $\mathcal{S}$ receives a word search query $\mathcal{Q}$. It takes as input $\mathcal{Q}$ and the index I and outputs a word search response $\mathcal{R}$.
- $\mathsf{Verify}(\mathcal{R}, \mathsf{St}_u) \rightarrow b$: It is a deterministic algorithm run by authorized users to verify $\mathcal{S}$'s responses. On input of $\mathcal{S}$'s response $\mathcal{R}$ and the current state St_u of authorized user $\mathcal{U}$, Verify outputs a bit $b = 1$ if $\omega \in F$ and $b = 0$ otherwise.
- $\mathsf{Revoke}(\mathsf{MK}, \mathsf{St}_o, \mathsf{id}_u) \rightarrow (\mathsf{St}'_o, \mathsf{St}'_s)$: This algorithm is run by the data owner O to revoke the access of previously authorized users. It has as input the master key MK, the current state St_o of data owner O and the identifier id_u of some previously authorized user $\mathcal{U}$, and it outputs an updated state St'_o for O and an updated state St'_s for cloud server $\mathcal{S}$.

3 ADVERSARY MODEL

The crucial privacy challenge to address when designing a privacy preserving delegated word search is assuring privacy against a misbehaving cloud server. Indeed, the cloud server may attempt to infer sensitive information about the outsourced files (and their owners thereof) from the ciphertexts and indexes it keeps. It may also try to derive information about those files from the word search queries it processes. Thus, it is of utmost importance to ensure that the ciphertexts and the indexes that the cloud stores together with the word search queries it processes do not leak any information about the data owners' files.

Furthermore, the delegation of search capabilities to third party users inherently raises the requirements of access authorization and revocation, and therewith the requirement of *privacy against revoked users*. For example, a previously authorized user may exploit the information it collected during its word search operations that occurred when it was still authorized to conduct lookup operation after its revocation so as to learn new information about the outsourced files. Therefore, one should ensure that even if revoked users can still issue valid search queries to the cloud server, they should not be able to decode the cloud server's responses.

Along these lines, we provide in the subsequent sections formal models for the notions of both privacy against cloud servers and privacy against revoked users, which we will employ to assess the security of our scheme in the appendix of this paper. Of course, solutions protected against misbehaving clouds and revoked users are inherently secure against any other type of external adversaries.

3.1 Privacy against Cloud Server

In accordance with the work of Blass et al. [3] and Curtmola et al. [6], we assume that the cloud server $\mathcal{S}$ is *semi-honest*: Although interested in discovering the content of the data and the queries, $\mathcal{S}$ still performs all the required operations correctly.

A privacy preserving delegated word search should ensure that the *semi-honest* cloud server $\mathcal{S}$ does not discover any information about the content of an outsourced file from either its encryption or its index. This means that in addition to not being able to break the confidentiality of the outsourced data, $\mathcal{S}$ should neither be able to mount statistical attacks on the outsourced files (e.g. occurrence of words) nor to tell whether two files contain (or do not contain) the same words. In compliance with the work of [3], we refer to this requirement as *storage privacy*. Moreover, a solution for privacy preserving delegated word search should as well guarantee *query privacy*: during the lookup phase, cloud server $\mathcal{S}$ should not be able to derive any useful information about the queries of authorized users. Namely, $\mathcal{S}$ should not be able to tell whether any two word search queries were issued for the same word or not (cf. [3]).

To formally capture the adversarial capabilities of $\mathcal{S}$ in the subsequent privacy definitions, we assume that $\mathcal{S}$ is given access to the following oracles:

- $\mathcal{O}_{\mathsf{encrypt}}(F, \mathsf{MK}) \rightarrow C$: This oracle takes a file F and the master key MK of some data owner O as inputs and computes an encryption C of file F by calling the algorithm Encrypt.
- $\mathcal{O}_{\mathsf{index}}(F, \mathsf{MK}) \rightarrow I$: On inputs of file F and the master key MK, this oracle executes the algorithm BuildIndex and returns the index I associated with file F.
- $\mathcal{O}_{\mathsf{search},\mathcal{S}}(I, \omega) \rightarrow \mathsf{view}_{\mathcal{S}}$: Cloud server $\mathcal{S}$ invokes this oracle whenever it wants to receive and process a word search query. On inputs of index I and word ω, this oracle starts an execution of

Algorithm 1: Learning phase of the storage privacy game

```
// $\mathcal{S}$ calls oracles $\mathcal{O}_{\mathsf{encrypt}}$ and $\mathcal{O}_{\mathsf{index}}$ a polynomial
// number of times
$F_i \leftarrow \mathcal{S}$ ;
$C_i \leftarrow \mathcal{O}_{\mathsf{encrypt}}(F_i, \mathsf{MK})$;
$I_i \leftarrow \mathcal{O}_{\mathsf{index}}(F_i, \mathsf{MK})$;
//$\mathcal{S}$ returns a challenge word
$\omega^* \leftarrow \mathcal{S}$;
```

Algorithm 2: Challenge phase of the storage privacy game

```
// Let $\mathsf{F}_0^*$ and $\mathsf{F}_1^*$ be two files s.t. $\mathsf{F}_1^*$ contains $\omega^*$
// while $\mathsf{F}_0^*$ does not
$b \leftarrow \{0,1\}$;
$C_b^* \leftarrow \mathcal{O}_{\mathsf{encrypt}}(F_b^*, \mathsf{MK})$;
$I_b^* \leftarrow \mathcal{O}_{\mathsf{index}}(F_b^*, \mathsf{MK})$;
$b^* \leftarrow \mathcal{S}$ ;
```

the word search protocol with cloud server $\mathcal{S}$ to check whether ω is in $\mathcal{I}$ or not. At the end of the word search operation, $\mathcal{O}_{\mathsf{search},s}$ returns the view $\mathsf{view}_s = (\mathsf{St}_s, \mathsf{rand}_s, M_{1,s}, M_{2,s}, ..., M_{l,s})$ of cloud server $\mathcal{S}$ during the word search, where St_s is the current state of cloud server $\mathcal{S}$, rand_s is its internal randomness that it used to generate its word search response and $M_{i,s}$ is the i^{th} message that $\mathcal{S}$ received during the word search from oracle $\mathcal{O}_{\mathsf{search},s}$.

3.1.1 Storage Privacy

We define storage privacy using an indistinguishability-based game that comprises two phases: A learning phase (cf. Algorithm 1) and a challenge phase (cf. Algorithm 2). The goal of cloud server $\mathcal{S}$ in this game is to tell whether a challenge file F_b^* contains some word ω^*. To this effect, cloud server $\mathcal{S}$ calls the oracles $\mathcal{O}_{\mathsf{encrypt}}$ and $\mathcal{O}_{\mathsf{index}}$ for a polynomial number of times in the learning phase. By the end of this phase, $\mathcal{S}$ outputs a challenge word ω^*.

Let F_0^* and F_1^* be two files such that F_1^* contains ω^* while F_0^* does not.

Now in the challenge phase, cloud server $\mathcal{S}$ is provided with the encryption C_b^* and the index I_b^* of file F_b^* where b is picked randomly from $\{0,1\}$. At the end of the challenge phase, $\mathcal{S}$ outputs its guess b^* for the bit b. We say that $\mathcal{S}$ succeeds in the storage privacy game if $b = b^*$.

Definition 1 (Storage privacy). *Let $\Pi^{\mathcal{S}}_{\mathsf{success}}$ denote the probability that $\mathcal{S}$ succeeds in the storage privacy game. We say that a word search protocol assures storage privacy,* **iff** *for any cloud server $\mathcal{S}$, $\Pi^{\mathcal{S}}_{\mathsf{success}} \leq \frac{1}{2} + \varepsilon$, where ε is a negligible function in the security parameter ζ.*

Algorithm 3: Learning phase of the query privacy game

```
// $\mathcal{S}$ calls oracles $\mathcal{O}_{\mathsf{encrypt}}$, $\mathcal{O}_{\mathsf{index}}$, and $\mathcal{O}_{\mathsf{search,s}}$
// a polynomial number of times
$(F_i, \omega_i) \leftarrow \mathcal{S}$ ;
$C_i \leftarrow \mathcal{O}_{\mathsf{encrypt}}(F_i, \mathsf{MK})$;
$I_i \leftarrow \mathcal{O}_{\mathsf{index}}(F_i, \mathsf{MK})$;
$\mathsf{view}_{s,i} \leftarrow \mathcal{O}_{\mathsf{search},s}(\mathcal{I}, \omega_i)$;
//$\mathcal{S}$ outputs achallenge file $\mathsf{F}^*$ and two distinct
// words $\omega_0$ and $\omega_1$
$(F^*, \omega_0^*, \omega_1^*) \leftarrow \mathcal{S}$;
```

Algorithm 4: Challenge phase of the query privacy game

```
$C^* \leftarrow \mathcal{O}_{\mathsf{encrypt}}(F^*, \mathsf{MK})$;
$I^* \leftarrow \mathcal{O}_{\mathsf{index}}(F^*, \mathsf{MK})$;
$b \leftarrow \{0,1\}$;
$\mathsf{view}_s^* \leftarrow \mathcal{O}_{\mathsf{search},s}(\mathcal{I}^*, \omega_b^*)$;
$b^* \leftarrow \mathcal{S}$;
```

3.1.2 Query Privacy

Similarly to storage privacy, we formalize query privacy through an indistinguishability-based game that runs in two phases: A learning phase and a challenge phase. In the learning phase as depicted in Algorithm 3, cloud server $\mathcal{S}$ picks adaptively a polynomial number of file and word pairs (F_i, ω_i). For each selected pair (F_i, ω_i), $\mathcal{S}$ calls first the oracles $\mathcal{O}_{\mathsf{encrypt}}$ and $\mathcal{O}_{\mathsf{index}}$ to encrypt F and build the corresponding index respectively, then it queries the oracle $\mathcal{O}_{\mathsf{search},s}$ to receive and process a search query for word ω_i in F_i. At the end of the learning phase, $\mathcal{S}$ outputs a challenge file F^* and two challenge words ω_0^* and ω_1^*.

In the challenge phase (cf. Algorithm 4), cloud server $\mathcal{S}$ queries the oracles $\mathcal{O}_{\mathsf{encrypt}}$ and $\mathcal{O}_{\mathsf{index}}$ which provide $\mathcal{S}$ with the encryption and the index of the challenge file F^* respectively. Then, the oracle $\mathcal{O}_{\mathsf{search},s}$ executes an instance of the word search protocol for word ω_b^* with $\mathcal{S}$, where b is a randomly selected bit. Finally, $\mathcal{S}$ outputs its guess b^* for the bit b. We say that $\mathcal{S}$ succeeds in the query privacy game if $b = b^*$.

Definition 2. *Let $\Pi^{\mathcal{S}}_{\mathsf{success}}$ denote the probability that $\mathcal{S}$ succeeds in the query privacy game. We say that a word search protocol ensures query privacy,* **iff** *for* *any cloud server $\mathcal{S}$, $\Pi^{\mathcal{S}}_{\mathsf{success}} \leq \frac{1}{2} + \varepsilon$, where ε is a negligible function in the security parameter ζ.*

3.2 Privacy against Revoked Users ("forward privacy")

Ideally, a privacy preserving delegated word search should assure that when an authorized user is revoked, it can no longer look for words in the cloud server's files (this does not imply that the revoked user cannot query the server's database, rather it means that it cannot successfully interpret the cloud server's responses). In other words, a privacy preserving delegated word search should make sure that even if a revoked user is able to issue word search queries, it cannot infer *any new information* about the outsourced files that it did not learn before its revocation. This requirement resembles the notion of forward secrecy whereby a user cannot have access to any data after its revocation. In the context of word search in addition to the content of the data, the revoked user should not infer any additional information from future queries as well.

Since in this paper we only focus on *static data* (i.e. the data owner does not update its file once outsourced to the cloud server), we argue that the above intuition can be captured by assuring that revoked users cannot look up a word for which they did not issue a search query when they were still authorized.

Without loss of generality, we assume that there is a data owner O that outsources its file F and the corresponding index I to cloud server $\mathcal{S}$, and that a user $\mathcal{U}$ is interested in searching the file F even after its revocation. To this effect, $\mathcal{U}$ may behave *maliciously* during the execution of the word search protocol. Namely, $\mathcal{U}$ may provide bogus word search queries to cloud server $\mathcal{S}$.

In order to formalize privacy against revoked users, we use a privacy game that similarly to the two previous games consists of a learning and a challenge phase. In addition to the oracles O_{encrypt} and O_{index}, user $\mathcal{U}$ has access to the following oracles.

- $\mathcal{O}_{\mathsf{delegate}}(\mathsf{MK}) \rightarrow K_u$: On input of the data owner O's master key MK, the oracle $\mathcal{O}_{\mathsf{delegate}}$ executes the algorithm Delegate to allow $\mathcal{U}$ to perform word search on O's file F and outputs the secret key K_u.
- $\mathcal{O}_{\mathsf{revoke}}$: This oracle revokes the right of $\mathcal{U}$ to search the file F by executing the algorithm Revoke which updates the states of data owner O and cloud server $\mathcal{S}$.
- $\mathcal{O}_{\mathsf{search},u}(I, \omega) \rightarrow \mathsf{view}_u$: $\mathcal{U}$ calls this oracle whenever it wants to perform a word search on the index I. It takes as input an index I and a word ω and outputs the view $\mathsf{view}_u = (\mathsf{St}_u, \mathsf{rand}_u, M_{1,u}, M_{2,u}, ..., M_{l',u})$ of user $\mathcal{U}$ during the word search, where St_u is the current state of user $\mathcal{U}$ and rand_u is its internal randomness that it used to generate its word search query, whereas $M_{i,u}$ corresponds to the i^{th} message that $\mathcal{U}$ received from $\mathcal{O}_{\mathsf{search},u}$ during the word search.
- $\mathcal{O}_{\mathsf{chal},u}(I, \omega) \rightarrow \mathsf{chal}_{u,b}$: When called with an index I and word ω, this oracle flips a random coin $b \in \{0,1\}$. If $b = 1$, then $\mathcal{O}_{\mathsf{chal},u}$ returns the actual view $\mathsf{chal}_{u,1} = \mathsf{view}_u = (\mathsf{St}_u, \mathsf{rand}_u, M^1_{1,u}, M^1_{2,u}, ..., M^1_{l',u})$ of user $\mathcal{U}$ during the word search for ω, such that St_u is the current state of user $\mathcal{U}$ and rand_u is its internal randomness, whereas $M_{i,u}$ corresponds to the i^{th} message that $\mathcal{U}$ received from $\mathcal{O}_{\mathsf{search},u}$ during the word search. If $b = 0$, then $\mathcal{O}_{\mathsf{chal},u}$ outputs $\mathsf{chal}_{u,0} = (\mathsf{St}_u, \mathsf{rand}_u, M^0_{1,u}, M^0_{2,u}, ..., M^0_{l',u})$, where St_u is the current state of user $\mathcal{U}$ and rand_u is its internal randomness, and $M^0_{i,u}$ are generated randomly by $\mathcal{O}_{\mathsf{chal},u}$.

Once user $\mathcal{U}$ enters the learning phase of the privacy game (see Algorithm 5), it first calls the oracle $\mathcal{O}_{\mathsf{index}}$ with a file F of its choosing to get the corresponding index I. Next user $\mathcal{U}$ invokes the oracle $\mathcal{O}_{\mathsf{delegate}}$ which supplies $\mathcal{U}$ with the secret key K_u. This key will enable $\mathcal{U}$ to execute the word search protocol with cloud server $\mathcal{S}$ on the index I and therewith on file F. Then user $\mathcal{U}$ queries the oracle $\mathcal{O}_{\mathsf{search},u}$ for a polynomial number of words ω_i of its choosing. Next, the oracle $\mathcal{O}_{\mathsf{revoke}}$ revokes $\mathcal{U}$. After the revocation, $\mathcal{U}$ can still issue a polynomial number of word search queries on file F by calling $\mathcal{O}_{\mathsf{search},u}$. Finally, $\mathcal{U}$ outputs a challenge word ω^* that is not present in file F.

In the challenge phase (see Algorithm 6), $\mathcal{U}$ queries the oracle $\mathcal{O}_{\mathsf{chal},u}$ with the word ω^* and the index I^* that corresponds to $F \cup \{\omega^*\}$. The oracle $\mathcal{O}_{\mathsf{chal},u}$ in turn flips a random coin $b \in \{0,1\}$ and outputs the challenge view $\mathsf{chal}^*_{u,b}$. At the end of the challenge phase, revoked user $\mathcal{U}$ outputs a guess b^* for bit b.

We say that $\mathcal{U}$ succeeds in the game of privacy against revoked users if **i.)** $b = b^*$ and if **ii.)** $\mathcal{U}$ did not issue a search query for the challenge word ω^* before calling the oracle $\mathcal{O}_{\mathsf{revoke}}$ (i.e. $\omega^* \neq \omega_i, \forall i$).

Definition 3. *Let $\Pi^{\mathcal{U}}_{\mathsf{success}}$ denote the probability that $\mathcal{U}$ succeeds in the privacy game against revoked users. We say that a delegated word search mechanism provides privacy against revoked users* **iff** *for any revoked user $\mathcal{U}$, $\Pi^{\mathcal{U}}_{\mathsf{success}} \leq \frac{1}{2} + \varepsilon$, where ε is a negligible function in the security parameter ζ.*

Algorithm 5: Learning phase of the privacy game against revoked users

```
$I \leftarrow \mathcal{O}_{\mathsf{index}}(F, \mathsf{MK})$;
$K_u \leftarrow \mathcal{O}_{\mathsf{delegate}}(I)$;
// $\mathcal{U}$ calls $\mathcal{O}_{\mathsf{search,u}}$ for a polynomial number of
// times
$\omega_i \leftarrow \mathcal{U}$;
$\mathsf{view}_{u,i} \leftarrow \mathcal{O}_{\mathsf{search},u}(I, \omega_i)$;
$\mathcal{O}_{\mathsf{revoke}}(\mathcal{U})$;
// $\mathcal{U}$ calls $\mathcal{O}_{\mathsf{search,u}}$ for a polynomial number of
// times after revocation
$\omega'_i \leftarrow \mathcal{U}$;
$\mathsf{view}'_{u,i} \leftarrow \mathcal{O}_{\mathsf{search},u}(I, \omega'_i)$;
//$\mathcal{U}$ returns a challenge word that is not in file F
$\omega^* \leftarrow \mathcal{U}$ ;
```

Algorithm 6: Challenge phase of the privacy game against revoked users

```
$I^* \leftarrow \mathcal{O}_{\mathsf{index}}(F \cup \{\omega^*\}, \mathsf{MK})$;
$\mathsf{chal}^*_{u,b} \leftarrow \mathcal{O}_{\mathsf{chal},u}(I^*, \omega^*)$;
$b^* \leftarrow \mathcal{U}$;
```

4 PRIVACY PRESERVING WORD SEARCH

In this section, we describe the first version of the proposed word search solution which does not offer any delegation capabilities and therefore only assures privacy against honest-but-curious cloud providers. Similarly to [3, 5], to assure query privacy against a *semi-honest* cloud server, we rely on *Private Information Retrieval* (PIR) to build our word-search scheme. Actually, PIR allows a user to retrieve a data block from a server's database without disclosing any information about the sought block. However, PIR protocols assume that the user know beforehand the position in the database of the data block to be retrieved, and therefore, they cannot be used directly in privacy preserving word search wherein a user only holds a list of words to look for. Fortunately, Chor et al. [5] proposed a technique that transforms any PIR mechanism into a protocol for private information retrieval by keyword, and thereby, into a privacy preserving word-search. The main idea is to first construct an index of all the distinct words present in the outsourced data and then apply a PIR to this index. As shown in [5], this can be achieved by representing the index by a hash-table that maps each word to a unique position in the table. During the search phase, the user first computes the position of the requested word in the hashtable (i.e. the index) and further runs PIR to fetch the block stored at that position. While the construction of [5] can be easily transformed into a privacy preserving word search, we believe that it can be further optimized by using *Cuckoo hashing* to build the hashtables (i.e. the indexes) of the words in the outsourced files.

Along these lines, we first formalize and describe the PIR and the Cuckoo hashing algorithms that will underpin our word search solution.

4.1 Building Blocks

4.1.1 Trapdoor Private Information Retrieval

For efficiency purposes, we opt for a PIR mechanism called *trapdoor PIR* which was proposed by Trostle and Parrish [16], and whose security is based on the *trapdoor group assumption*. We stress however that this particular PIR can be interchanged by any other efficient PIR algorithm.

In compliance with the work of Trostle and Parrish [16], we model the server's database on which private information retrieval is performed by a binary $(k,l)-$matrix $\mathcal{M}$. Trapdoor PIR allows a user to retrieve the bit b at position (x,y) in $\mathcal{M}$ as follows:

- $\mathsf{PIRQuery}(x) \rightarrow \vec{\alpha}$: The user picks a *secret* large number p (typically $|p| = 200$ bits) and selects randomly $u \in \mathbb{Z}_p^*$ and k other values $a_i \in \mathbb{Z}_p$. Next, it computes the k following values: $e_x = 1 + 2 \cdot a_x$ and $\forall\ i \neq x,\ e_i = 2 \cdot a_i$, and sends the vector $\vec{\alpha} = (\alpha_i)_{i=1}^k = (u \cdot e_i \bmod p)_{i=1}^k$ to the cloud.
- $\mathsf{PIRResponse}(\vec{\alpha}, \mathcal{M}) \rightarrow \vec{\beta}$: On receiving $\vec{\alpha}$, the server computes the matrix product $\vec{\beta} = (\beta_1, \beta_2, ..., \beta_l) = \vec{\alpha} \cdot \mathcal{M}$.
- $\mathsf{PIRAnalysis}(\vec{\beta}, y) \rightarrow b$: After receiving the server's response $\vec{\beta} = (\beta_1, \beta_2, ..., \beta_l)$, the user computes $\gamma_y = \beta_y \cdot u^{-1} \bmod p$, and retrieves b by computing $\gamma_y \bmod 2$.

4.1.2 Cuckoo Hashing

Cuckoo hashing was first proposed by Pagh and Rodler [14] to build efficient and practical data indexes. It ensures *worst-case* constant look-up and deletion time and *amortized* constant insertion time while minimizing the storage requirements.

In order to store n elements in some index I, Cuckoo hashing uses two hash tables T and T' containing L entries each, and two hash functions $H : \{0,1\}^* \rightarrow \{1,2,...,L\}$ and $H' : \{0,1\}^* \rightarrow \{1,2,...,L\}$. Now, an element τ_i is either stored in entry $H(\tau_i)$ in hash table T, or in entry $H'(\tau_i)$ in hash table T' but never in both.

The lookup operation in I is therefore simple: When given an element $\tau \in \{0,1\}^*$, the two entries at positions $H(\tau_i)$ and $H'(\tau_i)$ are queried in tables T and T' respectively. To delete an element τ_i from I, the entry corresponding to τ_i is removed. Finally, to insert a new element $\tau_i \in \{0,1\}^*$ into I, we first check whether the entry of T at position $H(\tau_i)$ is empty. If it is the case, then τ_i is inserted in this entry of T and the insertion algorithm converges. Otherwise, if that entry is already occupied by another element τ_j, then τ_j will be removed from its current entry in T and relocated to its other possible entry $H'(\tau_j)$ in T'. Now, if there is an element τ_k in the entry $H'(\tau_j)$ of T', then τ_j will be inserted in entry $H'(\tau_j)$ in table T' while τ_k will be moved to its other possible entry $H(\tau_k)$ in T. This insertion process is repeated iteratively until the insertion of all elements in either T or T'. If this process of insertion does not converge (i.e., there is an element that cannot be inserted), or it takes too long to converge, then all the elements in I will be rehashed with new hash functions $\mathbf{H}$ and $\mathbf{H}'$.

An analysis of Cuckoo hashing [13] shows that if $L \geq n$, then there is a family of universal hash functions that guarantees a small rehashing probability of order $O(\frac{1}{n})$ and a constant expected time for insertion. For a more comprehensive analysis of the performance of Cuckoo hashing, the reader may refer to [14].

4.2 Protocol Description

We recall that in this first version, the data owner O wants to upload a large file F to cloud server $\mathcal{S}$ and once its data uploaded O wants to further search for some words within the file without revealing any information to the semi-honest cloud server. The set of all distinct words within F is defined as $\mathcal{L}_\omega = \{\omega_1, \omega_2, ..., \omega_n\}$. The proposed protocol can be divided into two main phases:

- During the **upload** phase, before outsourcing its data, O builds the index corresponding to the n distinct words present in file F and encrypts F using a *semantically secure* symmetric encryption.

- During the **search** phase, O computes the position of the requested word ω in F's index and perform a PIR query to retrieve the information stored at that position in the index. Upon reception of server $\mathcal{S}$'s PIR response, O verifies this response and decides accordingly whether ω is present in F or not.

4.2.1 Setup

The data owner O calls the Setup algorithm which takes as input the security parameter ζ and outputs a master key MK and a set of public parameters $\mathcal{P}$ such that:

- The master key MK is composed of a symmetric encryption key K_{enc} and a MAC key K_{mac}.
- The public parameters $\mathcal{P}$ comprise a MAC $\mathcal{H}_{\mathsf{mac}} : \{0,1\}^\zeta \times \{0,1\}^* \to \{0,1\}^\kappa$ and a cryptographic hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^t$.

4.2.2 Upload

The file upload phase consists of **i.)** Encrypting the file F using a *semantically secure* encryption such as AES in counter mode (cf. Encrypt) and **ii.)** building a searchable index for $\mathcal{L}_\omega$ (cf. BuildIndex).

The data owner O first generates a unique file identifier fid for file F and then encrypts F by calling the algorithm Encrypt. This algorithm takes as inputs secret key K_{enc} and file F and outputs a semantically secure encryption $C = \mathsf{Enc}(K_{\mathsf{enc}}, F)$ of F. Next, O invokes the algorithm BuildIndex which on input of master key MK (more precisely MAC key K_{mac}), file identifier fid and the list of distinct words $\mathcal{L}_\omega = \{\omega_1, \omega_2, ..., \omega_n\}$ present in F outputs a list of MACs $\mathcal{L}_H = \{h_1, h_2 ..., h_n\}$, such that $h_i = \mathcal{H}_{\mathsf{mac}}(K_{\mathsf{mac}}, \omega_i || \mathsf{fid})$ where $||$ denotes concatenation. Then the algorithm BuildIndex constructs an index I for $\mathcal{L}_H = \{h_1, h_2 ..., h_n\}$ using Cuckoo hashing. In order to optimize the performance of the PIR underlying our word-search scheme, our index will differ from traditional Cuckoo hashing indexes by comprising two sets of t binary (rectangular) matrices $\{\mathcal{M}_j\}_{j=1}^t, \{\mathcal{M'}_j\}_{j=1}^t$ of size (k,l) rather than two hash-tables T and T'. Namely, instead of using two hash functions that hash into $\{1,2,...,L\}$, we employ two hash functions H and H' that hash into $\{1,2,...,k\} \times \{1,2,...,l\}$. For an element $h \in \{0,1\}^*$, the hash function H (H' resp.) returns a position (x,y) ((x',y') resp.) in matrices $\{\mathcal{M}_j\}$ ($\{\mathcal{M'}_j\}$ resp.). More precisely, the algorithm BuildIndex executes the following:

- First BuildIndex generates two sets of t binary matrices $\{\mathcal{M}_j\}$ and $\{\mathcal{M}'_j\}$ $(1 \leq j \leq t)$ of size (k,l) each, where each element is initialized to 0.
- BuildIndex then picks two hashes H and H' that map each element h_i in $\mathcal{L}_H$ to either a position $(x_i, y_i) = H(h_i)$ in matrices $\{\mathcal{M}_j\}$ or to a position $(x'_i, y'_i) = H'(h_i)$ in matrices $\{\mathcal{M}'_j\}$, by following the Cuckoo hashing algorithm described in Section 4.1.2. We recall that in order to ensure worst-case constant look-up using Cuckoo hashing, k

and l have to be chosen such that $kl \geq n$, where n is the size of $\mathcal{L}_H$.

- BuildIndex subsequently fills the binary matrices $\{\mathcal{M}_j\}$ and $\{\mathcal{M}'_j\}$ $(1 \leq j \leq t)$ as follows:
  - For each h_i, BuildIndex computes $\mathcal{H}(h_i) = (b_{i,1}, b_{i,2}, ..., b_{i,t})$, where $\mathcal{H}$ is a $t-$bits cryptographic hash function.
  - Now, if h_i is mapped to a position $(x_i, y_i) = H(h_i)$ in $\mathcal{M}_j$ (or to a position $(x'_i, y'_i) = H'(h_i)$ in $\mathcal{M}'_j$ resp.), then the bit at position (x_i, y_i) in $\mathcal{M}_j$ (the bit at position (x'_i, y'_i) in $\mathcal{M}'_j$ resp.) will be set to $b_{i,j}$. Hence, if h_i is mapped to a position $(x_i, y_i) = H(h_i)$ in $\{\mathcal{M}_j\}$ $(1 \leq j \leq t)$, then:
$$\mathcal{H}(h_i) = (\mathcal{M}_{1(x_i,y_i)}, \mathcal{M}_{2(x_i,y_i)}, ..., \mathcal{M}_{t(x_i,y_i)})$$
- Finally, BuildIndex outputs the searchable index $I = \{H, H', \mathbb{M}, \mathbb{M}'\}$ such that $\mathbb{M} = \{\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_t\}$ and $\mathbb{M}' = \{\mathcal{M}'_1, \mathcal{M}'_2, ..., \mathcal{M}'_t\}$.

At the end of this phase, data owner O sends the file identifier fid, the encryption C and the index I to cloud server $\mathcal{S}$.

4.2.3 Word Search

The search phase is divided into the three following steps:

Search Query To look for a word ω in file F, O calls the algorithm Token which computes the MAC $\mathsf{h} = \mathcal{H}_{\mathsf{mac}}(K_{\mathsf{mac}}, \omega || \mathsf{fid})$. Further, O runs the algorithm Query which computes $H(\mathsf{h}) = (x, y)$ and $H'(\mathsf{h}) = (x', y')$. We recall that (x, y) and (x', y') correspond to the potential position of h in $\{\mathcal{M}_j\}$ and $\{\mathcal{M}'_j\}$ respectively. Next, algorithm Query outputs two PIR queries $\vec{\alpha} = \mathsf{PIRQuery}(x) = (\alpha_1, \alpha_2, ..., \alpha_k)$ and $\vec{\alpha}' = \mathsf{PIRQuery}(x') = (\alpha'_1, \alpha'_2, ..., \alpha'_k)$ that will allow O to retrieve the x^{th} and x'^{th} rows respectively of (k, l) binary matrices, as depicted in Section 4.1.1. Finally, O sends its search query $\mathcal{Q} = (\vec{\alpha}, \vec{\alpha}')$ to server $\mathcal{S}$.

Search response On receiving O's search query $\mathcal{Q} = (\vec{\alpha}, \vec{\alpha}')$, $\mathcal{S}$ runs algorithm Response which on input of $\mathcal{Q}$, $\mathbb{M} = \{\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_t\}$ and $\mathbb{M}' = \{\mathcal{M}'_1, \mathcal{M}'_2, ..., \mathcal{M}'_t\}$, computes two sets of t PIR responses $\mathbb{R} = \{\vec{\beta}_1, \vec{\beta}_2, ..., \vec{\beta}_t\}$ and $\mathbb{R}' = \{\vec{\beta}'_1, \vec{\beta}'_2, ..., \vec{\beta}'_t\}$ such that for all $1 \leq j \leq t$:

$$\begin{aligned} \vec{\beta}_j &= \mathsf{PIRResponse}(\vec{\alpha}, \mathcal{M}_j) = \vec{\alpha} \cdot \mathcal{M}_j \\ \vec{\beta}'_j &= \mathsf{PIRResponse}(\vec{\alpha}', \mathcal{M}'_j) = \vec{\alpha}' \cdot \mathcal{M}'_j \end{aligned}$$

$\mathcal{S}$ sends then its word search response $\mathcal{R} = \{\mathbb{R}, \mathbb{R}'\}$ to O.

Verification To verify whether ω is in file F, the data owner O runs the algorithm Verify. When called, algorithm Verify unblinds the y^{th} element of each vector $\vec{\beta}_j$ by executing $\mathsf{PIRAnalysis}(y)$ and the y'^{th} element of each vector $\vec{\beta}'_j$ by running $\mathsf{PIRAnalysis}(y')$, as was shown in Section 4.1.1. This allows Verify to derive a bit b_j from $\vec{\beta}_j$ and a bit b'_j from $\vec{\beta}'_j$ respectively for all $1 \leq j \leq t$.

We denote by $\vec{b}$ and $\vec{b}'$ the string of bits $(b_1, b_2, ..., b_t)$ and $(b'_1, b'_2, ..., b'_t)$ respectively. After obtaining $\vec{b}$ and $\vec{b}'$, algorithm Verify computes the hash $\mathcal{H}(\mathsf{h})$ and checks whether $\vec{b} = \mathcal{H}(h)$ or $\vec{b}' = \mathcal{H}(\mathsf{h})$. If so, then Verify outputs 1 meaning that $\omega \in F$; otherwise, Verify outputs 0.

5 PRIVACY PRESERVING WORD SEARCH WITH DELEGATION

In this section we describe the entire solution including the delegation capabilities. We recall that data owner O wants to: **i.)** upload a large file F that contains n distinct words $\mathcal{L}_\omega = \{\omega_1, \omega_2, ..., \omega_n\}$ to cloud server $\mathcal{S}$, **ii.)** delegate the search capabilities on file F to third party users and finally **iii.)** be able to revoke these third party users at any point of time. Therefore the final solution involves in addition to the previously mentioned two phases from the basic protocol (i.e. **Upload** and **WdSearch**), a **Delegation** and a **Revocation** phase. We modify the **Upload** and **Word Search** phases so as to allow the data owner to upload the necessary material that will enable authorized users to perform search operations, whereas during the newly defined **Delegation** phase, the data owner provides authorized users with the MAC key used to build the index. Finally, the **Revocation** phase is defined in order to grant the data owner the capability to revoke authorized users efficiently.

The additional two phases are defined thanks to the use of *Ciphertext-Policy Attribute-Based Encryption* (CP-ABE) and *Oblivious Pseudo Random Functions* (OPRF). We stress here that by combining OPRF and ABE, we do not only allow for seamless revocation but also we ensure the *anonymity of authorized users*. As opposed to traditional access control mechanisms, the proposed solution does not require authorized users to identify and authenticate themselves to the cloud server.

Before providing a detailed description of our scheme, we summarize and formalize in the next section the algorithms underlying CP-ABE and OPRFs.

5.1 Building Blocks

5.1.1 Ciphertext-Policy Attribute-Based Encryption

A ciphertext-policy attribute-based encryption allows a user to encrypt a message M under some access policy AP in such a way that only parties possessing attributes that match AP can derive M from the ciphertext. Actually, a CP-ABE consists of the following algorithms, cf. [2]:

- $\mathsf{Setup}_{\mathsf{abe}}(\zeta) \rightarrow (\mathsf{MK}_{\mathsf{abe}}, \mathcal{P}_{\mathsf{abe}})$: It is a randomized algorithm that takes as input a security parameter ζ, and outputs a master key $\mathsf{MK}_{\mathsf{abe}}$ and a set of public parameters $\mathcal{P}_{\mathsf{abe}}$ that will be used by subsequent algorithms.
- $\mathsf{Enc}_{\mathsf{abe}}(M, \mathsf{AP}) \rightarrow \mathcal{C}$: It is a randomized algorithm that takes as input a message M and some access policy AP, and outputs a ciphertext $\mathcal{C} = \mathsf{Enc}_{\mathsf{abe}}(M, \mathsf{AP})$ such that only users holding the attributes satisfying the access policy AP can decrypt $\mathcal{C}$.
- $\mathsf{CredGen}_{\mathsf{abe}}(\mathsf{MK}_{\mathsf{abe}}, \mathcal{A}_i) \rightarrow \mathsf{cred}_i$: It is a randomized algorithm which on input of master key $\mathsf{MK}_{\mathsf{abe}}$ and a set of attributes $\mathcal{A}_i$, generates a set of credentials cred_i that are associated with $\mathcal{A}_i$. This algorithm is generally executed by a trusted third party (for instance a certification authority) whose aim is to define a set of admissible attributes $\mathbb{A}$ and to issue credentials cred_i to any user possessing attributes $\mathcal{A}_i \subset \mathbb{A}$.
- $\mathsf{Dec}_{\mathsf{abe}}(\mathcal{C}, \mathsf{cred}_i) \rightarrow \hat{M}$: It is a deterministic algorithm that takes as input a ciphertext $\mathcal{C}$ and a set of credentials cred_i. Assume that $\mathcal{C}$ encrypts a message M under the access policy AP (i.e., $\mathcal{C} = \mathsf{Enc}_{\mathsf{abe}}(M, \mathsf{AP})$) and that the credentials cred_i are associated with the set of attributes $\mathcal{A}_i$. If the attributes $\mathcal{A}_i$ satisfy the access policy AP, then $\mathsf{Dec}_{\mathsf{abe}}$ decrypts $\mathcal{C}$ successfully and outputs $\hat{M} = \mathsf{Dec}_{\mathsf{abe}}(\mathcal{C}, \mathsf{cred}_i) = M$. Otherwise, the decryption fails and $\mathsf{Dec}_{\mathsf{abe}}$ outputs $\hat{M} = \perp$.

5.1.2 Oblivious Pseudo-Random Functions

An OPRF [9, 11] is a two-party protocol that allows a sender S with input δ and a receiver R with input h to compute jointly the function $f_\delta(\mathsf{h})$ for some pseudo-random function family f_δ, in such a way that receiver R only learns the value $f_\delta(\mathsf{h})$, whereas sender S learns nothing from the protocol interaction.

Definition 4 (Oblivious Pseudo-Random Function [9]). *A two-party protocol π between a sender S of input δ and a receiver R of input h is said to be an oblivious pseudo-random function (OPRF), if there is some pseudo-random function family f_δ such that at the end of the execution of π:*

- *Receiver R gets $f_\delta(\mathsf{h})$ while learning nothing about S's input δ.*
- *Sender S learns nothing about R's input h or the value of $f_\delta(\mathsf{h})$.*

In the following, we provide a quick overview of the generic algorithms underpinning an OPRF that evaluates the output of some pseudo-random function family f_δ:

- $\mathsf{Setup}_{\mathsf{oprf}}(\zeta) \rightarrow (\delta, \mathcal{P}_{\mathsf{oprf}})$: It is a randomized algorithm that is run by the sender S. It takes as input the security parameter ζ and outputs an OPRF secret key δ and a set of public parameters $\mathcal{P}_{\mathsf{oprf}}$ that will be used by subsequent algorithms.
- $\mathsf{Query}_{\mathsf{oprf}}(\mathsf{h}) \rightarrow \mathcal{Q}_{\mathsf{oprf}}$: It is a randomized algorithm that is executed by the receiver R whenever R wants to generate an OPRF query. This algorithm has as input an element $\mathsf{h} \in \{0,1\}^\kappa$ and outputs a matching OPRF query $\mathcal{Q}_{\mathsf{oprf}}$ that will be sent later to sender S.
- $\mathsf{Response}_{\mathsf{oprf}}(\mathcal{Q}_{\mathsf{oprf}}, \delta) \rightarrow \mathcal{R}_{\mathsf{oprf}}$: It is a randomized algorithm which is operated by sender S whenever S receives an OPRF query. On input of an OPRF query $\mathcal{Q}_{\mathsf{oprf}}$, the algorithm $\mathsf{Response}_{\mathsf{oprf}}$ returns the corresponding OPRF response $\mathcal{R}_{\mathsf{oprf}}$ that will be forwarded to the receiver.
- $\mathsf{Result}_{\mathsf{oprf}}(\mathcal{R}_{\mathsf{oprf}}, \mathsf{St}_r) \rightarrow f_\delta(\mathsf{h})$: It is deterministic algorithm that is run by receiver R and takes as input an OPRF response $\mathcal{R}_{\mathsf{oprf}}$ and the current state St_r of R. Without loss of generality, we assume that R received the response $\mathcal{R}_{\mathsf{oprf}}$ as a follow-up to a previous OPRF query that was generated for $\mathsf{h} \in \{0,1\}^\kappa$. Accordingly, the algorithm $\mathsf{Result}_{\mathsf{oprf}}$ outputs $f_\delta(\mathsf{h})$, i.e. the evaluation of the pseudo-random function f_δ at point h.

In the remainder of this paper, we employ the OPRF proposed by Jarecki and Liu [11] which allows a receiver R and a sender S to compute jointly the evaluation of the pseudo-random function $f_\delta(\mathsf{h}) = \mathfrak{g}^{1/(\delta+\mathsf{h})}$ for any $\mathsf{h} \in \mathbb{Z}_N^*$, where N is an RSA safe modulus and $\mathfrak{g}$ is a random generator of a group $\mathbb{G}$ of order N. However for ease of exposition, we will omit the implementation details of this OPRF and we will only refer to the generic OPRF algorithms when describing our scheme.

5.2 Protocol Description

In the sequel of this paper and in accordance with the work of Curtmola et al. [6], we assume that the cloud server does not collude with revoked users. We indicate that if such a collusion happens, then our protocol will not be able to deter revoked users from searching the outsourced files.

Without loss of generality, we also assume that there is some certification authority which is in charge of: **i.)** defining the universe of admissible attributes $\mathbb{A} = \{\mathsf{att}_1, \mathsf{att}_2, ...\}$, **ii.)** providing potential data owners and potential authorized users with their credentials cred_i that match their attributes $\mathcal{A}_i \subset \mathbb{A}$ following for instance the CP-ABE scheme proposed by Bethencourt et al. [2].

5.2.1 Setup

As in the first version of the protocol, the data owner O calls the Setup algorithm which takes as input the security parameter ζ and outputs a master key MK and a set of public parameters $\mathcal{P}$ such that:

- The master key MK is composed of a symmetric encryption key K_{enc}, a MAC key K_{mac} and an OPRF secret key δ.
- The new public parameters $\mathcal{P}$ comprise a MAC $\mathcal{H}_{\mathsf{mac}} : \{0,1\}^{\zeta} \times \{0,1\}^* \to \mathbb{Z}_N^*$ (where N is a safe RSA modulus), a cryptographic hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^t$ and the public parameters $\mathcal{P}_{\mathsf{oprf}}$ of the OPRF $f_\delta(h) = \mathfrak{g}^{1/(\delta+h)}$.

5.2.2 Upload

The file upload phase amounts to **i.)** Encrypting the file F using AES encryption (cf. $\mathsf{Encrypt}$) **ii.)** building a searchable index for $\mathcal{L}_\omega$ (cf. $\mathsf{BuildIndex}$). Now instead of building the index I based on $\mathcal{L}_H = \{h_1, h_2 ..., h_n\}$ as was done previously, the index will be constructed using the OPRF values $f_\delta(h_i) = \mathfrak{g}^{1/(\delta+h_i)}$. Since the computation of OPRF is deemed to be demanding, we suggest that $\mathsf{BuildIndex}$ be executed jointly by O and the *semi-honest* cloud server $\mathcal{S}$ in such a way that O is only required to compute symmetric operations (e.g. hash functions and AES encryption) whereas the cloud server performs the more computationally intensive operations (i.e. OPRF and Cuckoo Hashing). Henceforth, we denote $\mathsf{BuildIndex}_{\mathsf{O}}$ the sub-algorithm of $\mathsf{BuildIndex}$ that is executed by data owner O and $\mathsf{BuildIndex}_{\mathsf{S}}$ the sub-algorithm of $\mathsf{BuildIndex}$ that is operated by cloud server $\mathcal{S}$.

Processing at the data owner As in the previous protocol, data owner O first generates a unique file identifier fid for file F and then encrypts F by calling the algorithm $\mathsf{Encrypt}$ which outputs an AES encryption $C = \mathsf{Enc}(K_{\mathsf{enc}}, F)$ of F. Then, O invokes the algorithm $\mathsf{BuildIndex}_{\mathsf{O}}$ which outputs a list of MACs $\mathcal{L}_H = \{h_1, h_2 ..., h_n\}$, such that $h_i = \mathcal{H}_{\mathsf{mac}}(K_{\mathsf{mac}}, \omega_i || \mathsf{fid})$. Next, O defines the access policy AP that will be associated with file F and finally forwards (*via a secure channel*) the file identifier fid, the encryption C, the list of MACs $\mathcal{L}_H = \{h_1, h_2, ..., h_n\}$, the access policy AP and the OPRF secret key δ to cloud server $\mathcal{S}$.

Processing at the cloud The processing at the cloud comprises two operations. The first one is to compute OPRF over the MACs in $\mathcal{L}_H = \{h_1, h_2, ..., h_n\}$ using the secret key δ. The second operation is to build an index with the resulting values using Cuckoo hashing. More precisely, upon receipt of file identifier fid, ciphertext C, list of keyed hashes $\mathcal{L}_H = \{h_1, h_2, ..., h_n\}$, access policy AP associated with C and the OPRF key δ, $\mathcal{S}$ calls the algorithm $\mathsf{BuildIndex}_{\mathsf{S}}$ which proceeds as explained below:

- First, $\mathsf{BuildIndex}_{\mathsf{S}}$ computes $\tau_i = f_\delta(h_i) = \mathfrak{g}^{1/(\delta+h_i)}$ for all $1 \leq i \leq n$.
- $\mathsf{BuildIndex}_{\mathsf{S}}$ prepares an index I for $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$ using Cuckoo hashing. Namely, $\mathsf{BuildIndex}_{\mathsf{S}}$ generates two sets of t binary matrices $\{\mathcal{M}_j\}$ and $\{\mathcal{M}'_j\}$ $(1 \leq j \leq t)$ of size (k,l) each, where each element is initialized to 0. $\mathsf{BuildIndex}_{\mathsf{S}}$ then selects two hashes H and H' that map each element τ_i in $\mathcal{T}$ to either a position $(x_i, y_i) = H(\tau_i)$ in matrices $\{\mathcal{M}_j\}$ or to a position $(x'_i, y'_i) = H'(\tau_i)$ in matrices $\{\mathcal{M}'_j\}$, by executing the Cuckoo hashing algorithm.
- $\mathsf{BuildIndex}_{\mathsf{S}}$ fills the binary matrices $\{\mathcal{M}_j\}$ and $\{\mathcal{M}'_j\}$ $(1 \leq j \leq t)$ similarly to the previous version of the protocol. The only difference is that instead of storing the hashes $\mathcal{H}(h_i)$ in $\{\mathcal{M}_j\}$ and $\{\mathcal{M}'_j\}$, we store the hashes $\mathcal{H}(\tau_i)$.
- Finally, $\mathsf{BuildIndex}_{\mathsf{S}}$ outputs the searchable index $I = \{H, H', \mathbb{M}, \mathbb{M}'\}$ such that $\mathbb{M} = \{\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_t\}$ and $\mathbb{M}' = \{\mathcal{M}'_1, \mathcal{M}'_2, ..., \mathcal{M}'_t\}$.

5.2.3 Delegation

To delegate the word search capabilities on the encrypted file F to third party users, data owner O encrypts its MAC key K_{mac} under its access policy AP using attribute-based encryption and provides cloud server $\mathcal{S}$ with the resulting ciphertext $\mathcal{C}_{\mathsf{mac}} = \mathsf{Enc}_{\mathsf{abe}}(K_{\mathsf{mac}}, \mathsf{AP})$. Thereafter, $\mathcal{S}$ publishes the ciphertext $\mathcal{C}_{\mathsf{mac}}$ and the file identifier fid.

We note that an authorized user $\mathcal{U}$ will in principle possesses a set of attributes $\mathcal{A}$ (and therewith a set of credentials cred) that satisfy the access policy AP.

Hence, $\mathcal{U}$ will be able to decrypt the ciphertext $\mathcal{C}_{\mathsf{mac}}$ using cred and derives the MAC key K_{mac}. This MAC key K_{mac} will be then used by $\mathcal{U}$ to perform word search on O's file as will be shown in the next section.

5.2.4 Word Search

To search the encrypted file C for some word ω, the authorized user $\mathcal{U}$ performs the following operations:

Token generation The token generation phase consists of executing an OPRF protocol between the authorized user $\mathcal{U}$ and the cloud server $\mathcal{S}$, where $\mathcal{U}$ corresponds to the receiver R and $\mathcal{S}$ to the sender S (following the notations in Section 5.1.2). Consequently, to generate a token τ for word ω, $\mathcal{U}$ executes algorithm Token as follows:

- On inputs of the word ω, the file identifier fid and the MAC key K_{mac}, the algorithm Token first computes $\mathsf{h} = \mathcal{H}_{\mathsf{mac}}(K_{\mathsf{mac}}, \omega||\mathsf{fid})$. Then it calls the algorithm $\mathsf{Query}_{\mathsf{oprf}}$ which on input of h outputs an OPRF query $\mathcal{Q}_{\mathsf{oprf}}$ to evaluate $f_\delta(\mathsf{h}) = \mathfrak{g}^{1/(\delta+\mathsf{h})}$. Next, the algorithm Token forwards the OPRF query $\mathcal{Q}_{\mathsf{oprf}}$ to cloud server $\mathcal{S}$.
- Upon receipt of $\mathcal{Q}_{\mathsf{oprf}}$, $\mathcal{S}$ calls the OPRF algorithm $\mathsf{Response}_{\mathsf{oprf}}$. This algorithm uses the secret OPRF key δ and the OPRF query $\mathcal{Q}_{\mathsf{oprf}}$ to output an OPRF response $\mathcal{R}_{\mathsf{oprf}}$.

  Here instead of sending the OPRF response $\mathcal{R}_{\mathsf{oprf}}$ in clear to $\mathcal{U}$, $\mathcal{S}$ will obfuscate it in such a way that only an authorized (i.e. non-revoked) user will be able to derive $\mathcal{R}_{\mathsf{oprf}}$. This obfuscation is performed as follows:

  - $\mathcal{S}$ picks randomly a symmetric encryption key K'_{enc} and encrypts the OPRF response $\mathcal{R}_{\mathsf{oprf}}$ using K'_{enc} and the *semantically secure* encryption Enc. This will result in a ciphertext $C' = \mathsf{Enc}(K'_{\mathsf{enc}}, \mathcal{R}_{\mathsf{oprf}})$.
  - Then it computes a CP attribute-based encryption $\mathcal{C}_{\mathsf{enc}} = \mathsf{Enc}_{\mathsf{abe}}(K'_{\mathsf{enc}}, \mathsf{AP})$ of the encryption key K'_{enc} under the access policy AP of the data owner O.

  Notice that in this manner, we make sure that only authorized users will be able to decrypt the OPRF response and therewith obtain the token $\tau = f_\delta(\mathsf{h}) = \mathfrak{g}^{1/(\delta+\mathsf{h})}$ necessary to perform the word search.

  At the end of this step, $\mathcal{S}$ forwards the ciphertexts C' and $\mathcal{C}_{\mathsf{enc}}$ to authorized user $\mathcal{U}$.
- On receiving the ciphertexts C' and $\mathcal{C}_{\mathsf{enc}}$, the algorithm Token first decrypts $\mathcal{C}_{\mathsf{enc}}$ using the credentials cred that $\mathcal{U}$ obtained from the CA and gets $K'_{\mathsf{enc}} = \mathsf{Dec}_{\mathsf{abe}}(\mathcal{C}_{\mathsf{enc}}, \mathsf{cred})$. Then it computes the OPRF response $\mathcal{R}_{\mathsf{oprf}}$ by decrypting the ciphertext $\mathcal{C}_{\mathsf{enc}}$ using the secret key K'_{enc}. Next, the algorithm Token calls the OPRF algorithm $\mathsf{Response}_{\mathsf{oprf}}$ which takes as input $\mathcal{R}_{\mathsf{oprf}}$ and outputs consequently the word search token $\tau = f_\delta(\mathsf{h}) = \mathfrak{g}^{1/(\delta+\mathsf{h})}$.

Search Query After obtaining the token τ corresponding to the word ω, $\mathcal{U}$ runs the algorithm Query which first computes $H(\tau) = (x,y)$ and $H'(\tau) = (x',y')$. Then, as in the previous solution, it computes two PIR queries $(\vec{\alpha}, \vec{\alpha}')$ to retrieve the x^{th} and the x'^{th} row of a (k,l) binary matrix and sends the word search query $\mathcal{Q} = (\vec{\alpha}, \vec{\alpha}')$ to cloud server $\mathcal{S}$.

Search response On receiving $\mathcal{U}$'s search query $\mathcal{Q} = (\vec{\alpha}, \vec{\alpha}')$, cloud server $\mathcal{S}$ runs algorithm Response which computes the two sets of t PIR responses $\mathbb{R} = \{\vec{\beta}_1, \vec{\beta}_2, ..., \vec{\beta}_t\}$ and $\mathbb{R}' = \{\vec{\beta}'_1, \vec{\beta}'_2, ..., \vec{\beta}'_t\}$ such that for all $1 \leq j \leq t$:

$$\begin{aligned}\vec{\beta}_j &= \mathsf{PIRResponse}(\vec{\alpha}, \mathcal{M}_j) = \vec{\alpha} \cdot \mathcal{M}_j \\ \vec{\beta}'_j &= \mathsf{PIRResponse}(\vec{\alpha}', \mathcal{M}'_j) = \vec{\alpha}' \cdot \mathcal{M}'_j\end{aligned}$$

$\mathcal{S}$ sends then its word search response $\mathcal{R} = \{\mathbb{R}, \mathbb{R}'\}$ to $\mathcal{U}$.

Verification To verify whether ω is in the encrypted file C, the authorized user $\mathcal{U}$ runs the original algorithm Verify as described in Section 4.2.3. But after obtaining $\vec{b}$ and $\vec{b}'$, algorithm Verify computes the hash $\mathcal{H}(\tau)$ instead of the hash $\mathcal{H}(\mathsf{h})$ and checks accordingly whether $\vec{b} = \mathcal{H}(\tau)$ or $\vec{b}' = \mathcal{H}(\tau)$. If it is the case, then Verify outputs 1 meaning that $\omega \in F$; otherwise, Verify outputs 0.

5.2.5 Revocation

For sake of simplicity, we assume that the data owner O revokes attributes $\mathsf{att}_i \in \mathbb{A}$ instead of individual users $\mathcal{U}$. We believe that this assumption is sufficient in the context of our application as described in Section 2, where the data owner delegates the word search capabilities to regulators or auditors that are not identified by their identities but by their attributes.

Now to revoke an attribute att_i, O runs the algorithm Revoke which outputs a new access policy AP' that will be given to the cloud server $\mathcal{S}$. For instance, if we assume that the initial access policy AP of O states that auditors from EU and the US can perform word search on O's files, then a revocation of attribute US will lead to a new access policy AP' that says that only auditors from the EU can perform word search. In this manner, auditors from the US will no longer have access to O's file.

6 PRIVACY ANALYSIS

In this section, we briefly analyze the privacy properties of the proposed scheme. The interested reader may refer to the appendix for a more formal analysis.

6.1 Storage Privacy

Our scheme insures storage privacy thanks to the use of *semantically secure encryption* and *message authentication code* during the upload phase. Actually, the semantically secure encryption assures that cloud server $\mathcal{S}$ cannot derive any information about the file F from its encryption C. In addition, by computing MACs that not only depend on the words present in the file but also on its unique identifier, we ensure that the index I does not leak any information about the outsourced file. Notably, cloud server $\mathcal{S}$ cannot tell whether two outsourced files have words in common or not, based on their indexes.

6.2 Query Privacy

Query privacy is assured by the use of both *OPRF* and *PIR*. On the one hand, OPRF allows authorized user $\mathcal{U}$ to generate a word search token τ without disclosing anything to cloud server $\mathcal{S}$ about the word ω that $\mathcal{U}$ is interested in. On the other hand, PIR enables $\mathcal{U}$ to preform word search on $\mathcal{S}$'s database while making sure that $\mathcal{S}$ learns nothing about the word search queries or their corresponding results.

6.3 Privacy against Revoked Users

Since in this paper, we only focus on the case where data owner O revokes attributes instead of individual users, it follows that using for instance the CP-ABE scheme proposed by Bethencourt et al. [2] suffices to ensure efficient revocation. As shown in the previous section, revocation is achieved by updating the access policy associated with file F and by exploiting the properties of OPRF: Obfuscating $\mathcal{S}$'s responses during the token generation phase (cf. Section 5.2) stops a revoked user from deriving new word search tokens and consequently from verifying $\mathcal{S}$'s responses.

Note also that even if revoked users gain access to the cloud server's database, they cannot decrypt the content of the outsourced files as they do not have access to the encryption key K_{enc}. All they can achieve is performing a dictionary attack on the index I using the MAC key K_{mac} and the OPRF secret key δ, which can be computationally intensive.

7 PERFORMANCE EVALUATION

During the upload phase, the data owner is only required to encrypt the file to be outsourced using a symmetric encryption and to compute a MAC h_i for each word $\omega_i \in \mathcal{L}_\omega$. On the other hand, the cloud server computes the OPRFs (i.e. tokens) $\tau_i = f_\delta(h_i)$ and builds the corresponding index I by following the algorithm of Cuckoo hashing. Although the computation of the OPRF proposed in [11] may be deemed computationally demanding as it calls for exponentiations, it can be efficiently parallelized at the cloud server. Actually, if the cloud server possesses N machines for instance, it can provide each one of its machines with $\frac{1}{N}$ fraction of the list of MACs $\mathcal{L}_H = \{h_1, h_2, ..., h_n\}$ supplied by the data owner. Each machine will consequently compute $\frac{n}{N}$ exponentiations whose results will be given back to the cloud server to construct the index I.

While some would argue that using PIR to compute the responses of the cloud server to word search queries is computationally intensive, we note that this computation consists of *matrix multiplications* which can easily be parallelized. Actually, the cloud server can store at each one of its machine $\frac{1}{N}$-fraction of the binary matrices $\{\mathcal{M}_j\}$ and $\{\mathcal{M}'_j\}$. Upon receipt of a word search query, $\mathcal{S}$ forwards the PIR queries it receives to its N machines which accordingly compute the corresponding PIR responses.

Furthermore, we emphasize that in this paper we employ PIR to retrieve a hash of word search tokens instead of their actual values. This fact drastically enhances the computation and the communication performances of our scheme. For example, if we instantiate the OPRF in the token generation phase with the OPRF presented in [11], then we will end up with tokens of size 1024 bits. This means that if we retrieve the actual values of the token to perform word search, then each search query will consist of retrieving 1024 bits which is far from being practical. Instead in our protocol, each search operation consists of fetching t-bit (t is typically 80) hash. We note also that setting the size (k,l) of the matrices $\{\mathcal{M}_j\}$ and $\{\mathcal{M}'_j\}$ to $(\sqrt{tn}, \sqrt{\frac{n}{t}})$ results in a minimal communication cost of $O(\sqrt{tn})$.

Finally, we stress that contrary to related work [6], revocation in our protocol does not require the re-encryption of the outsourced files. Rather, it only calls for an update of the access policy of the data owner at the cloud server.

8 RELATED WORK

As opposed to the proposed solution, most of existing word search mechanisms be them asymmetric [1, 4, 17] or symmetric [6, 10, 12, 15] seem to guarantee query privacy partially: Indeed, in these solutions, although the outsourced data and queries are encrypted, the cloud can discover the response to any encrypted query. Furthermore very few of current solutions [6, 7] propose the ability to delegate the search operation; unfortunately, these solutions provide the authorized user with the data encryption key and therefore revocation of a user requires the re-encryption of the entirely outsourced data and the distribution of this new key to the authorized users.

The first solution which transforms an original PIR mechanism into a privacy preserving word-search solution is proposed by Chor et. al. in [5]. Similarly to our solution, in [5], the owner of the data constructs an index based on all distinct words in the outsourced file. This index is a hash-table that is filled according to the perfect hashing algorithm of Fredman et al. [8]. Our solution outperforms the solution in [5] thanks to the use of Cuckoo hashing instead of perfect hashing. Namely, in the scheme of [5], a word search query consists of three PIR queries, whereas in our protocol it is composed of two PIR queries. Additionally, the PIR queries in the case of Cuckoo hashing are independent. This implies that the server can execute the two PIR instances in parallel to respond to the word search query.

Another solution that resembles the proposed solution is PRISM [3] where the cloud constructs some binary matrices in which each cell represents one or more words without knowing their content and the owner sends PIR requests to retrieve the content of one of these cells. Thanks to the use of Cuckoo hashing, our solution outperforms the original PRISM mechanism without lowering the security level. PRISM defines a matrix in which each cell corresponds to one or more words; therefore, two words can turn out to be represented by the same cell. In order to decrease the probability of such collisions, the data owner send multiple (q) queries for the same word. In the newly proposed mechanism, the probability of collisions within the binary matrices is 0 and the data owner and/or the authorized user need to send a single query for each word. Additionally, PRISM does not offer any delegation capability and a straightforward delegation operation would require the distribution of the data encryption key to authorized users which can increase privacy risks.

9 CONCLUSION

We introduced a protocol for privacy preserving delegated word search in the cloud. This protocol allows a data owner to outsource its encrypted data to a cloud server, while empowering the data owner with the capability to delegate word search operations to third parties. By employing keyed hash functions and oblivious pseudo-random functions, we ensure that authorized users only learn whether a given word is in the outsourced files or not. In addition, we use private information retrieval to make sure that the cloud server cannot infer any information about the outsourced files from the execution of the word search protocol. Furthermore, we combine attribute-based encryption and oblivious pseudo-random functions to accommodate efficient revocation. Finally, the data owner in our protocol is only required to perform symmetric operations, whereas the computationally intensive computations are performed by the cloud server, and they can easily be parallelized.

Bibliography

[1] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *Proceedings of the 27th Annual International Cryprology Conference on Advances in Cryptology, (CRYPTO'07)*, pages 535–552, 2007.

[2] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pages 321–334, 2007.

[3] E.-O. Blass, R. di Pietro, R. Molva, and M. Önen. PRISM - Privacy-Preserving Search in MapReduce. In *Proceedings of the 12th Privacy Enhancing Technologies Symposium (PETS 2012)*. LNCS, July 2012.

[4] D. Boneh, G. G. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proceedings of Eurocrypt 2004*, volume 3027, pages 506–522. LNCS, 2004.

[5] B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords, 1997.

[6] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 79–88. ACM, 2006. ISBN 1-59593-518-5.

[7] C. Dong, G. Russello, and N. Dulay. Shared and searchable encrypted data for untrusted servers. In *Proceeedings of the 22nd annual IFIP WG 11.3 working conference on Data and Applications Security*, pages 127–143, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-70566-6. doi: 10.1007/978-3-540-70567-3_10. URL http://dx.doi.org/10.1007/978-3-540-70567-3_10.

[8] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a Sparse Table with 0(1) Worst Case Access Time. *J. ACM*, 31(3):538–544, June 1984. ISSN 0004-5411.

[9] M.J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Proceedings of the Second international conference on Theory of Cryptography*, TCC'05, pages 303–324, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-24573-1, 978-3-540-24573-5.

[10] P. Golle, J. Staddon, and B. Waters. Secure conjunctive keyword search over encrypted data. In M. Jakobsson, M. Yung, and J. Zhou, editors, *Proc. of the 2004 Applied Cryptography and Network Security Conference*, pages 31–45. LNCS 3089, 2004.

[11] S. Jarecki and X. Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *Theory of Cryptography*, volume 5444 of *Lecture Notes in Computer Science*, pages 577–594. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-00456-8.

[12] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 965–976, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1651-4. doi: 10.1145/2382196.2382298. URL http://doi.acm.org/10.1145/2382196.2382298.

[13] R. Pagh. On the cell probe complexity of membership and perfect hashing. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, pages 425–432, New York, NY, USA, 2001. ACM. ISBN 1-58113-349-9.

[14] R. Pagh and F.F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[15] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, pages 44–, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0665-8. URL http://dl.acm.org/citation.cfm?id=882494.884426.

[16] J. Trostle and A. Parrish. Efficient Computationally Private Information Retrieval from Anonymity or Trapdoor Groups. In *Proceedings of Conference on Information Security*, pages 114–128, Boca Raton, USA, 2010.

[17] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters. Building an encrypted and searchable audit log. In *Proceedings of NDSS'04*, 2004.

APPENDIX

Storage Privacy

Theorem 1. *The protocol presented in Section 5 provides storage privacy under the semantic security of the encryption* Enc *and the security of the MAC* $\mathcal{H}_{\mathsf{mac}}$ *employed during the upload phase to encrypt the outsourced files and to build the corresponding indexes respectively.*

Due to space limitation, we omit the proof of this theorem. Notice that all the server has access to is the semantically secure encryption and the indexes that are computed using keyed hashes. Therefore, as discussed in Section 6, the cloud server cannot derive any information about the outsourced files.

Query Privacy

Theorem 2. *The protocol described in Section 5 achieves query privacy under the security of the trapdoor PIR and the security of the OPRF.*

Before introducing the full proof of the above theorem, we present here briefly a formal definition of the security properties of Private Information Retrieval.

Private Information Retrieval. Let $\mathcal{O}_{\mathsf{pir}}$ be an oracle that takes as input an (k,l) binary matrix $\mathcal{M}$ and two positions (x_0,y_0) and (x_1,y_1), flips a coin $b_{\mathsf{pir}} \in \{0,1\}$ and returns a PIR query to fetch the bit at position $(x_{b_{\mathsf{pir}}}, y_{b_{\mathsf{pir}}})$ from $\mathcal{M}$ as depicted in Section 4.1.1.

Let $\mathcal{S}$ be an adversary that submits two positions $\mathsf{pos}_0^* = (x_0^*, y_0^*)$ and $\mathsf{pos}_1^* = (x_1^*, y_1^*)$ to $\mathcal{O}_{\mathsf{pir}}$ to get a PIR query for $\mathsf{pos}_{b_{\mathsf{pir}}}^*$. Upon receipt of the PIR query, $\mathcal{S}$ outputs a guess b_{pir}^* for the bit b_{pir}.

Let $\Pi_{\mathsf{success}}^{\mathcal{A}}$ denote the probability that $\mathcal{S}$ outputs a correct guess for b_{pir} (i.e. $b_{\mathsf{pir}}^* = b_{\mathsf{pir}}$). We say that a PIR scheme is secure if for any adversary $\mathcal{S}$, $\Pi_{\mathsf{success}}^{\mathcal{A}} \leq \frac{1}{2} + \varepsilon_{\mathsf{pir}}$, where $\varepsilon_{\mathsf{pir}}$ is a negligible function.

Proof. Assume there exists an adversary $\mathcal{S}$ which is able to break the query privacy of our protocol with a non-negligible advantage ε. We here describe an adversary $\mathcal{A}$ which uses $\mathcal{S}$ to break the security of the PIR with a non-negligible $\varepsilon_{\mathsf{pir}}$ as long as the OPRF used in the token generation phase is secure.

Construction. To simulate the query privacy game for adversary $\mathcal{S}$, $\mathcal{A}$ picks an encryption key K_{enc}, a MAC key K_{mac} and an OPRF secret key δ which it provides to $\mathcal{S}$.

When $\mathcal{S}$ enters the learning phase, $\mathcal{A}$ simulates the oracles $\mathcal{O}_{\mathsf{encrypt}}$, $\mathcal{O}_{\mathsf{index}}$ and $\mathcal{O}_{\mathsf{search},\mathcal{S}}$ using the secret keys K_{enc} and K_{mac}.

At the end of the learning phase as shown in Algorithm 3, $\mathcal{S}$ outputs a challenge file F^* and a pair of challenge words (ω_0^*, ω_1^*).

In the challenge phase (see Algorithm 4), $\mathcal{A}$ first encrypts and builds the index of file F^*. Without loss of generality, we denote the resulting ciphertext and index C^* and $I^* = \{\mathbb{M}^*, \mathbb{M}'^*, H^*, H'^*\}$ respectively.

Now to simulate the oracle $\mathcal{O}_{\mathsf{view},\mathcal{S}}$ in the challenge phase, $\mathcal{A}$ executes the following steps:

- First $\mathcal{A}$ engages in a token generation with $\mathcal{S}$ by performing the OPRF protocol for word ω_0^*.
- Then, it provides the oracle $\mathcal{O}_{\mathsf{pir}}$ with a binary matrix $\mathcal{M}$ of size (k,l) and two positions $(x_0^*, y_0^*) = H(f_\delta(\mathsf{h}_0^*))$ and $(x_1^*, y_1^*) = H(f_\delta(\mathsf{h}_1^*))$, where $\mathsf{h}_i^* = \mathcal{H}(K_{\mathsf{mac}}, \omega_i^* || \mathsf{fid})$, $i \in \{0,1\}$. The PIR oracle picks randomly a bit $b_{\mathsf{pir}} \in \{0,1\}$ and returns a PIR query for the position (x_b^*, y_b^*). This PIR will be used by adversary $\mathcal{A}$ to fetch a bit from the matrices $\mathbb{M}^*$.
- Finally, $\mathcal{A}$ prepares another PIR query to retrieve the element at position $(x_0'^*, y_0'^*) = H'(f_\delta(\mathsf{h}_0^*))$ from the matrices $\mathbb{M}'^*$.

At the end of the challenge phase, adversary $\mathcal{S}$ outputs a bit b^*, such that $b^* = 0$ if $\mathcal{S}$ thinks that it has been queried for word ω_0^*; $b^* = 1$ otherwise.

Notice that since we use OPRF to generate search tokens, the token generation phase does not leak any information about the queried word ω_0^*. Therefore, to break query privacy $\mathcal{S}$ has to rely on the PIR queries it receives during the word search phase.

We remark also that when the PIR oracle chooses the bit $b_{\mathsf{pir}} = 0$, then the view $\mathsf{view}_{\mathcal{S}}^*$ of $\mathcal{S}$ that $\mathcal{A}$ simulates above is indistinguishable from the view of $\mathcal{S}$ during a word search protocol execution for word ω_0^* with an actual authorized user. This implies that whenever $b_{\mathsf{pir}} = 0$, $\mathcal{S}$ will be able to output a correct guess for the queried word ω_0^* (i.e. $\mathcal{S}$ will output $b^* = 0$) with a non-negligible advantage ε. However, if $\mathcal{O}_{\mathsf{pir}}$ picks $b_{\mathsf{pir}} = 1$, then the view of $\mathcal{S}$ will correspond to an OPRF execution for ω_0^* and two PIR queries one for ω_0^* and the other for ω_1^*.

Assume that adversary $\mathcal{S}$ can detect with some probability π that the PIR queries it receives when $b_{\mathsf{pir}} = 1$ do not correspond to the same word, and as a result, it aborts the query privacy game.

It follows that to break the security of the PIR, $\mathcal{A}$ outputs $b_{\mathsf{pir}}^* = b^*$ when $\mathcal{S}$ does not stop the query privacy game; otherwise, $\mathcal{A}$ outputs $b_{\mathsf{pir}}^* = 1$. Actually, when $\mathcal{S}$ aborts the game, this means that $\mathcal{S}$ has received two incompatible PIR queries, i.e., the query generated by the PIR oracle $\mathcal{O}_{\mathsf{pir}}$ correspond to position $(x_1^*, y_1^*) = H(f_\delta(\mathsf{h}_1^*))$.

$\square$

Privacy against Revoked Users

Theorem 3. *The protocol presented in Section 5 ensures privacy against revoked users under the indistinguishability of the OPRF, the security of CP-ABE and the semantic security of encryption* Enc.

Before presenting the formal proof of the above theorem, we provide below a brief formalization of OPRF indistinguishability.

OPRF indistinguishability. Let $\mathcal{A}$ be an adversary against the indistinguishability of the OPRF. The goal of adversary $\mathcal{A}$ is given $\mathsf{h}^* \in \{0,1\}^\kappa$ and σ^*, it should be able to tell whether $\sigma^* = f_\delta(\mathsf{h}^*)$ or not.

Accordingly, $\mathcal{A}$ is given access to the following oracles:

- $\mathcal{O}_{\mathsf{oprf}}^F(h) \rightarrow \mathcal{R}_{\mathsf{oprf}}$: It is an oracle that acts as the sender in the OPRF protocol. It takes as input an OPRF query Q_{oprf} for some $h \in \{0,1\}^\kappa$ and outputs an OPRF response $\mathcal{R}_{\mathsf{oprf}}$.
- $\mathcal{O}_{\mathsf{oprf}}^I(h) \rightarrow \sigma$: It is an oracle that on input of h, selects randomly a bit $b_{\mathsf{oprf}} \in \{0,1\}$. If $b = 1$, then $\mathcal{O}_{\mathsf{oprf}}^I$ outputs $\sigma = f_\delta(h)$; otherwise it sets σ to a randomly generated value.

To break the indistinguishability of the OPRF f_δ, $\mathcal{A}$ is allowed to issue OPRF queries to the oracle $\mathcal{O}_{\mathsf{oprf}}^F$ for a polynomial number of values h_i. Next, $\mathcal{A}$ chooses $\mathsf{h}^* \notin \{h_i\}$ and submits h^* to the oracle $\mathcal{O}_{\mathsf{oprf}}^I$. Upon receipt of h^*, $\mathcal{O}_{\mathsf{oprf}}^I$ selects a random bit b_{oprf} and outputs σ^* as shown above.

We say that $\mathcal{A}$ succeeds in breaking the indistinguishability of the OPRF if given h^* and σ^*, $\mathcal{A}$ is able to output a correct guess b_{oprf}^* for the bit b_{oprf}. That is, if $\mathcal{A}$ is able to tell whether $\sigma^* = f_\delta(\mathsf{h})$ or whether it was randomly generated. Hereafter, let $\Pi_{\mathsf{success}}^{\mathcal{A}}$ denote the probability that $\mathcal{A}$ outputs a correct guess for b_{oprf} (i.e. the probability that $b_{\mathsf{oprf}}^* = b_{\mathsf{oprf}}$).

We recall that an OPRF is said to insure indistinguishability if for any adversary $\mathcal{A}$, $\Pi_{\mathsf{success}}^{\mathcal{A}} \leq \frac{1}{2} + \varepsilon_{\mathsf{oprf}}$, where $\varepsilon_{\mathsf{oprf}}$ is a negligible function.

Proof. Assume there exists a user $\mathcal{U}$ that breaks the privacy against revoked users with a non-negligible advantage ε. We show in the following how to construct an adversary $\mathcal{A}$ which uses $\mathcal{U}$ to break the indistinguishability of the OPRF with a non-negligible advantage $\varepsilon_{\mathsf{oprf}}$.

Construction. To break the indistinguishability property of the OPRF, adversary $\mathcal{A}$ picks a symmetric encryption key K_{enc} and a MAC key K_{mac} that it uses to encrypt file F and to compute the MACs necessary to build the index I respectively. Without loss of generality, we denote C the encryption of the challenge file F and $\mathcal{L}_H = \{h_1, h_2, ..., h_n\}$ the MACs of the words present in F. To complete the construction of the index I, adversary $\mathcal{A}$ invokes the oracle $\mathcal{O}^F_{\mathsf{oprf}}$ for each $h_i \in \mathcal{L}_H$. This oracle returns for each $h_i \in \mathcal{L}_H$ the corresponding OPRF response from which $\mathcal{A}$ derives $f_\delta(h_i)$.

When $\mathcal{U}$ enters the learning phase, $\mathcal{A}$ first simulates the oracle $\mathcal{O}_{\mathsf{delegate}}$ by giving $\mathcal{U}$ the MAC key K_{mac} and the public parameters of the OPRF. Besides to simulate the oracle $\mathcal{O}_{\mathsf{search},u}$, $\mathcal{A}$ proceeds as follows:

- If $\mathcal{O}_{\mathsf{search},u}$ receives an OPRF query for some MAC $h_i = \mathcal{H}_{\mathsf{mac}}(K_{\mathsf{mac}}, \omega_i || \mathsf{fid})$ before the revocation of $\mathcal{U}$, then $\mathcal{A}$ forwards this query to the oracle $\mathcal{O}^F_{\mathsf{oprf}}$ which in turn outputs a matching OPRF response. Next, $\mathcal{A}$ obfuscates this OPRF response using CP-ABE and the encryption Enc as was shown in the protocol.
- If $\mathcal{O}_{\mathsf{search},u}$ receives an OPRF query after the revocation of $\mathcal{U}$, then $\mathcal{A}$ generates randomly an OPRF response which it obfuscates using CP-ABE and the semantically secure encryption Enc. By combining CP-ABE and semantically secure symmetric encryption, $\mathcal{A}$ makes sure that revoked user $\mathcal{U}$ cannot tell whether it is receiving an actual OPRF response or a randomly generated one.

Then, when $\mathcal{U}$ issues a PIR query to perform word search, then $\mathcal{A}$ computes its PIR response as expected by $\mathcal{U}$.

Finally, $\mathcal{A}$ simulates $\mathcal{O}_{\mathsf{revoke}}$ by modifying the access policy in such a way that $\mathcal{U}$ is no longer allowed to perform lookups.

At the end of the learning phase, $\mathcal{U}$ outputs a challenge word ω^* for which it did not issue a search query when it was still authorized.

Now if ω^* was already in the challenge file F, then $\mathcal{A}$ aborts the game, otherwise it proceeds with the challenge phase. In the challenge phase, $\mathcal{A}$ simulates the oracle $\mathcal{O}_{\mathsf{chal},u}$ as depicted below:

- $\mathcal{A}$ first computes the MAC $\mathsf{h}^* = \mathcal{H}_{\mathsf{mac}}(K_{\mathsf{mac}}, \omega^* || \mathsf{fid})$ and calls the oracle $\mathcal{O}^I_{\mathsf{oprf}}$ with h^*. This oracle flips a coin b_{oprf} and returns σ^* as depicted above. $\mathcal{A}$ then inserts σ^* into index the challenge index I^*.
- Upon receipt of the OPRF query from $\mathcal{U}$, $\mathcal{A}$ generates randomly an OPRF response which it obfuscates using CP-ABE and the encryption Enc.
- When $\mathcal{A}$ receives the PIR queries from $\mathcal{U}$, it computes its PIR response on index I^* as expected by $\mathcal{U}$.

At the end of the challenge phase, $\mathcal{U}$ outputs a bit b^*. To break the indistinguishability of OPRF, $\mathcal{A}$ outputs $b^*_{\mathsf{oprf}} = b^*$.

Note that on the one hand, if $b^* = 1$ then this means that $f_\delta(\mathsf{h}^*)$ is in I^* (i.e. $\sigma^* = f_\delta(\mathsf{h}^*)$). On the other hand if $b^* = 0$ then this entails that $\sigma^* \neq f_\delta(\mathsf{h}^*)$ (i.e. σ^* was generated randomly).

We point out here that $\mathcal{A}$ breaks the indistinguishability of OPRF if it does not abort the game of privacy against revoked users and if $\mathcal{U}$ outputs a correct guess b^*. This occurs with probability $\frac{1}{2} + \varepsilon\pi$, where π is the probability that $\mathcal{A}$ does not stop the game.

To summarize, if there is an adversary $\mathcal{U}$ which breaks the privacy against revoked users with a non-negligible advantage ε, then there exists another adversary $\mathcal{A}$ which breaks the indistinguishability of OPRF with a non-negligible advantage $\varepsilon_{\mathsf{oprf}} = \varepsilon\pi$.

$\square$