RESEARCH ARTICLE

Attribute-Based Proxy Re-Encryption with Keyword Search

Yanfeng Shi[1]*, Jiqiang Liu[1], Zhen Han[1], Qingji Zheng[3], Rui Zhang[2], Shuo Qiu[1]

1. School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China, **2.** The State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China, **3.** Department of Computer Science, University of Texas at San Antonio, San Antonio, Texas, United States of America

*schwannfeng@bjtu.edu.cn

Citation: Shi Y, Liu J, Han Z, Zheng Q, Zhang R, et al. (2014) Attribute-Based Proxy Re-Encryption with Keyword Search. PLoS ONE 9(12): e116325. doi:10.1371/journal.pone.0116325

Editor: Cheng-Yi Xia, Tianjin University of Technology, China

Received: July 30, 2014

Accepted: December 4, 2014

Published: December 30, 2014

Data Availability: The authors confirm that all data underlying the findings are fully available without restriction. All relevant data are within the paper.

Funding: This work is supported by the 111 project, Program for New Century Excellent Talents in University (NCET-11-0565), the Fundamental Research Funds for the Central Universities (2012JBZ010) and PCSIRT (No.IRT 201206). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The authors have declared that no competing interests exist.

Abstract

Keyword search on encrypted data allows one to issue the search token and conduct search operations on encrypted data while still preserving keyword privacy. In the present paper, we consider the keyword search problem further and introduce a novel notion called *attribute-based proxy re-encryption with keyword search* ($\mathcal{ABRKS}$), which introduces a promising feature: In addition to supporting keyword search on encrypted data, it enables data owners to delegate the keyword search capability to some other data users complying with the specific access control policy. To be specific, $\mathcal{ABRKS}$ allows (i) the data owner to outsource his encrypted data to the cloud and then ask the cloud to conduct keyword search on outsourced encrypted data with the given search token, and (ii) the data owner to delegate other data users keyword search capability in the fine-grained access control manner through allowing the cloud to re-encrypted stored encrypted data with a re-encrypted data (embedding with some form of access control policy). We formalize the syntax and security definitions for $\mathcal{ABRKS}$, and propose two concrete constructions for $\mathcal{ABRKS}$: key-policy $\mathcal{ABRKS}$ and ciphertext-policy $\mathcal{ABRKS}$. In the nutshell, our constructions can be treated as the integration of technologies in the fields of attribute-based cryptography and proxy re-encryption cryptography.

Introduction

Cloud computing platforms assemble vast computational resources and make them available to users as a service. The cloud users can outsource their heavy computation tasks and/or storage to cloud providers while still enjoying promising properties, e.g., low maintenance cost and pervasive accessing. While it is promising, cloud computing also confronts many challenges against data privacy/system vulnerabilities [1–3] and service quality [4, 5]. One possible solution to prevent these problems is to use the private cloud, where the underlying infrastructure (i.e., servers, network and storage) is owned and operated by the cloud users themselves. However, this might depress the benefits bringing from the cloud computing, when comparing with the public cloud that is more reliable, elastic (i.e., computational resources can be increased and decreased quickly) and cost-saving. As such, individual and organizations are considering migrating from their owned infrastructure to the public cloud.

In order to preserve data privacy against any possible attacks in the public cloud, it is inevitable for data owners to encrypt their data before outsourcing it to the cloud, which might hinder the data usage. For example, how the data owner can search on their outsourced encrypted data? How the data owner can delegate his search capability to other users in a fine-grained manner? In this paper, we continue the line of keyword search on encrypted data and attempt to solve the above questions simultaneously.

To explain the motivation for solving the above questions, we consider the following motivational application: The data owner, say Alice, encrypted her personal health data that was collected by sensors attached her and outsourced the encrypted data to the cloud. In order to facilitate the examination on health condition, Alice may need to share the encrypted data with professionals, e.g. doctors that work in some specific department, so that the professionals can retrieve qualified records from the cloud. In order to assure that only certain professionals satisfying some policy can conduct keyword search and retrieve corresponding encrypted data of their interests, Alice needs to delegate keyword search capability by specifying the fine-grained access control policy.

A straightforward solution toward the above questions can work as follows: the data owner encrypts his data with attribute-based encryption, and issues proper keys to data users so that only authorized data users can access these encrypted data. Unfortunately, solutions based on attribute-based encryption in the literature do not support keyword search. That is, even satisfying the access control policy, the authorized user has to download entire encrypted data, rather than portion of encrypted data of his interest, which will bring in huge communication overhead. In light of this, we propose a novel notion, dubbed attributed-based proxy re-encryption with keyword search (ABRKS), allowing data owners to grant keyword search capability to authorized users complying with access control policies.

Our Contribution

We introduce a novel notion called attribute-based proxy re-encryption with keyword search (ABRKS), which allows a data owner to delegate keyword search capability over his encrypted data to authorized users by while complying with access control policies. We formally define its syntax and rigorously formalize the security definitions. We present two flavors of ABRKS constructions, key-policy ABRKS and ciphertext-policy ABRKS, the security of which are based on the standard Multilinear Decisional Diffie-Hellman Assumption in the random oracle model. Our solutions perfectly solve the motivation example and enjoy three distinctive properties: (i) The data owner could conduct keyword search on outsourced encrypted data; (ii) The data owner could delegate keyword search capability to users by specifying fine-grained access control policies so that only authorized users satisfying the access control policy can conduct keyword search; and (iii) There is no interaction happening between data owners and users. Moreover, the tedious work, e.g., performing keyword search and re-encrypting encrypted data, can be outsourced to the cloud without compromising data privacy.

Related Work

Here we briefly survey the works that are relevant to the problem we attempt to solve in this paper, while cannot solve it. We summarize the features of the most relevant techniques, proxy re-encryption with keyword search, attribute-based encryption, attribute-based encryption with keyword search and attribute-based proxy re-encryption, and compare them with our ABRKS solutions as shown in Table 1.

Proxy Re-encryption with Keyword Search

Proxy re-encryption with keyword search (PRES) was introduced in [6], which allows a data owner to delegate keyword search capability to other users. PRES was further revised by [7] and/or enhanced by various papers, e.g., [8–11]. However, all these PRES solutions only considered coarse-grained access control enforcement, i.e., delegating the search capability to one specific authorized user. In contrast, we consider the fine-grained access control enforcement when the data owner needs to delegate search capability in this paper.

Attribute-based Encryption

Attribute-based encryption (ABE) was first introduced by [12], which is to specify fine-grained access control on encrypted data, such that only data users with proper credentials (i.e., satisfying the access control policy) can decrypt the ciphertexts. There are two flavors of ABE depending on the manner of associating access control policy: key-policy ABE (KP-ABE) [13–15] associates the decryption key with the access control policy and ciphertext-policy ABE (CP-ABE) associates the ciphertext with the access control policy [16–18]. While ABE allows data owners to achieve fine-grained access control enforcement on encrypted data, unfortunately it cannot support keyword search.

Attribute-based Encryption with Keyword Search

The concept of attribute-based encryption with keyword search (ABKS) was introduced by [19] and [20] independently. It allows data owner to grant search capability to authorized users by specifying fine-grained access control when encrypting plaintext. However, it does not support the data owner delegating

Table 1. Property summary for PRES, ABE, ABPRE, ABKS in the literature and the solution in this paper.

Scheme	Proxy Re-encryption	Keyword Search	Access Control
PRES [6–11]	√	√	×
ABE [12–18]	×	×	√
ABKS [19, 20]	×	√	√
ABPRE [21–26]	√	×	√
ABRKS(Our solution)	√	√	√

doi:10.1371/journal.pone.0116325.t001

search capability to authorized users when encrypted data were stored in the cloud.

Attribute-based Proxy Re-encryption

Attribute-based proxy re-encryption (ABPRE) was introduced by [21] and enriched by [22–26] with various features. However, these solutions do not support the function of keyword search on encrypted data. Generally speaking, the solution in this paper can be regarded as an extension to ABPRE with the feature of keyword search on encrypted data.

Preliminary

Cryptographic Assumptions

Multilinear Maps

The concept of multilinear maps was introduced in [27] and came to reality thanks to [28, 29]. Given a security parameter ℓ and an ℓ-bit prime p, a 4-multilinear map consists of 4 cyclic groups (G_0, G_1, G_2, G_3) of order p, and 3 mappings $e_i : G_0 \times G_i \to G_{i+1}$, $i = 0,1,2$. The 4-multilinear map should satisfy the following properties with respect to i, $i = 0,1,2$: (i) Given that $g_0 \in G_0$ is a generator of G_0, then $g_{i+1} = e_i(g_0, g_i)$ is a generator of G_{i+1}; (ii) $\forall \alpha, \beta \in \mathbb{Z}_p$, $e_i(g_0^{\alpha}, g_i^{\beta}) = e_i(g_0, g_i)^{\alpha\beta}$; and (iii) e_i can be efficiently computed.

4-Multilinear Decisional Diffie-Hellman Assumption (4-MDDH)

Given the 4-multilinear map and $g_0, g_0^a, g_0^b, g_0^c, g_0^w, g_0^r, Z$, where $a, b, c, w, r \overset{R}{\leftarrow} \mathbb{Z}_p$ that are unknown, $Z \overset{R}{\leftarrow} G_3$, $g_1 = e_0(g_0, g_0) \in G_1, g_2 = e_1(g_0, g_1) \in G_2$ and $g_3 = e_2(g_0, g_2) \in G_3$, there exists no probabilistic polynomial algorithm $\mathcal{A}$ that can determine whether $g_3^{abcwr} = Z$ or not with a non-negligible advantage with respect to security parameter ℓ, where the advantage is defined as

$$\begin{aligned} &|\Pr[\mathcal{A}(g_3^{abcwr}, g_0, g_0^a, g_0^b, g_0^c, g_0^w, g_0^r) = 1] \\ &- \Pr[\mathcal{A}(Z, g_0, g_0^a, g_0^b, g_0^c, g_0^w, g_0^r) = 1]|. \end{aligned}$$

Access Control Policy

Linear Secret Sharing Scheme

A linear secret sharing scheme (LSSS) can be used to represent an access control policy P via (M,π), where $\mathrm{M}=(\mathbb{Z}_p)^{l\times k}$ is an $l\times k$ dimensional matrix with entries belonging to $\mathbb{Z}_p$ and $\pi:\{1,\ldots,l\}\rightarrow\mathrm{UAtt}$ is an injective function that maps a row into an attribute. Given an attribute set $\mathrm{S}\subset\mathrm{UAtt}$ where UAtt is the attribute universe, we denote $F(\mathrm{S},\mathrm{P})=1$ if S satisfies the access control policy P. Specifically, an LSSS consists of two algorithms:

Share$((\mathrm{M},\pi),s)$: This algorithm is to distribute a secret value s with respect to the attributes specified by π: It selects $\upsilon_2,\ldots,\upsilon_k\overset{R}{\leftarrow}\mathbb{Z}_p$, sets $\mathbf{v}=(s,\upsilon_2,\ldots,\upsilon_k)$ and computes $\lambda_{\pi(i)}=\mathrm{M}_i\cdot\mathbf{v}$ where M_i is the ith row of M. Then it assigns secret share $\lambda_{\pi(i)}$ to the attribute $\pi(i)$.

Combine$(\mathrm{S},(\lambda_{\pi(i)},\ldots,\lambda_{\pi(l)}),(\mathrm{M},\pi))$: This algorithm is to assemble the secret value from the secret shares associated with respect to the attributes: It selects a subset $I=\{i|\pi(i)\in\mathrm{S}\}$ such that the attribute set $\{\pi(i)|i\in I\}$ satisfies the access control policy (M,π), and then computes the coefficients $c_i,i\in I$ such that $\sum_{i\in I}c_i\mathrm{M}_i=(1,0,\ldots,0)$. The recovered secret will be $\sum_{i\in I}c_i\lambda_{\pi(i)}=s$.

The correctness of algorithm Combine is guaranteed by the following lemma:

Lemma 1 ([17]) *Let* (M,π) *be an* LSSS *representing an access control policy* P. *For all attributes in* S *that do not satisfy* P, *there is a polynomial-time algorithm that outputs vector* $\mathbf{w}=(w_1,\ldots,w_k)\in\mathbb{Z}_p^k$ *such that* $w_1=1$ *and* $M_i\cdot\mathbf{w}=0$ *for all* $i\in[1,\ldots,l]$, *where* $\pi(i)\in\mathrm{S}$.

Definition

System Model

The system model of attribute-based proxy re-encryption with keyword search is shown in Fig. 1, consisting of three parties: the trusted authority, the cloud server and cloud users that can be either data owner or data users wishing to share the data owner's data. The trusted authority is responsible for initiating system public parameters and issuing private keys to cloud users with respect to their attributes. A data owner (say Alice) encrypts her data and the keyword index and outsource the encrypted data and the associated encrypted keyword index to the cloud server. Moreover, the data owner can retrieve encrypted data of her interest by issuing a search token with respect to some keyword to the cloud. On the other hand, the data owner is capable of granting search capability to other authorized users by issuing re-encryption keys (which is associated with access control policies) to the cloud. The cloud server provides storage and computation service for cloud users. Especially, the cloud server can transform the stored encrypted data with re-encryption keys from the data owner, so that the authorized data user (say Bob) is able to generate search tokens and ask the cloud server to conduct keyword search on the re-encrypted data for retrieving encrypted data of his

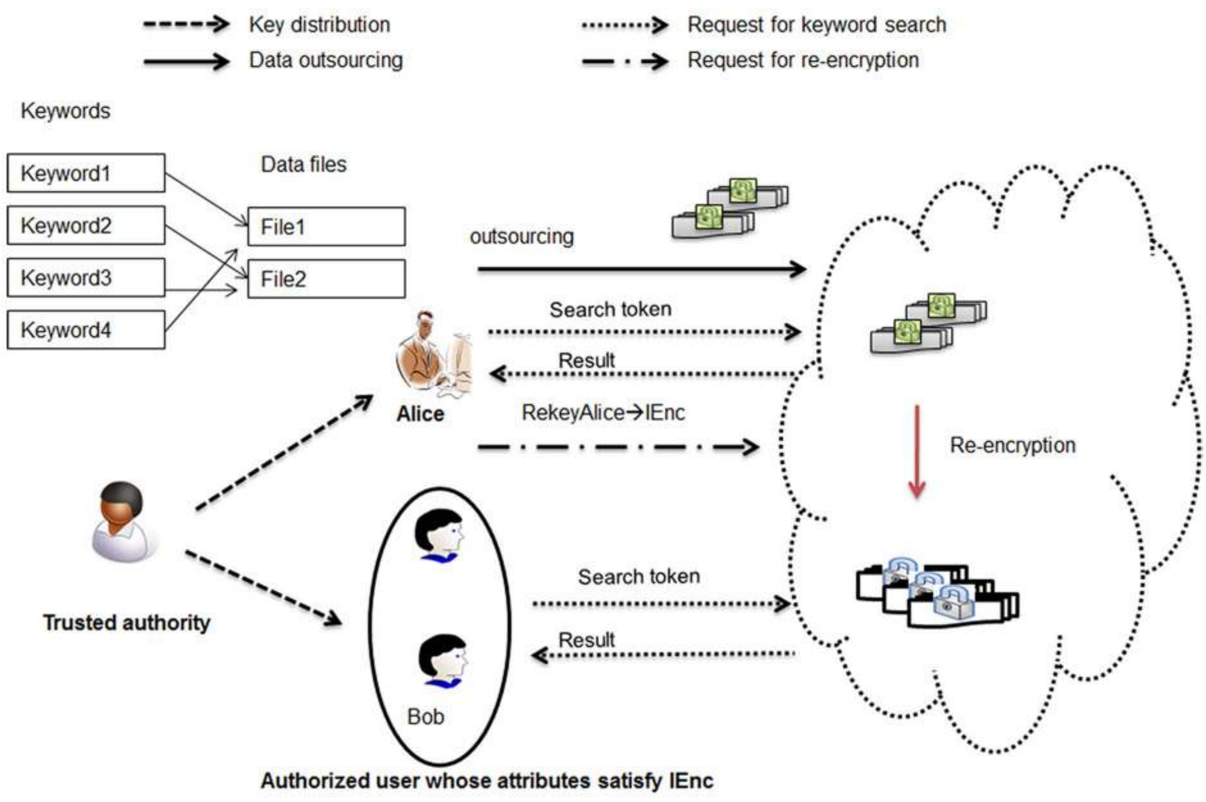

Fig. 1. System model of attribute-based access control for proxy re-encryption with keyword search.

doi:10.1371/journal.pone.0116325.g001

interest. In this model, we assume that the data owner and data users require no direct interaction.

Functional Definition

We now present the formal definition of attribute-based proxy re-encryption with keyword search, which consists of two variants: key-policy ABRKS (KP - ABRKS) whose private keys are associated with access control policies, and ciphertext-policy ABRKS (CP - ABRKS) whose ciphertexts after re-encryption are associated with access control policies. To unify the presentation, let I_{Enc} denote the input of the encryption function ReKeyGen and I_{KeyGen} denote the input of the key generation function KeyGen. Therefore, I_{Enc} and I_{KeyGen} respectively correspond to an attribute set and an access policy in KP - ABRKS, whereas I_{Enc} and I_{KeyGen} respectively correspond to an access policy and an attribute set in CP - ABRKS. We denote $F(I_{\mathrm{KeyGen}},I_{\mathrm{Enc}})=1$ if and only if I_{Enc} satisfies I_{KeyGen} in KP-ABRKS or I_{KeyGen} satisfies I_{Enc} in CP-ABRKS.

To be specific, an ABRKS scheme consists of algorithms as follows:

(param,mk)$\leftarrow$Setup(1^{ℓ}): Taking as input a security parameter ℓ, this algorithm is run by the trusted authority to initiate the public parameter param and a master private key mk.

$\mathrm{sk}_{I_{\mathrm{KeyGen}}}\leftarrow$KeyGen(mk,param,$I_{\mathrm{KeyGen}}$): Taking as input I_{KeyGen}, the master key mk and public parameter param, this algorithm is run by the trusted authority to issue a private key $\mathrm{sk}_{I_{\mathrm{KeyGen}}}$ associated with I_{KeyGen} for a data user.

($\mathrm{sk}_{\mathrm{uid}}$,$\mathrm{pk}_{\mathrm{uid}}$)$\leftarrow$PrivKeyGen(param,uid): Taking as input a user's identity uid, the master key mk and public parameter param, this algorithm is run by the trusted authority to generate a pair of keys ($\mathrm{sk}_{\mathrm{uid}}$,$\mathrm{pk}_{\mathrm{uid}}$).

$\mathrm{rk}_{\mathrm{uid}\rightarrow I_{\mathrm{Enc}}}\leftarrow$ReKeyGen($\mathrm{sk}_{\mathrm{uid}}$,$I_{\mathrm{Enc}}$): Taking as input a user's private key $\mathrm{sk}_{\mathrm{uid}}$ and I_{Enc}, this algorithm is run by the data owner to generate the re-encryption key $\mathrm{rk}_{\mathrm{uid}\rightarrow I_{\mathrm{Enc}}}$.

cph$\leftarrow$Enc(kw, param,$\mathrm{pk}_{\mathrm{uid}}$): Given a keyword kw, the public parameter param, and the data owner's public key $\mathrm{pk}_{\mathrm{uid}}$, this algorithm is run by the data owner to output an original ciphertext cph.

$\mathrm{cph}^{R}\leftarrow$ReEnc(cph, param, $\mathrm{rk}_{\mathrm{uid}\rightarrow I_{\mathrm{Enc}}}$): Given a ciphertext of uid, the public parameter param, and a re-encryption key $\mathrm{rk}_{\mathrm{uid}\rightarrow I_{\mathrm{Enc}}}$, this algorithm is run by the cloud server to output a re-encrypted ciphertext cph^{R}.

token$\leftarrow$TokenGen($\mathrm{sk}_{\mathrm{uid}}$, kw): This algorithm is run by the data owner to generate a token token, which can be used to conduct the search operation over original encrypted keywords.

$\mathrm{token}^{\mathrm{R}}\leftarrow\mathrm{TokenGen}^{\mathrm{R}}$($\mathrm{sk}_{\mathrm{I}_{\mathrm{KeyGen}}}$,kw): This algorithm is run by a data user to generate a token $\mathrm{token}^{\mathrm{R}}$, which can be used to conduct the keyword operation over re-encrypted keywords.

Search(token,cph): This algorithm, run by the cloud server, returns 1 if the original encrypted keyword cph and the token token correspond to the same keyword; otherwise it returns 0.

Search^{R}($\mathrm{token}^{\mathrm{R}}$,$\mathrm{cph}^{\mathrm{R}}$): This algorithm, run by the cloud server, returns 1 if (i) $F(I_{\mathrm{KeyGen}},I_{\mathrm{Enc}})=1$ and (ii) the re-encrypted keyword cph^{R} and the token $\mathrm{token}^{\mathrm{R}}$ correspond to the same keyword; otherwise it returns 0.

Correctness We say an ABRKS scheme is secure if, for (param,mk)$\leftarrow$Setup(1^{ℓ}), ($\mathrm{sk}_{\mathrm{uid}}$,$\mathrm{pk}_{\mathrm{uid}}$)$\leftarrow$PrivKeyGen(param,uid), $\mathrm{sk}_{I_{\mathrm{KeyGen}}}\leftarrow$ KeyGen(mk, param, I_{KeyGen}), then the follows should hold:

- Given cph$\leftarrow$Enc(kw,param,$\mathrm{pk}_{\mathrm{uid}}$) and token$\leftarrow$TokenGen($\mathrm{sk}_{\mathrm{uid}}$,kw), Search (token,cph) always returns 1;
- Given $\mathrm{cph}^{R}\leftarrow$ReEnc(cph,param,$\mathrm{rk}_{\mathrm{uid}\rightarrow I_{\mathrm{Enc}}}$) and $\mathrm{token}^{\mathrm{R}}\leftarrow$TokenGen($\mathrm{sk}_{\mathrm{I}_{\mathrm{KeyGen}}}$, kw), where $\mathrm{rk}_{\mathrm{uid}\rightarrow I_{\mathrm{Enc}}}\leftarrow$ReKeyGen($\mathrm{sk}_{\mathrm{uid}}$,$I_{\mathrm{Enc}}$), Search^{R}($\mathrm{token}^{\mathrm{R}}$,$\mathrm{cph}^{\mathrm{R}}$) always returns 1 if $F(I_{\mathrm{KeyGen}},I_{\mathrm{Enc}})=1$.

Security Definitions

The security of ABRKS requires that the ciphertexts and tokens leak nothing about the underlying keywords. Informally, the adversary is allowed to query ciphertext of any plaintext and tokens except those corresponding to two keywords in the challenge phase. We expect that the adversary cannot distinguish the challenge ciphertext that is generated from one of keywords kw_0 and kw_1. To formalize aforementioned security notion, we define the selective chosen keyword security game as follows. Note that in our corruption model, the adversary is not allowed to get the re-encryption key from uncorrupted users to corrupted users. Note that in our security model we consider the static corrupted model in the sense that the set of corrupted users has to be selected in the setup phase.

Setup

The adversary $\mathcal{A}$ selects a set of corrupted users denoted by CoList and I^*_{Enc}, and sends them to the challenger. The challenger runs Setup to produce param,mk, sends param to $\mathcal{A}$ and keeps mk private.

Phase 1

$\mathcal{A}$ can query the following oracles in polynomially many times:

- $\mathcal{O}_{\mathrm{pk,sk}}(\mathrm{uid})$: It runs $(\mathrm{sk}_{\mathrm{uid}},\mathrm{pk}_{\mathrm{uid}})\leftarrow\mathrm{PrivKeyGen}(\mathrm{param},\mathrm{uid})$. If $\mathrm{uid}\notin\mathrm{CoList}$, it returns the public key $\mathrm{pk}_{\mathrm{uid}}$ to $\mathcal{A}$; otherwise $\mathrm{uid}\in\mathrm{CoList}$, then it returns the key pair $(\mathrm{pk}_{\mathrm{uid}},\mathrm{sk}_{\mathrm{uid}})$ to $\mathcal{A}$. We assume that before querying oracles $\mathcal{O}_{\mathrm{rk}}$, $\mathcal{O}_{\mathrm{ReEnc}}$ and $\mathcal{O}_{\mathrm{token}}$, the user's private key $\mathrm{sk}_{\mathrm{uid}}$ has been generated.
- $\mathcal{O}_{\mathrm{KeyGen}}(I_{\mathrm{KeyGen}})$: If $F(I_{\mathrm{KeyGen}},I^*_{\mathrm{Enc}})=1$, it aborts. Otherwise, it runs $\mathrm{sk}_{I_{\mathrm{KeyGen}}}\leftarrow\mathrm{KeyGen}(\mathrm{mk},\ \mathrm{param},\ I_{\mathrm{KeyGen}})$ and returns the private key $\mathrm{sk}_{I_{\mathrm{KeyGen}}}$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathrm{rk}}(\mathrm{uid},I_{\mathrm{Enc}})$: If $I_{\mathrm{Enc}}\nsubseteq\ I^*_{\mathrm{Enc}}$ and $\mathrm{uid}\notin\mathrm{CoList}$, it aborts because it is not allowed to query re-encrypted key from an uncorrupted user to I_{KeyGen} where $F(I_{\mathrm{KeyGen}},I_{\mathrm{Enc}})=1$. Otherwise, it runs $(\mathrm{sk}_{\mathrm{uid}},\mathrm{pk}_{\mathrm{uid}})\leftarrow\mathrm{PrivKeyGen}$ $(\mathrm{param},\mathrm{uid})$ and $\mathrm{rk}_{\mathrm{uid}\rightarrow I_{\mathrm{Enc}}}\leftarrow\mathrm{ReKeyGen}(\mathrm{sk}_{\mathrm{uid}},I_{\mathrm{Enc}})$, and returns the re-encryption key $\mathrm{rk}_{\mathrm{uid}\rightarrow I_{\mathrm{Enc}}}$.
- $\mathcal{O}_{\mathrm{ReEnc}}(\mathrm{uid},I_{\mathrm{Enc}})$: It runs $(\mathrm{sk}_{\mathrm{uid}},\mathrm{pk}_{\mathrm{uid}})\leftarrow\mathrm{PrivKeyGen}(\mathrm{param},\mathrm{uid})$, $\mathrm{rk}_{\mathrm{uid}\rightarrow I_{\mathrm{Enc}}}$ $\leftarrow\mathrm{ReKeyGen}(\mathrm{sk}_{\mathrm{uid}},I_{\mathrm{Enc}})$ and $\mathrm{cph}^R\leftarrow\mathrm{ReEnc}(\mathrm{cph},\mathrm{param},\mathrm{rk}_{\mathrm{uid}\rightarrow I_{\mathrm{Enc}}})$, and returns re-encrypted keyword cph^R to $\mathcal{A}$.
- $\mathcal{O}_{\mathrm{token}}(\mathrm{uid},\mathrm{kw})$: It runs $\mathrm{token}\leftarrow\mathrm{TokenGen}(\mathrm{sk}_{\mathrm{uid}},\mathrm{kw})$, and returns the token token for kw over original encrypted keyword to $\mathcal{A}$.
- $\mathcal{O}_{\mathrm{token}^{\mathrm{R}}}(I_{\mathrm{KeyGen}},\mathrm{kw})$: It runs $\mathrm{token}^{\mathrm{R}}\leftarrow\mathrm{TokenGen}(\mathrm{sk}_{\mathrm{I}_{\mathrm{KeyGen}}},\mathrm{kw})$ and returns the token $\mathrm{token}^{\mathrm{R}}$ for kw over re-encrypted keyword to $\mathcal{A}$.

Challenge

$\mathcal{A}$ selects an uncorrupted user $\mathrm{uid}^*\notin\mathrm{CoList}$ and two equal-length keywords $(\mathrm{kw}_0,\mathrm{kw}_1)$, where (i) $(\mathrm{uid}^*,\mathrm{kw}_0)$ or $(\mathrm{uid}^*,\mathrm{kw}_1)$ have never been queried on $\mathcal{O}_{\mathrm{token}}$ and (ii) if $(I_{\mathrm{KeyGen}},\mathrm{kw}_1)$, then $(I_{\mathrm{KeyGen}},\mathrm{kw}_0)$ and $(I_{\mathrm{KeyGen}},\mathrm{kw}_1)$ have not been queried to $\mathcal{O}_{\mathrm{token}^{\mathrm{R}}}$. $\mathcal{A}$ sends them to the challenger. The challenger selects $\sigma \overset{R}{\leftarrow} \{0,1\}$, runs $\mathrm{cph}^* \leftarrow \mathrm{Enc}(\mathrm{kw}_\sigma, \mathrm{param}, \mathrm{pk}_{\mathrm{uid}^*})$ and forwards cph^* to $\mathcal{A}$.

Phase 2

$\mathcal{A}$ queries the oracles the same as Phase 1 except that

- $(\mathrm{uid}^*, \mathrm{kw}_0)$ and $(\mathrm{uid}^*, \mathrm{kw}_1)$ are not allowed to query on $\mathcal{O}_{\mathrm{token}}$.
- If $F(I_{\mathrm{KeyGen}}, I^*_{\mathrm{Enc}}) = 1$, then $(I_{\mathrm{KeyGen}}, \mathrm{kw}_0)$ and $(I_{\mathrm{KeyGen}}, \mathrm{kw}_1)$ should not been queried to $\mathcal{O}_{\mathrm{token}^{\mathrm{R}}}$

Guess

$\mathcal{A}$ outputs a guess σ'. We say that $\mathcal{A}$ wins the game if $\sigma = \sigma'$.

Definition 1

We say that an ABRKS *scheme achieves selective security against chosen-keyword attack if any probabilistic polynomial-time adversary $\mathcal{A}$ wins the selective security game defined above with a negligible advantage with respect to the security parameter ℓ, where the advantage is defined as* $|\Pr[\sigma' = \sigma] - 1/2|$.

Methods

The Basic Idea

In our ABRKS scheme, the critical part is how to support keyword search over re-encrypted ciphertexts while being able to enforce access control. In order to achieve this, our intuition (shown in Fig. 2) is to compose the re-encrypted ciphertext with two components: one is associated with the keyword and is transformed from original encrypted ciphertext; the other one is associated with the access control policy and can be derived from the re-encryption key where the access control policy is determined by the data owner.

KP - ABRKS Construction

Recall that an access control policy is represented by (M, π), where M is an $l \times k$ dimensional matrix and Max is the maximum number of attributes associated with a ciphertext. Note that let $x \overset{R}{\leftarrow} X$ denote selecting element x from the set X uniformly at random. The KP-ABRKS scheme can be constructed as follows:

Setup(1^ℓ): Given the security parameter ℓ, the algorithm generates the public parameters and the master key as follows:

- Generate a 4 multi-linear map: $\{e_i : G_0 \times G_i \rightarrow G_{i+1} | i = 0,1,2\}$, where $(G_0, \ldots, G_3)$ are cyclic groups of order p respectively. Let $g_0 \in G_0$ be a generator of G_0, and $g_{i+1} = e_i(g_0, g_i)$ be the generator of G_{i+1} for $i = 0,1,2$.

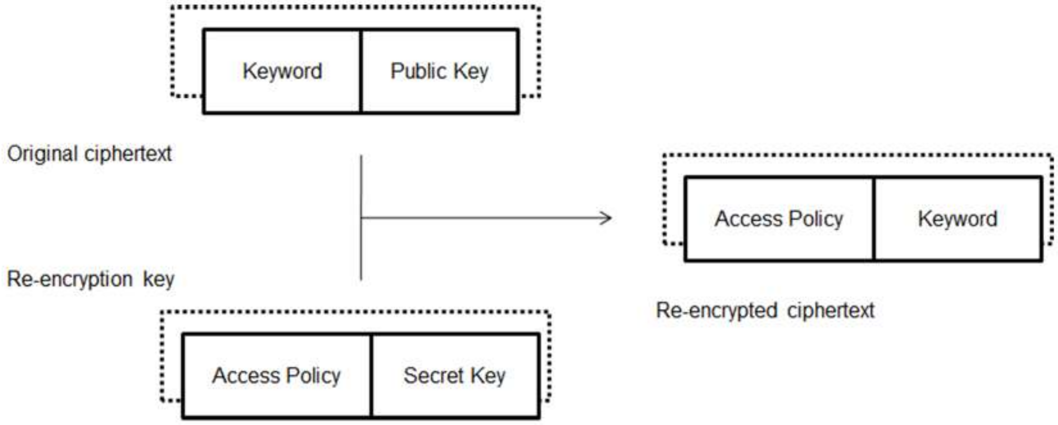

Fig. 2. The high level idea of enabling keyword search over re-encrypted ciphertext by re-encryption.

doi:10.1371/journal.pone.0116325.g002

- Let $H:\{0,1\}^*\rightarrow G_0,H_1:\{0,1\}^*\rightarrow\mathbb{Z}_p$ be two secure hash functions modeled as random oracles.
- Let $h_j\overset{R}{\leftarrow}G_0,j=0,\ldots,\text{Max}$ and define a function $Q(y)=\Pi_{j=0}^{\text{Max}}h_j^{(y^j)}$ where $y\in\mathbb{Z}_p$.
- Choose $a,b\overset{R}{\leftarrow}\mathbb{Z}_p^*$ and set the public parameters and master key as

$$\text{param}=(e_0,e_1,e_2,G_0,G_1,G_2,G_3,g_0,g_1,g_2,g_0^a,g_0^b,H,H_1,h_0,\ldots,h_{\max}),$$
$$\text{mk}=(a,b).$$

KeyGen(mk, (M,π)): Given an access control policy (M,π), this algorithm generates the private key as follows:

- Select $\upsilon_2,\ldots,\upsilon_k\overset{R}{\leftarrow}\mathbb{Z}_p$, set $\mathbf{v}=(ab,\upsilon_2,\ldots,\upsilon_k)$, and compute $\lambda_{\pi(i)}=\mathrm{M}_i\cdot\mathbf{v}$ for $i=1,\ldots,l$.
- For each $i\in[1,l]$, select $r_i\overset{R}{\leftarrow}\mathbb{Z}_p^*$ and set

$$A_i=g_0^{\lambda_{\pi(i)}}Q(H_1(\pi(i)))^{r_i},B_i=g_0^{r_i}.$$

- The private key is set to

$$\text{sk}=((M,\pi),(A_1,B_1),\ldots(A_l,B_l)).$$

PrivKeyGen(mk,param,uid): Given a user's identity uid, this algorithm selects $x_{\text{uid}}\overset{R}{\leftarrow}\mathbb{Z}_p$ and sets

$$\text{sk}_{\text{uid}}=x_{\text{uid}},\text{pk}_{\text{uid}}=g_0^{x_{\text{uid}}}.$$

Enc(kw,param,pk$_{\text{uid}}$): Given a keyword kw$\in\{0,1\}^*$, this algorithm selects $r \overset{R}{\leftarrow} \mathbb{Z}_p$, and sets $C_1 = g_0^r$ and $C_2 = e_2(H(\text{kw})^r, e_1(g_0^a, e_0(\text{pk}_{\text{uid}}, g_0^b)))$. It sets the original encrypted keyword as

$$\text{cph} = (C_1, C_2).$$

ReKeyGen(sk$_{\text{uid}}$,S): Taking as input the data owner's private key $\text{sk}_{\text{uid}} = x_{\text{uid}}$ and an attribute set S, this algorithm generates the re-encryption key as follows:

- Select $d \overset{R}{\leftarrow} \mathbb{Z}_p$ and set $R_1 = d/x_{\text{uid}}, R_2 = g_0^d$.
- Set $R_{\text{at}_j} = Q(H_1(\text{at}_j))^d$ for each at$_j\in$S.
- Set the re-encryption key as $\text{rk}_{\text{uid}\to\text{S}} = (R_1, R_2, \{R_{\text{at}_j}\}_{\text{at}_j\in\text{S}})$.

ReEnc(cph,param,rk$_{\text{uid}\to\text{S}}$): Given the original ciphertext $\text{cph} = (C_1, C_2)$ and the re-encryption key rk$_{\text{uid}\to\text{S}}$, it computes $C'_2 = C_2^{R_1}$ and re-encrypts cph to $\text{cph}^R = (C_1, C'_2, R_2, \{R_{\text{at}_j}\}_{\text{at}_j\in\text{S}})$.

TokenGen(sk$_{\text{uid}}$,kw): Given the private key $\text{sk}_{\text{uid}} = x_{\text{uid}}$ of data user uid and a keyword kw, this algorithm sets the token for the keyword kw over original encrypted keywords as

$$\text{token} = \text{H}(\text{kw})^{\text{x}_{\text{uid}}}.$$

TokenGen^R(sk,kw): Given the data user's private key sk, this algorithm computes $A'_i = e_0(H(\text{kw}), A_i)$ and $B'_i = e_0(H(\text{kw}), B_i)$ for $i = 1, \ldots, l$. It sets the token for the keyword kw over re-encrypted keywords as

$$\text{token}^{\text{R}} = ((\text{M}, \pi), (\text{A}'_{\text{i}}, \text{B}'_{\text{i}})_{\text{i}\in[1,\text{l}]}).$$

Search(token,cph): Given the original encrypted keyword cph and a token token generated by the data owner, this algorithm outputs 1 if $e_2(\text{token}, \text{e}_1(\text{C}_1, \text{e}_0(\text{g}_0^{\text{a}}, \text{g}_0^{\text{b}}))) = \text{C}_2$, and 0 otherwise.

Search^R(token$^{\text{R}}$,cph$^{\text{R}}$): Given the re-encrypted keyword cph^R and a token token$^{\text{R}}$ generated by the data users, the search can be done as follows:

- If the attribute set S associated with cph^R satisfies the access control policy specified by (M,π) associated with token$^{\text{R}}$, compute c_i such that $\sum_{\pi(i)\in\text{S}} c_i\text{M}_i = (1, 0, \ldots, 0)$, and let

$$K = \prod_{\pi(i)\in\text{S}} \left(\frac{e_1(R_2, A'_i)}{e_1(R_{\pi(i)}, B'_i)}\right)^{c_i}.$$

  If $e_2(K, C_1) = C'_2$, output 1 and 0 otherwise.

- Otherwise, output 0.

Correctness

The correctness of the KP - ABRKS scheme can be verified as follows:

If token and the original ciphertext correspond to the same keyword, we have

$$\begin{aligned} e_2(\mathrm{token},e_1(\mathrm{C}_1,\mathrm{e}_0(\mathrm{g}_0^{\mathrm{a}},\mathrm{g}_0^{\mathrm{b}}))) &= e_2(H(\mathrm{kw})^{x_{id}},e_1(g_0^r,e_0(g_0^a,g_0^b))) \\ &= e_2(H(\mathrm{kw}),e_1(g_0^{x_{id}},e_0(g_0^a,g_0^b)))^r \\ &= C_2. \end{aligned}$$

If the attribute set S satisfies the access control policy specified by (M,π), and token$^{\mathrm{R}}$ and the re-encrypted ciphertext correspond to the same keyword,

$$\begin{aligned} K = &\prod_{\pi(i)\in \mathrm{S}} (\frac{e_1(R_2,A'_i)}{e_1(R_{\pi(i)},B'_i)})^{c_i}. \\ = &\prod_{\pi(i)\in \mathrm{S}} (\frac{e_1(g_0^d,e_0(H(\mathrm{kw}),g_0^{\lambda_{\pi(i)}}Q(H_1(\pi(i)))^{r_i}))}{e_1(Q(H_1(\pi(i)))^d,e_0(H(\mathrm{kw}),g^{r_i}))})^{c_i} \\ = &\, e_1(g_0,e_0(H(\mathrm{kw}),g_0))^{abd}. \end{aligned}$$

such that

$$\begin{aligned} e_2(K,C_1) &= e_2(H(\mathrm{kw}),e_1(g_0,e_0(g_0,g_0)))^{abdr} \\ &= C'_2. \end{aligned}$$

CP - ABRKS Construction

We also elaborate the construction of the CP-ABRKS scheme as follows.

Setup (N,n_{max}): This algorithm takes as input N, the number of attributes in the system and n_{max} the maximum of columns of M. It generates the public parameters and the master key as follows:

- Generate a 4 multi-linear map: $\{e_i : G_0 \times G_i \to G_{i+1} | i=0,1,2\}$, where $(G_0,\ldots,G_3)$ are cyclic groups of order p respectively. Let $g_0 \in G_0$ be a generator of G_0, and $g_{i+1}=e_i(g_0,g_i)$ be a generator of G_{i+1} for $i=0,1,2$.
- Select elements $h_{1,1},h_{1,2},\ldots,h_{n_{max},N}$ from G_0 uniformly at random.
- Let $H : \{0,1\}^* \to G_0$ be a secure hash function modeled as a random oracle.
- Select $a,b,c \overset{R}{\leftarrow} \mathbb{Z}_p$ and set the public parameters and master key as

$$\begin{aligned} \mathrm{param} = (&e_0,e_1,e_2,G_0,G_1,G_2,G_3,g_0,g_1,g_2,g_0^a,g_0^b, \\ &h_{1,1},\ldots,h_{n_{max},N},H), \\ \mathrm{mk} = (&a,b). \end{aligned}$$

KeyGen(mk,S): Given an attribute set S, this algorithm generates the private key as follows:

- Choose $t_1,\ldots,t_{n_{max}} \overset{R}{\leftarrow} \mathbb{Z}_p$, and set $D = g_0^{ab} g_0^{at_1}$.
- For each $j \in [1,n_{max}]$ set $L_j = g_0^{t_j}$ and for each $x \in \mathrm{S}$ set $D_x = \Pi_{j=1}^{n_{max}} h_{j,x}^{t_j}$.
- The private key is set to

$$\mathrm{sk} = (D,\{L_j\}_{j\in [1,n_{max}]},\{D_x\}_{x\in \mathrm{S}}).$$

PrivKeyGen(param,uid): This algorithm is the same as PrivKeyGen algorithm in KP - ABRKS.

Enc(kw,param,$\mathrm{pk}_{\mathrm{uid}}$): This algorithm is the same as Enc algorithm in KP - ABRKS.

ReKeyGen($\mathrm{sk}_{\mathrm{uid}}$,(M,$\pi$)): Taking as input a user's private key $\mathrm{sk}_{\mathrm{uid}} = x_{\mathrm{uid}}$ and an access control policy (M,π), where M is an $l \times n_{max}$ matrix (If the number of columns of M is $k < n_{max}$, it can simply "pad out" the rightmost $n_{max} - k$ columns with zeros.), this algorithm generates the re-encryption key as follows:

- Select $d \overset{R}{\leftarrow} \mathbb{Z}_p$ and set $R_1 = d/x_{\mathrm{uid}}, R_2 = g_0^d$.
- Choose $n_{max} - 1$ random elements $\upsilon_2,\ldots,\upsilon_{n_{max}} \overset{R}{\leftarrow} \mathbb{Z}_p$, let the vector $\vec{\upsilon} = (\upsilon_1 = d, \upsilon_2,\ldots,\upsilon_{n_{max}}) \in \mathbb{Z}_p^{n_{max}}$, and set $R_{i,j} = g_0^{a\mathrm{M}_{i,j}\upsilon_j} h_{j,\pi(i)}^{-d}$ for each $i = 1,\ldots,l$ and $j = 1,\ldots, n_{max}$.
- Set the re-encryption key as $\mathrm{rk}_{\mathrm{uid}\rightarrow(\mathrm{M},\pi)} = (R_1,R_2,\{R_{i,j}\}_{i\in [1,l],j\in [1,n_{max}]})$.

ReEnc(cph,param,$\mathrm{rk}_{\mathrm{uid}\rightarrow(\mathrm{M},\pi)}$): Given an original encrypted keyword $\mathrm{cph} = (C_1,C_2)$ and the re-encryption key $\mathrm{rk}_{\mathrm{uid}\rightarrow(\mathrm{M},\pi)}$, the algorithm computes $C'_2 = C_2^{R_1}$ and re-encrypts cph to

$$\mathrm{cph}^R = ((\mathrm{M},\pi),C_1,C'_2,R_2,\{R_{i,j}\}_{i\in [1,l],j\in [1,n_{max}]}).$$

TokenGen($\mathrm{sk}_{\mathrm{uid}}$,kw): This algorithm performs as same as TokenGen algorithm in KP - ABRKS.

TokenGen^R(sk,kw): Given credentials sk, this algorithm computes $D' = e_0(H(\mathrm{kw}),D)$, $L'_j = e_0(H(\mathrm{kw}),L_j)$ for $j = 1,\ldots,n_{max}$ and $D'_x = e_0(H(\mathrm{kw}),D_x)$ for $x \in \mathrm{S}$. It sets the token for the keyword kw over re-encrypted keywords as

$$\mathrm{token}^{\mathrm{R}} = (\mathrm{S},\mathrm{D}',\{\mathrm{L}'_{\mathrm{j}}\}_{\mathrm{j}\in [1,\mathrm{n_{max}}]},\{\mathrm{D}'_{\mathrm{x}}\}_{\mathrm{x}\in \mathrm{S}}).$$

Search(token,cph): This algorithm performs the same as Search algorithm in KP - ABRKS.

Search^R(token$^{\mathrm{R}}$,cph$^{\mathrm{R}}$): Given the re-encrypted keyword cph^R and a token token$^{\mathrm{R}}$ generated by the data users, the search can be done as follows:

- If the attribute set S associated with token$^{\mathrm{R}}$ satisfies the access control policy specified by (M,π) associated with cph^R, compute c_i such that

$\sum_{\pi(i) \in \mathrm{S}} c_i \mathrm{M}_i = (1,0,\ldots,0)$, and let

$$K = e_1(R_2,D')/(\prod_{j=1,\ldots,n_{max}} e_1(\prod_{\pi(i)\in \mathrm{S}} R_{i,j}^{c_i},L'_j)) \prod_{\pi \in \mathrm{S}} e_1(R_2,D'_{\pi(i)}).$$

If $e_2(K,C_1) = C'_2$, output 1 and 0 otherwise.

- Otherwise, output 0.

Correctness

The correctness of the CP-ABRKS scheme can be verified similar to that of KP-ABRKS scheme.

Discussion

KP - ABRKS Security

Theorem 1

Assume that 4-*MDDH assumption holds, our* KP-ABRKS *scheme achieves selective security against chosen-keyword attack in the random oracle model.*

Proof: The proof strategy is to reduce the security of our construction to the hardness of 4-MDDH assumption. That is, we show that if there exists a probabilistic polynomial time adversary $\mathcal{A}$ breaking selective security game of KP-ABRKS against chosen-keyword attack with a non-negligible advantage ϵ, then we can simulate a challenger solving 4-MDDH problem with a non-negligible advantage $(1/e + 1/q_T)\frac{\epsilon}{2}$, where q_T is a polynomial large number, which should be larger than the number of oracle queries for $\mathcal{O}_{\mathrm{ReEnc}},\mathcal{O}_{\mathrm{token}}$ and $\mathcal{O}_{\mathrm{token}^{\mathrm{R}}}$.

Given an instance of 4-MDDH problem $(g_0,g_0^a,g_0^b,g_0^c,g_0^w,g_0^r,Z)$, where $a,b,c,w,r \overset{R}{\leftarrow} \mathbb{Z}_p$ are unknown, the challenger simulates the game as follows:

Setup

$\mathcal{A}$ selects a set of corrupted users denoted by CoList and an attribute set S^*, and sends them to the challenger. The challenger generates the public parameters and master key as follows:

- Given the attribute set S^*, let $\phi(y) = y^{\mathrm{Max}\text{-}|\mathrm{S}^*|} \cdot \prod_{\mathrm{at} \in \mathrm{S}^*} (y - H_1(\mathrm{at}))$, which can be rewritten as $\phi(y) = \sum_{j=0}^{\mathrm{Max}} \phi_j y^j$, where ϕ_j is the coefficient of y^j and therefore $\phi_j = 0$ for $j = 0,\ldots,\mathrm{Max}\text{-}|\mathrm{S}^*|$.
- Select $\varphi_0,\ldots,\varphi_{\mathrm{Max}} \overset{R}{\leftarrow} \mathbb{Z}_p$, and define $\varphi(y) = \sum_{j=0}^{\mathrm{Max}} \varphi_j y^j$.
- Let $h_j = g_0^{a\phi_j + \varphi_j}, 0 \le j \le \mathrm{Max}$, and define $Q(y) = g_0^{a\phi(y) + \varphi(y)} = \prod_{j=0}^{\mathrm{Max}} g_0^{(a\phi_j + \varphi_j)y^j}$.
- The public parameters is set to
$$\mathrm{param} = (e_0,e_1,e_2,G_0,G_1,G_2,G_3,g_0,g_1,g_2,g_0^a,g_0^b,H,H_1,h_0,\ldots,h_{\mathrm{Max}}),$$
by implicitly setting the master private key $\mathrm{mk} = (a,b)$.

Moreover, the challenger simulates the oracles H,H_1 as follows:

- $\mathcal{O}_H(\mathrm{kw})$: Given a keyword kw, it proceeds as follows:
  - If kw has not been queried before, then select $a_i \overset{R}{\leftarrow} \mathbb{Z}_p$ and toss a random coin $c_i \in \{0,1\}$ with the probability that $\Pr[c_i = 0] = 1/(q_T + 1)$, where q_T is a polynomial large number. We require that q_T should be larger than the number of oracle queries for $\mathcal{O}_{\mathrm{ReEnc}}, \mathcal{O}_{\mathrm{token}}$ and $\mathcal{O}_{\mathrm{token}^{\mathrm{R}}}$. If $c_i = 0$, then compute $H(\mathrm{kw}) \leftarrow g_0^w g_0^{a_i}$; Otherwise, compute $H(\mathrm{kw}) \leftarrow g_0^{a_i}$. Add $(\mathrm{kw}, a_i, H(\mathrm{kw}), c_i)$ to L_H and return $H(\mathrm{kw})$.
  - Otherwise, retrieve $H(\mathrm{kw})$ from L_H with respect to kw and return $H(\mathrm{kw})$.
- $\mathcal{O}_{H_1}(\mathrm{at})$: If the attribute at has not been queried before, select $u \overset{R}{\leftarrow} \mathbb{Z}_p$, set $H_1(\mathrm{at}) = u$, and add $(\mathrm{at}, H_1(\mathrm{at}))$ to the list L_{H_1}. Otherwise, retrieve $H_1(\mathrm{at})$ from L_{H_1} with respect to at. Eventually, it returns $H_1(\mathrm{at})$.

Phase 1

$\mathcal{A}$ can query the following oracles in polynomially many times:

- $\mathcal{O}_{\mathrm{pk,sk}}(\mathrm{uid})$: Given a user identity uid, the challenger proceeds as follows:
  - If uid has been queried before, retrieve $(\mathrm{sk}_{\mathrm{uid}}, \mathrm{pk}_{\mathrm{uid}})$ from L_U with respect to uid and return $(\mathrm{sk}_{\mathrm{uid}}, \mathrm{pk}_{\mathrm{uid}})$.
  - Otherwise, select $x_{\mathrm{uid}} \overset{R}{\leftarrow} \mathbb{Z}_p$. If uid∈CoList, compute $\mathrm{sk}_{\mathrm{uid}} \leftarrow \mathrm{x}_{\mathrm{uid}}$ and $\mathrm{pk}_{\mathrm{uid}} \leftarrow g_0^{x_{\mathrm{uid}}}$; otherwise set $\mathrm{sk}_{\mathrm{uid}} = \perp$ and $\mathrm{pk}_{\mathrm{uid}} \leftarrow g_0^{c x_{\mathrm{uid}}}$. Finally add $(\mathrm{uid}, \mathrm{sk}_{\mathrm{uid}}, \mathrm{pk}_{\mathrm{uid}})$ to L_U and return $(\mathrm{sk}_{\mathrm{uid}}, \mathrm{pk}_{\mathrm{uid}})$.
- $\mathcal{O}_{\mathrm{KeyGen}}(\mathrm{P})$: Given an access control policy P specified by (M, π), the challenger proceeds as follows:
  - If $F(\mathrm{S}^*, \mathrm{P}) = 1$, then abort.
  - Otherwise, because S^* does not satisfy the access structure (M, π), there exists a vector $\mathbf{w} = (w_1, \ldots, w_k) \in \mathbb{Z}_p^k$ such that $w_1 = 1$ and $\forall \pi(i) \in \mathrm{S}^*, \mathrm{M}_i \cdot \mathbf{w} = 0$. Choose $v'_i \overset{R}{\leftarrow} \mathbb{Z}_p$ for $i = 2, \ldots, k$, and set $\mathbf{v}' = (0, v'_2, \ldots, v'_k)$. By implicitly setting $\mathbf{v} = ab\mathbf{w} + \mathbf{v}'$, it generates A_i and B_i as follows:
    - ✶ If $\pi(i) \in \mathrm{S}^*$, select $r_i \overset{R}{\leftarrow} \mathbb{Z}_p$ compute $\lambda_{\pi(i)} = \mathrm{M}_i \cdot \mathbf{v}' = \mathrm{M}_i \cdot \mathbf{v}$, and set $A_i = g^{\lambda_{\pi(i)}} Q(H_1(\pi(i)))^{r_i}$ and $B_i = g^{r_i}$.

- ⋆ Otherwise, select $r'_i \overset{R}{\leftarrow} \mathbb{Z}_p$ and compute

$$A_i = g^{\lambda_{\pi(i)}} Q(H_1(\pi(i)))^{r_i}$$

$$= g^{\mathrm{M}_i \cdot [ab\mathbf{w} + \mathbf{v}']} . g^{[a\phi(H_1(\pi(i))) + \varphi(H_1(\pi(i)))] r_i}$$

$$= \frac{g^{\mathrm{M}_i \cdot \mathbf{v}' + a\phi(H_1(\pi(i))) r'_i + \varphi(H_1(\pi(i))) r'_i}}{g^{b\varphi(H_1(\pi(i))) \mathrm{M}_i \cdot \mathbf{w} / \phi(H_1(\pi(i)))}}$$

$$B_i = g^{-b\mathrm{M}_i \cdot \mathbf{w} / \phi(H_1(\pi(i))) + r'_i}$$

by implicitly setting $r_i = r'_i - b\mathrm{M}_i \cdot \mathbf{w} / \phi(H_1(\pi(i)))$.

- $\mathcal{O}_{\mathrm{rk}}$(uid,S): Given a user identity uid and an attribute set S, the challenger proceeds as follows:
  - If $\mathrm{S} \not\subseteq \mathrm{S}^* \wedge$ uid $\notin$ CoList, then abort.
  - If uid∈CoList, choose a random $d \in \mathbb{Z}_p$ and set

$$\mathrm{rk}_{\mathrm{uid} \to \mathrm{S}} = (R_1 = d / x_{\mathrm{uid}}, R_2 = g_0^d, \{R_{\mathrm{at}_j}\}_{\mathrm{at}_j \in \mathrm{S}}),$$

where $d \overset{R}{\leftarrow} \mathbb{Z}_p$ and $R_{\mathrm{at}_j} = Q(H_1(\mathrm{at}_j))^d$.

  - Otherwise, choose $d' \overset{R}{\leftarrow} \mathbb{Z}_p$ and set

$$\mathrm{rk}_{\mathrm{uid} \to \mathrm{S}} = (R_1 = cd' / cx_{\mathrm{uid}} = d' / x_{\mathrm{uid}}, R_2 = g_0^{cd'}, \{R_{\mathrm{at}_j}\}_{\mathrm{at}_j \in \mathrm{S}}),$$

by implicitly letting $d = cd'$. Note that $R_{\mathrm{at}_j} = Q(H_1(\mathrm{at}_j))^{cd'} = \Pi_{j=0}^{\mathrm{Max}} (g_0^c)^{d' \varphi_j y^j}$, since $Q(H_1(\mathrm{at}_j)) = \Pi_{j=0}^{\mathrm{Max}} g_0^{(a\phi_j + \varphi_j) y^j} = \Pi_{j=0}^{\mathrm{Max}} g_0^{\varphi_j y^j}$.

- $\mathrm{O}_{\mathrm{ReEnc}}$(uid,S,cph): Given a user identity uid, an original encrypted keyword cph and an attribute set S, the challenger proceeds as follows:
  - If uid∈CoList $\vee$ ($\mathrm{S} \subseteq \mathrm{S}^* \wedge$ uid $\notin$ CoList), it queries $\mathcal{O}_{\mathrm{rk}}$ with (uid, S) to get the re-encryption key $\mathrm{rk}_{\mathrm{uid} \to \mathrm{S}}$ and computes $\mathrm{cph}^R \leftarrow \mathrm{ReEnc}(\mathrm{cph}, \mathrm{param}, \mathrm{rk}_{\mathrm{uid} \to \mathrm{S}})$.
  - Otherwise, if there exists kw_i in L_H such that $c_i = 1$ and $e_2(e_1(e_0(g_0^a, g_0^b), g_0^c), g_0^r)^{a_i x_{\mathrm{uid}}} = C_2$, it selects $d \overset{R}{\leftarrow} \mathbb{Z}_p$, sets $C'_2 = e_2(e_1(e_0(g_0^a, g_0^b) g_0^r), g_0)^{a_i d}$, $R_2 = g_0^d$ and $R_{\mathrm{at}_j} = Q(H_1(\mathrm{at}_j))^d$ for each $\mathrm{at}_j \in$S, and returns $\mathrm{cph}^R = (C_1, C'_2, R_2, \{R_{\mathrm{at}_j}\}_{\mathrm{at}_j \in \mathrm{S}})$;
  - Otherwise, it reports failure and terminates.
- $\mathcal{O}_{\mathrm{token}}$(uid,kw): Given a user identity uid and a keyword kw, the challenger proceeds as follows:
  - It queries O_H with kw to obtain $(a_i, H(\mathrm{kw}), c_i)$.

  - If $c_i = 1$, set token $= \mathrm{H}(\mathrm{kw})^{\mathrm{sk}_{\mathrm{uid}}} = (g_0^{\mathrm{a_i}})^{\mathrm{sk}_{\mathrm{uid}}} = \mathrm{pk}_{\mathrm{uid}}^{\mathrm{a_i}}$;
  - If $c_i = 0 \wedge \mathrm{uid} \in \mathrm{CoList}$, set token $= \mathrm{H}(\mathrm{kw})^{\mathrm{sk}_{\mathrm{uid}}} = g_0^{\mathrm{wx}_{\mathrm{uid}}}$;
  - Otherwise, report failure and terminate.

- $\mathcal{O}_{\mathrm{token}^{\mathrm{R}}}(\mathrm{P},\mathrm{kw})$: Given an access control policy P and a keyword kw, the challenger proceeds as follows:

  - If $c_i = 1$, select $\upsilon_2,\upsilon_3,\ldots,\upsilon_k \overset{R}{\leftarrow} \mathbb{Z}_p$, implicitly set $\mathbf{v} = (ab,\upsilon_2,\ldots,\upsilon_k)$ and $\lambda_{\pi(i)} = \mathrm{M}_i\mathbf{v}$ for $i = 1,\ldots,l$. Compute for $i = 1,\ldots,l$,

$$\begin{aligned} A'_i &= e_0(H(\mathrm{kw}),A_i) \\ &= e_0(g_0^{\lambda_{\pi(i)}} Q(H_1(\pi(i)))^{r_i}, g_0^{a_i}) \\ &= e_0(g_0^a,g_0^b)^{\mathrm{M}_{i1}a_i} e_0(g_0,g_0)^{\sum_{j=2}^{k} \mathrm{M}_{ij}\upsilon_j a_i} \qquad e_0(Q(H_1(\pi(i)))^{r_i}, g_0^{a_i}), \end{aligned}$$

$$B'_i = e_0(H(\mathrm{kw}),B_i) = e_0(g_0,g_0)^{r_i a_i}.$$

  - If $c_i = 0 \wedge F(\mathrm{S}^*,\mathrm{P}) = 0$, make a query P on $\mathcal{O}_{\mathrm{KeyGen}}$ to get sk, and compute $A'_i = e_0(H(\mathrm{kw}),A_i)$ and $B'_i = e_0(H(\mathrm{kw}),B_i)$ for $i = 1,\ldots,l$.
  - Otherwise, report failure and terminate.

Challenge

$\mathcal{A}$ selects an uncorrupted user $\mathrm{uid}^* \notin \mathrm{CoList}$ and two keywords $(\mathrm{kw}_0,\mathrm{kw}_1)$ of equal length. Given kw_0 and kw_1, if $c_0 = 1 \wedge c_1 = 1$, the challenger reports failure and terminates; otherwise, let σ be a bit which is selected as follows:

- If $c_0 = 1$ and $c_1 = 0$, then set $\sigma = 1$,
- If $c_0 = 0$ and $c_1 = 1$, then set $\sigma = 0$,
- Otherwise, let $\sigma \overset{R}{\leftarrow} \{0,1\}$.

The challenger responses $\mathcal{A}$ with $\mathrm{cph}^* = (C_1 = g^r, C_2 = Z^{a_i x_{\mathrm{uid}}})$.

Phase 2

$\mathcal{A}$ executes the same as Phase 1.

Guess

$\mathcal{A}$ outputs a guess σ'. The challenger outputs $Z = g_3^{abcwr}$ if $\sigma' = \sigma$; Otherwise, it outputs $Z \neq g_3^{abcwr}$.

This completes the simulation. In what follows let us analyze the probability that the challenger will not report failure and terminate due to the following two independent events:

- When $\mathcal{A}$ queries $\mathcal{O}_{\mathrm{token}},\mathcal{O}_{\mathrm{token}^{\mathrm{R}}}$ and $\mathcal{O}_{\mathrm{ReEnc}}$, it happens that $c_i = 0$ for some keyword. Note that for each query with respect to some keyword,

$\Pr[c_i=0]=1/(q_T+1)$. Therefore, as $\mathcal{A}$ makes at most q_T oracle queries, the probability of the challenger not reporting failure and terminating can be $(1-1/(q_T+1))^{q_T}\geq 1/e$.

- When $\mathcal{A}$ presents kw_0 and kw_1, it happens that $c_0=1$ and $c_1=1$. Since $\Pr[c_i=0]=1/(q_T+1)$ for $i=0,1$, we have $\Pr[c_0=1\wedge c_1=1]=(1-1/(q_T+1))^2\leq 1-1/q_T$. Hence, the probability that the challenger has no failure is at least $1/q_T$.

Therefore the challenger simulates without failure with the probability at least $1/e+1/q_T$.

Now let us analyze the advantage of the challenger solving 4-MDDH problem on condition that the simulation completes perfectly. In the challenge phase, if $Z=g_3^{abcwr}$, then cph^* is indeed a valid ciphertext of kw_σ. Then the probability of $\mathcal{A}$ outputting $\sigma=\sigma'$ is $\frac{1}{2}+\epsilon$. If Z is an element randomly selected from G_3, the probability of $\mathcal{A}$ outputting $\sigma=\sigma'$ is $\frac{1}{2}$. Therefore, the probability of the challenger correctly guessing $Z\overset{?}{=}g_3^{abcwr}$ is $\frac{1}{2}(\frac{1}{2}+\epsilon)+\frac{1}{2}\frac{1}{2}=\frac{1}{2}+\frac{\epsilon}{2}$. That is, the challenger solves the 4-MDDH problem with advantage $(1/e+1/q_T)\frac{\epsilon}{2}$ if $\mathcal{A}$ wins the selective security game with advantage ϵ. □

CP - ABRKS Security

Security of the CP - ABRKS scheme can be proven as the following theorem.

Theorem 2

Assume that 4-*MDDH assumption holds, our* CP-ABRKS *scheme achieves selective security against chosen-keyword attack in the random oracle model.*

Proof: The main idea is to reduce the security of our CP-ABRKS to the hardness of 4-MDDH assumption. That's, we show that if there exists a probabilistic polynomial time adversary $\mathcal{A}$ breaking the selective security game of our CP-ABRKS scheme against chosen-keyword attack with a non-negligible advantage ϵ, then we can construct a challenger solving 4-MDDH problem with a non-negligible advantage $(1/e+1/q_T)\frac{\epsilon}{2}$, where q_T is a polynomial large number, which should be larger than the number of oracle queries for $\mathcal{O}_{\mathrm{ReEnc}},\mathcal{O}_{\mathrm{token}}$ and $\mathcal{O}_{\mathrm{token}^{\mathrm{R}}}$. In this part, $\mathrm{P}\subseteq\mathrm{P}^*$ means P is a substructure of P^*.

Given an instance of 4-MDDH problem $(g_0,g_0^a,g_0^b,g_0^c,g_0^w,g_0^r,Z)$ where $a,b,c,w,r\leftarrow\mathbb{Z}_p$ are unknown, the challenger simulates the game as follows:

Setup

$\mathcal{A}$ selects a set of corrupted users denoted by CoList and an access control policy (M^*,π^*), where M is an $l\times k^*$ matrix, and sends them to the challenger. The challenger generates the public parameters and master key as follows:

- Given the access control policy (M^*,π^*), for each (j,x) where $1 \leq x \leq N$ and $1 \leq j \leq n_{max}$, choose $z_{j,x} \overset{R}{\leftarrow} \mathbb{Z}_p$. If there exists an i such that $\pi^*(i) = x$ and $i \leq k^*$, let $h_{j,x} = g_0^{z_{j,x} + a\mathrm{M}^*_{i,j}}$; Otherwise, let $h_{j,x} = g_0^{z_{j,x}}$.

  The public parameters is set to

$$\mathrm{param} = (e_0,e_1,e_2,G_0,G_1,G_2,G_3,g_0,g_1,g_2,g_0^a,g_0^b,$$

$$h_{1,1},h_{1,2},\ldots,h_{n_{max},N},H),$$

by implicitly setting the master private key $\mathrm{mk} = (a,b)$.

The random oracle O_H is simulated as same as the proof of Theorem 1.

Phase 1

$\mathcal{A}$ can query the following oracles in polynomial many times:

- $\mathcal{O}_{\mathrm{pk,sk}}(\mathrm{uid})$: Same as the proof of Theorem 1.
- $\mathcal{O}_{\mathrm{KeyGen}}(\mathrm{S})$: Given an attribute set S, the challenger proceeds as follows:
  - If $F(\mathrm{S},\mathrm{P}^*) = 1$, then abort.
  - Otherwise, because S does not satisfy the access structure (M^*,π^*), there exists a vector $\mathbf{w} = (w_1,\ldots,w_{n_{max}}) \in \mathbb{Z}_p^{n_{max}}$ such that $w_1 = -1$ and $\forall \pi^*(i) \in \mathrm{S}, \mathrm{M}^*_i \cdot \mathbf{w} = 0$. Note that we simply let $w_j = 0$ and $\mathrm{M}^*_{i,j} = 0$ for $k^* < j \leq n_{max}$. Compute $D = g_0^{ab} g_0^{at_1} = g_0^{ar_1}$ and $L_j = g^{r_j}(g^b)^{w_j}$ for $j = 1,\ldots,n_{max}$, by choosing $r_1,\ldots,r_{n_{max}} \overset{R}{\leftarrow} \mathbb{Z}_p$ and implicitly defining $t_j = r_j + w_j b$, and set D_x for each $x \in \mathrm{S}$ as follows:
    - ⋆ If there exists i such that $\pi^*(i) = x$, set

$$\begin{aligned} D_x &= \prod_{j=1}^{n_{max}} h_{j,x}^{t_j} \\ &= \prod_{j=1}^{n_{max}} g_0^{(z_{j,x} + a\mathrm{M}^*_{i,j})(r_j + w_j b)} \\ &= \prod_{j=1}^{n_{max}} g_0^{z_{j,x} r_j + z_{j,x} w_j b + a\mathrm{M}^*_{i,j} r_j}. \end{aligned}$$

    - ⋆ Otherwise set $D_x = \prod_{j=1}^{n_{max}} L_j^{z_{j,x}}$.
- $\mathcal{O}_{\mathrm{rk}}(\mathrm{uid},\mathrm{P})$: Given a user identity uid and an access control policy $\mathrm{P} = (\mathrm{M},\pi)$, where M is an $l \times k$ matrix, the challenger proceeds as follows:
  - If $\mathrm{P} \not\subseteq \mathrm{P}^* \wedge \mathrm{uid} \notin \mathrm{CoList}$, then abort.
  - If $\mathrm{uid} \in \mathrm{CoList}$, choose random elements $d,v_2,\ldots,v_{n_{max}} \overset{R}{\leftarrow} \mathbb{Z}_p$, let $\vec{v} = (v_1 = d,v_2,\ldots,v_{n_{max}}) \in \mathbb{Z}_p^{n_{max}}$, and set

$$\mathrm{rk}_{\mathrm{uid}\to \mathrm{P}}=(R_1=d/x_{\mathrm{uid}},R_2=g_0^d,\{R_{i,j}\}_{i=1,\ldots,l,j=1,\ldots,n_{max}}),$$

where $R_{i,j}=g_0^{\mathrm{M}_{i,j}\upsilon_j}$.

- Otherwise, we consider $\mathrm{P}=\mathrm{P}^*$ first. Choose random elements $d',\upsilon'_2,\ldots,\upsilon'_{n_{max}}\overset{R}{\leftarrow}\mathbb{Z}_p$ and set

$$\mathrm{rk}_{\mathrm{uid}\to \mathrm{P}}=(R_1=cd'/cx_{\mathrm{uid}}=d'/x_{\mathrm{uid}},R_2=g_0^{cd'},\{R_{i,j}\}_{i=1,\ldots,l,j=1,\ldots,n_{max}}),$$

where

$$\begin{aligned}R_{i,j}&=g_0^{a\mathrm{M}^*_{i,j}\upsilon_j}h_{j,\pi^*(i)}^{-d}\\&=g_0^{a\mathrm{M}^*_{i,j}(cd'+\nu'_j)}(g_0^{a\mathrm{M}^*_{i,j}+z_{\pi^*(i),j}})^{-cd'}\\&=g_0^{\upsilon'_j}(g_0^c)^{-d'z_{\pi^*(i),j}},\end{aligned}$$

by implicitly defining $d=cd'$ and $\vec{\upsilon}=(\upsilon_1=cd',cd'+\upsilon'_2,\ldots,cd'+\upsilon'_{n_{max}})\in\mathbb{Z}_p^{n_{max}}$(We set $\upsilon'_1=0$). Note that the form of our re-encryption key is similar to that of the ciphertext of Water's CP-ABE[17]. So if $\mathrm{P}=(M,\pi)\subseteq\mathrm{P}^*=(\mathrm{M}^*,\pi^*)$, the re-encryption key can be derived from (M^*,π^*) through the technology of ciphertext delegation proposed in [30].

- $\mathcal{O}_{\mathrm{ReEnc}}$(uid,S,cph): Given a user identity uid, an original encrypted keyword cph and an access control policy P, the challenger proceeds as follows:
  - If $\mathrm{uid}\in\mathrm{CoList}\ \vee(\mathrm{P}\in\mathrm{P}^*\wedge\mathrm{uid}\notin\mathrm{CoList})$, it queries $\mathcal{O}_{\mathrm{rk}}$ with (uid, P) to get the re-encryption key $\mathrm{rk}_{\mathrm{uid}\to\mathrm{P}}$ and computes $\mathrm{cph}^R\leftarrow\mathrm{ReEnc}(\mathrm{cph},\mathrm{param},\mathrm{rk}_{\mathrm{uid}\to\mathrm{P}})$.
  - Otherwise, if there exists kw_i in L_H such that $c_i=1$ and $e_2(e_1(e_0(g_0^a,g_0^b),g_0^c),g_0^r)^{a_ix_{\mathrm{uid}}}=C_2$, it picks $d\in\mathbb{Z}_p$, sets $C'_2=e_2(e_1(e_0(g_0^a,g_0^b)g_0^r),g_0)^{a_id}$, $R_2=g_0^d$ and $R_{i,j}=g_0^{a\mathrm{M}_{i,j}\upsilon_j}h_{j,\pi(i)}^{-d}$ for each $i=1,\ldots,N$ and $j=1,\ldots,n_{max}$, and returns $\mathrm{cph}^R=(C_1,C'_2,R_2,\{R_{i,j}\}_{i\in[1,N],j\in[1,n_{max}]})$;
  - Otherwise, it reports failure and terminates.
- $\mathcal{O}_{\mathrm{token}}$(uid,kw): Same as the proof of Theorem 1.
- $\mathcal{O}_{\mathrm{token}^{\mathrm{R}}}$(S,kw): Given an attribute set S and a keyword kw, the challenger proceeds as follows:
  - If $c_i=1$, select $t_1,t_2,\ldots,t_{n_{max}}\overset{R}{\leftarrow}\mathbb{Z}_p$. Compute

$$\begin{aligned}D'&=e_0(g_0^{ab}g_0^{at_1},g^{a_i})\\&=e_0(g_0^a,g_0^b)^{a_i}e_0(g_0^a,g_0)^{t_1a_i},\end{aligned}$$

for $j=1,\ldots,n_{max}$,

$$\begin{aligned} L'_j &= e_0(H(\mathrm{kw}),L_j) \\ &= e_0(g_0,g_0)^{t_j a_i}, \end{aligned}$$

and for each $x \in \mathrm{S}$,

$$\begin{aligned} \hat{D}_x &= e_0(D_x,H(\mathrm{kw})) \\ &= e_0(\prod_{j=1}^{n_{max}} h_{j,x}^{t_j},g_0^{a_i}). \end{aligned}$$

- If $c_i = 0 \wedge F(\mathrm{S}^*,\mathrm{P}) = 0$, make a query S on $\mathcal{O}_{\mathrm{KeyGen}}$ to get sk, and compute $D' = e_0(H(\mathrm{kw}),D)$, $L'_j = e_0(H(\mathrm{kw}),L_j)$ for $j = 1,\ldots,n_{max}$ and $D'_x = e_0(H(\mathrm{kw}),D_x)$ for $x \in \mathrm{S}$.
- Otherwise, report failure and terminate.

Challenge

$\mathcal{A}$ selects an uncorrupted user $\mathrm{uid}^* \notin \mathrm{CoList}$ and two equal-length keywords $(\mathrm{kw}_0,\mathrm{kw}_1)$. If $c_0 = 1 \wedge c_1 = 1$, the challenger reports failure and terminates; otherwise, let σ be a bit which is selected as follows:

- If $c_0 = 1$ and $c_1 = 0$, then set $\sigma = 1$,
- If $c_0 = 0$ and $c_1 = 1$, then set $\sigma = 0$,
- Otherwise, let $\sigma \overset{R}{\leftarrow} \{0,1\}$.

The challenger responses $\mathcal{A}$ with $\mathrm{cph}^* = (C_1 = g^r, C_2 = Z^{a_i x_{\mathrm{uid}}})$.

Phase 2

$\mathcal{A}$ executes the same as Phase 1.

Guess

$\mathcal{A}$ outputs a guess σ'. The challenger outputs $Z = g_3^{abcwr}$ if $\sigma' = \sigma$. Otherwise, it outputs $Z \neq g_3^{abcwr}$.

This completes the simulation. We can show that the challenger solves the 4-MDDH problem with advantage $(1/e + 1/q_T)\frac{\epsilon}{2}$ if $\mathcal{A}$ wins the selective security game of CP-ABRKS with advantage ϵ similar to the analysis of Theorem 1. □

Application

Our ABRKS schemes fit very well for many applications in the cloud computing environment. One of the prominent applications is about Personal Health Records (PHR) for patients: The data owner encrypted his own health records and outsourced these encrypted records to the cloud which hosts the PHR service. The data owner always needs to fetch the related health records upon some keywords since it is too costly to download all encrypted records and decrypt them to get desired records. In addition, the data owner might need to share these encrypted

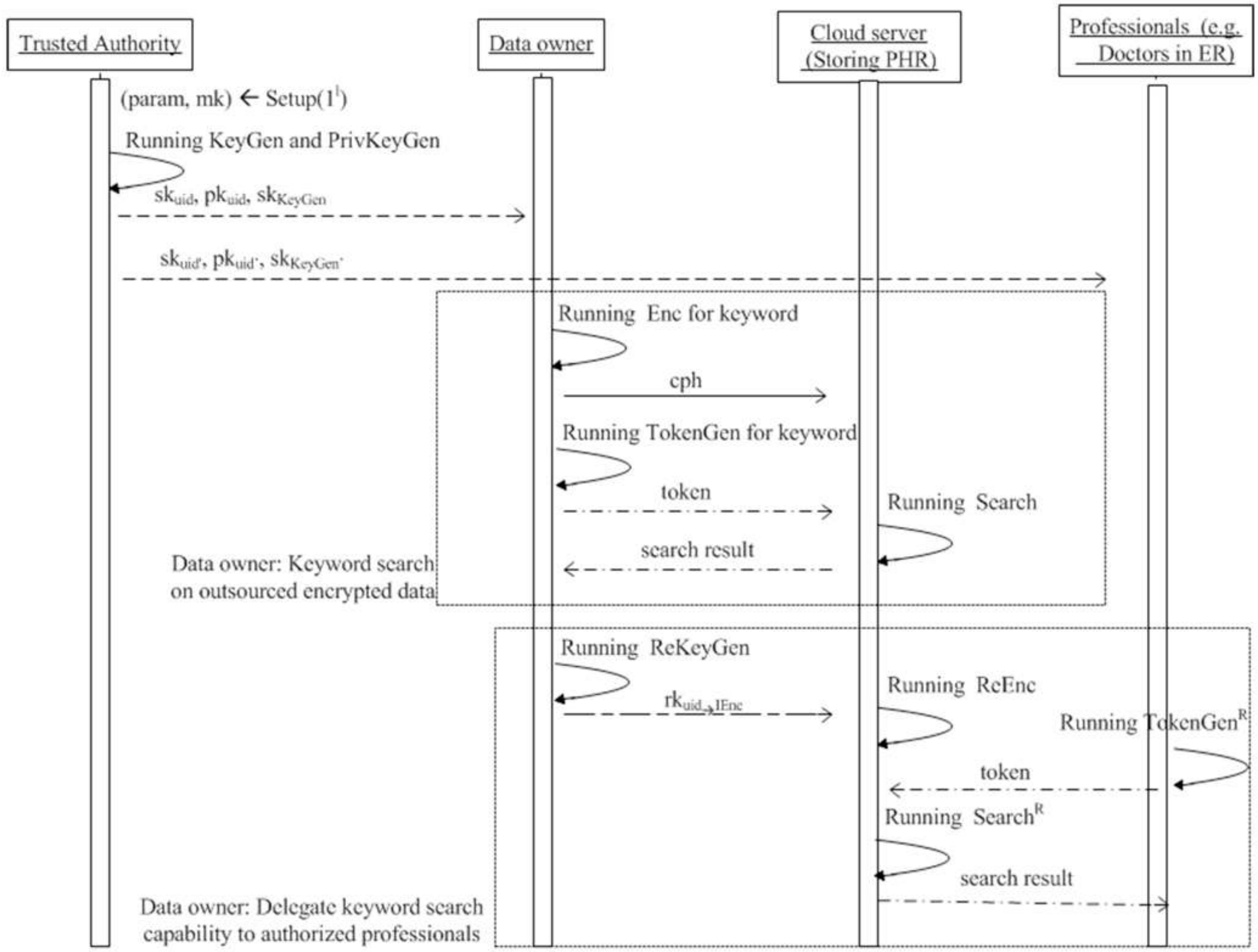
Trusted Authority
Data owner
Cloud server (Storing PHR)
Professionals (e.g. Doctors in ER)
(param, mk) ← Setup(1^l)
Running KeyGen and PrivKeyGen
sk_{uid}, pk_{uid}, sk_{KeyGen}
$sk_{uid'}$, $pk_{uid'}$, $sk_{KeyGen'}$
Running Enc for keyword
cph
Running TokenGen for keyword
token
Running Search
search result
Data owner: Keyword search on outsourced encrypted data
Running ReKeyGen
$rk_{uid \to IEnc}$
Running ReEnc
Running TokenGen^R
token
Running Search^R
search result
Data owner: Delegate keyword search capability to authorized professionals

Fig. 3. Sequence diagram for using ABRKS in the application where the data owner shares his medical records with some professionals such that only authorized professionals can retrieve medical records of their interests.

doi:10.1371/journal.pone.0116325.g003

health records with some professionals, for example, heart doctors in Emergency Room. In order to attain this goal, the data owner has to delegate the search capability. Fig. 3 shows the sequence diagram that how the entities in the PHR application make use of the proposed ABRKS schemes to achieve these goals.

Conclusions

In this paper, we propose a novel notion called attribute-based proxy re-encryption with keyword search (ABRKS). Our solutions can be used in the cloud setting, such that (1) a data owner can delegate the search capability to a group of users by specifying fine-grained access control policies; (2) the data owner and data users can delegate the tedious re-encryption and search process to the cloud without compromising data confidentiality.

Author Contributions

Conceived and designed the experiments: YFS. Performed the experiments: SQ. Analyzed the data: YFS SQ. Contributed reagents/materials/analysis tools: JQL ZH. Wrote the paper: YFS RZ QJZ.

References

1. **Zhang S, Zhang XW, Ou XM** (2014) After we knew it: empirical study and modeling of cost-effectiveness of exploiting prevalent known vulnerabilities across iaas cloud. In: 9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '14, Kyoto, Japan - June 03 - 06, 2014. pp. 317–328.
2. **Zhang S, Caragea D, Ou XM** (2011) An empirical study on using the national vulnerability database to predict software vulnerabilities. In: Database and Expert Systems Applications - 22nd International Conference, DEXA 2011, Toulouse, France, August 29 - September 2, 2011. Proceedings, Part I. pp. 217–231.
3. **Huang HQ, Zhang S, Ou XM, Prakash A, Sakallah KA** (2011) Distilling critical attack graph surface iteratively through minimum-cost SAT solving. In: Twenty-Seventh Annual Computer Security Applications Conference, ACSAC 2011, Orlando, FL, USA, 5-9 December 2011. pp. 31–40.
4. **Ding S, Yang SL, Zhang YT, Liang CY, Xia CY** (2014) Combining qos prediction and customer satisfaction estimation to solve cloud service trustworthiness evaluation problems. Knowl-Based Syst 56: 216–225.
5. **Ding S, Xia CY, Zhou KL, Yang SL, Shang JS** (2014) Decision support for personalized cloud service selection through multi-attribute trustworthiness evaluation. PloS one 9(6): e97762.
6. **Shao J, Cao ZF, Liang XH, Lin H** (2010) Proxy re-encryption with keyword search. Information Sciences 180: 2576–2587.
7. **Yau WC, Phan RCW, Heng SH, Goi BM** (2010) Proxy re-encryption with keyword search: new definitions and algorithms. In: Security Technology, Disaster Recovery and Business Continuity, Springer. pp. 149–160.
8. **Fang LM, Susilo W, Ge CP, Wang JD** (2012) Chosen-ciphertext secure anonymous conditional proxy re-encryption with keyword search. Theoretical Computer Science 462: 39–58.
9. **Wang XA, Huang XY, Yang XY, Liu LF, Wu XG** (2012) Further observation on proxy re-encryption with keyword search. Journal of Systems and Software 85: 643–654.
10. **Zhong WD, Wang XA, Wang ZQ, Ding Y** (2011) Proxy re-encryption with keyword search from anonymous conditional proxy re-encryption. In: Computational Intelligence and Security (CIS), 2011 Seventh International Conference on. IEEE, pp. 969–973.
11. **Chen X, Li Y** (2011) Efficient proxy re-encryption with private keyword searching in untrusted storage. International Journal of Computer Network and Information Security (IJCNIS) 3: 50–56.
12. **Sahai A, Waters B** (2005) Fuzzy identity-based encryption. In: Advances in Cryptology–EUROCRYPT 2005, Springer. pp. 457–473.
13. **Attrapadung N, Libert B, De Panafieu E** (2011) Expressive key-policy attribute-based encryption with constant-size ciphertexts. In: Public Key Cryptography–PKC 2011, Springer. pp. 90–108.
14. **Goyal V, Pandey O, Sahai A, Waters B** (2006) Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM conference on Computer and communications security. ACM, pp. 89–98.
15. **Rao YS, Dutta R** (2013) Computationally efficient expressive key-policy attribute based encryption schemes with constant-size ciphertext. In: Information and Communications Security, Springer. pp. 346–362.
16. **Bethencourt J, Sahai A, Waters B** (2007) Ciphertext-policy attribute-based encryption. In: Security and Privacy, 2007. SP'07. IEEE Symposium on. IEEE, pp. 321–334.

17. **Waters B** (2011) Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In: Public Key Cryptography–PKC 2011, Springer. pp. 53–70.

18. **Ibraimi L, Tang Q, Hartel P, Jonker W** (2009) Efficient and provable secure ciphertext-policy attribute-based encryption schemes. In: Information Security Practice and Experience, Springer. pp. 1–12.

19. **Zheng QJ, Xu SH, Ateniese G** (2014) VABKS: verifiable attribute-based keyword search over outsourced encrypted data. In: 2014 IEEE Conference on Computer Communikations, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014. pp. 522–530.

20. **Sun WH, Yu SC, Lou WJ, Hou YT, Li H** (2014) Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. In: 2014 IEEE Conference on Computer Communikations, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014. pp. 226–234.

21. **Guo SQ, Zeng YP, Wei J, Xu QL** (2008) Attribute-based re-encryption scheme in the standard model. Wuhan University Journal of Natural Sciences 13: 621–625.

22. **Li KY, Wang JF, Zhang YH, Ma H** (2014) Key policy attribute-based proxy re-encryption and rcca secure scheme. Journal of Internet Services and Information Security (JISIS) 4: 70–82.

23. **Liang XH, Cao ZF, Lin H, Shao J** (2009) Attribute based proxy re-encryption with delegating capabilities. In: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security. ACM, pp. 276–286.

24. **Luo S, Hu JB, Chen Z** (2010) Ciphertext policy attribute-based proxy re-encryption. In: Information and Communications Security, Springer. pp. 401–415.

25. **Mizuno T, Doi H** (2011) Hybrid proxy re-encryption scheme for attribute-based encryption. In: Information Security and Cryptology. Springer, pp. 288–302.

26. **Liang KT, Fang LM, Susilo W, Wong DS** (2013) A ciphertext-policy attribute-based proxy re-encryption with chosen-ciphertext security. In: Intelligent Networking and Collaborative Systems (INCoS), 2013 5th International Conference on. IEEE, pp. 552–559.

27. **Boneh D, Silverberg A** (2003) Applications of multilinear forms to cryptography. Contemporary Mathematics 324: 71–90.

28. **Garg S, Gentry C, Halevi S** (2013) Candidate multilinear maps from ideal lattices. In: Advances in Cryptology–EUROCRYPT 2013, Springer. pp. 1–17.

29. **Coron JS, Lepoint T, Tibouchi M** (2013) Practical multilinear maps over the integers. In: Advances in Cryptology–CRYPTO 2013, Springer. pp. 476–493.

30. **Sahai A, Seyalioglu H, Waters B** (2012) Dynamic credentials and ciphertext delegation for attribute-based encryption. In: Advances in Cryptology–CRYPTO 2012, Springer. pp. 199–217.