11-2013

# Achieving revocable fine-grained cryptographic access control over cloud data

Yanjiang YANG

Xuhua DING
*Singapore Management University*, xhding@smu.edu.sg

Haibing LU

Zhiguo WAN

Jianying ZHOU

# Achieving Revocable Fine-Grained Cryptographic Access Control over Cloud Data

Yanjiang Yang[1](✉), Xuhua Ding[2], Haibing Lu[3], Zhiguo Wan[4], and Jianying Zhou[1]

[1] Institute for Infocomm Research, Singapore, Singapore
{yyang,jyzhou}@i2r.a-star.edu.sg

[2] School of Information Systems, Singapore Management University, Singapore, Singapore
xhding@smu.edu.sg

[3] The Leavey School of Business, Santa Clara University, Santa Clara, USA
hlu@scu.edu

[4] School of Software, Tsinghua University, Beijing, China
wanzhiguo@tsinghua.edu.cn

**Abstract.** Attribute-based encryption (ABE) is well suited for fine-grained access control for data residing on a cloud server. However, existing approaches for user revocation are not satisfactory. In this work, we propose a new approach which works by splitting an authorized user's decryption capability between the cloud and the user herself. User revocation is attained by simply nullifying the decryption ability at the cloud, requiring neither key update nor re-generation of cloud data. We propose a concrete scheme instantiating the approach, which features lightweight computation at the user side. This makes it possible for users to use resource-constrained devices such as mobile phones to access cloud data. We implement our scheme, and also empirically evaluate its performance.

## 1 Introduction

Data residing on a cloud storage need to be encrypted in order to safeguard their secrecy against the untrusted cloud provider [8–10], and to serve as an access control mechanism where a user's decryption capability is assigned according to the access control policy. For instance, a hospital encrypts its medical data outsourced to a cloud storage such that a patient's medical records are only allowed to be decrypted by her doctors and nurses. Attribute-based encryption (ABE) [6,12,18], has been hailed as the solution to cloud data encryption because it enforces fine-grained access control over the decryption capabilities. Informally, as an one-to-many encryption mechanism, ABE allows data to be encrypted with certain policy/attributes while each decryption key is associated with certain attributes/policy. Only when the attributes satisfy the policy can the ciphertext be deciphered correctly by the key.

Nonetheless, *user revocation* remains a thorny problem in the setting of cloud data encryption without a satisfactory solution. In the following, we review several possible approaches to user revocation. Firstly, authentication-based revocation used in conventional access control systems are expensive for the cloud setting. This approach requires an extra authentication mechanism at the user-cloud interface. It costs the user to possess extra secrets and the cloud server is burdened with the authentication task. The second approach is key-update based revocation as proposed in [1,19,20]. This method suffers from poor scalability as all data must be re-encrypted and all remaining legitimate user keys are to be updated or re-distributed, whose cost is tremendously high when the data volume or the number of users scales up.

The third approach is to retrofit ABE schemes with revocation support by assigning revocation related attributes. The ABE schemes in [6,12] propose to include an "expiry time" attribute in the attribute set such that each decryption key is effective only for a period of time. The shortcoming of this method is that it does not allow for immediate revocation due to the time gap. In [16], Ostrovsky et. al. propose *negative* constrains in the access policy, such that a revocation of certain attributes amounts to negating these attributes. This mechanism is not scalable in revoking individual users, as each encryption has to involve information of all revoked users each of which is treated as a distinctive attribute.

The fourth approach is key splitting [4], where an untrusted security mediator holds one part of a decryption key. Once a user is revoked the mediator immediately refuses to participate in her decryption. Although this paradigm is suitable for the cloud setting where the cloud server plays the role as a security mediator, it requires the underlying decryption algorithm to be homomorphic. However, existing homomorphic encryption schemes do not allow for fine-grained access control policies. Another downside of this approach is that it requires a TTP (Trusted Third Party) to perform key splitting which is not suitable for certain cloud storage applications.

In short, cloud storage applications desire an encryption mechanism with the following two properties: (a) fine-grained access control with strong expressiveness to describe complex access policies; (b) scalable and efficient revocation to instantly nullify a user's decryption privilege without affecting legitimate users and inflicting high cost on the cloud server. This paper presents such an encryption system for cloud storage. We first envision a general approach solving the user revocation problem in encryption of cloud storage, with the basic idea of splitting a user's decryption capability between the cloud server and the user herself, such that a full decryption requires the cloud server's assistance. User revocation is achieved by instructing the cloud server not to offer the needed assistance. We then propose a concrete scheme realizing the approach, which achieves the same level of fine-grained access control as ABE. Different from key splitting, decryption-capability splitting does not require a TTP to know all users' secrets. Another feature of our scheme is the lightweight computation at the user end, such that a user only performs power exponentiations in a regular algebraic group, though the scheme is based on bilinear mapping. We have implemented our scheme, and empirically tested on a smartphone where the decryption at the user side only takes 12 ms.

**Organization**. In Sect. 2, we elaborate our decryption-capability splitting approach for user revocation, and formulate a system model. A concrete instantiation is presented in Sect. 3. Experimental results are given in Sect. 4, and Sect. 5 introduces related work. Section 6 concludes the paper.

## 2 Synopsis

### 2.1 System Setting

A data owner (denoted as Owner) uploads her data records to a cloud storage server (denoted by Server). Without fully trusting the Server, the Owner encrypts her cloud data such that data privacy is protected against the Server. This encrypted cloud data storage is to be accessed by a group of users authorized by the Owner. Data encryption enforces fine-grained access control, such that different users have different decryption capabilities matching their respective functional roles. In particular, a user's decryption capability is delineated by a set of attributes according to her role. Each data encryption is associated with an access control policy such that a user can successfully decipher the encrypted record only if her attributes satisfy the record's policy. Under certain circumstances, the Owner may need to revoke a user, in the sense that the revoked user is not allowed to decipher any record in the cloud. We consider the Server as an honest-but-curious adversary, which honestly runs the algorithms or tasks assigned to it while attempting to attack data privacy. Figure 1 depicts an overview of the system.

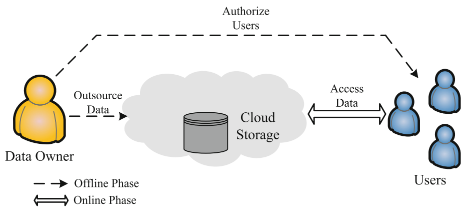

**Fig. 1.** An overview of the cloud storage system

### 2.2 Fine-Grained Access Control

To facilitate fine-grained decryption capabilities, we use "attribute" and "access structure" utilized in [18] to describe our access control model.

**Attributes.** Let $\Lambda$ denote the dictionary of attributes used in the system. Each user $u$ of the cloud storage is assigned with a set of attributes $S_u \subseteq \Lambda$. The attribute assignment procedure is application specific which is beyond the scope of this paper.

**Policy and Access Structure.** In our system, an access control policy is expressed by a monotonic access structure which is a subset of $2^{\Lambda}$. In particular, a collection $\mathbb{A} \subset 2^{\Lambda}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$, then $C \in \mathbb{A}$. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) $\mathbb{A}$ of non-empty subsets of $\Lambda$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{A}$ are called the unauthorized sets. We restrict our attention to monotone access structures in this work.

**Linear Secret-Sharing Schemes.** Linear secret-sharing schemes will be used to establish access structures. A secret-sharing scheme $\Pi$ over a set $S$ of attributes is linear if (1) the shares for each attribute form a vector over $Z_p$; (2) there exists a matrix $M$ (with $\ell$ rows and $n$ columns) called the share-generating matrix for $\Pi$. For the $i^{th}$ row of $M$, i = 1, ..., $\ell$, we let the function $\rho$ define the attribute labeling row $i$ as $\rho(i)$. When we consider the column vector $\boldsymbol{v} = (x, y_2, \cdots, y_n)$, where $x \in Z_p$ is the secret to be shared, and $y_2, \cdots, y_n \in Z_p$ are random, then $M \cdot \boldsymbol{v}$ is the vector of $\ell$ shares of the secret $x$ according to $\Pi$. The share $(M \cdot \boldsymbol{v})_i$ belongs to attribute $\rho(i)$.

It is shown in [2] that every linear secret-sharing scheme as above also enjoys the linear reconstruction property: Suppose that $\Pi$ is a linear secret-sharing scheme for the access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ be any authorized set, and let $I = \{i : \rho(i) \in S\} \subset \{1, 2, ..., \ell\}$. Then, there exist a set of constants $\{\omega_i\}_{i \in I}$ such that if $\{\lambda_i\}_{i \in I}$ are valid shares of any secret $x$ according to $\Pi$, then $\sum_{i \in I} \omega_i \lambda_i = x$. Furthermore, these constants $\{\omega_i\}_{i \in I}$ can be found in time polynomial in the size of the share-generating matrix $M$.

## 2.3 Our Approach

A notable difference (in terms of data encryption) between the cloud storage setting and the conventional PKI or group communication settings is that the untrusted cloud server is always involved in users' data accesses. To leverage this fact towards addressing user revocation, we split a user's decryption capability between the Server and the user herself, such that decryption of encrypted cloud record depends on the cooperation between Server and the user. Specifically, a user's decryption capability is rendered by a *server-side key* and a *user-side key*, where the former is held by the Server and the latter is possessed by the user. To manage users' server-side keys, the Server maintains a Server-side Key list, with each entry containing a user's identity and her server-side key. When the user requests a data record from cloud, the Server executes a *server decryption* operation over the data with the user's server-side key, generating an intermediate value. The intermediate value is then returned to the user, who gets the plaintext data by a *user decryption* operation using her user-side key. We remark that the pair of server-side key and user-side key are not the result of key splitting. Therefore, our approach does not require a trusted party (i.e., the Owner in our case) to compute the user-side key on behalf of the user.

In line with our approach, a secure cloud storage system involves three types of entities, the Owner, a set of users, and the Server. These entities interact with one another in the following algorithms.

**Definition 1.** *A secure cloud storage system (SCSS) is defined as a collection of the following seven algorithms.*

**Setup**$(1^\kappa) \to (params, msk)$**:** *Taking as input a security parameter $1^\kappa$, the Owner executes this algorithm to set up public parameters params and a master secret key msk. Below we assume that params is implicit in the input of the rest algorithms, unless stated explicitly.*

**UsKGen**$(u) \to (Upk_u, Usk_u)$**:** *The user-side key generation algorithm takes as input the public parameters and a user identity $u$, and outputs a pair of user-side public/private keys $(Upk_u, Usk_u)$ for $u$.*
*Each user executes this algorithm to generate a key pair for herself.*

**SsKGen**$(msk, Upk_u, S) \to SsK_u$**:** *The server-side key generation algorithm takes as input master secret key msk, user-side public key $Upk_u$, and a set $S \subset \Lambda$ of attributes, and outputs a server-side key $SsK_u$ for $u$.*
*The Owner executes this algorithm to authorize a user, based on her attributes. The server-side key $SsK_u$ will be secretly given to the Server who then adds a new entry in the Server-side Key list $\mathcal{L}_{SsK}$, i.e., $\mathcal{L}_{SsK} = \mathcal{L}_{SsK} \cup \{u, SsK_u\}$.*

**Encrypt**$(m, \mathbb{A}) \to c$**:** *The encryption algorithm takes as input a message $m$ and an access structure $\mathbb{A}$, and outputs a ciphertext $c$ by encrypting data $m$ under the access structure.*
*The Owner executes this algorithm to encrypt a data record to be uploaded to the Server.*

**SDecrypt**$(SsK_u, c) \to v$**:** *The server decryption algorithm takes as input a server-side key $SsK_u$ and a ciphertext $c$, and outputs an intermediate value $v$.*
*The Server executes this algorithm to help a user decrypt a scrambled data record requested by the user $u$ with her server-side key.*

**UDecrypt**$(Usk_u, v) \to m$**:** *The user decryption algorithm takes as input a user-side private key $Usk_u$ and an intermediate value $v$, and outputs a plaintext $m$.*
*An authorized users executes this algorithm to obtain the desired data from the intermediate value returned by the Server.*

**Revoke**$(u, \mathcal{L}_{SsK}) \to \mathcal{L}_{SsK} \setminus \{u, SsK_u\}$**:** *Taking as input a user identity $u$ and the Server-side Key list $\mathcal{L}_{SsK}$, the algorithm revokes $u$'s decryption capability by removing her entry from $\mathcal{L}_{SsK}$, and outputs the updated $\mathcal{L}_{SsK} = \mathcal{L}_{SsK} - \{u, SsK_u\}$.*

Correctness of the system requires that $\texttt{UDecrypt}(Usk_u, \texttt{SDecrypt}(SsK_u, c)) = m$ if $S$ satisfies $\mathbb{A}$, for all $(Upk_u, Usk_u) \leftarrow \texttt{UsKGen}(u), SsK_u \leftarrow \texttt{SsKGen}(msk, Upk_u, S)$ and $c \leftarrow \texttt{Encrypt}(m, \mathbb{A})$, where $(params, msk) \leftarrow \texttt{Setup}(1^\kappa)$.

Caveat. To highlight the distinction between decryption-capability splitting and key splitting syntactically, we point out that for the latter, the UsKGen algorithm and the SsKGen algorithm above could be combined into a single algorithm which is to be executed by the same entity (i.e., the Owner in our case).

**Security Requirements.** Below we explain the intuitions of three security requirements imposed upon the system, while the formulations are deferred to Appendix A.

*Data Privacy Against Cloud.* In our adversarial model, the Server is an honest-but-curious adversary. It stores the Owner's data and performs server decryptions to serve users' accesses by applying the corresponding server-side keys. It mandates that the Server cannot learn any semantic information about the data in the storage, but it should behave honestly in terms of managing cloud data, processing user access requests, and other administrative activities.

*Data Privacy Against Users.* It mandates that a user cannot obtain data beyond the authorized access rights stipulated by the Owner. In particular, the collusion between a set of malicious users does not yield new decryption capabilities beyond those privileges assigned to those users.

*User Revocation Support.* When a user's decryption capability is revoked, she is no long able to decipher the encrypted data on the storage. In other words, without the assistance of the Server, no user can decipher the encrypted data by herself, even when her attributes satisfy the policy attached to the ciphertext.

## 3 A Concrete Instantiation

In this section, we present a concrete instantiation of the above approach and algorithms. Of particular interest of our scheme is the computation efficiency at the user side, i.e., the user decryption algorithm UDecrypt only involves exponentiation operations in regular algebraic groups, while the scheme itself is based on bilinear mapping. This makes it possible for users to use resource-constrained device such as mobile phone to access the cloud data. This is one of the features that distinguish our scheme from all other proposals on using ABE for encryption of cloud data, e.g., those reviewed in Sect. 5.

### 3.1 Construction Details

Our construction is based on the scheme in [18], which is proved secure under the decisional $q$-BDHE assumption. The main trick we have rests with the generation of a user's server-side key, such that an ABE ciphertext can be transformed into one under the user's (standard) public key.

Let $s \in_R S$ denote an element $s$ randomly drawn from a set $S$. The details of our scheme are as follows.

**Setup**$(1^\kappa) \rightarrow (params, msk)$**:** On input a security parameter $1^\kappa$, the setup algorithm

- determines a bilinear map $e : G_0 \times G_0 \rightarrow G_1$, where $G_0$ and $G_1$ are cyclic multiplicative groups of $\kappa$-bit prime order $p$.
- selects random exponents $z, a \in Z_p$, and a generator $g$ of $G_0$.
- chooses a cryptographic hash function $H : \{0,1\}^* \rightarrow G_0$.
- decides a standard public key encryption scheme. Below we use $\mathsf{Enc}(\cdot,\cdot)$ to denote both the description of the scheme and its encryption function.
- sets $params = (g, e(g,g)^z, g^a, H, \mathsf{Enc})$, and $msk = g^z$.

**UsKGen**$(u) \rightarrow (Upk_u, Usk_u)$**:** On input a user identity $u$, the user-side key generation algorithm outputs a public/private key pair $(Upk_u, Usk_u)$ for $\mathsf{Enc}$.

**SsKGen**$(msk, Upk_u, S) \rightarrow SsK_u$**:** On input master secret key $msk = g^z$, a user public key $Upk_u$, and a set of attributes $S$, the server-side key generation algorithm picks $\alpha, t \in_R Z_p$, and sets the server-side key $SsK_u$ for $u$ as

$$SsK_u = \Big(K = g^{z\alpha} g^{at}, K' = \mathsf{Enc}(Upk_u, \alpha), L = g^t, \{\forall s \in S : K_s = H(s)^t\}\Big),$$

where $\mathsf{Enc}(Upk_u, \alpha)$ denotes encryption of $\alpha$ by $\mathsf{Enc}$ under $Upk_u$.

**Encrypt**$(m, \mathbb{A}) \rightarrow c$**:** Taking as input a message $m$, and an access structure $\mathbb{A} = (M, \rho)$, and implicitly $params$, the encryption algorithm proceeds as follows.
We limit $\rho$ to be injective function, i.e., an attribute is associated with at most one row of $M$. Let $M$ be an $\ell \times n$ matrix. It first selects a random column vector $\boldsymbol{v} = (x, y_2, \ldots, y_n) \in Z_p^n$, which will be used to secret-share the exponent $x$. Calculates $\lambda_i = M_i \cdot \boldsymbol{v}, i = 1$ to $\ell$, where $M_i$ denotes the $i^{th}$ row of $M$. Sets the ciphertext $c$ as

$$c = (C = m \cdot e(g,g)^{zx}, C' = g^x, \{\forall i \in [1 \cdot \cdot l] : C_i = g^{a\lambda_i} H(\rho(i))^{-x}\}, (M, \rho))$$

**SDecrypt**$(SsK_u, c) \rightarrow v$**:** On input a server-side key $SsK_u$ associating with a set of attributes $S$, and a ciphertext $c$ under access structure $(M, \rho)$, the server decryption algorithm outputs an intermediate value $v$ if $S$ satisfies the access structure, or $\perp$ otherwise. Suppose $S$ satisfies the access structure and $I = \{i : \rho(i) \in S\} \subset \{1, 2, \cdots, \ell\}$. Then, let $\{\omega_i\}_{i \in I}$ be a set of constants such that if $\{\lambda_i\}$ are valid shares of a secret $x$ according to $M$, then $\sum_{i \in I} \omega_i \lambda_i = x$. Note that it suffices to determine $\{\omega_i\}_{i \in I}$ based on $M$ and $I$. The algorithm first computes

$$\begin{aligned}
&\frac{e(C', K)}{\prod_{i \in I}(e(C_i, L) e(C', K_{\rho(i)}))^{\omega_i}} \\
&= \frac{e(g^x, g^{z\alpha} g^{at})}{\prod_{i \in I}(e(g^{a\lambda_i} H(\rho(i))^{-x}, g^t) e(g^x, H(\rho(i))^t))^{\omega_i}} \\
&= \frac{e(g,g)^{xz\alpha} e(g,g)^{xat}}{\prod_{i \in I}(e(g,g)^{a\lambda_i t} e(H(\rho(i)), g)^{-xt} e(g, H(\rho(i)))^{xt})^{\omega_i}} \\
&= \frac{e(g,g)^{xz\alpha} e(g,g)^{xat}}{\prod_{i \in I} e(g,g)^{a\lambda_i t \omega_i}} \\
&= e(g,g)^{xz\alpha}.
\end{aligned}$$

Then, sets $v$ as

$$\begin{aligned} v &= (V_1, V_2, V_3) \\ &= (C = m \cdot e(g,g)^{zx}, e(g,g)^{xz\alpha}, K' = \mathsf{Enc}(Upk_u, \alpha)) \end{aligned}$$

**UDecrypt**$(Usk_u, v) \rightarrow m$**:** On input a user-side private key $Usk_u$ and an intermediate value $v$ resulting from the server decryption algorithm with $u$'s server-side key, the user decryption algorithm works as follows. Recall that $v = (V_1 = m \cdot e(g,g)^{zx}, V_2 = e(g,g)^{xz\alpha}, v_3 = \mathsf{Enc}(Upk_u, \alpha))$. Decryption is straightforward, by decrypting $V_3$ using $Usk_u$ to get $\alpha$, and then computing $e(g,g)^{xz} = (V_2)^{1/\alpha}$ and finally getting $m = V_1/e(g,g)^{xz}$.

**Revoke**$(u, \mathcal{L}_{SsK}) \rightarrow \mathcal{L}_{SsK} \setminus \{u, SsK_u\}$**:** On input of a user identity $u$ and the Server-side Key list $\mathcal{L}_{SsK}$, the entry corresponding to $u$ is removed from $\mathcal{L}_{SsK}$. Depending on applications, either the Server updates $\mathcal{L}_{SsK}$ instructed by the Owner, or an interface is provided to the Owner so that he does the deletion.

**Correctness.** Correctness of the construction is easily verified, so we do not elaborate.

**Performance.** The bit length of the intermediate value $v$ returned by the SDecrypt algorithm is $2|G_1| + |\mathsf{Enc}(\cdot)|$, independent of the complexity of the encryption access policy (represented by the total number of rows of the share-generating matrix). So is the computation overhead of the UDecrypt algorithm, which consists mainly of a decryption operation of a standard public key encryption scheme and a power exponentiation in $G_1$. This means that no pairing operation is involved at the user side, although the scheme bases itself on bilinear mapping and enjoys the fine grained encryption/decryption capabilities comparable to the ABE schemes in [6,18].

## 3.2 Security Analysis

Security of our scheme is reduced to the decisional $q$-Bilinear Diffie-Hellman Exponnet ($q$-BDHE) assumption defined in [18].

**Assumption 1** *[Decisional q-BDHE Assumption]***.** Let $G_0$ be a group of prime order $p$. Let $a, x \in Z_p$ be randomly chosen, and $g$ be a generator of $G_0$. Let $\boldsymbol{v} = (g, g^x, g^a, g^{a^2}, g^{a^3}, \cdots, g^{a^q}, g^{a^{q+2}}, \cdots, g^{a^{2q}})$. The decisional $q$-BDHE assumption holds if for any PPT adversary $\mathcal{A}$ and $Z \in_R G_0$,

$$|\Pr[\mathcal{A}(\boldsymbol{v}, g^{a^{x(q+1)}}) = 1] - \Pr[\mathcal{A}(\boldsymbol{v}, Z) = 1]| < \nu$$

where $\nu$ is a negligible function, and the the probability is taken over the random choice of $a, x \in Z_p$, $Z \in G_1$, and of the random coins of $\mathcal{A}$.

We have the following theorem, regarding data privacy and user revocation support of our scheme, and the security proof can be found in Appendix B.

**Theorem 1.** *The above scheme achieves data privacy and user revocation support (specified in Definitions 2 and 3), if the decisional q-BDHE assumption holds, the standard public key encryption* Enc *is semantically secure, and $H(\cdot)$ is a random oracle.*

## 4 Experimental Results

To gauge its performance, we have implemented and experimented with our proposed scheme. The implementation is coded in Java, and the cryptographic algorithms are implemented based on the Bouncy Castle Java Crypto Library[1]. We instantiated the bilinear map in the scheme with a 512-bit supersingular curve with embedding degree 2.

### 4.1 Experimental Results

*Cost in Cloud and Owner.* In our scheme, the computation costs of both Encrypt and SDecrypt are dependent on the complexity of the access control policy, linear to the number of rows of the share-generating matrix. These costs constitute the most demanding part of our construction.

To empirically evaluate the computational costs of the Encrypt and SDecrypt algorithms (while avoiding compounding factors such as network latency, instability of on-demand computing, etc.), we run these algorithms on a PC with 2.66 GHz Intel Core2Duo and 3.25 GB RAM. We experiment them with a set of access structures, whose share-generating matrixes are of $\ell$ rows and $\ell$ columns. Access structures in such a form ensure that all involved attributes are used in the SDecrypt algorithm, thus imposing the heaviest workload. We repeat each experiment for 100 times and calculate the average. The experimental results are shown in Fig. 2, which displays the time of the two algorithms with respect to the number of attributes (i.e., the number of rows of the share-generating matrix).

As evident from the figure, the results corroborate the fact that both algorithms perform linear computations with the number of attributes. Note that the cost of SDecrypt with 60 attributes is about 1.7 s. This performance should be acceptable for practical applications, since 50–60 attributes should suffice for specifying access control policies based on functional roles.

*Cost in User End.* We have also implemented and tested the UDecrypt algorithm of our scheme on an HTC HD2 smartphone, which is configured with a 1 GHz Scorpion CPU and 448 MB RAM. We instantiate the standard public-key encryption scheme Enc associated with the user-side public/private key pair by the ElGamal-type encryption in $G_0$, i.e., $(Upk_u, Usk_u) = (g^{x_u}, x_u \in Z_p^*)$. The experimental results indicate that on average, it takes about 12 ms to decrypt a ciphertext of the form $(m \cdot e(g,g)^{zx}, e(g,g)^{xz\alpha}, Enc(Upk_u, \alpha))$. On the other hand, the communication overhead for the user is $2|G_1| + 2|G_0|$, about 1.5 Kbits in our implementation. These results suggest that it is affordable for a user to access the cloud storage using a low-end device like a smartphone.

[1] http://www.bouncycastle.org/java.html.

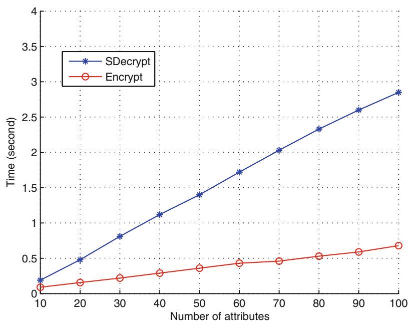

**Fig. 2.** Experimental results

## 5 Related Work

This section provides an overview of the related work on encryption of cloud data and user revocation, respectively.

**Encryption of Cloud Data with ABE.** Among many proposals on enforcement of cryptographic access control upon cloud data with encryption, we focus on those using ABE and PRE. The proposal by Yu *et al.* applies KP-ABE for encryption of cloud data to achieve fine-grained data sharing [20]. For user revocation support, they suggest adopting the specific technique of PRE scheme [3] to update users' decryption keys. The advantage is that the cloud is entrusted to taking the majority of the workload for re-generation of cloud data and re-distribution of new keys. While their scheme improves considerably over the trivial solution, i.e., the Owner is fully responsible for data re-generation and key re-distribution, it is always preferable not to burden the cloud if possible. In addition, their scheme requires the cloud to maintain user revocation information for legitimate users to gradually complete key update. Our decryption-capability splitting approach avoids these problems and the overhead incurred due to user revocation is minor.

Wang *et al.* achieve hierarchical attribute-based encryption for cloud storage, by augmenting CP-ABE with hierarchical identity-based encryption [19]. Hierarchical attribute-based encryption could cope with more complicated application requirements, but for user revocation, their scheme is similar to [20]. The proposal by Liu *et al.* also aims at hierarchical attribute-set-based encryption for cloud storage [15], and its approach for user revocation is the "expiry time" mechanism.

As a final note, our proposed scheme distinguishes itself from all the above work because the computation at the user side is lightweight, independent of the complexity of the access control policy of the underlying ABE scheme.

**Key-Split Cryptography.** We realize that the idea of splitting key for revocation of cryptographic capabilities is not new. Boneh *et al.* propose "mediated RSA" to split the private key of RSA into two shares, such that one share is delegated to an online "mediator" and the other is given to the user [4]. As RSA decryption and signing require the engagement of both parties, the user's cryptographic capabilities are immediately revoked if the mediator does not cooperate. Our approach for user revocation follows a similar rationale, but it is more precisely split of decryption capability (rather than simply split of key). A technical difference resulting from this distinction is that the party responsible for key split knows both shares, but the party in charge of capability split does not necessarily know both shares, i.e. in our scheme, the Owner does not learn users' private keys.

A broader class of cryptographic primitives related to key split is threshold cryptography, e.g., [13,17]. A threshold cryptosystem works by splitting a private key among $n$ parties, in a way that any $t$ out of $n$ parties together can decrypt or sign. Threshold cryptography is by no means designed for user revocation purposes; rather, it is intended to work in a distributed environment where individual parties are restricted from abusing cryptographic capabilities.

The recent work by Green *et al.* [11] proposes delegating the bulk of decryption overhead of ABE to a powerful proxy server in order to mitigate the burden at the user side. As a result, the user only performs a standard ElGamal decryption operation (similar to ours). While their constructions are not intended for user revocation, they can be directly used to instantiate our approach, but resulting in a scheme of key splitting. In contrast, our scheme implements decryption-capability splitting with the advantage that users do not need to disclose their secret keys to the Owner.

## 6 Conclusions

A main issue to be addressed when using ABE for encryption of cloud storage is user revocation. In this work we proposed a decryption-capability splitting approach for user revocation, which is advantageous over existing solutions. A user's decryption capability is split between the cloud and the user herself, such that user revocation is achieved by simply invalidating the cloud's decryption ability. As a result, neither key update nor re-generation of cloud data is required. We further proposed a concrete scheme instantiating the approach, which is featured with lightweight computation at the user side such that users can use resource-constrained devices to access cloud data.

**Acknowledgments.** This work is supported in part by A*STAR funded project SecDC-112172014 (Singapore), and the second author is funded by the Singapore Management University through the research grant MSS12C004 from the Ministry of Education Academic Research Fund Tier 1.

## A Formulation of Security Notions

The definitional model for data privacy against cloud is captured in the following game between a challenger managing a SCSS system and an adversary who wants to break the system.

**Definition 2** *[Data Privacy Against Cloud]. A secure cloud storage system (SCSS) satisfies data privacy against cloud if for any PPT adversary, the probability of the following game returns* $1$ *is* $1/2 + \nu(\kappa)$*, where* $\nu(.)$ *is a negligible function.*

**Setup.** *The challenger runs the Setup algorithm and gives the public parameters params to the adversary.*

**Phase 1.** *The adversary makes repeated queries to the server-side key generation oracle by submitting sets of attributes* $S_1, ..., S_{q_1}$*. For each query, the challenger first runs the UsKGen algorithm to get a user-side public/private key pair; with the user-side public key and the attribute set* $S_i$*, the challenger then runs the SsKGen algorithm to get a server-side key; the challenger returns the server-side key together with the user-side public key to the adversary.*

**Challenge.** *The adversary submits two equal length messages* $m_0$ *and* $m_1$*, together with a challenge access structure* $\mathbb{A}^*$*. The challenger flips a random coin* $b$*, runs the Encrypt algorithm on* $m_b$ *and* $\mathbb{A}^*$*, and returns the ciphertext* $c^*$ *to the adversary.*

**Phase 2.** *Phase 1 is repeated.*

**Guess.** *The adversary outputs a guess* $b'$ *on* $b$*. If* $b' = b$*, then the challenger returns 1; otherwise returns 0.*

The formulation of data privacy against authorized users and of user revocation support bases on the same fact that without an appropriate server-side key, a user cannot decrypt even with her user private key. Following is the formal security model.

**Definition 3** *[Data Privacy against Users & Revocation Support]. A secure cloud storage system (SCSS) satisfies data privacy against users and user revocation support if for any PPT adversary, the probability of the following game returns* $1$ *is* $1/2 + \nu(\kappa)$*, where* $\nu(.)$ *is a negligible function.*

**Init.** *The adversary declares the access structure* $\mathbb{A}^*$ *he wants to be challenged upon.*

**Setup.** *The challenger runs the Setup algorithm and returns the public parameters params to the adversary.*

**Phase 1.** *The adversary makes repeated queries to the user-side key generation oracle (UsKGen), and the server-side key generation oracle (SsKGen). For the former, the challenger returns the resulting user-side key (both public and private) to the adversary; for the latter, the adversary submits sets of attributes* $S_1, ..., S_{q_1}$ *with the restriction that each* $S_i$ *does not satisfy* $\mathbb{A}^*$*, and the challenger returns the resulting server-side key to the adversary.*

**Challenge.** *The adversary submits two equal length messages* $m_0$ *and* $m_1$*, together with the challenge access structure* $\mathbb{A}^*$*. The challenger flips a random* *coin $b$, runs the Encrypt algorithm on $m_b$ and $\mathbb{A}^*$, and returns the ciphertext $c^*$ to the adversary.*

**Phase 2.** *Phase 1 is repeated.*

**Guess.** *The adversary outputs a guess $b'$ on $b$. If $b' = b$, then the challenger returns 1; otherwise returns 0.*

We stress that giving the server-side keys to the adversary models authorized users's ability to get intermediate values from the $\mathsf{Server}$ (who uses the server-side keys). Intuitively, user revocation support (in which case the adversary does not have any server-side key) is implied by the fact the adversary even cannot decrypt the challenge ciphertext without *appropriate* server-side keys.

## B Security Proof for Theorem 1

*Proof.* We prove that our scheme satisfies Definitions 2 and 3, respectively.

*Satisfying Definition 2.* The proof is much simpler than in [18], due to the use of semantically secure public-key encryption $\mathsf{Enc}$, and the fact that the adversary does not have the private key. Satisfaction of Definition 2 is actually based on the DBDH (Decisional Bilinear Hiffie-Hellman) assumption, which states that it is infeasible to distinguish between $(g \in G_0, g^c, g^d, g^x, e(g,g)^{cdx} \in G_1)$ and $(g, g^c, g^d, g^x, Z \in_R G_1)$. The DBDH assumption clearly is weaker than the decisional $q$-BDHE assumption.

Suppose we have an adversary $\mathcal{A}$ with non-negligible advantage $\mathsf{Adv}_{\mathcal{A}}$ in the game of Definition 2 against our scheme. We build a challenger $\mathcal{C}$ breaking the DBDH assumption. Details follow.

**Setup.** The challenger takes in the DBDH challenge $(g, g^c, g^d, g^x, Z)$. The challenger implicitly sets $z = cd$ by setting $e(g,g)^z = e(g^c, g^d)$, and selects a random number in $G_0$ as $g^a$. In addition, the challenger programs the random oracle $H$ by building a table as follows. Consider a call to $H(s)$. If $H(s)$ was already defined in the table, then simply return the same answer as before; otherwise, select a random value $\tau_s \in Z_p$ and define $H(s) = g^{\tau_s}$.

**Phase 1.** The challenger answers server-side key generation queries from the adversary. For a query, the challenger first generates a public/private key pair for $\mathsf{Enc}$ by executing $\mathsf{UsKGen}$; then chooses $z', t \in_R Z_p$ and a random $K'$ from the range of $\mathsf{Enc}$, and computes $K = g^{z'} g^{at}, L = g^t, \forall s \in S : K_s = g^{\tau_s t}$. We argue that the distribution of the simulated key $(K, K', L, \{K_s\})$ so generated is computationally indistinguishable from the actual server-side key. First, due to the semantic security of $\mathsf{Enc}$, the randomly chosen $K'$ is indistinguishable from $\mathsf{Enc}(\alpha)$ in the actual key. Second, conditioned on the random $K'$ replacing $\mathsf{Enc}(\alpha)$, the $g^{z\alpha}$ in the actual $K$ is no different from $g^{z'}$ for a random $z'$. Our argument thus holds.

**Challenge.** The challenger builds the challenge ciphertext. The challenger flips a coin $b$. Then it computes $C = m_b Z$, and sets $C' = g^x$. Suppose the challenge access structure is $\mathbb{A}^* = (M^*, \rho^*)$, where the share-generating matrix $M^*$ has $\ell^*$ rows. The challenger computes the shares $\{\lambda_i\}$ as usual according to $M^*$, and then computes $\forall i = 1, \cdots, \ell^*, C_i = g^{a\lambda_i}(g^x)^{\tau_{\rho^*(i)}}$.

**Phase 2.** Same as Phase 1.

**Guess.** The adversary outputs a guess $b'$ of $b$. If $b = b'$ then the challenger outputs 1 to indicate that $Z = e(g,g)^{cdx}$; otherwise, it outputs 0 to indicate that $Z$ is a random element in $G_1$.

When $Z = e(g,g)^{cdx}$, then the above simulation by the challenger $\mathcal{C}$ for the challenge ciphertext is perfect. Thus we have $\Pr[\mathcal{C}(g, g^c, g^d, g^x, g^{cdx}) = 1] = \Pr[b' = b] = 1/2 + \mathsf{Adv}_{\mathcal{A}}$. On the other hand, if $Z$ is random number in $G_1$, then $m_b$ in the challenge ciphertext of the above simulation is completely hidden from the adversary. Thus we have $\Pr[\mathcal{C}(g, g^c, g^d, g^x, T) = 1] = \Pr[b' = b] = 1/2$. Combined, we get $|\Pr[\mathcal{C}(g, g^c, g^d, g^x, g^{cdx}) = 1] - \Pr[\mathcal{C}(g, g^c, g^d, g^x, T) = 1]| = \mathsf{Adv}_{\mathcal{A}}$. This completes the proof.

*Satisfying Definition* 3. We prove this by presenting a reduction from Waters' scheme which is proved secure under the decisional $q$-BDHE assumption in [18] to ours. To this end, we first point out that the main differences between our scheme and Waters' that are relevant to the proof here are the format of the server-side key in our scheme and of the private key in Waters' scheme. In Waters' scheme, the format of a private key is $(K = g^z g^{at}, L = g^t, \{\forall s \in S : K_s = H(s)^t\})$. Bearing this difference in mind, we build an adversary $\mathcal{B}$ against Waters' scheme, given an adversary $\mathcal{A}$ of our scheme. Details follow.

$\mathcal{B}$ acts as the challenger in the game in Definition 3.

**Init.** $\mathcal{A}$ declares a challenge access structure $\mathbb{A}^*$ to $\mathcal{B}$, who then declares $\mathbb{A}^*$ to the challenger of Waters' scheme.

**Setup.** $\mathcal{B}$ takes in the public parameters of a Waters' scheme, and gives them to $\mathcal{A}$. $\mathcal{B}$ also determines a standard public-key encryption scheme $\mathsf{Enc}$ and gives the description to $\mathcal{A}$.

**Phase 1.** $\mathcal{B}$ answers user-side key generation and server-side key generation queries from $\mathcal{A}$. To answer a user-side key generation query, $\mathcal{B}$ simply generates a public/private key pair according to $\mathsf{Enc}$. To answer a server-side key generation query on a set $S$ of attributes and a user public key $Upk$, $\mathcal{B}$ submits a key generation (KeyGen) query to the challenger of Waters' scheme with $S$ (if $S$ does not satisfy $\mathbb{A}^*$), and as a response $\mathcal{B}$ is returned a key of the form $(K = g^z g^{at}, L = g^t, \{\forall s \in S : K_s = H(s)^t\})$. Then $\mathcal{B}$ selects $\alpha \in_R Z_p$, and computes $\mathsf{Enc}(Upk, \alpha)$ and sets the server-side key as $(K^\alpha, \mathsf{Enc}(Upk, \alpha), L^\alpha, \{\forall s \in S : K_s^\alpha\})$. It can easily see that the resulting server-side key is valid with respect to our scheme.

**Challenge.** $\mathcal{B}$ builds a challenge ciphertext under $\mathbb{A}^*$, given $m_0, m_1$ from $\mathcal{A}$. To this end, $\mathcal{B}$ submits $m_0$ and $m_1$ to the challenger of Waters' scheme as a challenge, and gets back a challenge ciphertext $c^*$. $\mathcal{B}$ returns $c^*$ as the challenge ciphertext to $\mathcal{A}$.

**Phase 2.** Same as Phase 1.

**Guess.** $\mathcal{A}$ outputs a guess $b'$, which is also used by $\mathcal{B}$ as the guess to the challenger of Waters' scheme. It is easily seen that the simulation by $\mathcal{B}$ is perfect, and the advantage of $\mathcal{B}$ is at least that of $\mathcal{A}$. This completes the proof. □

## References

1. Attrapadung, N., Imai, H.: Attribute-based encryption supporting direct/indirect revocation modes. In: Proceedings IMA International Conference on Cryptography and Coding, pp. 278–300 (2009)
2. Beimel, A.: Secure schemes for secret sharing and key distribution, Ph.D. thesis, Israel Institute of Technology, Technion, Haifa, Israel (1996)
3. Blaze, M., Bleumer, G., Strauss, M.J.: Divertible protocols and atomic proxy cryptography. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 127–144. Springer, Heidelberg (1998)
4. Boneh, D., Ding, X., Tsudik, G., Wong, C.M.: A method for fast revocation of public key certificates and security capabilities. In: Proceedings USENIX Security (2001)
5. Bobba, R., Khurana, H., Prabhakaran, M.: A pracitically motivated enhancement to attribute-based encryption. In: Proceedings ESORICs (2009)
6. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: Proceedings IEEE S&P (2007)
7. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Heidelberg (2005)
8. Cloud security alliance: security guidance for critical areas of focus in cloud computing (2009). http://www.cloudsecurityalliance.org
9. European network and information security agency: cloud computing risk assessment (2009). http://www.enisa.europa.eu/act/rm/_les/deliverables/cloud-computing-risk-assessment
10. Gartner: don't trust cloud provider to protect your corporate assets, 28 May 2012. http://www.mis-asia.com/resource/cloud-computing/gartner-dont-trust-cloud-provider-to-protect-your-corporate-assets
11. Green, M., Hohenberger, S., Waters, B.: Outsourcing the decryption of ABE ciphertexts. In: Proceedings USENIX Security (2011)
12. Goyal, V., Pandy, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings ACM CCS 2006 (2006)
13. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust threshold DSS signatures. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 354–371. Springer, Heidelberg (1996)
14. Liang, X., Cao, Z., Lin, H., Shao, J.: Attribute-based proxy re-encrytpion with delegating capabilities. In: Proceedings ACM ASIACCS 2009, pp. 276–286 (2009)
15. Liu, J., Wan, Z., Gu, M.: Hierarchical attribute-set based encryption for scalable, flexible and fine-grained access control in cloud computing. In: Proceedings 7th Information Security Practice and Experience Conference, ISPEC 2011 (2011)
16. Ostrovsky, R., Sahai, A., Waters, B.: Attribute-based encryption with non-monotonic access structures. In: Proceedings ACM CCS 2007, pp. 195–203 (2007)
17. Shoup, V., Gennaro, R.: Securing threshold cryptosystems against chosen ciphertext attack. J. Cryptol. **15**(2), 75–96 (2002)

18. Waters, B.: Ciphertext-policy attribute-Based encryption: an expressive, efficient, and provably aecure realization. In: Proceedings Practice and Theory in Public Key Cryptography, PKC 2011, pp. 53–70 (2011)
19. Wang, G., Liu, Q., Wu, J.: Hierarhical attribute-based encryption for fine-grained access control in cloud storage services. In: Proceedings ACM CCS 2010 (2010)
20. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable, and fine-grained data access control in cloud computing. In: Proceedings IEEE INFOCOM 2010 (2010)