J Supercomput (2012) 62:431–460
DOI 10.1007/s11227-011-0727-9

SAPDS: self-healing attribute-based privacy aware data sharing in cloud

Zeeshan Pervez · Asad Masood Khattak · Sungyoung Lee · Young-Koo Lee

Published online: 7 January 2012

Abstract This paper addresses the issue of data governance in a cloud-based storage system. To achieve fine-grained access control over the outsourced data, we propose Self-Healing Attribute-based Privacy Aware Data Sharing in Cloud (SAPDS). The proposed system delegates the key distribution and management process to a cloud server without seeping out any confidential information. It facilitates data owner to restrain access of the user with whom data has been shared. User revocation is achieved by merely changing one attribute associated with the decryption policy, instead of modifying the entire access control policy. It enables authorized users to update their decryption keys followed by each user revocation, making it self-healing, without ever interacting with the data owner. Computation analysis of the proposed system shows that data owner can revoke n' users with the complexity of $O(n')$. Besides this, legitimate users can update their decryption keys with the complexity of $O(1)$.

Keywords Cloud storage · Data privacy · Remote storage

1 Introduction

Emergence of virtualization technologies, availability of high-speed Internet to populace and adoption of Service Oriented Architecture (SOA) has derived a new comput-

Z. Pervez · A.M. Khattak · S. Lee (✉) · Y.-K. Lee
Ubiquitous Computing Lab, Department of Computer Engineering, Kyung Hee University, Seocheon-dong, Giheung-gu, Yongin-si, Gyeonggi-do 446-701, Republic of Korea
e-mail: sylee@oslab.khu.ac.kr

Z. Pervez
e-mail: zeeshan@oslab.khu.ac.kr

A.M. Khattak
e-mail: asad.masood@oslab.khu.ac.kr

Y.-K. Lee
e-mail: yklee@khu.ac.kr

ing paradigm known as cloud computing [1]. Benefits provided by this on-demand, massive, and scalable computing facility can be primarily lumped into one category—cost [2, 3]. Businesses seeking computing resources can tap into public cloud without the need to build datacenter of their own [4, 5]. Similarly, enterprises can utilize their computing resources in efficient manner endorsing green computing through private cloud [6, 7]. Annually Gartner publishes a list of organizational strategic technologies; for the past two years it has rated cloud computing as one of the core technologies, which organizations must consider during their strategic planning for the next ten years [8].

Although materialization of cloud computing is seeded by comparatively mature technologies like virtualization, and distributed processing/storage, still it faces research challenges that are yet to be answered [9]. In a cloud based data sharing system data owner owns the data. It utilizes storage facility provided by a cloud service provider to share the outsourced data with the legitimate user(s). Privacy and data confidentiality are at the top of the most concerned research issues of cloud storage system, which can impend its adoption with overwhelming consequences [10]. This is because the cloud service provider resides outside the federated and trusted domain of the data owner, thus circumscribing his control over the outsourced data [11–14].

Migrating from federated domain (*desktop computers, corporate servers*) to an un-federated domain (*cloud server*) raises privacy concerns, which entail the protection and appropriate use of personal information of the customers, and meeting their expectations about its usage [15]. Security requirements of Personally Identifiable Information (PII), especially healthcare information, dramatically changes when stored on the un-trusted servers [16, 38]. One feasible solution to these problems would be, encrypting the outsourced data and handing over the decryption key to the legitimate users with whom data need to be shared. Similar type of solutions exists in the domain of Private Information Retrieval (PIR), which store and share the encrypted data in remote storage services [17]. However, these methodologies face issues of scalability and realism when applied to emerging epitome of virtualized computing to achieve fine-grained access control over the outsourced data.

To achieve desire level of access control, mainly two methodologies are adopted, i.e., file based [18, 19], and file-group based access control list [20–22]. Although, these schemes manages to restrain illicit data access; however, their feasibly is questioned rigorously with system scalability. As for file based scheme, the complexity would be equal to number of users in the system, whereas file-group based methodology only provides coarse-grained data access control. Furthermore, applying these methodologies raise concerns of user revocation, increased complexity of key management, and dependency on a trusted entity to disseminate appropriate decryption keys. Restricting users from accessing the data, which they have been accessing previously (*user revocation*) is one of the prime concern in the realization of fine-grained access over the outsourced data. The intensity of this issue escalates with the fact that the data owner possesses no control over that outsourced data and does not want to exclusively rely on the cloud provider [13, 14].

Since, encryption is the only way to conceal the confidential data (i.e., PII), it requires exchange of decryption key among the involved entities. However, it is not feasible for the data owner to remain always online to provide decryption key to the

individual users. This leaves data owner with limited number of options; either, rely on trusted third party (TTP), or delegate decryption key distribution to the storage provider. On one side reliance on TTP increases additional privacy concern—how to ensure that third parties protect the data and do not use it for their own purposes. On the other hand, risk of privacy infringement escalates when decryption keys are distributed via the storage provider. In either case, the usual practice is to conceal decryption key with public key of the individual user, and then transmit it to the respective legitimate user. The obvious problem with this solution is scalability. As the number of legitimate users increases so does the load of asymmetric encryption on the data owner.

In order to address the issue of data governance in cloud-based storage system, we propose SAPDS, enabling the data owner in maintaining fine-grained access control over the outsourced data. With SAPDS, data owner generates single decryption key for similar interest group, whilst limiting the interaction between the data owner and the cloud provider. Furthermore, SAPDS evades the need of data owner's availability for the dissemination of data decryption key. The notion of fine-grained access control is realized by modeling the access control policy in a tree structure, called access tree. Leaf node of the access tree stores an attribute value and intermediate node defines the logical operation (threshold gates) on multiple leaf nodes. Data that needs to be shared is encrypted in such a way that combination of attributes and threshold gates reveals the decryption key.

To achieve fine-grained data governance in a cloud-based storage system SAPDS combines Cipher-Text Policy Attribute Based Encryption (CP-ABE) [23], along with Proxy Re-encryption (PRE) [24] and Key Derivation (KD) methodologies. CP-ABE ensures the compliance of access control policy, whereas PRE eliminates the need of direct interaction between the data owner and the users for decryption key dissemination. KD assists legitimate users in updating their decryption keys, besides this KD ensures that legitimate users prior to the current revocation continue their access to the outsourced data, without the need to interact with the data owner. By uniquely combining these cryptographic methodologies SAPDS realizes a cloud based data sharing system which delegates decryption key distribution and management to an un-trusted cloud service provider. Consequently, it eliminates the need of trusted third party to manage decryption keys for authorized user access and guaranteed user revocation.

SAPDS exhibits that the data sharing complexity is not more than the number of leaf nodes in the access control policy (*i.e.*, $O(numberOfLeaf(\tau))$ *where* τ *is access structure*). Furthermore, it can revoke the user(s) with the complexity of $O(n')$ (*where* n' *is the number of revoked users*). For the cloud user the complexity is always $O(1)$. SAPDS inherits the computational complexity of CP-ABE and PRE, for Chosen-Plaintext-Attach (CPA) and Chosen-Ciphertext-Attack (CCA) [23].

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 outlines models and assumptions of the proposed scheme. Section 4 presents SAPDS. Section 5 examines the performance of SAPDS. Section 6 presents the security analysis of SAPDS. Section 7 provides the computation and communication assessment of CP-ABE in the context of SAPDS. Section 8 compares the performance of SAPDS with existing methodologies; finally, Sect. 9 concludes the paper along with the future directions.

2 Related work

The most recent work addressing the privacy issues in a cloud-based storage is carried out by Shucheng Yu et al. [25]. They proposed a KP-ABE [26] based cloud storage system, which utilizes lazy re-encryption along with proxy re-encryption for user revocation. Their work mainly focuses on restricting revoked user in regaining access to the confidential data. In order to revoke a particular user, their proposed technique relies on outlining the access structure that can not be satisfied by the revoked user. They achieve this by identifying the minimum set of attributes without which decryption of the shared content cannot be carried out successfully. The problem with this type of technique is obvious, its dependency on the access control mechanism. On each user revocation, a new access policy is defined to prohibit access to the outsourced data, which would eventually lead to a situation where it is not possible to construct further more access structure. Apart from that, on each user revocation individual user's master and public key components of user secret key are redefined, requiring resource intensive computation after each revocation.

SiRiUS is a secure file systems designed to layer over insecure network and P2P file system [18]. It works by maintaining an access structure in a meta data file (*md*). Each entry in *md* contains a file encryption key (*FEK*) and a signature key (*FSK*), encrypted with user's public key. Each legitimate user can decrypt the respective entry in *md* by using his private key. This type of solution works well when data is shared among smaller groups; however, for collaboration among bigger groups SiRiUS proves to be inefficient. Since, the size of *md* is directly proportional to the number of legitimate users, scalability of SiRiUS is greatly threatened as number of legitimate user increases. Another important consideration, which is overlooked during its design is user revocation. Whenever, user is revoked, *FEK* and *FSK* are updated and new keys are encrypted with public key of the individual user, making it impracticable for dynamic user sets.

Key Regression [20] is a key generation scheme, which addresses the pseudorandom issue of key rotation proposed in [27]. In their construction instead of handing over the encryption keys, member states (stm_i) are given to the legitimate users. They can then generate their encryption key K_i for i-th time period, using $Keyder(stm_i)$ function. Separating the encryption keys from the member states ensures that encryption key K_i is pseudorandom for the member possessing stm_l, where $l < i$. However, keys generated by *Keyder* are limited to maximum-wind MW. This leads to the problem that after certain number of revocations *Keyder* will not be able to generate the new member states, thus reducing its practicability to a limited number of revocations.

In [21], Backes et al. proposed a key updating system for Lazy Revocation. Encryption keys are updated merely on the file access, which is followed by a user revocation. In their proposed system, files are maintained in groups of identical access permissions. Organizing keys in groups has an obvious advantage i.e., key of a particular group is updated from which user is revoked. However, important point that is overlooked during its design is key distribution process; owner himself distributes updated keys to the legitimate users. On one hand, this type of distribution certainly avoids trusted third party, but on the other hand, it requires guaranteed availability of the data owner until all of the legitimate users update their keys.

CRUST [19] is a cryptographic remote storage system, which avoids the use of public key encryption for the purpose of speed and efficiency in cryptographic operations. CRUST assumes the availability of a trusted agent (*i.e.*, *trusted third party*), responsible for key management. For each new user, trusted agent generates an encryption key by using system master key stored locally on it. CRUST maintains file in blocks each encrypted with a separate key. Apart from that, it utilizes lazy re-encryption strategy which ensure that only the updated block of a file is re-encrypted to avoid unnecessary cryptographic operations. Methodology adopted by CRUST uses trusted agent, thus making it highly dependent on the trusted third party. It also suffers from key management problem. Separate encryption key for each block of a file increases key management burden on each user as well as on the trusted agent, thus making it unfeasible for the practical deployment.

Patient Controlled Encryption, PCE [22] is a privacy preserving medical health record system. In PCE data is stored as a hierarchical structure on a remote location which is not under the control of patient. However, PCE facilitates sharing and searching of the encrypted data. According to the authors, the proposed scheme can be realized using symmetric and asymmetric encryption, with their inherited benefits and limitations. For both strategies, they proposed a generalized key revocation scheme in which data is re-encrypted on each revocation, and decryption key is distributed to the legitimate users, personally by the data owner. Keeping in view the sensitivity of the involved data PCE uses a simple revocation scheme whose complexity is directly proportional to the number of legitimate users.

3 Models, design goals and assumptions

3.1 System model

To realize a cloud-based storage system: Data Provider, Data Consumer, and Cloud Service Provider are considered as the involved entities; for brevity, we shall be using owner, user, and cloud server respectively. Owner harnesses the storage facility provided by the cloud server by uploading the encrypted data he wants to share (*e.g., pictures, text and multimedia files*). User obtains a copy of data from the cloud server and decrypts it (*if allowed*) by using his decryption key. Neither the owner nor the user is always online; only the cloud server's availability is assured which can be effortlessly guaranteed by signing service level agreement [28]. This model represents a typical cloud storage business model in which owner goes offline after uploading the data and user only interacts with the cloud server to access the outsourced data.

3.2 Security model

In the proposed system we assume that cloud server is "honest but curious", similar to that of [29, 30]. Consequently, cloud server will honestly execute the task delegated by the owner, however cloud server will try to find out the data contents for some business needs *i.e.*, *related ad serving, or understanding the access patterns*. This leads to the situation where cloud server can team up with the revoked user(s) to

decrypt the encrypted data for some wicked motives. In addition to that, multiple users revoked at t_i and t_j time intervals (where $i < j$) can work together to gain access of the outsourced data otherwise not allowed. The proposed technique focuses on fine-grained access control in a cloud-based storage system, we believe that there exists a secure channel (*i.e.*, *SSL*) between the involved entities though which data is exchanged.

3.3 Design goal

The pivotal design goal of the proposed system is to achieve fine-grained access control in a cloud-based storage system, with negligible execution overhead on both the owner and the user, whist allowing guaranteed user revocation. More importantly, the proposed system should not involve any trusted third party for key management. Trusted third party provides centralized key management. However, it is hard to find a common trustable entity when owner and user belongs from different administrative domains i.e., organizations, countries, having their own data exchange policies. Additionally, relying on a trusted party would incur extra cost to pay for the data decryption key management and distribution services. Thus, it would increase the subscription cost of a cloud based data sharing system. Besides all that, the proposed system should provide counter measures for key abuses, i.e., reconstructing a valid decryption key $\partial(K_i) \rightarrow K_j$, from a revoked key K_i, where ∂ is a specialized key reconstruction function. Furthermore, from security perspective the proposed system should affirm the following properties.

1. Decryption key[1] should be update after each user revocation.
2. Owner should not remain always online to distribute new decryption key among the legitimate users.
3. Legitimate user should be allowed to update their secret key, after each user revocation without interacting with the owner.
4. Cloud server should not be able to learn any information about the contents of the outsourced data.

4 SAPDS: self-healing attribute-based privacy aware data sharing in cloud

SAPDS realizes data sharing in a cloud-based storage system with privacy considerations, and devoid of relying on any trusted third party. Considerations described in system design goal escort the fundamental properties of a cloud-based storage system in which owner has limited control over the storage system [11–14]. However, owner can govern data access policies and key management protocol(s). To develop a privacy aware data-sharing scheme with said properties, we combine; CP-ABE [23], PRE [24] along with KD. The combination of CP-ABE, PRE and KD assists the owners in harnessing the cloud resources whilst maintaining the same level of data security they enjoy in standalone storage systems [31].

[1] Throughout this text, we shall be using "decryption key" to represent symmetric encryption key, "secret key" to symbolize CP-ABE decryption key (user's key), and "pre key" as Proxy Re-Encryption transformation key.

4.1 Main idea

Suppose, Alice is a professional photographer and has subscribed to a cloud storage facility owned by Eve. She wants to share some photographs with Bob who is running a publishing business. Prior to uploading the photographs to the cloud-based storage, Alice informs Bob about the address (*URL*) of the cloud storage, along with his secret key and transformation key. She then encrypts the pictures with symmetric encryption algorithm. The decryption key is then encrypted with CP-ABE under the policy that can be satisfied by the attributes attached to Bob's secret key. In the end encrypted photographs and CP-ABE encrypted decryption key are uploaded to the cloud server. Now Bob can access the file, by simply requesting the encrypted file along with the decryption key. Cloud server replies back with encrypted photographs along with the concealed decryption key. On receiving the response Bob uses his secret key to decipher the decryption key. At the end he uses decryption key to decipher the photographs shared by Alice. At anytime if Alice wants to revoke Bob, she updates the policy with which decryption key was encrypted; without ever relying on cloud server. Instead of altering the entire access control policy, only one attribute (*legitimacy attribute*) is updated, restraining Bob's secret key to satisfy the decryption policy. To revoke multiple users e.g., Bob and Mallory, Alice simply updates the single legitimacy attribute, and restrains revoked users to update their set key by using KD. Figure 1 illustrates SAPDS enabled data sharing in a cloud-based storage system.

By coalescing CP-ABE, PRE, and KD, SAPDS leverages owner to ensure fine-grained access control over the outsourced data without relying on cloud server. Additionally, SAPDS delegates the key distribution task to the cloud server without seeping out any confidential information. Nevertheless, legitimate users can re-gain their accessibility by merely updating single attribute (*legitimacy attribute*) of their secret key.

Before getting into the details of SAPDS, we mention some of the assumptions considered during its design.

1. We assume that legitimate users behave honestly, by which we mean that they never share their decryption key with the revoked users.[2]
2. We believe that cloud server performs his duty honestly as dictated by the owner. However, can team-up with revoked user to access the shared data.

4.2 Proposed scheme

SAPDS uses amalgam of cryptographic primitives to overcome the problem of user revocation in an un-trusted domain. Table 1 explain the notation used in the descriptive detail of SAPDS.

Throughout this text, superscript used on these abbreviations represents the encryption algorithm applied to conceal the secret information e.g., F^{se} symbolizes a

[2]Decryption key is protected with an access control policy which can only be satisfied by a legitimate and authorized user. Thus sharing decryption key is equivalent to delegating access privileges without the consent of data owner.

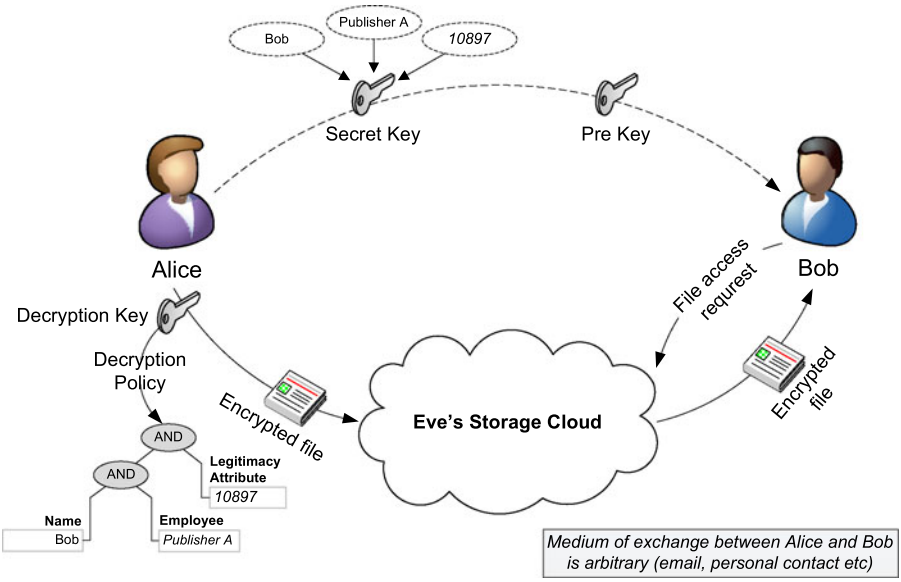


Fig. 1 SAPDS enabled cloud-based storage system

Table 1 Abbreviations used in proposed technique

Notation	Description
F	Data file shared on cloud-based storage system
PK, MK	Public and master keys for CP-ABE
SK	User secret key for CP-ABE encrypted cipher text
D_{EK_i}	Data decryption key
L_{num}	Random legitimacy number
L_{att_i}	Legitimacy attribute
Ver_{att_i}	Version number of legitimacy attribute
$\rho_{att_{i+1}}$	Public key component of the user secret key
H	One way hash function
ω	Cipher text transformation key for the owner
θ	Cipher text transformation key for the user
ψ	Cipher text transformation key for the cloud server
$\tau_{\backslash U_{rvk}}$	Access structure in which revoked user is not allowed to decrypt cipher text
U_{rvk}	Revoked user identity
sk_{att}	Attribute associated with SK

file encrypted with symmetric encryption algorithm. Similarly, $D_{EK_i}^{cp\_abe}$ represents the data decryption key encrypted with CP-ABE.

Each F is encrypted with D_{EK_i}. Since cloud server is not a trustable entity, and it is not feasible for the owner to remain always online to handover the decryption

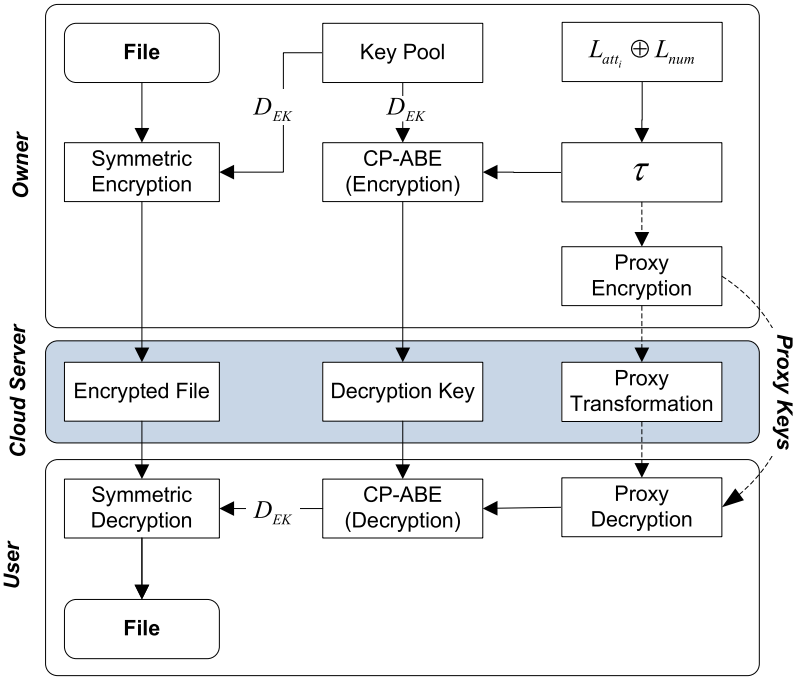


Fig. 2 Self-healing attribute-based privacy aware data sharing in cloud

key to the individual user(s), D_{EK_i} is encrypted with the access policy (τ) such that attributes associated with *SK* would be able to decipher it. To avoid generating the decryption key for individual user by owner himself; single D_{EK_i} is generated for similar interest group. Finally, F^{se} and $D_{EK_i}^{cp\_abe}$ are uploaded to the cloud-based storage. Individual user in possession of *SK* along with θ, requests the cloud server to access F^{se}, cloud server then replies back with the encrypted outsourced data along with the concealed decryption key. Legitimate user, then uses sk_{att} associated to his *SK* to reveal the hidden value with which $D_{EK_i}^{cp\_abe}$ is encrypted. After that D_{EK_i} is utilized to decipher F^{se}. At any time if there is a need to revoke a user, decryption policy of $D_{EK_i}^{cp\_abe}$ is updated by simply changing L_{att_i}. With L_{num} legitimate users can update their *SK* confirming to the new decryption policy. SAPDS component model is shown in Fig. 2.

4.2.1 Data preparation

It is the first step in SAPDS enabled cloud-based storage system. It encrypts the data in such a way that it can be managed according to the security requirements specified in the system design goals. Owner selects the data encryption key D_{EK_i} from his local key pool $\mathcal{K}$. D_{EK_i} is used to encrypt the confidential data with an arbitrary symmetric encryption algorithm. The rationale of encrypting data with symmetric encryption algorithm is obvious—with symmetric encryption, data is encrypted for

similar interest group and decryption key is distributed among the legitimate users of that particular group.

4.2.2 Initialization

This step is prerequisite of data sharing on the cloud-based storage system, it initializes CP-ABE algorithm. It takes φ as a security parameter and outputs PK and MK. φ specifies the order of Bilinear Group $\mathbb{G}$. SK is derived form MK while PK is used to encrypt D_{EK_i}. For more detail on CP-ABE initialization, please refer to Appendix A.

4.2.3 File creation

Once F is encrypted (i.e., F^{se}), next step is to upload it to the cloud-based storage along with D_{EK_i}. To avoid dependency on TTP and storage provider; D_{EK_i} is secured by utilizing attribute base encryption (CP-ABE). Owner specifies the set of attributes for access structure τ, it then encrypts D_{EK_i} with CP-ABE (i.e., $D_{EK_i}^{cp\_abe}$). τ enumerates the attributes which are required for the decryption of $D_{EK_i}^{cp\_abe}$, these attributes are associated with the SK of a legitimate user. Appendix A illustrates the encryption process of CP-ABE in more detail. Finally, owner uploads both F^{se}, and $D_{EK_i}^{cp\_abe}$ to the cloud-based storage system.

4.2.4 Key generation

Key generation process generates SK to decrypt $D_{EK_i}^{cp\_abe}$. Keys are generated by utilizing MK and τ. SK represents the attributes (sk_{att}) involved in τ along with L_{att_i}. sk_{att} are computed to represent the secret value used in the encryption process, and then during decryption process these attributes are interpolated to reveal the secret value. Please refer to KeyGen in Appendix A, for more details.

4.2.5 Key distribution

In SAPDS we are dealing with three different kinds of keys, i.e., data decryption key (D_{EK_i}), secret key (SK), and transformation key. Legitimate user must possess each of them to access the outsourced data. According to the security model D_{EK_i} is subject to change on each user revocation. However, SK is partially updated (*legitimacy attribute*) on each user revocation.

$D_{EK_i}^{cp\_abe}$ is uploaded to the cloud server along with F^{se}, each outsourced F^{se} has its own D_{EK_i}. Secret key and transformation key are handed over to each user during the subscription process. We are following the usual subscription process in which confirmation code is send to the individual user through an email to complete the subscription process. Similarly, secret and transformation keys are sent to individual user through the email, once owner approve their subscription. This is the only exchange of confidential information between the owner and the user. Decryption key and secret key are updated via cloud server utilizing proxy re-encryption. For mathematical proof of proxy re-encryption please refer to Appendix B.

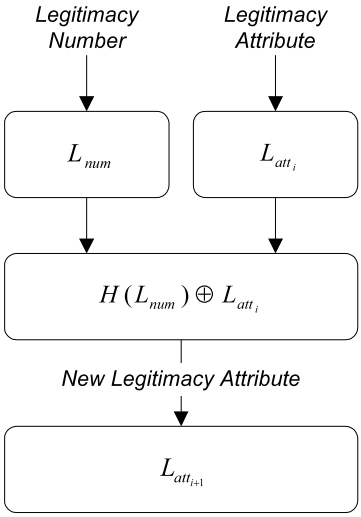


Fig. 3 Legitimacy attribute generation

4.2.6 File access

User requests the outsourced data, and in response cloud server replies with F^{se}, and $D_{EK_i}^{cp\_abe}$. Legitimate user possessing SK, can decrypt $D_{EK_i}^{cp\_abe}$, and ultimately can gain access to the shared contents by decrypting F^{se} with D_{EK_i}. We believe that there exists a mechanism through which legitimate users are intimated about the newly outsourced data. For simplicity, we are intentionally overlooking these details.

4.2.7 User revocation

User revocation is executed whenever there is a need to restrict the user from accessing the outsourced data, which was previously accessible to him. Encrypting D_{EK_i} with static set of attributes associated with SK creates problem during user revocation. On each revocation owner needs to redefine τ, so that revoked user can not decrypt $D_{EK_i}^{cp\_abe}$. To overcome the issue of increased complexity of redefining τ we introduce the concept of legitimacy attribute L_{att_i}, it specifies whether user should be able to decrypt $D_{EK_i}^{cp\_abe}$. L_{att_i} is append to the root of access structure τ with an AND gate making it mandatory field in decryption of $D_{EK_i}^{cp\_abe}$. Instead of modify the entire access structure to revoke a particular user, L_{att_i} is updated to $L_{att_{i+1}}$, and only legitimate users are allowed to update legitimacy attribute. On each user revocation, owner generates a new decryption key ($D_{EK_{i+1}}$) and uploads the newly encrypted file to the cloud server. Next owner derives $L_{att_{i+1}}$, by randomly generating a legitimacy number L_{num}, and computing XOR of L_{att_i} with hash value of L_{num} as shown in Fig. 3.

Once $L_{att_{i+1}}$ is derived, $D_{EK_{i+1}}$ is encrypted with CP-ABE under access structure τ substituting the value of L_{att_i} with $L_{att_{i+1}}$. Legitimate user must possess L_{num} to update the existing legitimacy attribute, as illustrated in Fig. 3. Since, cloud server can team up with the revoked user to decrypt the outsourced data, to overcome this problem, L_{num} is not directly sent to the cloud server. Instead of that, owner constructs a

new access policy $\tau_{\backslash U_{rvk}}$, encrypts L_{num} with CP-ABE under $\tau_{\backslash U_{rvk}}$, and uploads it to the cloud server. $\tau_{\backslash U_{rvk}}$ specifies that legitimate user can decrypt L_{num} except from the recently revoked user(s). Owner then sends the PRE encrypted public key component $\rho^{pre_\omega}_{att_{i+1}}$ of the legitimacy attribute, along with $L^{cp\_abe}_{num}$, $H(L_{att_{i+1}})$, $D^{cp\_abe}_{EK_{i+1}}$ to the cloud server. Availability of the owner is no longer required, from onward cloud server can manage the key update process without compromising data privacy. Algorithm 1 illustrates the process of updating the secret component of the SK.

Algorithm 1 Owner: upload secret component to cloud server

1: Update data symmetric key $\mathcal{K} \rightarrow D_{EK_{i+1}}$
2: $F^{se} \leftarrow symmetric\_encryption(F, D_{EK_{i+1}})$
3: Compute new legitimacy attribute
   a: Generate random number L_{num}
   b: $L_{att_{i+1}} \leftarrow L_{att_i} \oplus H(L_{num})$
   c: Compute public key component $\rho_{att_{i+1}}$ of $L_{att_{i+1}}$
4: $\rho^{pre_\omega}_{att_{i+1}} \leftarrow pre\_encrypt(\rho_{att_{i+1}}, \omega)$
5: Upload $F^{D_{EK_{i+1}}}$ to cloud server
6: Construct new access structure $\tau_{\backslash U_{rvk}}$
7: $L^{cp\_abe}_{num} \leftarrow cp\_encrypt(L_{num}, \tau_{\backslash U_{rvk}})$
8: Send $L^{cp\_abe}_{num}$, $\rho^{pre_\omega}_{att_{i+1}}$, $H(L_{att_{i+1}})$, $D^{cp\_abe}_{EK_{i+1}}$ to cloud server

Cloud server stores $D^{cp\_abe}_{EK_{i+1}}$, public key component ($\rho^{pre_\omega}_{att_{i+1}}$), CP-ABE encrypted legitimacy number $L^{cp\_abe}_{num}$, along with the hash value of $L_{att_{i+1}}$. Besides this, it also archives the older version of these newly transmitted values. The archived values ensure that the user who missed the previous update revocation cycle, manages to update his secret key. On receiving the user's request cloud server checks the version of legitimacy attribute Ver_{att_i}, if user possesses the older version of L_{att_i} a challenge is offered to him by sending $L^{cp\_abe}_{num}$. The challenge requires to successfully compute the hash value of newly generated legitimacy attribute. $L^{cp\_abe}_{num}$ encryption policy enforces that revoked user should not be able to decrypt it, thus obstructing him to generate the new legitimacy attribute. Contrary, legitimate user decrypts $L^{cp\_abe}_{num}$, computes $H(L_{att_{i+1}})$ and sends the resultant value to the cloud server, which validates it; if succeeded $\rho^{pre_\theta}_{att_{i+1}}$ is send to the user. Algorithm 2 elaborates the delegated responsibility of secret key update process to the cloud server.

In order to get $D_{EK_{i+1}}$, user needs to prove his legitimacy, by decrypting $L^{cp\_abe}_{num}$, and computing $L_{att_{i+1}}$ as illustrated in Fig. 3. After that hash value of $L_{att_{i+1}}$ is sent to the cloud server, it compares the hash value provided by the owner and the value sent by the user, if matches it replies back with ($\rho^{pre_\theta}_{att_{i+1}}$). On receiving ($\rho^{pre_\theta}_{att_{i+1}}$) user reverts the transformation using θ and update his SK, thus manages to continue his access to the outsourced data. Algorithm 3 illustrates the user secret key update process.

Algorithm 2 Cloud server: manage secret key on cloud server

1: $Replace(F^{D_{EK_i}}, F^{D_{EK_{i+1}}})$
2: $Replace(D_{EK_i}^{cp\_abe}, D_{EK_{i+1}}^{cp\_abe})$
3: Store $L_{num}^{cp\_abe}$, $\rho_{att_{i+1}}^{pre_\omega}$, $H(L_{att_{i+1}})$
4: Respond with $L_{num}^{cp\_abe}$ to user possessing older Ver_{att_i}
5: Send $\rho_{att_{i+1}}^{pre_\theta}$ if user successfully computes $H(L_{att_{i+1}})$

Algorithm 3 User: update secret key

1: Generate data request with Ver_{att_i}
2: $L_{num} \leftarrow cp\_dencrypt(L_{num}^{cp\_abe}, U_{idd})$
3: $L_{att_{i+1}} \leftarrow L_{att_i} \oplus H(L_num)$
4: Send $H(L_{att_{i+1}})$ to cloud server
5: Get updated $\rho_{att_{i+1}}^{pre_\theta}$
6: $\rho_{att_{i+1}} \leftarrow pre\_decrypt(\rho_{att_{i+1}}^{pre_\theta}, \theta)$
7: Update secret key, to decrypt $D_{EK_{i+1}}^{cp\_abe}$

5 Performance analysis

This section evaluates the computational complexity and communication overhead that SAPDS exerts on each involved entity. In the following analysis, we are not considering the complexity of initializing the SAPDS, since it is only executed once and depends on the size of φ. For more detail on complexity of initialization, please refer to [32].

5.1 Data preparation

As discussed earlier, this step is prerequisite for transferring data from a trusted domain to an un-trusted domain, executed exclusively by the owner. It involves the selection of symmetric encryption key D_{EK_i} from $\mathcal{K}$. The computation cost of this step is directly proportional to the size of data (M), where selection of D_{EK_i} can be carried out in constant time. Thus, the complexity of data perpetration would be $O(sizeOf(M))$.

5.2 File creation

This step involves both the owner and the cloud server, complexity of the former entity is higher than that of the later one. From owner's point of view, this step can be divided into two sub steps, (a) creation of legitimacy attribute and (b) encoding access policy through collating polynomials of each involved attribute. Creation of legitimacy attribute consists of calculating the hash value of L_{num} followed by an XOR between hashed value with previous legitimacy attribute (see Fig. 3). Complexity of defining polynomials is directly proportional to the number of leaf nodes in

τ including legitimacy attribute. Thus, the complexity of this step can be measure as $O(sizeOfLeaf(\tau))$. For cloud server the overhead is equivalent to the signature verification of the transmitted data, which can be computed as $O(1)$.

5.3 Key generation

Key generation engages the owner and the cloud server. The complexity for the owner mainly constitutes of selection of sk_{att} that can satisfy τ. This step is similar to the CP-ABE KeyGen, except from the fact that in SAPDS, the selected attributes must contain a L_{att_i}. We consider its complexity to be directly proportional to the number of access attributes in τ, thus complexity can be computed as $O(|sk_{att}|)$, where $|sk_{att}|$ represents the number of selected attributes along with the legitimacy attribute. Similar, to the previous step cloud server only verifies the signature of the transmitted data, thus its complexity is equivalent to $O(1)$.

5.4 Key distribution

Key distribution assists the data owner to overcome the problem of guaranteed availability. As D_{EK_i} is subject to change after each user revocation, it is distributed to the legitimate users via cloud server. Considering n legitimate users, the computational complexity of key distribution for the cloud server would be $O(n)$. Once key is received user verifies the signature of response, making its complexity $O(1)$. Secret and transformation keys are transmitted to the individual users on successful completion of the subscription process, thus marking its complexity $O(n)$ for owner. However, one important point which must be considered that secret and transformation keys are exchanged only once. After that secret key is updated via cloud server.

5.5 File access

In order to access the outsourced data legitimate user need to carry out series of step. First, it needs to get the outsourced data along with its decryption key, after that it decrypts $D_{EK_i}^{cp\_abe}$ to get D_{EK_i}, in the end it deciphers the outsourced data (i.e., F^{se}). Since we are dealing with fine-grained access control, we are not considering the complexity of gaining access to the outsourced data from the cloud server. Similarly, decryption of outsourced data depends on the symmetric encryption algorithm. SAPDS does not specify any symmetric encryption algorithm, it depends on owner's choice. Instead, our main concern is to reduce the computational complexity of decryption key management in an un-trusted storage system. As D_{EK_i} is encrypted with attribute based encryption, the complexity of decrypting it would be equivalent to interpolating the attributes associated to SK, which is $O(|\tau|)$.

5.6 User revocation

All three entities are involved in this step. Owner executes this set into two phases; in the first phase new symmetric key $D_{EK_{i+1}}$ is derived from $\mathcal{K}$ and data is encrypted with $D_{EK_{i+1}}$, making it similar to data preparation step. For the second phase new

Table 2 Computational complexity of SAPDS

Operation	Involved entities		
	Owner	Cloud server	User
Data Perpetration	$O(sizeOf(M))$	–	–
File Creation	$O(numberOfLeaf(\tau))$	$O(1)$	–
Key Generation	$O(\|sk_{att}\|)$	$O(1)$	–
Key Distribution	$O(n)$	$O(n)$	$O(1)$
File Access	–	–	$O(\|\tau\|)$
User Revocation	$O(n')$	$O(n-n')$	$O(1)$

legitimacy number is generated which is encrypted with CP-ABE under $\tau_{\backslash U_{rvk}}$ specifying the revoked user id, prohibiting him to decrypt $L_{num}^{cp\_abe}$. As described earlier, applying CP-ABE is directly proportional to the number of leaf nodes in τ. Considering n' users are revoked, overhead of computing $L_{num}^{cp\_abe}$ would be $O(n')$ where $n' < n$ (*in worst case all of the user can be revoked increasing number of leaf node in* $\tau_{\backslash U_{rvk}}$). Computing $\rho_{att_{i+1}}^{pre_\omega}$, and $H(L_{att_{i+1}})$ are considered to be constant and consequently can be ignored, thus complexity for the owner is $O(n')$.

Complexity for the cloud server involves signature verification of the transmitted data, along with the transformation of $\rho_{att_{i+1}}^{pre_\omega}$ to $\rho_{att_{i+1}}^{pre_\theta}$, for $n - n'$ legitimate users. Additional step that can be ignored is verification of $H(L_{att_{i+1}})$. Thus, the complexity of user revocation on cloud server is $O(n - n')$.

Each legitimate user needs to generate new legitimacy attribute $L_{att_{i+1}}$ and reply back the hashed value of $L_{att_{i+1}}$ to the cloud sever. On successfully computing the new legitimacy attribute user needs to decrypt the public key component of legitimacy attribute ($\rho_{att_{i+1}}^{pre_\theta}$) sent by the cloud server. Both the legitimacy attribute calculation and decryption of $\rho_{att_{i+1}}^{pre_\theta}$ can be carried out in constant time, thus we consider complexity for user is $O(1)$.

SAPDS is designed with the consideration that every request executed on the cloud server cost money. Having massive computation capacity does not mean that we can delegate every task to the cloud server. At the same time we do not want to escalate the processing load on the owner or on the user as well, to avoid the situation where migrating to cloud does not seem to be a lucrative option. Table 2 summaries the computation complexity SAPDS for each involved entities.

SAPDS reduces the dependency between each entity by utilizing proxy re-encryption, enabling owner to go offline after each revocation, and delegating secret key update task to the cloud server. Majority of the overheard experienced by the owner is during initial steps (*encrypting data before uploading it to cloud-based storage*), which is inevitable, as data need to be encrypted before it can be outsourced. Whereas, cloud server experiences the maximum execution overhead during Key Distribution and User Revocation. Overhead on cloud server is obvious as the design goal of SAPDS is to avoid dependency on the trusted third party and on the cloud server as well. However, at any step the execution complexity for user is always $O(1)$, except form file access which is $O(|\tau|)$.

6 Security analysis

This section examines the security aspect of SAPDS. Our analysis investigates the possibility of privacy breach when attackers team up each other or with the cloud service provider to gain access to the outsourced data otherwise not allowed. The security analysis investigates the robustness of SAPDS on each of its fundamental steps. However, as Data initialization and Key generation are similar to Setup and KeyGen of CP-ABE reader may refer to [23] for their cryptographic analysis.

6.1 Data preparation

Outsourced data is encrypted with symmetric encryption to restrain any illicit data access by the revoked users or by the cloud service provider. SAPDS does not impose any restriction on the choice of symmetric encryption algorithm. Data owner can choose encryption algorithm according to the sensitivity of the outsourced data and computational capabilities of the users with whom data is shared. The confidentiality of the outsourced data is directly related to the computational effort required to attack the symmetric encryption algorithm chosen by the data owner. As SAPDS uses symmetric encryption to conceal outsourced data, it inherits all of the cryptographic properties of the chosen encryption algorithm.

6.2 File creation

This step starts the actual interaction between the data owner and cloud server provider. Data owner stores encrypted outsourced data (F^{se}) along with the attribute based encrypted decryption key ($D_{EK_i}^{cp\_abe}$). In order to illicitly gain access to the outsourced data attacker would need D_{EK_i}. However, as D_{EK_i} is encrypted with the access control policy (τ) it would further require CP-ABE secret key (SK) that can satisfy τ. Since, SK is only shared with the legitimate users by the data owner, the computational complexity for an attacker would be equal to deciphering CP-ABE without SK. Even if multiple attackers (i.e., revoked users) combine their revoked secret keys to compromise privacy of the outsourced data, they would not succeed as they do not have access to the updated legitimacy number ($L_{num}^{cp\_abe}$), along with the updated public key component ($\rho_{att_{i+1}}^{pre_{\omega}}$) of SK.

6.3 Key distribution

SAPDS uses three different types of keys to achieve fine-grained access control over the outsourced data. Data encryption key (D_{EK_i}) is outsourced to the cloud server, whereas user secret key (SK) and transformation key (θ) are disseminated to the legitimate users during the subscription process. Both of these keys are concealed with legitimate user's public key by the data owner. User possessing the respective private key can decipher them and can gain access to the outsourced data, if required set of attributes associated with SK can satisfy τ. Thus for any attacker either revoked user or even cloud service provider the computational complexity to gain access to SK and θ would be equal of reverting asymmetric encryption without valid private key. For the cryptographic proof of public key infrastructure readers may refer to [33].

6.4 File access

SAPDS does not rely on cloud service provider to govern data access. On each user request cloud service provider replies back with the requested outsourced data (F^{se}) along with its respective decryption key ($D_{EK_i}^{cp\_abe}$). Outsourced data and decryption key are concealed to assure authorized data access. To illicitly gain access attacker would need to decipher D_{EK_i} without valid secret key (SK). Even if attacker teams up with each other (i.e., revoked users working together to illicitly gain access), still they would not able to retrieve D_{EK_i} as it is encrypted under the access control policy (τ). In order to satisfy τ valid set of attributes along with updated legitimacy number (L_{num}) are required. Since, attacker does not possess the valid set of attributes and even does not have access to L_{num}, $D_{EK_i}^{cp\_abe}$ cannot be deciphered, thus privacy of the outsourced data is preserved.

6.5 User revocation

SAPDS does not rely on any trusted third party for key management. It utilizes cloud service provider to disseminate the updated decryption key ($D_{EK_{i+1}}$) to the legitimate users after each user revocation. On each user revocation a new legitimacy attribute ($L_{att_{i+1}}$) is generated by using a randomly selected legitimacy number (L_{num}) in association with the current legitimacy attribute (L_{att_i}), see Fig. 3. To ensure that legitimate users continue their access to the outsourced data a PRE encrypted public key component ($\rho_{att_{i+1}}^{pre_\omega}$) of the user's secret key is outsourced to the cloud service provider. Each legitimate user needs L_{num} to update his secret key (SK). In order to prevent revoked users from updating their SK, L_{num} is concealed with the access control policy ($\tau_{\setminus U_{rvk}}$). For a revoked user to access the outsourced data, $\rho_{att_{i+1}}^{pre_\omega}$ is required. However, as L_{num} is encrypted with $\tau_{\setminus U_{rvk}}$, it would not be able to learn L_{num}, thus restraining revoked user to update his secret key.

Even if multiple users, revoked at different time intervals (i.e., h and i where $h < i$) team up with each other to learn L_{num}, it will be of no use. As for each revoked user the computational complexity would be equal to XOR of L_{num_j} and L_{att_j}, where j is the time interval in which user is revoked. User revoked earlier then the current revocation would have to overcome the computational complexity of all revocation till the current time interval.

7 Results and evaluation

In this section, we provide the performance assessment of the proposed scheme. Particularly, our assessment focuses on the computation and communication overhead exerted by SAPDS on each entity (*owner, user, and cloud server*).

7.1 Performance and evaluation setup

To assess SAPDS's performance for the owner and user, evaluation process is carried out on 32-bit Pentium IV, Ubuntu 9.10 with a 2.60 GHz Dual-Core processor and

3.0 GB RAM. We evaluated SAPDS on three different encryption algorithms, Data Encryption Standard (DES), Triple DES, and Advance Encryption Standard (AES).[3] To evaluate SAPDS's performance on a cloud server we chose GoogleApp Engine [34] as a cloud service provider. Cloud server is only responsible for data storage and key management. The evaluation process measures the computational overhead of Proxy Re-encryption on GoogleApp Engine, by realizing a service which assist legitimate users to update their secret key.

Through this evaluation process, we highlight the feasibility of SAPDS. For that particular reason, we use computing power, which we believe is accessible to average cloud user (owner). We have shown that complexity of getting secret key is $O(1)$, whereas decryption of $D_{EK_i}^{cp\_abe}$ depends on number of sk_{att} associated to SK, thus user can access cloud storage from devices having limited computation power (*i.e., Smartphone*). In the underlying evaluation process, we exhibit computation and communication overhead of SAPDS on each entity. For evaluation, each step of SAPDS is executed 5,000 times and then average values are plotted as final verdict. For encryption and decryption of CP-ABE, we use the library available at [32].

7.2 Computation overhead

The very first step in SAPDS is defining the access policy τ under which D_{EK_i} is concealed. CP-ABE implementation [32] supports two kinds of policy generation mainly due to the type of attributes (sk_{att}) associated with the SK. First one associates simple literal attributes with the leaf nodes of τ e.g., System Administrator, IT Department. The second one associates complex attributes with logical conditions e.g., $Age > 24$ *years*, $EmployeeRank < 10$. Figure 4 illustrates that generation of SK with ten different simple attributes would take maximum 0.2 seconds. However, it would take at most 14 seconds to generate a secret key associated with ten complex attributes. Current implementation of CP-ABE utilizes Bison and YACC parser languages to convert the access policy into a machine-readable format. Each token generated by the parser is stored into GSList data structure, that results in consuming more CPU cycles in parsing the complex policy as compared to the simple policy.

In SAPDS user executes CP-ABE decryption process to retrieve D_{EK_i}. Decryption is primarily required when user gets the $D_{EK_i}^{cp\_abe}$ for the very first time, and during the decryption of $L_{num}^{cp\_abe}$. As CP-ABE support two types of policies, Fig. 5(a) and (b) show the computation overhead of CP-ABE decryption process over 56-bit, 128-bit, and 256-bit keys of DES, Triple DES and AES respectively.

CP-ABE [32] exhibits same decryption time for different sizes of cipher text, encrypted under similar policies. However, type of attribute do effect the complexity of decryption process. Shown in Fig. 6, cipher text associated with the complex attributes tends to consume slightly more CPU cycles as compared to the simple attributes. Furthermore, CP-ABE shows linear decryption overhead with the increase in number of associated attributes.

[3]The rationale of evaluating SAPDS on different configurations is obvious, as it does not impose any restriction on underlying symmetric encryption algorithm. One can choose the encryption algorithm he/she desires, to conceal PII then utilize SAPDS to achieve fine-grained access control over the outsourced data.

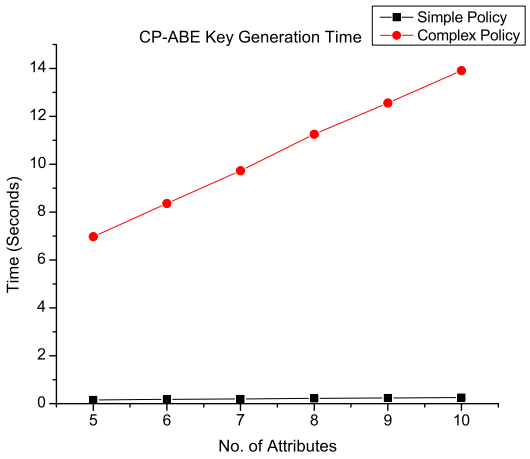


Fig. 4 Computation overhead of secret key generation process of CP-ABE

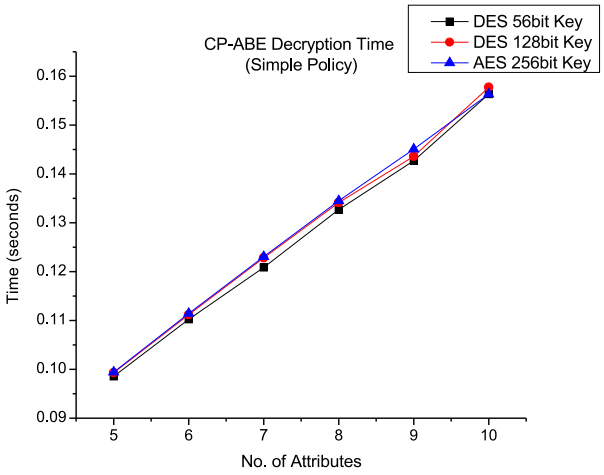
Table 3 Performance assessment of user secret key update on GoogleApp Engine

Data size (bytes)	GoogleApp Engine		
	Computation time (ms)	Response time (ms)	Cost ($)
8	1.4	32.2	0.000873
16	1.4	32.2	0.000874
32	1.8	32.8	0.000875
64	2.6	33.0	0.000876
128	4.4	35.4	0.000878

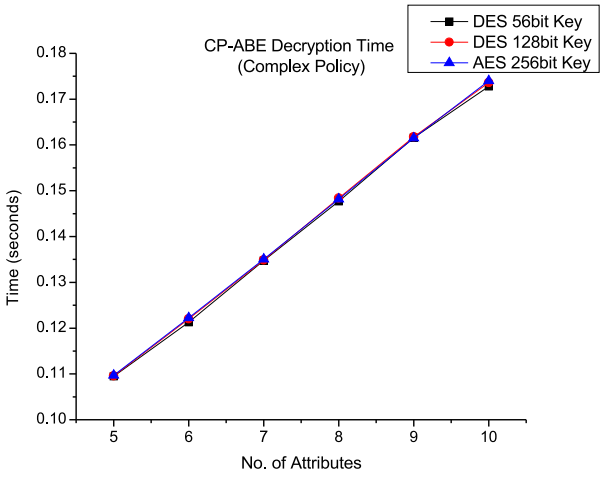
Complex attributes certainly provide more expressivity of access policy. However, they tend to exert more computation overhead on the owner during secret key generation. However, CP-ABE decryption is not affected by the choice of attributes made by the owner. Moreover, as illustrated in Fig. 6 increase in size of access policy increase the decryption time, nevertheless it shows the linear behavior in the increase of time required to decrypted cipher text, for both simple and complex attributes.

SAPDS achieves user revocation by updating the legitimacy attribute ($L_{att_{i+1}}$) associated with the access control policy (τ) which conceals the data decryption key ($D_{EK_i}^{cp\_abe}$). User secret key (SK) update task is delegated to cloud server by outsourcing the public key component of legitimacy attribute encrypted with Proxy Re-encryption i.e., $\rho_{att_{i+1}}^{pre_\omega}$. Cloud server transforms $\rho_{att_{i+1}}^{pre_\omega}$ to $\rho_{att_{i+1}}^{pre_\theta}$ by using the transformation key (ψ). To evaluate the computational complexity of Proxy Re-encryption we realized a secret key update service on GoogleApp Engine which transforms the public key component of varied sizes. Table 3 presents the computational over head of GoogleApp Engine to transform the public key component.

Computational complexity to transform $\rho_{att_{i+1}}^{pre_\omega}$ is measured by analyzing the CPU Time (*execution time on physical CPU*) and estimated CPU usage cost of 1000 requests of similar computational complexity. Our evaluation result shows that to transform public key component of size 8 to 128 bytes, secret key update service takes 1.4



(a) Simple Access Policy



(b) Complex access policy

Fig. 5 Computation overhead of CP-ABE decryption process

to 4.4 milliseconds, with an average response time of 32.2 to 35.4 milliseconds respectively. Whereas, it only costs between 0.000873 to 0.000878 dollar for every 1000 requests of secret key update. In short, if data is shared between 1001 users and data owner updates the legitimacy attribute of size 8 bytes to revoke a single user, then data owner only has to pay 0.000873 dollar to let rest of 1000 legitimate users to update their secret keys.

The evaluation result highlights the fact that delegating key management to cloud server has multifaceted benefits. First, data owner does not need to rely on any trusted third party to disseminate data decryption to the legitimate users. Second, the cost of decryption key dissemination is reasonably low. Thus assisting data owner to achieve fine-grained access control over the outsourced data with minimal expenses of key management.

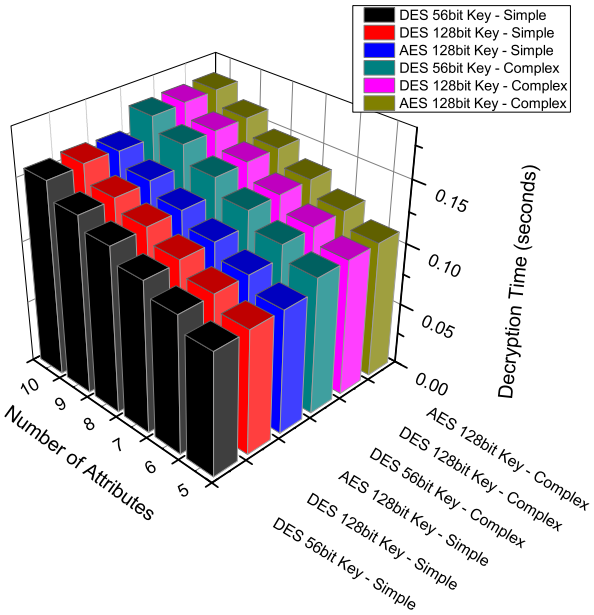


Fig. 6 Comparison of CP-ABE decryption overhead for simple and complex attributes

7.3 Communication overhead

With the adoption of cloud computing, new-fangled business models are coming forth, which consider the amount of data exchanged between the cloud server and its users. The decryption keys streaming between entities can greatly affect the invoice which owner will receive from cloud provider. In this context, for communication overhead we measure the size of encrypted data flowing between the entities except from the encrypted file; as SAPDS does not impose any restriction on symmetric encryption algorithm.

Other than the encrypted file, D_{EK_i} is stored on the cloud server. CP-ABE can greatly reduce the amount of data exchanged between the entities in order to achieve fine-grained access control, if policies are defined for groups instead of individual users. However, for the fairness of evolution process we consider that decryption keys are generated for individual users rather than groups. Figure 7 shows that the size of secret key is in direct proportion to the number of attributes associated with it. For complex policy CP-ABE outputs bigger sized keys as compared to the simple policy. The primary reason for the difference in size is, CP-ABE assign a 64 Bit integer value to each of the token generated by the Bison and YACC parsers.

CP-ABE is applied to different size of symmetric encryption algorithm decryption keys. Figure 8 depicts the difference between cipher text size when D_{EK_i} is encrypted with simple and complex attributes. However, the difference between cipher text size is relatively low, since only the access policy is associated with the cipher text, whereas attribute values are associated with the SK. The difference between the size of cipher text is due to the fact that complex policy includes logical operators, whereas in simple policy no logical operator is used.

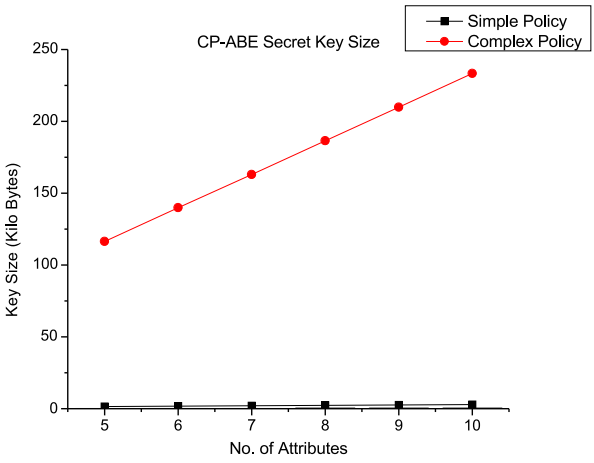


Fig. 7 CP-ABE user secret key size with different attributes

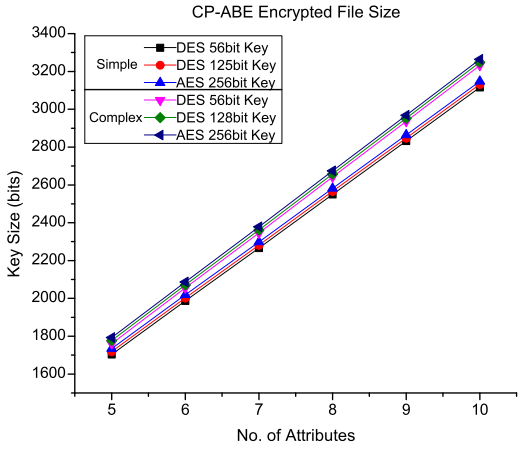


Fig. 8 Encrypted file size

8 Discussion

SAPDS elegantly enables the owner to migrate their data to an un-trusted domain without compromising privacy. In addition, it enables the owner to a maintain fine-grained access control over the outsourced data even if owner is not in command of the underlying computing resources and infrastructure. With SAPDS, it is possible for the owner to revoke a particular user or a group of users, with $O(n')$ complexity and without ever disseminating updating decryption key to the legitimate users by himself. Through the proposed scheme, owner delegates the key management task to the cloud server devoid of revealing any confidential information, which could impend the privacy of the outsourced data.

Similarly, from user's point of view SAPDS enables legitimate user to seamlessly obtain updated decryption keys with minimum number of interactions with the cloud server. It does not rely on any trusted third party to govern the key update protocol; in

fact, it utilizes attribute-based encryption to ensure that only the legitimate user can derive a new decryption key from his previous key.

Through SAPDS, cloud server not only provisions the outsourced data it also works a proxy for the owner to disseminate the decryption keys. Although, cloud server provides the decryption key, nevertheless the key is encrypted with attribute-based encryption to guarantee data confidentiality. Access policy associated with decryption key governs the decryption process, according to the access control policy defined by the owner.

As the core purpose of SAPDS is to ensure fine-grained access control over the cloud-based storage system, which ensures that at any quantum of time all of the keys circulating among users must confirm the policy associated with the hosted data. This policy enforcement requires that as soon as user is revoked all of the legitimate users must update their decryption keys cornering off the revoked user.

Through experiments, we demonstrated that SAPDS can affect the bill which owner would receive from the cloud server. Selecting the correct combinations of attributes for the access control policy is very important. On one had complex attributes assist the owner to define more comprehensive access policy, on the other hand they tend to consume more processing power and bandwidth. Whilst simple attributes provide fewer comprehensibility; however, they are more resource friendly in terms of using computation and bandwidth usage requirements. Nevertheless, CP-ABE used with either of the policies provides the same level of security against chosen-plain text, and chosen-ciphertext attack with the assumption of Decisional Bilinear Diffie Hellman Assumption (DBDH); which states that no polynomial algorithm β can distinguish between (1) and (2) with more than a negligible advantage.

$$A = g^a, \quad B = g^b, \quad C = g^c, \quad e(g,g)^{abc} \tag{1}$$

$$A = g^a, \quad B = g^b, \quad C = g^c, \quad e(g,g)^{z} \tag{2}$$

The advantage of β is can modeled as (3)

$$\left|\Pr\left(\beta_r, \left(A, B, C, e(g,g)^{abc}\right) = 0\right) - \Pr\left(\beta_r, \left(A, B, C, e(g,g)^{z}\right) = 0\right)\right| \tag{3}$$

The probability is taken over the random choice of the generator g, the random choice of a, b, c in $\mathbb{Z}_p$ and the random bits consumed by β_r [26].

Existing system providing privacy aware data sharing tends to add complexity and does not consider the computation and bandwidth cost. Fine-grained access control over the outsourced data achieved by SAPDS, exhibits far less computation complexity as compared to the existing ones.

Shucheng Yu et al. [25] proposed a technique that utilizes KP-ABE to protect the data hosted in the cloud server. Their technique utilizes Lazy re-encryption to eliminate unnecessary cryptographic overhead after each user revocation. With confidential information hosted in the un-trusted domain, lazy re-encryption could create serious privacy issues. Imagine the medical record of a person who does not visit doctor frequently as he is living a healthy life, would be available to the revoked users of a cloud based EHR, just for the reason that it is not being modified.

For user revocation complexity of Shucheng Yu et al. [25] scheme is $O(n)$, where n is total number of legitimate users, besides this each user needs to update every public component of his secret key making it $O(sizeOf(\tau))$. Whereas, SAPDS exhibits complexity $O(n')$, where n' is total number of revoked users at current annulment without considering the previously revoked user, besides this user only need to update their legitimacy attribute making its complexity $O(1)$. In [25] on each user revocation a new access control policy is defined which is then used to conceal the file encryption key. While defining new access control policy, owner has to make sure that users revoked at previous time interval are not able to access data. In contrast to that SAPDS only requires to update a single legitimacy attribute without considering revoked users at previous time intervals. This is because it is computationally infeasible for the revoked users to learn all the legitimacy values till the current revocation.

Key Regression [20] is similar to key rotation scheme, except it introduces the concept of member state which assists in generating the decryption keys for accessing current as well as for the historical data. However, it lacks scalability, as during the initialization phase maximum-wind (*maximum rotation*) is provided as the security parameter, restricting the maximum number of decryption keys that can be generated. In contrast to that, SAPDS does not specify any number that restricts number of secret keys that can be produced, making it scalable with dynamic user set. Key Regression provides the capabilities to access the historical data encrypted with different decryption key than the current one. However, to share the updated member state it does not specify any methodology. To enable legitimate users to continue their access to the outsourced data, owner has to adopt outbound methods to share member states with the respective users. In contrast to that SAPDS utilizes cloud service provider to disseminate the updated data decryption key.

SiRiUS [18], also claim to provide secure remote storage system in an un-trusted domain by defining an access structure that contains the file decryption and signature keys. It also suffers from the issue of scalability, as number of users increase so does the size of access structure. Whereas, SAPDS maintains an access control policy with the outsourced data, instead of maintaining the decryption key. Since, keys are distributed to the individual users, number of users do not affect the size of outsourced data. Whereas, in case of SiRiUS increase in number of users tend to grow the size of outsourced data. To ensure data confidentiality SiRiUS stores the asymmetrically encrypted file decryption key on remote storage. Each legitimate user has its own copy of concealed file decryption key, which only he can decrypt by using his private key. SiRiUS provides perfect secrecy with the computational complexity of public key cryptography in case of an attack. However, in case of user revocation owner has to encrypt the newly generated file decryption key for all the legitimate users by their respective public keys. Contrary to that, SAPDS provides user revocation which only requires owner to update the data decryption key and outsourced it to the cloud service provider by encrypting it under the access control policy with updated legitimacy attribute.

CRUST [19] proposed a cryptographic remote system which utilizes symmetric encryption to conceal the data. For key management CRUST assumes the availability of a trusted agent to disseminate the decryption keys on the behalf of owner. Dissimilar to CRUST, SAPDS does not rely on any trusted third party for decryption

key distribution. In fact, SAPDS utilizes CP-ABE for the delegated responsibilities of key distribution without compromising privacy of the outsourced data. To avoid unnecessary cryptographic operations in case of file update, CRUST divides file into blocks each encrypted with a separate key. This methodology ensures that only modified blocks are re-encrypted during the file update process. However, in case of user revocation it increases the number of keys trusted agent has to update, in order to ensure that legitimate users continue their access to the outsourced data. SAPDS does not support this type of block encryption which is useful in case of large files. However, SAPDS does provide a more robust user revocation in term of decryption key update and dissemination, by utilizing CP-ABE to conceal the updated data decryption key. Most importantly SAPDS does not rely on any trusted third party or agent for decryption key management.

PCE [22] proposed a hierarchical data sharing system for electronic health record, by encrypting each level of hierarchy PCE can share the decryption keys among appropriate users. This strategy works well when shared data contains diverse information which needs to be shared among variant users. Since PCE utilizes symmetric and asymmetric encryption it suffers from the problem of user revocation, in which each legitimate user needs to obtain the new decryption key from owner or by the trusted third party. This type of hierarchical data sharing is not supported by SAPDS; however, it can be easily realized by encrypting each portion of the outsourced data with a different access policy, and then distributing the decryption keys to the appropriate users via cloud server. For the user revocation PCE suffers from the problem of key update and dissemination similar to CRUST. Whereas, SAPDS provides user revocation which exerts reasonable computational load on owner and on user as well, without the intervention of trusted third party. Thus, to achieve fine-grained access control over the outsourced data SAPDS provides more realistic data sharing capabilities than PCE.

SAPDS is a privacy aware fine-grained access control scheme for the data hosted in the cloud-based storage system. It exhibits fewer computation and bandwidth consumption on each involved entity as compared to the existing methodologies. Furthermore, it does not rely on any trusted thirty party to disseminate the updated decryption keys, additionally it leverages the users to update their decryption keys without even interacting with the owner.

9 Conclusion and future directions

Cloud computing promises to reduce costs and increase the on-demand availability of compute resources. However, data privacy issues are slowing down adoption of this lucrative computing paradigm. In this paper, we proposed SAPDS, a novel privacy aware data-sharing scheme, whilst achieving fine-grained access over the outsourced data hosted in the cloud. SAPDS uniquely combines the attribute-based encryption along with proxy re-encryption and user secret key updating capability without relying on any trusted third party. It leverages data owner in availing multi-faceted benefits of cloud computing architecture, without compromising the privacy of the outsourced data. It enables data owner to delegate the toiled work of key management to

the cloud server, while remains himself offline. SAPDS exhibits less computing and communication complexity as compared to the existing privacy aware data sharing schemes. With SAPDS data owner can revoke a user with the computation complexity of $O(n')$. Whilst, cloud server can disseminate the updated decryption keys with the complexity of $O(n - n')$, and complexity of $O(1)$ is exhibited by the user during secret key derivation.

Smartphone combined with collaborative software provide the ultimate computing experience while on the move [35]. Sharing data for collaborative purpose by different owners would require a multiple authority framework that can govern consent of involved authorities. Deploying SAPDS in a multiple authority domain would require modification in access policy modeling. Nevertheless, its underlying principle would remain the same making it feasible where there is a need to avoid reliance on a trusted third party and to achieve delegated responsibilities.

Acknowledgements This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2011-0020515).

Appendix A: Cipher-text attribute based encryption (CP-ABE)

CP-ABE is a cryptographic primitive in which cipher text is associated with an access policy defined over the set of attributes. These attributes are associated with the user secret key which can decrypt the cipher text, if conforms to the access control policy. CP-ABE is a relatively new cryptosystem as compared to PKI [35] and secret sharing schemes [36], which address the issues related to private information retrieval [17]. One of the primary objective of CP-ABE is to ensure access control policy of the data residing in a domain over which owner possesses no control. CP-ABE leverages the owner to govern data access without relying on any trusted third party. CP-ABE consists of four fundamental algorithms: SETUP, ENCRYPT, KEYGEN, and DECRYPT.

A.1 SETUP

This algorithm initializes CP-ABE by selecting a Bilinear Group $\mathbb{G}$ of prime order p with g generator. Once selection of $\mathbb{G}$ is finalized two random numbers $\mathbb{Z}_p$ are selected which constitute the Public Key (PK) and Master Key (MK) as

$$PK = \left(g, h = g^{\beta}, e(g, g)^{\alpha}\right) \tag{4}$$

$$MK = (\beta, g)^{\alpha} \tag{5}$$

A.2 ENCRYPT

In CP-ABE access control policy is modeled in the form of access tree τ. Each leaf node of τ represents an attribute $y \in Y$, where Y is a set of attributes involved in the

access control policy. The encryption algorithm encrypts the plain text M under τ by selecting a random number $s \in \mathbb{Z}_p$ and defining a polynomial for each leaf and non-leaf node, starting from root node R. The degree d_x of each polynomial is set one less than its threshold value. If node is non-leaf node having "AND" logical gate then its threshold value is set equal to its number of children, in case of "OR" logical gate it will be one. Polynomial for the root is defined as $q_R(0) = s$. Encryption algorithm outputs the cipher text as

$$CT = \left(\tau, \hat{C}, C = h^s, \forall y \in Y : C_y = g^{q_y(0)}, C'_y = H\left(att(y)^{q_y(0)}\right)\right) \tag{6}$$

where $C = M.e(g, g)^{\alpha s}$. As polynomials are selected in top down approach starting for the root node, each polynomial of a node (*excluding the root node*) is calculated as $q_x(0 = q_{parent(x)}(index(x)))$, where $parent(x)$ represents the parent of node x and $index(x)$ stands for unique index number assigned to each node of τ.

A.3 KEYGEN

It takes input a set of attributes S along with the system master key and generates the user *Secret Key* (*SK*), which possess the ability to decrypt the cipher text.

$$SK = \left(D = g^{(\alpha+r)/\beta}, \forall j \in S : D_j.H(j)^{r_j}, D'_j = g^{r_j}\right) \tag{7}$$

A.4 DECRYPT

CP-ABE decryption algorithm decrypts the cipher text using polynomial interpolation [37]. Cipher text along with user secret key *SK* is provided as input to decryption algorithm. For each leaf node x in τ it computes the value

$$Decrypt(CT, SK, x) = \frac{e(D_j, C_x)}{e(D'_j, C'_x)} \tag{8}$$

After computing the value of $Decrypt(CT, SK, x)$ function for each leaf node, it executes polynomial interpolation to estimate the value of hidden factor. For polynomial interpolation, it uses Lagrange Coefficient $\Delta_{i,s}(x) = \prod j \in s, j \neq i \frac{x-j}{i-j}$; applying polynomial interpolation will reveal the hidden factor $e(g, g)^{\alpha s}$, which was used to encrypt the fact.

Appendix B: Proxy re-encryption (PRE)

Proxy Re-Encrypt is a cryptographic primitive, which transforms the cipher text from one secret key to another without revealing the secret key to a semi-trusted party. Through PRE, cipher text encrypted with Alice secret key can be transformed to another cipher text, which Bob can decrypt without revealing any information to the intermediary (*semi-trusted server*). PRE consists of four fundamental steps, Key Generation, Encryption, Re-Encryption, and Decryption algorithm. Suppose Alice wants to send a message m to Bob through an intermediary server by using PRE, following are the steps, which will be executed.

B.1 Key generation

Alice first selects a Bilinear Group $\mathbb{G}$ of prime order q with g generator. Two random numbers a and b of order q are generated. a and b are then use to generate secret respective keys $SK_a = a$ and $SK_b = b$, consequently public key are produced as $PK_a = g^a$ and $PK_b = g^b$. Once public key are defined Alice will select a random number $r \in \mathbb{Z}^*{}_p$, along with a Bilinear Map of $\mathbb{G}$ as $Z = e(g, g)$. Finally, proxy-key is generated as $RK_{a\to b} = (g^b)^{1/a}$ and is handed over to the semi-trusted server responsible for cipher text transformation.

B.2 Encryption

In order to encrypt message m, with Alice public key, cipher text is computed as $C_a = (Z^r.m, g^{ra})$.

B.3 Re-encryption

This step is executed by a semi-trusted server. Cipher text is transformed from $C_a \to C_b$ by using proxy-key $RK_{a\to b}$

$$C_b = \left(Z^r.m, e\left(g^{ra}, RK_{a\to b}\right)\right) \tag{9}$$

$$C_b = \left(Z^r.m, e\left(g^{ra}, g^{b/a}\right)\right) \tag{10}$$

$$C_b = \left(Z^r.m, Z^{rb}\right) \tag{11}$$

B.4 Decryption

To decrypt the cipher C_b, Bob uses his secret key SK_b, communicated to him by Alice through secure means i.e., *SSL*. Message m can be obtained as $m = \frac{Z^r.m}{(Z^{rb})^{1/b}}$.

Appendix C: Key derivation (KD)

Key derivation is a process through which new encryption keys can be derived using the old keys. To ensure authorized access of resources, encryption keys are updated periodically or on each user revocation, which leads to the problem of key management. Each time user(s) is revoked, owner needs to update the encryption keys of entire legitimate user set, and distribute them either through email or through centralized secure repository. To overcome this problem, many key derivation techniques have been proposed by research community. The notion behind key derivation is to delegate the arduous task of key generation to the respective client devoid of legitimate user's dependency on the owner (*key distribution, guaranteed availability of owner after each user revocation*).

References

1. Armbrust M et al (2010) A view of cloud computing. Commun ACM 53:50–58
2. Kondo D, Javadi B, Malecot P, Cappello F, Anderson DP (2009) Cost-benefit analysis of cloud computing versus desktop grids. In: Proceedings of the 2009 IEEE international symposium on parallel&distributed processing. IEEE Computer Society, Washington, DC, pp 1–12
3. Assunção MD, Costanzo A, Buyya R (2010) A cost-benefit analysis of using cloud computing to extend the capacity of clusters. Clust Comput 13:335–347
4. Amazon Web Services LLC, Amazon Simple Storage Service (2009). http://aws.amazon.com/s3/
5. Amazon Web Services LLC, Amazon Elastic Compute Cloud (2009). http://aws.amazon.com/ec2/
6. Wood T, Gerber A, Ramakrishnan KK, Shenoy P, Van der Merwe J (2009) The case for enterprise-ready virtual private clouds. In: Proceedings of the 2009 conference on Hot topics in cloud computing, Berkeley, CA, USA, HotCloud'09, USENIX Association, p 4
7. Head MR, Sailer A, Shaikh H, Shea DG (2010) Towards self-assisted troubleshooting for the deployment of private clouds. In: Proceedings of the 2010 IEEE 3rd international conference on cloud computing, CLOUD '10. IEEE Computer Society, Washington, DC, pp 156–163
8. Gartner Rating 2011. http://www.gartner.com/it/page.jsp?id=1454221
9. Naghshineh M et al (2009) Ibm research division cloud computing initiative. IBM J Res Dev 53:499–508
10. Pearson S (2009) Taking account of privacy when designing cloud computing services. In: Proceedings of the 2009 ICSE workshop on software engineering challenges of cloud computing, CLOUD '09. IEEE Computer Society, Washington, DC, pp 44–52
11. Jaeger T, Schiffman J (2010) Outlook: Cloudy with a chance of security challenges and improvements. IEEE Secur Priv 8:77–80
12. Ristenpart T, Tromer E, Shacham H, Savage S (2009) Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: Proceedings of the 16th ACM conference on computer and communications security, CCS '09. ACM, New York, pp 199–212
13. Michael B (2009) In clouds shall we trust? IEEE Secur Priv 7:3
14. Kaufman LM (2009) Data security in the world of cloud computing. IEEE Secur Priv 7:61–64
15. Pearson S, Charlesworth A (2009) Accountability as a way forward for privacy protection in the cloud. In: Proceedings of the 1st international conference on cloud computing, CloudCom '09. Springer, Berlin, pp 131–144
16. U.S. Department of Health and Human Services, http://www.hhs.gov/ocr/privacy/ (2003) Protecting Personal Health Information in Research: Understanding the HIPAA Privacy Rule
17. Yekhanin S (2010) Private information retrieval. Commun ACM 53:68–73
18. Goh E-J, Shacham H, Modadugu N, Boneh D (2006) Sirius: Securing remote untrusted storage. In: Proceedings of the fifth workshop on the economics of information security (WEIS 2006)
19. Geron E, Wool A (2009) Crust: cryptographic remote untrusted storage without public keys. Int J Inf Secur 8:357–377
20. Fu K, Kamara S, Kohno T (2006) Key regression: Enabling efficient key distribution for secure distributed storage. In: Proc. network and distributed systems security symposium (NDSS)
21. Backes M, Oprea A (2005) Lazy revocation in cryptographic file systems. In: Proceedings of the third IEEE international security in storage workshop. IEEE Computer Society, Washington, DC, pp 1–11
22. Benaloh J, Chase M, Horvitz E, Lauter K (2009) Patient controlled encryption: ensuring privacy of electronic medical records. In: Proceedings of the 2009 ACM workshop on cloud computing security, CCSW '09. ACM, New York, pp 103–114
23. Bethencourt J, Sahai A, Waters B (2007) Ciphertext-policy attribute-based encryption. In: Proceedings of the 2007 IEEE symposium on security and privacy, SP '07. IEEE Computer Society, Washington, DC, pp 321–334
24. Ateniese G, Fu K, Green M, Hohenberger S (2006) Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Trans Inf Syst Secur 9:1–30
25. Yu S, Wang C, Ren K, Lou W (2010) Achieving secure, scalable, and fine-grained data access control in cloud computing. In: Proceedings of the 29th conference on information communications, INFOCOM'10. IEEE Press, Piscataway, pp 534–542
26. 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, CCS '06. ACM, New York, pp 89–98

27. Kallahalla M, Riedel E, Swaminathan R, Wang Q, Fu K (2003) Plutus: Scalable secure file sharing on untrusted storage. In: Proceedings of the 2nd USENIX conference on file and storage technologies. USENIX Association, Berkeley, pp 29–42
28. di Vimercati SDC, Foresti S, Jajodia S, Paraboschi S, Samarati P (2007) Over-encryption: management of access control evolution on outsourced data. In: Proceedings of the 33rd international conference on very large data bases, VLDB Endowment, VLDB '07, pp. 123–134
29. Yu S, Wang C, Ren K, Lou W (2010) Attribute based data sharing with attribute revocation. In: Proceedings of the 5th ACM symposium on information, computer and communications security, ASIACCS '10. ACM, New York, pp 261–270
30. Brickell E, Li J (2007) Enhanced privacy id: a direct anonymous attestation scheme with enhanced revocation capabilities. In: Proceedings of the 2007 ACM workshop on privacy in electronic society, WPES '07. ACM, New York, pp 21–30
31. Jingmin He MW (2001) Cryptography and relational database management systems. In: Proceedings of the 2001 international symposium on database engineering & applications. IEEE Computer Society, Washington, DC, pp 273–284
32. Bethencourt J, Waters B, Sahai A Ciphertext-policy attribute-based encryption. http://acsc.cs.utexas.edu/cpabe/, accessed on Dec 10, 2010
33. Menezes AJ, Oorschot PCV, Vanstone SA, Rivest RL (1997) Handbook of applied cryptography
34. Google App Engine—Run your web applications on Google's infrastructure. http://code.google.com/appengine/
35. Adams C, Farrell S (1999) Internet x.509 public key infrastructure certificate management protocols
36. Shamir A (1979) How to share a secret. Commun ACM 22:612–613
37. Interpolating Polynomial The Wolfram Demonstrations Project. http://demonstrations.wolfram.com/InterpolatingPolynomial/
38. Guo H, Viktor H, Paquet E (2011) Privacy disclosure and preservation in learning with multi-relational databases. J Comput Sci Eng 5(3):183–196