Cloud Storage: Securely Transferring, Hosting, and Searching Encrypted Data

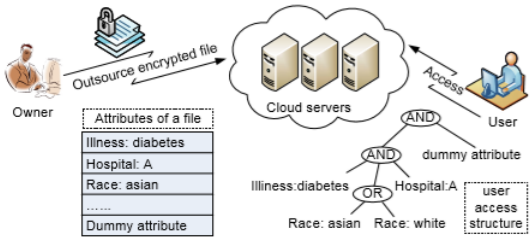
A Survey Paper by Robin Ward

***Abstract*—Cloud computing, specifically cloud data storage, is beginning to become the new normal for many companies. The ability to have someone else host your data, manage it, back it up, among many other tasks is invaluable. In many cases, it is also more reliable. Big name companies like Amazon, Microsoft, and Google are spending billions on ensuring that their infrastructure is scalable, fast, and redundant is something that many smaller companies hosting on premise physical storage are simply not able to do. That is where cloud storage will come in to play. For a service fee every month, these smaller companies can entrust their data with other companies to manage, allowing them to focus on what really matters, running their business. So, what is the downside? Security. These security concerns are since the end user is entrusting their potentially highly sensitive information to that of a company outside of their domain. In many systems, protecting the data is not only the desire of the client, but also a legal need. For example, all healthcare providers will need to store Personal Identifiable Information (PHI) in their database. Keeping patient data such as addresses, names, and Social Security Numbers is a legal requirement instated by the Health Insurance Portability and Accountability Act (HIPPA) [1]. While this need is vital to ensure data security, once the information is encrypted, there will always be a need to sift through encrypted data. Thankfully, there has been extensive research and development done in these areas as outlined in the referenced papers. The two main aspects that this survey will cover are secure and scalable access control and preserving the privacy of keyword search over encrypted data [1,2].**

I. State of the Art

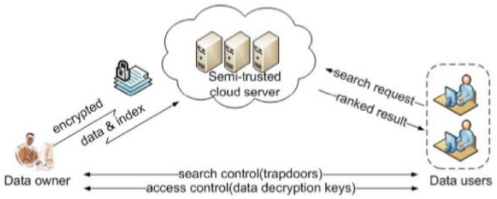
THE first issue to discuss is the need to achieve secure, scalable, and fine-grained data access control within these cloud storage companies. In order to maintain data security over these potentially very sensitive data files, these solutions will provide a cryptographic method of with a purpose of ensuring that the data is not intercepted and viewed by anyone other than the intended parties. To do this, all the data will be encrypted, with then passing the decryption keys to the end user. The only problem with this is that there is significant computation overhead for the data owner and key distribution when fine grained data access control is needed [1]. Requesting these small amounts of data with such a large overhead can become problematic very fast and will not scale well. Bottom line, business owners and end users would like to take advantage of the scalability, simplicity, and profitability of a cloud storage solution without these untrusted domains having access to their data. The first paper, “Achieving Secure, Scalable, and Fine-Grained Data Access Control in Cloud Computing” will cover this need and ongoing issue. The first hurdle to cross is to resolve the issue of performance when encrypting and decrypting large amounts of data. There are two main resolutions that have been previously outlined and mentioned: the first is to introduce a per file access control list (ACL) for fine grained data, or by categorizing the data into multiple filegroups for efficiency. As the system scales however, the complexity of the ACL would be a direct relation to the number of users in the system [1]. The problem with the filegroup based scheme is that it is only usable in course grained systems. Applying a set of complex attributes to the data file which correspond to the users ACL is an effective way to solve this problem. Each of the attributes in the data will be assigned a

public key. Then the user decrypt keys will be created based on if their attributes allow them to access the ciphertext. Pictured below is an example of how the attribute-based encryption model would apply to a hospital.



Revoking a user access is also another potential issue that will most likely arise at some point. This would require a reissue of public and private keys for the users, which could prove to be very resource heavy. To solve this issue, the authors came up with a way to assign these resource heavy re-encryption and user secret key update tasks to the cloud server without exposing any of the data. They were able to do this by utilizing the cryptographic key policy attribute-based encryption (KP-ABE) and combine it with proxy re-encryption (PRE) and lazy encryption [1].

The second issue to address is in relation to searching over encrypted cloud data. End user sensitive data will need to be encrypted before being sent to the server. This will terminate the possibility to perform a plaintext keyword search over the data. So, the need to have the ability to search through encrypted cloud files using multiple keywords is of upmost importance. Furthermore, there is also a need to have a result ranked search implemented. This will enable the end user to find relevant information much faster than sifting through every result in hopes it is the correct file. This would also reduce network traffic as there would be less bandwidth used when opening unimportant files. In the paper "Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data", authors Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou solve this problem of privacy-preserving multi-keyword ranked search over encrypted cloud data, which has been abbreviated as "MRSE". The architecture of this search model is illustrated below.



In reference to the diagram above, the client would have an array of documents, denoted as F, to be uploaded to the cloud server in encrypted form, denoted as C. In order for the searching to be effective, the client would first build an encrypted searchable index, denoted as I, from F. After that is complete, the client will then upload the index I and the encrypted document collection C to the cloud server. The next obvious step in this process would be search functionality. The client would then search the entire document collection with a specific number of keywords, denoted as t, the authenticated user would then acquire a cooresponding trapdoor, denoted as T. The client would then send T to the cloud server, which would then initiate a server search of the index I to return the specified encrypted documents. In an effort to reduce overhead and cost, the client may also send an optional number k asking that the server only return the top-k resultset.

Subsequently, the access control mechanism (ACM) will be used to decrypt the files, and also allow the client to edit, add, and delete as needed. This newly discovered method of keyword search would be considered a success when the following criteria is met: multi-keyword ranked search, privacy preserving, and efficiency. Listed below are notation definitions used in formulas.

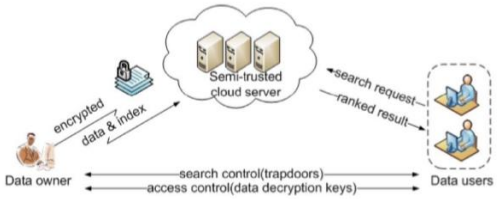
- F – the plaintext document collection, denoted as a set of m data documents $F = (F_1, F_2, \ldots, F_m)$.
- C – the encrypted document collection stored in the cloud server, denoted as $C = (C1, C2, \ldots, C_m)$.
- W – the dictionary, i.e., the keyword set consisting of n keyword, denoted as $W = (W1, W2, \ldots, W_n)$.
- I – the searchable index associated with C, denoted as $(I_1, I_2, \ldots, I_m)$ where each subindex I_i is built for F_i.
- $\tilde{W}$ – the subset of W representing the keywords in the search request, denoted as

$\tilde{W} = (W_{j1}, W_{j2}, \ldots, W_{jt})$.

- $T_{\tilde{W}}$ - The trapdoor for the search request $\tilde{W}$.
- $F_{\tilde{W}}$ – the ranked id list of all documents according to their relevance to $\tilde{W}$.

II. Method Analysis of [1]

Prior to understanding the method, it is necessary to ensure that the reader understands some of the techniques used in this paper. In KP-ABE, the secret keys that the users will receive are based on an access tree, that defines the privileges of that specific user, and then the data is encrypted over a set of attributes [3]. Proxy re-encryption (PRE) is another very important piece to this method. PRE is the act of converting the ciphertext under Alice's private key, so that Bob's private key can view the plaintext. The proxy will not be able to view the plaintext data. This will be a very important feature for file sharing. In order to enforce the attribute access control, KP-ABE will be used to send the encryption keys of the data files. Utilizing just the KP-ABE would solve the issue of fine-graininess file access. However, this alone would prove to be very problematic because there would be much overhead and resource usage, and user revocation would not be possible for when a user is no longer allowed to have access. For access control to function property, each file owner will set several attributes for the file. It is possible to have attributes in common between files. Furthermore, each attribute will be assigned a version number, and a record of changes will be stored in an attribute history list (AHL). There will also be a dummy attribute, which is denoted as Att_D, that is used for key management. This dummy attribute will be required in every file and will never be updated. As previously mentioned, each user will have an access tree that will act as what rights they have to a file. Interior nodes of the tree are threshold gates, for the dummy attribute to function properly, it will need to be assigned an AND gate. After this gate, sets of attributes can be assigned as leaves off each gate. In addition, the cloud server will store a list of valid users and associate a unique ID to them. This will be vital to removing access from certain users. The dummy node will not be affiliated with any other node in the system. The diagram below will provide a perfect example of how this would be setup.

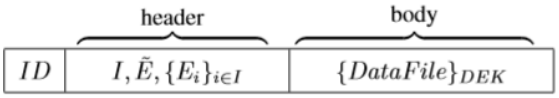


The following table will explain the notation that is used in this solution.

Notation	Description
PK, MK	system public key and master key
T_i	public key component for attribute i
t_i	master key component for attribute i
SK	user secret key
sk_i	user secret key component for attribute i
E_i	ciphertext component for attribute i
I	attribute set assigned to a data file
DEK	symmetric data encryption key of a data file
P	user access structure
L_P	set of attributes attached to leaf nodes of P
Att_D	the dummy attribute
UL	the system user list
AHL_i	attribute history list for attribute i
$rk_{i \leftrightarrow i'}$	proxy re-encryption key for attribute i from its current version to the updated version i'
$\delta_{O,X}$	the data owner's signature on message X

During the system setup, the data file owner will call the algorithm level interface function, which will output the system public key (PK) and the system master key (MK). After the data owner signs the keys, they will then be sent to the cloud servers. Prior to uploading a file to the server, it will need to be encrypted propertly. The steps and diagram below will outline this process.

- select a unique ID for this data file;
- randomly select a symmetric data encryption key $DEK \xleftarrow{R} \mathcal{K}$, where $\mathcal{K}$ is the key space, and encrypt the data file using DEK;
- define a set of attribute I for the data file and encrypt DEK with I using KP-ABE, i.e., $(\tilde{E}, \{E_i\}_{i \in I}) \leftarrow AEncrypt(I, DEK, PK)$.



After the encryption is complete, the file will be stored in the cloud server based on the header and body structure mentioned above.

In summary, the authors have designed a way to use the technique of hybrid encryption to protect data

files that are incoming into the cloud server. this method is accomplished first by utilizing the KP-ABE to introduce the use of fine-grained data access control and the file create, file delete, and new user operations. In order to remove a user successfully, the combination of proxy re-encryption with KP-ABE will be used and the delegating of resource heavy tasks to the cloud server. additionally, after obtaining the PRE keys, the data owner can logoff and not be connected because the cloud server will be handling the tasks. Lazy re-encryption can also be used in addition to enable cloud servers to aggregate multiple update and file re-encryption operations into one, which will save computation overhead.

III. Method Analysis of [2]

The MSRE will consist of the following four algorithms:

- Setup (1^ℓ). *Taking a security parameter ℓ as input, the data owner outputs a symmetric key as SK.*
- BuildIndex $(\mathcal{F}, SK)$. *Based on the data set $\mathcal{F}$, the data owner builds a searchable index $\mathcal{I}$ which is encrypted by the symmetric key SK and then outsourced to the cloud server. After the index construction, the document collection can be independently encrypted and outsourced.*
- Trapdoor $(\widetilde{\mathcal{W}})$. *With t keywords of interest in $\widetilde{\mathcal{W}}$ as input, this algorithm generates a corresponding trapdoor $T_{\widetilde{\mathcal{W}}}$.*
- Query $(T_{\widetilde{\mathcal{W}}}, k, \mathcal{I})$. *When the cloud server receives a query request as $(T_{\widetilde{\mathcal{W}}}, k)$, it performs the ranked search on the index $\mathcal{I}$ with the help of trapdoor $T_{\widetilde{\mathcal{W}}}$, and finally returns $\mathcal{F}_{\widetilde{\mathcal{W}}}$, the ranked id list of top-k documents sorted by their similarity with $\widetilde{\mathcal{W}}$.*

In order to achieve the method of effective searching of encrypted data that has been outsourced, the design will need to achieve security and performance. The following three bullets will explain these in detail [2].

- Multi-keyword ranked search - To design search schemes which allow multi-keyword query and provide result similarity ranking for effective data retrieval, instead of returning undifferentiated results.
- Privacy-preserving - To prevent the cloud server from learning additional information from the data set and the index, and to meet privacy requirements specified in Section
- Efficiency - Above goals on functionality and privacy should be achieved with low communication and computation overhead.

Privacy - As for privacy, the server should only be learning the search results. As for the data privacy, the data owner can utilize the symmetric key cryptography prior to uploading to the cloud, which will prevent the cloud server from prying into the data files. Index privacy will be another item to mention here. If the cloud server figures out any association between the search terms and the files, it may learn the subject of the document or even all the content in a small file. Because of this, the searchable index should be designed to prevent the cloud server from accomplishing this [2]. The second important part to privacy is hiding the contents of the keywords indicated by the corresponding trapdoor. The number of documents containing the keyword is enough statistical information to identify the keyword with high probability [2]. If the cloud server has a vast knowledge on the data set, it can potentially be used to reverse engineer the keyword. The function to generate a trapdoor should be random; this will prevent the cloud server from determining any relationship between the trapdoors. So, to unlink any trapdoors, there must be no determinacy in the trapdoor generation function.

GLOSSARY

- Key-policy attribute-based encryption (KP-ABE) - In KP-ABE, users' secret keys are generated based on an access tree that defines the privileges scope of the concerned user, and data are encrypted over a set of attributes. However, CP-ABE uses access trees to encrypt data and users' secret keys are generated over a set of attributes.
- Proxy re-encryption (PRE) - Proxy re-encryption (PRE) schemes are cryptosystems which allow third parties (proxies) to alter a ciphertext which has been encrypted for one party, so that it may be decrypted by another.
- Lazy re-encryption – this is utilized to reduce overhead and computation. The idea is that there is only a need to re-encrypt if the data has changed. The revoked user already knows what the file contained prior, so only new information should be encrypted.
- Access control list (ACL) - An access-control list (ACL), with respect to a computer file system, is a list of permissions attached to an object. An ACL specifies which users or

system processes are granted access to objects, as well as what operations are allowed on given objects.[1] Each entry in a typical ACL specifies a subject and an operation. For instance, if a file object has an ACL that contains (Alice: read, write; Bob: read), this would give Alice permission to read and write the file and Bob to only read it.

- Access control mechanism (ACM)
- Trapdoor

REFERENCES

[1] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure and fine-grained access control in cloud computing," IEEE INFOCOM 2010.

[2] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," IEEE INFOCOM 2011

[3] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in Proc. of CCS'06, 2006.