**Provably Secure Key-Aggregate Cryptosystems with Broadcast Aggregate Keys for Online Data Sharing on the Cloud**

**ABSTRACT:**

Online data sharing for increased productivity and efficiency is one of the primary requirements today for any organization. The advent of cloud computing has pushed the limits of sharing across geographical boundaries, and has enabled a multitude of users to contribute and collaborate on shared data. However, protecting online data is critical to the success of the cloud, which leads to the requirement of efficient and secure cryptographic schemes for the same. Data owners would ideally want to store their data/files online in an encrypted manner, and delegate decryption rights for some of these to users, while retaining the power to revoke access at any point of time. An efficient solution in this regard would be one that allows users to decrypt multiple classes of data using a single key of constant size that can be efficiently broadcast to multiple users. Chu et al. proposed a key aggregate cryptosystem (KAC) in 2014 to address this problem, albeit without formal proofs of security. In this paper, we propose CPA and CCA secure KAC constructions that are efficiently implementable using elliptic curves and are suitable for implementation on cloud based data sharing environments. We lay special focus on how the standalone KAC scheme can be efficiently combined with broadcast encryption to cater to m data users and m0 data owners while reducing the reducing the secure channel requirement from O(mm0) in the standalone case to O(m + m0).

**EXISTING SYSTEM:**

* Current technology for secure online data sharing comes in two major flavors - trusting a third party auditor, or using the user’s own key to encrypt her data while preserving anonymity.
* This system is popularly known as the key-aggregate cryptosystem (KAC), and derives its roots from the seminal work on broadcast encryption by Boneh et.al..
* KAC may essentially be considered as a dual notion of broadcast encryption. In broadcast encryption, a single ciphertext is broadcast among multiple users, each of whom may decrypt the same using their own individual private keys. In KAC, a single aggregate key is distributed among multiple users and may be used to decrypt ciphertexts encrypted with respect to different classes. For broadcast encryption, the focus is on having shorter ciphertexts and low overhead individual decryption keys, while in KAC, the focus is in having short ciphertexts and low overhead aggregate keys.

**DISADVANTAGES OF EXISTING SYSTEM:**

* The cloud is susceptible to privacy and security attacks, that are a major hindrance to its wholesome acceptance as the primary means of data sharing in todays world.
* This scheme is not practically deployable for two major reasons. Firstly, the number of secret keys would grow with the number of data classes. Secondly, any user revocation event would require Alice to entirely re-encrypt the corresponding subset of data, and distribute the new set of keys to the other existing valid users.
* This makes the scheme inefficient and difficult to scale.
* Since the decryption key in public key cryptosystems is usually sent via a secure channel, smaller key sizes are desirable.
* Moreover, resource constrained devices such as wireless sensor nodes and smart phones cannot afford large expensive storage for the decryption keys either.
* Firstly, no concrete proofs of cryptographic security for KAC are provided by the authors.
* Secondly, the scheme does not explicitly address the issue of aggregate key distribution among multiple users. In a practical data sharing environment with millions of users, it is neither practical nor efficient to depend on the existence of dedicated one-to-one secure channels for key distribution. A public key based solution for broadcasting the aggregate key among an arbitrarily large number of users is hence desirable.

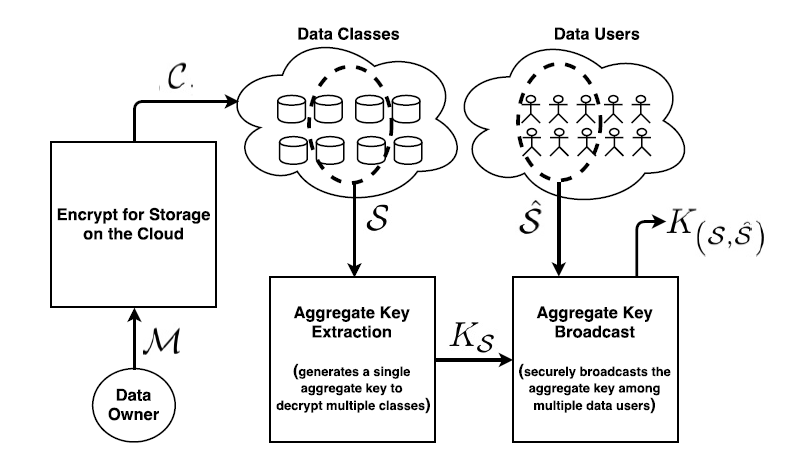
**PROPOSED SYSTEM:**

* In this paper, we attempt to build precisely such a data sharing framework that is provably secure and at the same time, efficiently implementable.
* In this paper we propose an efficiently implementable version of the basic key-aggregate cryptosystem (KAC) using asymmetric bilinear pairings.
* We propose a CCA-secure fully collusion resistant construction for the basic KAC scheme with low overhead ciphertexts and aggregate keys.
* We demonstrate how the basic KAC framework may be efficiently extended and combined with broadcast encryption schemes for distributing the aggregate key among an arbitrary number of data users in a real-life data sharing environment. The extension has a secure channel requirement of O(m + m0) for m data users and m0 data owners.
* In addition, the extended construction continues to have the same overhead for the public parameters, ciphertexts and aggregate keys, and does not require any secure storage for the aggregate keys, which are publicly broadcast.

**ADVANTAGES OF PROPOSED SYSTEM:**

* We prove our construction to be semantically secure against a non-adaptive adversary in the standard model under appropriate security assumptions.
* We also demonstrate that the construction is collusion resistant against any number of colluding parties.
* To the best of our knowledge, this is the first KAC construction in the cryptographic literature proven to be CCA secure in the standard model.

**SYSTEM ARCHITECTURE:**



**MODULES:**

* File Upload
* File Download
* File Update

**MODULE DESCRIPTIONS:**

**File Upload:**

Whenever a need to share data among the group arises, the owner of the file sends the encryption request to the CS. The request is accompanied by the file (*F*) and a list (*L*) of users that are to be granted access to the file. *L* also contains the access rights for each of the users. The users may have READ-only and/or READ–WRITE access to the file. Other parameters can be also set to enforce fine-grained access control over the data. *L* is used to generate the ACL for the data by the CS. *L* is sent to the CS only if the data are to be shared with a new proposed group. If the group already exists, the encryption request will not contain *L*; rather, the group ID of the existing group will be sent. The CS, after receiving the encryption request for the file, generates the ACL from the list and creates a group of the users. The ACL is separately maintained for each file. The ACL contains information regarding the file such as its unique ID, size, owner ID, the list of the user IDs with whom the file is being shared, and other metadata. If the group already existed, only the ACL for the file is created. Next, the CS generates *K* according to the procedure defined in Section III-B and encrypts the file with an appropriate symmetric block cipher (we have used the AES for encryption purposes). The result is an encrypted file (*C*). Subsequently, the CS generates *Ki* and *K\_ i* for every user and deletes *K* by secure overwriting. Secure overwriting is a concept in which the bits in the memory are constantly flipped to make sure that a memory cell never grips a charge for enough duration for it to be remembered and recovered. The *Ki* for each user is inserted into the ACL for later use. To protect the integrity of the file, the CS also computes the hash-based message authentication code (HMAC) signature on every encrypted file. A similar procedure for the HMAC key is adopted. However, the HMAC key is kept by the CS only. The encrypted data, the group ID (in the case of a newly generated group), and the *K\_ i* for the owner are sent to the requesting data owner. The group ID and the *K\_ i* for the rest of the group users are directly sent to them over a secure communication channel. The public keys of the group users can be also used to transmit the user portion of the key. We have used the public keys of the users to transmit the key portions. The user, after receiving *C*, uploads it to the cloud. *K* is deleted via secure overwriting from the CS after the encryption process. It is noteworthy that the key generation process is executed once when the group is initiated and the first file is submitted for encryption. Moreover, a newly joining member also activates the key generation but only for the new member.

**File Download:**

The authorized user sends a download request to the CS or downloads the encrypted file (*C*) from the cloud and sends the decryption request to the CS. The cloud verifies the authorization of the user through a locally maintained ACL. The decryption request is accompanied by the user portion of the key, i.e., *K\_ i*, along with other authentication credentials. The CS computes *K* by applying XOR operation over *K\_ i* and the corresponding *Ki* from the ACL. As each of the users correspond to a different pair of *Ki* and *K\_ i*, none of the users can use other users’ *K\_ i* to masquerade identity. Subsequently, the CS proceeds with the decryption process after verifying the integrity of the file. If the correct *K\_ i* is received by the CS, the result will be a successful decryption process; otherwise, the decryption will fail. After successful decryption, the file is sent to the requesting user through a secure communication channel that could be Secure Sockets Layer (SSL) or Internet Protocol Security (IPSec) channels. *K* is deleted via secure overwriting from the CS after decryption. The users are authenticated before the request processing according to standard procedures. Similar to the file upload process, the downloading of the file can be also done by the CS on behalf of the user. In the aforesaid case, the decryption request is sent to the CS. The CS, after authenticating the user, sends the download request to the cloud for the specified file. The cloud sends the encrypted file (*C*) to the CS. The rest of the process for the decryption is the same.

**File Update:**

Updating the file has a similar procedure to that of uploading the file. The difference is that, while updating, all of the activities related to the creation of the ACL and key generation are not carried out. The user, who has downloaded the file and made any changes, sends an update request to the CS. The request contains the group ID, the file ID, and *K\_i*, along with the file to be encrypted after changes. The CS verifies that the user has the WRITE access to the file from the corresponding ACL. In the case of a valid update request, the CS computes *K* by XORing *Ki* and *K\_ i*, encrypts the file, and performs the HMAC calculations. The encrypted file is sent to the user or uploaded to the cloud. *K* is deleted afterward.

**SYSTEM REQUIREMENTS:**

**HARDWARE REQUIREMENTS:**

* System : Pentium Dual Core.
* Hard Disk : 120 GB.
* Monitor : 15’’ LED
* Input Devices : Keyboard, Mouse
* Ram : 1 GB

**SOFTWARE REQUIREMENTS:**

* Operating system : Windows 7.
* Coding Language : JAVA/J2EE
* Tool : Netbeans 7.2.1
* Database : MYSQL

**REFERENCE:**

Sikhar Patranabis, Yash Shrivastava and Debdeep Mukhopadhyaym, “Provably Secure Key-Aggregate Cryptosystems with Broadcast Aggregate Keys for Online Data Sharing on the Cloud”, **IEEE Transactions on Computers, 2017.**