Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 2, Issue. 11, November 2013, pg.356 – 363

RESEARCH ARTICLE

POLICY BASED FILE ASSURED DELETION WITH SECURE OVERLAY CLOUD STORAGE

M. Geetha[1], V. Ravi kumar[2]

PG Scholar, Department of Software Engineering, TKR College of Engineering and Technology, Hyderabad, A.P-500 097, India
Email: geethareddy.it@gmail.com
Associate Professor, Department of Software Engineering, TKR College of Engineering and Technology, Hyderabad, A.P-500 097, India

Abstract: ***The outsource data backup to third-party cloud storage services so as to reduce data management costs, security concerns arise in terms of ensuring the privacy and integrity of out-sourced data. Design* Policy Base *a practical, implementable, and readily deployable cloud storage system that focuses on protecting deleted data with Policy Based file secured deletion. Policy Base is built upon standard cryptographic techniques, such that it encrypts outsourced data files to guarantee their privacy and integrity, and most importantly, securedly deletes files to make them unrecoverable to anyone upon revocations of file access policies. In par-titular, the design of Policy Base is geared toward the objective that it acts as an overlay system that works seamlessly atop today's cloud storage services. To demonstrate this objective, we implement a working prototype of Policy Base a top Amazon S3, one of today's cloud storage services, and empirically show that Policy Base provides Policy Based file secured deletion with a minimal trade-off of performance overhead. It provides insights of how to incorporate value-added security features into current data outsourcing applications.***

Key words: ***Policy Based f lie secured deletion, cloud storage, prototype, implementation***

Introduction

Cloud storage e.g., Amazon S3, My Asia Cloud offers an abstraction of infinite storage space for clients to host data, in a pay-as-you-go manner .For example, Smug Mug, a photo sharing website, chose to host terabytes of photos on Amazon S3 in 2009 and saved about 500K US dollars on storage devices . Thus, instead of self-maintaining data centers, enterprises can now outsource the storage of a bulk amount of digitized content to those third-party cloud storage providers so as to save the financial overhead in data management. Apart from enterprises, individuals can also benefit from cloud storage as a result of the advent of mobile devices e.g., smart phones, laptops. Given that mobile devices have limited storage space in general, individuals can move audio/video files to the cloud and make effective use of space in their mobile devices. However, privacy and integrity concerns become relevant as we now count on third parities to host possibly sensitive data. To protect outsourced data, a straightforward approach is to apply cryptographic encryption onto sensitive data with a set of encryption keys, yet maintaining and protecting such encryption keys will create another security issue. One specific issue is that upon requests of deletion of files,

cloud storage providers may not completely remove all file copies e.g., cloud storage providers may make multiple file backup copies and distribute them over the cloud for reliability, and clients do not know the number or even the existence of these backup copies, and eventually have the data disclosed if the encryption keys are unexpectedly obtained, either by accidents or by malicious attacks. Therefore, we seek to achieve a major security goal called file secured deletion, meaning that files are reliably deleted and remain permanently unrecoverable and inaccessible by any party. The security concerns motivate us, as cloud clients, to develop a secure cloud storage system that provides file secured deletion. However, a key challenge of building such a system is that cloud storage infrastructures are externally owned and managed by third-party cloud providers, and hence the system should never assume any structural changes in protocol or hardware levels in cloud infrastructures. Thus, it is important to design a secure overlay cloud storage system that can work seamlessly atop existing cloud storage services. In this paper, we present Policy Base a secure overlay cloud storage system that ensures file secured deletion and works seamlessly atop today's cloud storage services. Policy Base decouples the management of encrypted data and encryption A motivating application of Policy Base is cloud-based backup systemse.g. Jungle Disk , Cumulus Policy Base can be viewed as a value-added security service that further enhances the security properties of the existing cloud-based backup systems.
a new policy based file secured deletion scheme that reliably deletes files with regard to revoked file access policies. In this context, we design the key management schemes for various file manipulation operations. Implement a working prototype of policy base atop Amazon S3 .implementation aims to illustrate that various applications can benefit from Policy base, such as cloud-based backup systems. policy base consists of a set of API interfaces that we can export, so that we can adapt policy base into different cloud target implementations. Files with regard to revoked file access policies.

2 Policy based File Secure Deletion

The policy-based file secured deletion, the major design building block of our POLICY BASE architecture. Our main focus is to deal with the cryptographic key operations that enable file secured deletion. We first review time-based file secured deletion. Then explain how it can be extended to policy-based file secured deletion.

2.1 Conditions

Time based file secured deletion, which is first introduced in, means that files can be securely deleted and remain permanently inaccessible after a predefined duration. The main idea is that a file is encrypted with a data key, and this data key is further encrypted with a control key that is maintained by separate key Administrator service known as Euhemerize In, the control key is time-based, meaning that it will be completely removed by the key Administrator when an expiration time is reached, where the expiration time is specified when the file is first declared. Without the control key, the data key and hence the data file remain encrypted and are deemed to be inaccessible. Thus, the main security property of file secured deletion is that even if a cloud provider does not remove expired file copies from its storage, those files remain encrypted and unrecoverable. Time-based file secured deletion is later prototyped in Vanish. Vanish divides a data key into multiple key shares, which are then stored in different nodes of a peer-to-peer network. Nodes remove the key shares that reside in their caches for 8 hours. If a file needs to remain accessible after 8 hours, then the file owner needs to update the key shares in node caches. However, both and target only the secured deletion upon time expiration, and do not consider a more fine-grained control of secured deletion with respect to different file access policies.

2.2 Policy-based Deletion

Here associate each file with a single atomic file access policy or policy for short, or more generally, a Boolean combination of atomic policies. Each atomic policy is associated with a control key, and all the control keys are maintained by the key Administrator. Similar to time-based deletion, the file content is encrypted with a data key, and the data key is further encrypted with the control keys corresponding to the policy combination. When a policy is revoked, the corresponding control key will be removed from the key Administrator. Thus, when the policy combination associated with a file is revoked and no longer holds the data key and hence the encrypted content of the file cannot be recovered with the control keys of the policies. In this case, we say the file is deleted. The main idea of policy-based deletion is to delete files that are associated with revoked policies. The definitions of policies vary depending on applications. Time-based deletion is a special case under our framework, and policies with other access right scan be defined. To motivate the use of policy-based deletion, let us consider a scenario where a company outsources its data to the cloud. We consider four practical cases where policy-based deletion will be useful:

2.2.1 Storing files for tenured employees.

For each employee, can define a user-based policy P: Alice is an employee, and associate this policy with all files of Alice. If Alice quits her job, then the key Administrator will expunge the control key of policy P. Thus, nobody including Alice can access the files associated with P on the cloud, and those files are said to be deleted**.**

2.2.3 Storing files for contract-based employees.

An employee may be affiliated with the company for only a fixed length of time. Then can form a combination of the user-based and time-based policies for employees' files. For example, for a contract-based employee Bob whose contract expires on 2013-01-01, we have two policies "P\: Bob is an employee" and "P_2: valid before 2013-01-01". Then all files of Bob are associated with the policy combination Pi A P_2. If either Pi or P_2 is revoked, then Bob's files are deleted..

2.2.4 Switching a cloud provider.

The company can define a customer-based policy P: a customer of cloud provider X and all files that are stored on cloud X are tied with policy P. If the company switches to a new cloud provider, then it can revoke policy P. Thus, all files on cloud X will be deleted. Policy-based deletion follows the similar notion of attribute-based encryption, in which data can be accessed only if a subset of attributes is satisfied. However, our work is different from ABE in two aspects. First, focus on how to delete data, while ABE focuses on how to access database on attributes. Second, because of the different design objectives, ABE gives users the decryption keys of the associated attributes, so that they can access files that satisfy the attributes. On the other hand, in policy-based deletion, we do not share with users any decryption keys of policies, which instead are all maintained in the key Administrator. Our focus is to appropriately remove keys in the key Administrator so as to guarantee file secured deletion, which is an important security property when we outsource data storage to the cloud. This guides us into a different design space in contrast with existing ABE approaches.

2.3 Participants in the System

This system is composed of three participants: the data owner, the key Administrator, and the storage cloud. They are described as follows.

2.3.1 Data owner.

The data owner is the entity that originates file data to be stored on the cloud. It may be a file system of a PC, a user-level program, a mobile device, or even in the form of a plug-in of a client application.

2.3..2 Key Administrator.

The key Administrator maintains the policy-based control keys that are used to encrypt data keys. It responds to the data owner's requests by performing encryption, decryption, renewal, and revocation to the control keys.

2.3.3 Storage cloud.

The storage cloud is maintained by a third-party cloud provider e.g., Amazon S3 and keeps the data on behalf of the data owner. emphasize that we do not require any protocol and implementation changes. on the storage cloud to support our system. Even a naive storage service that merely provides file upload/download operations will be suitable.

2.4 Hazard Models and Statements

Here main design goal is to provide secured deletion of files produced by the data owner. A file is deleted or permanently inaccessible if its policy is revoked and becomes obsolete. Here, assume that the control key associated with a revoked policy is reliably removed by the key Administrator. Thus, by secured deletion of files, we mean that any existing file copy that are associated with revoked policies will remain permanently encrypted and unrecoverable. The key Administrator can be deployed as a minimally trusted third-party service. By minimally trusted, we mean that the key Administrator reliably removes the control keys of revoked policies. However, it is possible that the key Administrator can be compromised. In this case, an attacker can recover the files that are associated with existing active policies. On the other hand, files that are associated with revoked policies still remain inaccessible, as the control keys are removed. Hence, file secured deletion is achieved. It is still important to improve the robustness of the key Administrator service to minimize its chance of being compromised. To achieve this, we can use a quorum of key Administrators , in which we create n key shares for a key, such that any k $< n$ of the key shares can be used to recover the key. While the quorum scheme increases the storage overhead of keys, this is justified as keys are of much smaller size than data files. Before accessing the active keys in the key Administrator, the data owner needs to present authentication credentials e.g., based on public key infrastructure certificates to the key Administrator to show that it satisfies the proper policies Policy Revocation for File Secured Deletion If a policy P_i is revoked, then the key Administrator completely removes the private key d_i and the secret prime numbers p_i and g_i. Thus, we cannot recover S_i forms?, and hence cannot recover K and the file F. We say that the file F, which is tied to policy Pi, is securely deleted. Note that the policy revocation operations do not involve interactions with the storage cloud.ng that the file is not securely deleted. It is important to note that it is a non-trivial task to enforce the condition. of policy renewal, as the old policy may be associated with other existing files. Here do not consider this issue and we pose it as future work. Suppose that we have enforced the condition of policy renewal. A straightforward approach of implementing policy renewal is to combine the file upload and download operations, but without retrieving the encrypted file from the cloud.

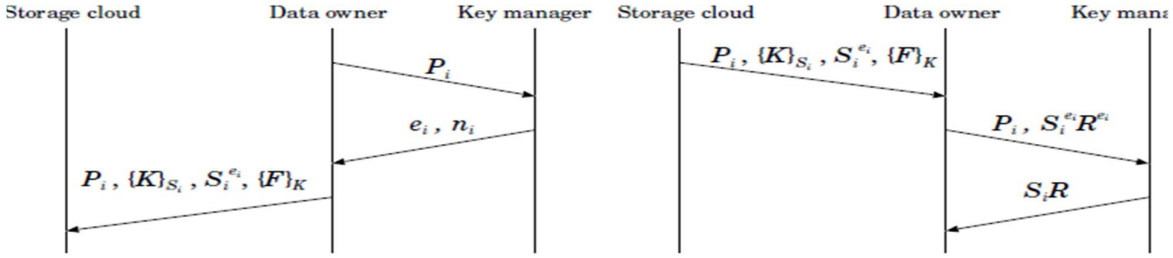
2.4.1The procedures can be summarized as follows:

(i) download all encrypted keys from the storage cloud,
(ii) send them to the key Administrator for decryption,
(iii) recover the data key,
(iv) re-encrypt the data key with the control keys of the new policies, and finally
(v) send the newly encrypted keys back to the cloud.

In some special cases, optimization can be made so as to save the operations of decrypting and re-encrypting the data key. Suppose that the Boolean com-bination structure of policies remain unchanged, but one of the atomic policies is changed Pj'. For example, when we extend the contract date of Bob we may need to update the particular time-based policy of Bob without changing other policies. In this case, the data owner simply sends theblinded version $S_i^{ei}R^{ei}$ to the key Administrator, which then returns S_iR. The data owner then recovers S_i. Now, the data owner

2.5 File Upload And Down Load

Here now introduce the basics of uploading/downloading files to/from the cloud storage. We first assume that each file is associated with a single policy, and then explain how a file is associated with multiple policies this design is based on blinded RSA , in which the data owner requests the key Administrator to decrypt a blinded version of the encrypted data key. If the associated policy is



return the blinded version of the original data key. The data owner can then recover the data key. In this way, the actual content of the data key remains confidential to the key Administrator as well as to any attacker that sniffs the communication between the data owner and the key Administrator. ere first summarize the major notation used throughout the paper. For each policy i, the key anger generates two secret large RSA prime numbers pi and qi and computes the product ni = piqi. The key Administrator then randomly chooses the RSA public-private control key pair (ei, di). The parameters (ni, ei) will be publicized, while di is securely stored in the key Administrator. On the other hand, when the data owner encrypts a file F, it randomly generates a data key K, and a secret key Si that corresponds to policy Pi. We let {m}k denote a message m encrypted with key k using symmetric-key encryption e.g., AES. Here let R be the blinded component when we use blinded RSA for the exchanges of cryptographic keys. Suppose that F is associated with policy Pi. Our goal here is to ensure that K, and hence F, are accessible only when policy Pi is satisfied. Note that the only present the operations on cryptographic keys, while the implementation subtleties, such as metadata, will be discussed in Section 3. Also, when The raise some number to exponents ei or di, it must be done over modulo ni. For brevity.

Integrity.

To protect the integrity of a file, the data owner needs to compute an HMAC on every encrypted file and stores the HMAC, together with the encrypted file, in the cloud storage. When a file is downloaded, the data owner will check whether the HMAC is valid before decrypting the file.Here assume that the data owner has a long-term private secret for the HMAC computational

rencrypts S_i into $S®^{3'}$ (mod n_i'),where (n_i', e_i') is the public key of policy P_i', and sends it to the cloud.

2.6 Procedure Restoration.

The encrypted data key K remains intact. Figure 3 illustrates this special case of policy renewal. if a user wants to extend the expiration time of a file, then the user can update the old policy that specifies an earlier expiration time to the new policy that specifies a later expiration time. However, to guarantee file assured deletion, policy renewal can be performed only when the following condition holds: the old policy will always be revoked first before the new policy is revoked. The reason is that after policy renewal, there will be two versions of a file: ones protected with the old policy, and one is protected with the new policy. If the new policy is revoked first, then the file version that is protected with the old policy may still be accessible when the control keys of the old policy are compromised, meaning that the file is not assuredly deleted. It is important to note that it is a non-trivial task to enforce the condition of policy renewal, as the old policy may be associated with other existing files. Here do not consider this issue and we pose it as future work. Suppose that we have enforced the condition of policy renewal. A straight forward approach of implementing policy renewal is to combine the file upload and download operations, but without retrieving the encrypted file from the cloud. The procedures can be summarized as follows: (i) download all encrypted keys from the storage cloud, (ii) send them to the key Administrator for decryption, (iii)recover the data key, (iv) re-encrypt the data key with the control keys of the new policies, and finally (v) send the newly encrypted keys back to the cloud. In some special cases, optimization can be made so as to save the operations of decrypting and re-encrypting the data key. Suppose that the Boolean combination structure of policies remain unchanged, but one of the atomic policies is changed Pi′ . For example, when we extend the contract date of Bob , we may need to update the particular time-based policy of Bob without changing other policies. In this case, the data owner simply sends the blinded version Soiree to the key Administrator, which then returns SiR. The dataowner then recovers Si. Now, the data owner re-encrypts Si into Sei′i (mod ni′),where (ni′ , ei′) is the public key of policy Pi′ , and sends it to the cloud.

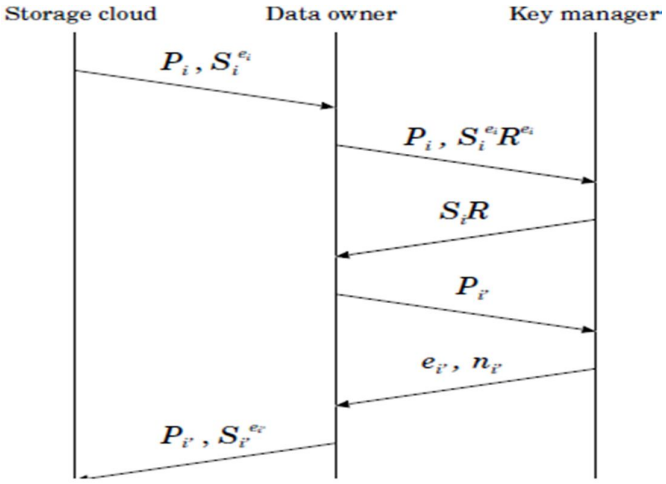


Fig. 3: Policy renewal.

3 The Policy Base Architecture

The implement a working prototype of POLICY BASE using C on Linux, and the use the Open SSL library for the cryptographic operations. In addition, The use Amazon S3 as our storage cloud. This is to address the implementation issues of our POLICY BASE architecture, based on our experience in prototyping policy base**.**The main goal is to show the practicality of policy base when it is deployed with today'scloud storage services. the policy base architecture. In the following, The define the meta-data of policy base attached to individual files. Then describe how The implement the data owner and the key Administrator, and how the data owner interacts with the storage cloud.

3.1 Illustration of Metadata

For each file protected by policy base we include the metadata that describes the policies associated with the file as well as a set of encrypted keys. In policy base there are two types of metadata: file metadata and policy metadata.

3.1.1File metadata.

The file metadata mainly contains two pieces of information:file size and HMAC. We hash the encrypted file with HMAC-SHA1 for integritychecking. The file metadata is of fixed size with 8 bytes of file size and 20 bytes**.**of HMAC and attached at the beginning of the encrypted file. Both the file

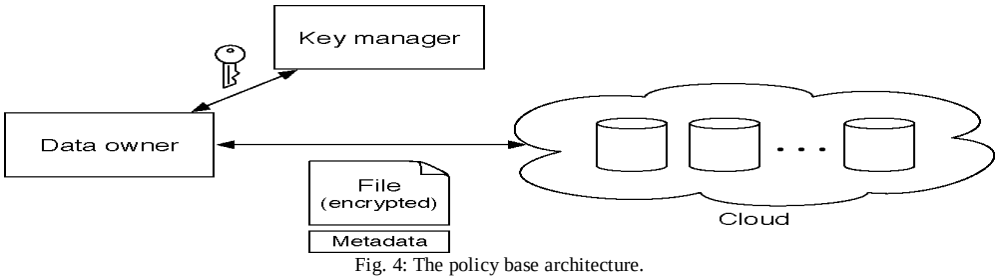


Fig. 4: The policy base architecture.

metadata and the encrypted data file will then be treated as a single file to be uploaded to the storage cloud.

3.1.2 Policy metadata.

The policy metadata includes the specification of the Boolean combination of policies and the corresponding encrypted cryptographic keys. Here, we assume that each single policy is specified by a unique 4-byteinteger identifier. To represent a Boolean combination of policies, we express it in disjunctive canonical form, i.e., the disjunction or of conjunctive policies,and use the characters '*' and '+' to denote the and or operators. Then the upload the policy metadata as a separate file to the storage cloud. This enables us to renew policies directly on the policy metadata without retrieving the entire file from the storage cloud. In this implementation, individual files have their own policy metadata,although we allow multiple files to be associated with the same policy which is the expected behavior of policy base In other words, for two data files that are under the same policy, they will have different policy metadata files that specify different data keys, and the data keys are protected by the control key of the same policy. we discuss how we may associate the same policy metadata file with multiple data files so as to reduce the metadata overhead.

3.2 Data Owner and Storage Cloud

In this implementation of the data owner uses the following four function calls toenable end users to interact with the storage cloud:

3.2.1Upload(file, policy).

The data owner encrypts the input file using thespecified policy or a Boolean combination of policies. It then sends the encrypted file and the metadata onto the cloud. In this implementation, the file is encrypted using the 128-bit AES algorithm with the cipher block chaining (CBC) mode, yet we can adopt a different symmetric-key encryption algorithm
depending on applications.

3.2.2Policy File Download

The data owner retrieves the file and the policy metadata from the cloud, checks the integrity of the file, and decrypts the file.

3.2.3Policy File Revoke

The data owner tells the key Administrator to permanently re-voke the specified policy. All files associated with the policy will be securely deleted.

3.2.4 Renew(file, new policy).

The data owner first fetches the policy metadata for the given file from the cloud. It then updates the policy metadata with the new policy. Finally, it sends the policy metadata back to the cloud. The above function calls can be exported as library APIs that can be embedded into different implementations of the data owner. In our current prototype, we implement the data owner as a user-level program that can access files under a working directory of a desktop PC.The above exported interfaces wrap the third-party APIs for interacting with the storage cloud. As an example, the LibAWS++ use a C++ library for interfacing with Amazon S3. We note that the LibAWS++ library uses HTTP to communicate with the cloud, and it does not provide any security protection on the data being transferred. To interact with different cloud storage services, The use different third-party APIs, provided that the APIs support the basic file upload/download operations.

3.3 Key Administrator

key Administrator that supports the following four basic functions:
Creating a policy. The key Administrator creates a new policy and returns the corresponding public control key. Retrieving the public control key of a policy. If the policy is accessible, then the key Administrator returns the public control key. Otherwise, it returns an error. Decrypting a key with respect to a policy. If the policy is accessible, then the key Administrator decrypts the blinded key. Otherwise, it returns an error. Revoking a policy. The key Administrator revokes the policy and removes the corresponding keys.

The basic functionalities of the key Administrator so that it can perform the required operations on the cryptographic keys. In particular, all the policy control keys are built upon 1024-bit blinded RSA& MIDP . To make the key Administrator more robust, we can extend the key Administrator to a quorum of key Administrators as stated in , and implement a PKI-based certification of system for policy checking .

4 .Examine

A prototype of policy base atop Microsoft S3 I-cloud, and evaluate the empirical performance of policy base. It is crucial that policy base does not introduce substantial performance overhead that will lead to a big increase in data management costs.

4.1 Time Presentation of Policy Base

The time performance of policy base using our prototype. In order to identify the time overhead of policy base we divide the running time of each measurement into three components:

data transmission time, the data uploading/downloading time between the data owner and the storage cloud. We further divide it into two components: the file component, which measures the transmission time for the file body and the file metadata, and the policy component, which measures the transmission time for the policy metadata. The upload/download these two components as two separate copies to/from the storage cloud.AES and HMAC time, the total computational time used for performing AES and HMAC on the file, key management time, the time for the data owner to coordinate with the key Administrator on operating the cryptographic keys. Experiment.

File size	Total time	Data transmission				AES+HMAC		Key management	
		File	(%)	Policy	(%)	Time	(%)	Time	(%)
1KB	1.260s	0.724s	57.4%	0.537s	42.6%	0.000s	0.0%	0.000s	0.0%
10KB	1.552s	1.020s	65.7%	0.532s	34.3%	0.001s	0.0%	0.000s	0.0%
100KB	2.452s	1.903s	77.6%	0.546s	22.3%	0.002s	0.1%	0.001s	0.0%
1MB	4.194s	3.646s	86.9%	0.527s	12.6%	0.022s	0.5%	0.000s	0.0%
10MB	16.275s	15.463s	95.0%	0.595s	3.7%	0.218s	1.3%	0.000s	0.0%

Policy Base: Secure Overlay Cloud Storage with File Assured Deletion

(a) Upload

File size	Total time	Data transmission				AES+HMAC		Key management	
		File	(%)	Policy	(%)	Time	(%)	Time	(%)
1KB	0.843s	0.485s	57.5%	0.355s	42.1%	0.000s	0.0%	0.003s	0.4%
10KB	0.912s	0.615s	67.4%	0.294s	32.2%	0.000s	0.0%	0.003s	0.3%
100KB	1.968s	1.682s	85.5%	0.282s	14.3%	0.002s	0.1%	0.002s	0.1%
1MB	4.696s	4.360s	92.8%	0.317s	6.7%	0.017s	0.4%	0.002s	0.1%
10MB	33.746s	33.182s	98.3%	0.395s	1.2%	0.166s	0.5%	0.002s	0.0%

(b) Download

Table 1: Experiment 1 Performance of upload/download operations.

4.2Performance of Multiple Policies

The performance of POLICY BASE when multiple policies are associated th a file . Here, we focus on the file upload operation, and fix the file size at1MB. We look at two specific combinations of policies, one on the conjunctive case and one on the disjunctive case. a shows different components of time for different numbers of conjunctive policies, and Table 3b shows the case for disjunctive policies. A key observation is that the AES and HMAC and the key management time remain vey low.

5. Instruction Erudite

The performance of policy base in terms of the overheads of time, space utilization, and data transfer. It is important to note that the performance results depend on the deployment environment. For instance, if the data owner and the key Administrator all reside in the United States as Amazon S3, then the transmission times for files and metadata will significantly reduce; or if the policy metadata contains more descriptive information, the overhead will increase. Nevertheless, we emphasize that our experiments can show the feasibility of policy base in

providing an additional level of security protection for today's cloud storage. The performance overhead of policy base becomes less significant when the size of the actual file content increases on the order of megabytes or even bigger.

5.1 Adding an Additional Layer of Encryption.

The current design of policy base if the key Administrator colludes with the storage cloud, then the storage cloud can decrypt the files of data owner. To prevent this from happening, one solution is to add an additional layer of encryption in the data owner. The idea is that the data owner first encrypts a file with a long-term secret key, and then encrypts the encrypted file with the data key. In this way, even if the key Administrator colludes with the storage cloud, the files of the data owner remain encrypted.

5.2 Multiple Files with the Same Policy Metadata.

The operations of policy base are on a per-file basis, such that each data file has one corresponding policy metadata file. To reduce the metadata overhead of policy base we can associate a batch of multiple data files under the same directory with the same policy metadata and the same set of cryptographic keys including the data key and the control keys of policies. The advantage of the batch-based approach is that the single policy metadata for multiple data files. Thus, if the data files are of small size, then the batch-based approach can reduce the storage overhead due to the policy metadata. It is possible to add a new data file into the batch of files that are currently associated with the same policy metadata. To achieve this, the data owner first downloads the policy metadata from the storage cloud and recovers the data key. Then it uses the same data key to encrypt the new file. Note that the content of the policy metadata remains unchanged. Also, the data key can be cached in the data owner's volatile storage so as to include new files into the batch later. Space while providing essential encryption on outsourced data. The above systems mainly put the protocol functionalities on the client side, and the cloud storage providers merely provide the storage space. The concept of attributed-based encryption is first introduced, in which attributes are associated with encrypted data. Goyal et al. Extend the idea to key policy ABE, in which attributes are associated with private keys, and encrypted data can be decrypted only when a threshold of attributes are satisfied. Pirate et al. Implement ABE and conduct empirical studies. Nair et al. Consider a similar idea of ABE, and they seek to enforce a fine grained access control of files based on identity-based public key cryptography. Perlman et al. Also address the Boolean combinations of policies, but they focus on digital rights management rather than file secured deletion and their operations of cryptographic keys are different from our work because of the different frameworks., ABE and our work have different design objectives and hence different key management mechanisms.

6 .Associated Hustle

Security solutions that are compatible with existing public cloud storage serviceshave been proposed. a cryptographic file system that provides privacy and integrity guarantees for outsourced data using a universal hash based MAC tree. They prototype a system that can interact with an unfrosted storage server via a modified file system. Jungle Disk and Cumulus are proposed to protect the privacy of outsourced data, and their implementation use Amazon S3 as the storage backend. Specifically, umulus focuses on making effective use of storage space while providing essential encryption on outsourced data. The above systems mainly put the protocol functionalities on the client side, and the cloud storage providers merely provide the storage space.The concept of attributed-based encryption is first introduced in ,in which attributes are associated with encrypted data. Goyal et al. extend the idea to key-policy ABE, in which attributes are associated with private keys, and encrypted data can be decrypted only when a threshold of attributes are satisfied. Pirretti et al. implement ABE and conduct empirical studies.Nair et al. consider a similar idea of ABE, and they seek to enforce a finegrained access control of files based on identity-based public key cryptography. Perlman et al. also address the Boolean combinations of policies, but they focus on digital rights management rather than file assured deletion and their operations of cryptographic keys are different from our work because of thedifferent frameworks. As argued, ABE and our work have different design objectives and hence different key management mechanisms**.**

7 Conclusions

A cloud storage system called policy base which aims to provide secured deletion for files that are hosted by today's cloud storage services. the design of policy-based file secured deletion, in which files are securely deleted and made unrecoverable by anyone when their associated file access policies are revoked. The essential operations on cryptographic keys so as to achieve policy-based file secured deletion. Implement a prototype of policy base to demonstrate its practicality, and empirically study its performance overhead.

Authors:

1. M. Geetha Scholar, Department of Computer Science and Engineering, TKR College of Engineering and Technology Hyderabad, A.P-500 097, India

2. V. Ravi Kumar, Associate Professor, Department of Computer Science and Engineering, TKR College of Engineering and Technology

References

1. Microsoft.SmugMug Case Study: Amazon Web Services. ***http://aws.amazon.com/*** solutions/case-studies/smugmug/**,** 2006.
2. Amazon Simple Storage Service (Amazon S3). ***http://aws.amazon.com/s3/.***
3. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Kaminski, G. Lee,

D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report ***UCB/EECS-2009-28, EECS Department, University of California***" Berkeley, Feb 2009.

4. G. Ateniese, R. D. Pietro, L. V. Mancini, and G. ***"Tsudik. Scalable and Efficient Provable Data Possession"***. In Proc. of SecureComm, 2008.

5. R. Geambasu, T. Kohno, A. Levy, and H. M. Levy. Vanish***: Increasing Data Privacy with Self-Destructing Data. In* Proc"**. of USENIX Security Symposium, Aug 2009.

6. V. Goyal, O. Pandey, A. Sahai, and B. Waters***. Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data"***. In Proc. of ACM CCS, 2006.

FADE: Secure Overlay Cloud Storage with File Assured Deletion
3
63

