Secure and Efficient Access to Outsourced Data

Weichao Wang
Department of SIS
UNC Charlotte
Charlotte, NC 28223
weichaowang@uncc.edu

Zhiwei Li
CS Department
UNC Charlotte
Charlotte, NC 28223
zli19@uncc.edu

Rodney Owens
Department of SIS
UNC Charlotte
Charlotte, NC 28223
rvowens@uncc.edu

Bharat Bhargava
CS Department
Purdue University
W. Lafayette, IN 47906
bb@cs.purdue.edu

ABSTRACT

Providing secure and efficient access to large scale outsourced data is an important component of cloud computing. In this paper, we propose a mechanism to solve this problem in owner-write-users-read applications. We propose to encrypt every data block with a different key so that flexible cryptography-based access control can be achieved. Through the adoption of key derivation methods, the owner needs to maintain only a few secrets. Analysis shows that the key derivation procedure using hash functions will introduce very limited computation overhead. We propose to use over-encryption and/or lazy revocation to prevent revoked users from getting access to updated data blocks. We design mechanisms to handle both updates to outsourced data and changes in user access rights. We investigate the overhead and safety of the proposed approach, and study mechanisms to improve data access efficiency.

Categories and Subject Descriptors

E.2 [**Data Storage Representations**]; H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*; H.3.2 [**Information Storage and Retrieval**]: Information Storage; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Data sharing*

General Terms

Security, Management

Keywords

Secure Data Access, Outsourced Data, Dynamic Environment

CCSW'09, November 13, 2009, Chicago, Illinois, USA.

1. INTRODUCTION

Since the daily operations of modern corporations heavily depend on their information processing capabilities, the costs and overhead to manage their computation resources start to pose serious challenges to these companies. To free the companies and their personnel from the burden of IT services, the concept of cloud computing has been proposed. In this new environment, a client may choose to outsource its data storage, information processing, or even the whole information infrastructure to a service provider. These new services allow companies to focus more on their core business and leave the IT operations to professionals. A large number of services on infrastructure, platform, and software have been developed and provided by various parties [24].

While the concept of cloud computing provides a new method for information processing, the security problems must be properly solved before these services can be widely deployed. Since many service providers are untrusted, the confidentiality, integrity, and privacy of the clients' information must be protected by some mechanisms. The Federal CIO Vivek Kundra recently emphasizes that data security is still a top concern about cloud computing [30].

Among various services of cloud computing, enabling secure access to outsourced data lays a solid foundation for information management and other operations. However, more research efforts are needed to achieve flexible access control to large-scale dynamic data. For example, using asymmetric encryption to protect data or metadata [14] will impact the adoption of the outsourcing platform by devices with limited computational power (e.g. mobile devices). At the same time, user-group-based data encryption may lead to a complicated access hierarchy after a series of grant and revocation operations [13].

In this paper, we focus on the data outsourcing scenario investigated in [9, 10, 12, 13]. In this environment, the data can be updated only by the original owner. At the same time, end users with different access rights need to read the information in an efficient and secure way. Both data and user dynamics must be properly handled to preserve the performance and safety of the outsourced storage system. Before presenting the details of the proposed approach, we use an example to illustrate the potential applications.

The world's largest collider accelerator LHC (Large Hadron Collider) can generate about 10 PB (Peta-Bytes, 10^{15} bytes)

data each year [27]. The data can be stored on a third party server and the European Organization for Nuclear Research (CERN) may publish new data, update existing records, and delete expired information. The data can be accessed by scientists in different countries. Since the scientists may have different security clearance levels, encryption based access control will be adopted. At the same time, methods must be designed to support dynamic changes of the access rights of end users.

Enforcing data security in this scenario puts new challenges to researchers. First, since the size of the outsourced data could be huge, we want the server to store only one copy of each data block (the data should be encrypted). Second, since a popular storage outsourcing pricing model is payper-use (e.g. Amazon S3), we want to reduce the number of operations on the storage server except for information access and updates. Specifically, we want to avoid data reencryption caused by changes of user access rights. Last but not least, we want to provide fine-grained access control to the end users.

In this paper, we propose to develop a new approach that integrates several advanced techniques to solve these problems. First, we encrypt every data block with a different symmetric key and adopt the key derivation method [3, 10] to reduce the number of secrets that the data owner and end users need to maintain. Different from [10], we do not organize users into groups based on their access rights. This method, although leads to more data encryption keys, will simplify operations during user access right changes. Second, we adopt over-encryption by the server [13] to achieve data isolation among end users even when they have the same access rights. For the servers that refuse to conduct over-encryption, we propose to use lazy revocation [17] to prevent revoked users from getting access to updated data. Finally, we present detailed methods to handle dynamics in both user access rights and outsourced data. To summarize, the contributions of the research include:

- The proposed approach provides fine grained access control to outsourced data with flexible and efficient management. The data owner needs to maintain only a few secrets for key derivation. It does not need to access the storage server except for data updates.
- We propose comprehensive mechanisms to handle dynamics in user access rights and updates to outsourced data. We study mechanisms to further improve the efficiency and safety of the proposed approach.

The remainder of the paper is organized as follows. Section 2 discusses the related work. We focus on securing untrusted storage and key management for outsourced data. In Section 3, we describe assumptions in the investigated scenario. We also discuss requirements to the proposed approach. Section 4 presents the details of the data access procedure. We introduce the construction of the key hierarchy and the key derivation procedures. In Section 5, we describe mechanisms to handle dynamics in outsourced data blocks and user access rights. Section 6 investigates the overhead of the proposed approach for data retrieval from scientific databases. Section 7 studies the efficiency, scalability, and safety of the proposed approach. Finally, Section 8 discusses future extensions and concludes the paper.

2. RELATED WORK

Although the official name of 'cloud computing' is proposed in recent years, the concept of treating data, storage, software, platform, and even infrastructure as a service has been investigated for a long time. To provide computation as a public utility as water and electricity, new challenges arise for the confidentiality, privacy, and integrity of the information and resources in the system. Although tens of research papers have been published on related topics, security research for cloud computing is still in its early stage. Therefore, below we first review the expected properties of data security in cloud computing and map them to the investigated scenario. We will then discuss the achievements in two research directions: secure remote untrusted storage and key management for access hierarchies, from which our proposed approach benefits.

Requirements of data security in cloud computing

Different from many fields in which a big gap exists between academic research and industry applications, cloud computing has attracted attentions from both sides since the very beginning. For example, secure data storage and management is an important component of the security guidance recently published by Cloud Security Alliance [8], the membership of which includes the leading corporations in cloud computing such as Sun, eBay, Visa, and McAfee. In the guidance, a secure data outsourcing service should be evaluated from at least the following aspects: (1) strong encryption and scalable key management; (2) user provisioning, de-provisioning, and information lifecycle management; and (3) system availability and performance.

For the first aspect, in this paper we propose to use data block level symmetric encryption. Since the proposed mechanism does not depend on any specific encryption algorithms, the end users can make their choices based on the requirements of the applications. The key derivation tree structure will allow data consumers to use a few keys to generate all secrets in need. For the second aspect, we provide detailed description on handling dynamics in user access rights and data blocks. For the last aspect, the performance and overhead analysis is conducted in Section 6.

Secure remote storage

Securing outsourced data for multi-user accesses can be achieved through encrypted file systems. However, the following analysis will show that existing approaches cannot satisfy the requirements of the example application discussed in Section 1. To allow users to get secure and efficient access to outsourced data files, both data and metadata must be properly protected. An early approach [23] presents the basic idea to encrypt the information and use hash values and digital signatures to guarantee information integrity. FARSITE [1] uses symmetric secrets to encrypt files and uses every reader's public key to encrypt the symmetric keys. In Plutus [17], both files and directory information are encrypted. It uses *sign* and *verify* keys respectively to determine whether or not a user can write or read a file. Since the key generation procedure depends on power-modular computation, the overhead is relatively heavy. SiRiUS [14] adopts a more complicated structure. Every data file has an encryption key and a sign key. At the same time, every user has a public/private key pair. The approach continuously signs the metadata tree to guarantee the freshness of the information and prevent rollback attacks. Every time a

secret is revoked, it will generate a new key and reencrypt corresponding data files. In SUNDR [19] the authors use hash trees and chains to prevent fork attacks and guarantee that users have the same view of files. They use update certificates to handle concurrent updates. The advantages and disadvantages of many approaches can be found in [18]. Recently, a data sharing platform for outsourced information using asymmetric encryption is proposed in [28]. We find that all these approaches adopt asymmetric encryption to protect data confidentiality. At the same time, encrypting information at the data block level will make the key management mechanism of secure file systems very cumbersome. Therefore, a new approach is needed to protect the safety of the outsourced data.

Key management for access hierarchies

To enable secure and efficient access to outsourced data, investigators have tried to integrate key derivation mechanisms [5, 6, 21, 32] with encryption-based data access control. Atallah *et al.* [3] propose a generic method that uses only hash functions to derive a descendant's key in a hierarchy. The method can handle updates locally and avoid propagation. Although the proposed key derivation tree structure can be viewed as a special case of access hierarchies, analysis in Section 7 will show that our method serves the studied application better. In [10], the authors divide users into groups based on their access rights to the data. The users are then organized into a hierarchy and further transformed to a tree structure to reduce the number of encryption keys. This method also helps to reduce the number of keys that are given to each user during the initiation procedure. In [13], data records are organized into groups based on the users that can access them. Since the data in the same group are encrypted by the same key, changes to user access rights will lead to updates in data organization. While a creative idea in this approach is to allow servers to conduct a second level encryption (over-encryption) to control access, repeated access revocation and grant may lead to a very complicated hierarchy structure for key management. In [12], the approach will store multiple copies of the same data record encrypted by different keys. At the same time, when access rights change, reencryption and data updates to the server must be conducted. These operations will cause extra overhead on the server and do not fit into our application scenarios. An experimental evaluation of these approaches can be found in [9].

3. PROBLEM DEFINITION

In this section, we briefly sketch out the application scenario under investigation and the system assumptions. We also describe the requirements to the proposed data outsourcing mechanism.

3.1 Application Scenario and Assumptions

As the example in Section 1 illustrates, the owner-write-users-read scenario is a popular case in the storage outsourcing applications. Figure 1 provides an abstract illustration of the scenario under investigation. The data owner stores a large amount of information on the service provider. Since the service provider is untrusted, the owner will encrypt the outsourced data before putting them on the server. Here we assume that the smallest information access unit is called a 'block'. This is an abstract concept and it may have different meanings in different systems. To provide fine-grained access control, the encryption will be conducted at the block level. Only the owner can make updates to the outsourced data. Here the operations include updates to data blocks, and deletion, insertion, and appending of blocks. We also assume that there exist pre-distributed secrets between data owner and service provider, and between data owner and end-users. The key distribution and update problem is beyond the scope of this paper and we refer interested readers to existing approaches such as [4].

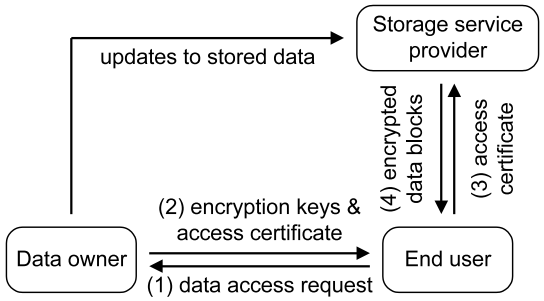

Figure 1: Illustration of the application scenario.

The outsourced data can be accessed by many different end users that are distributed all over the network. Since the end users may use devices with weak processing capabilities such as PDAs, we want to avoid computationally expensive operations such as asymmetric encryption of data blocks. At the same time, we want to reduce the amount of information that is stored on the end users. The access rights of the end users are different and they may change (grant and revocation) as time proceeds. Therefore, right keys must be provided to the end users to control their access.

We assume that the service provider adopts a curious but not malicious model. That means, the provider will not intentionally send wrong data blocks to an end user but it will try to get access to the plaintext of the stored information. To preserve confidentiality of the outsourced data, the owner may ask the service provider to conduct a second level encryption (over-encryption) [13] before the data is sent to the end users. For providers that refuse to offer this service, we adopt the lazy revocation method [17] to reduce information leakage through eavesdropping.

With the introduction to the roles of the data owner, service provider and end users, we can describe the data access procedure as follows. An end user will send a data access request to the owner. The owner will refer to its access control matrix and send back corresponding encryption keys through the secure channel between them. At the same time, the owner will send back a data access certificate to the end user. The user will then present the certificate to the service provider. The provider will verify it and send the corresponding encrypted data blocks to the end user. We assume that the end user has information to verify the integrity of the received data [15, 31]. In this way, end users directly communicate with the service provider to get the data blocks and the owner will not become the data transfer bottleneck.

3.2 System Requirements

In this subsection, we describe the requirements on efficiency and security to the proposed approach. First, since the outsourced data could be huge and the service provider may charge the owner based on used space, we want to

maintain only one encrypted copy for each data block on the outsourced storage. This will put new challenges to the key management mechanism. Second, in addition to providing the storage space, the service provider may or may not offer the service of over-encryption [13] when it sends the data blocks to end users. The proposed mechanism must properly handle both cases to preserve data confidentiality. Third, since the service provider may have a pay-per-use pricing policy, the data owner should reduce the number of information accesses to the service provider when they are not caused by updates to data blocks. That means we want to avoid data block reencryption when the access rights of end users change. Last but not least, we want to reduce the storage, communication, and computation overhead on the data owner and end users to promote the wide adoption of the proposed approach. Later discussion will show that some of the requirements conflict with each other and the designed approach will try to achieve a tradeoff.

4. SECURE AND EFFICIENT DATA ACCESS

In this section, we present the details of the proposed approach. We first introduce key-derivation-based data block encryption. We will then describe the data access procedure. Mechanisms to reduce the overhead on the data owner and prevent information access from revoked users will also be discussed.

4.1 Determining Keys for Data Encryption

As we introduce in Section 3, the smallest information access units are data blocks. Therefore, to provide fine-grained access control, we propose to encrypt every data block with a different secret. Here we do not assume the adoption of any specific symmetric encryption algorithm and leave the choice to the system when it is deployed. However, an efficient mechanism must be designed to allow data owner and end users to manage the encryption keys. In the worst case, if the outsourced data contain n blocks $\{D_1, D_2, \cdots, D_n\}$ and each block is encrypted with a randomly generated secret k_i (i=1 *to* n), the storage overhead on the owner will be linear to n. At the same time, when an end user needs to access l data blocks, the communication overhead between the owner and the user for key distribution will also be linear to l. This overhead can be overwhelming for many end users when we consider that the outsourced data can easily contain millions of blocks. Therefore, a more efficient key management method must be designed.

To solve this problem, we propose to adopt the key derivation method [3]. The basic idea is to generate the data block encryption keys through a hierarchy. Every key in the hierarchy can be derived by combining its parent node and some public information. Since the derivation procedure uses a one-way function, we cannot calculate the secret keys of the parent node and sibling nodes. In this way, the data owner will need to maintain only the root nodes of the hierarchies. During the key distribution procedure, the owner can send the secrets in the hierarchy to end users based on their access rights. The end user will derive the leaf nodes in the hierarchy to decrypt the data blocks. The cost of this approach is the calculation of one-way functions during key derivation. Since previous experiments [29] show that even mobile devices can accomplish the calculation very efficiently, in this paper we propose to trade computation for storage and communication overhead.

While there are different choices of the organization of key hierarchies and key derivation functions, below we present an approach using a binary tree structure and hash functions. Without losing generality, we assume that the outsourced data contain n blocks and $2^{p-1} \leq n < 2^p$. Therefore, we can build a binary tree with height p as follows. The data owner will choose a root secret $k_{0,1}$. Here the first index of the key represents its level in the hierarchy, and the second index represents its sequence in the level. For example, for level x in the hierarchy, the sequence numbers are from 1 to 2^x. In this way, for node $k_{i,j}$ in the hierarchy, its parent is $k_{(i-1),(\lceil j/2 \rceil)}$ (when $i \neq 0$), and its children are $k_{(i+1),(2*j-1)}$ and $k_{(i+1),(2*j)}$ (when $k_{i,j}$ is not a leaf node). The keys in level p will be used to encrypt the data blocks. The hierarchy is illustrated in Figure 2.

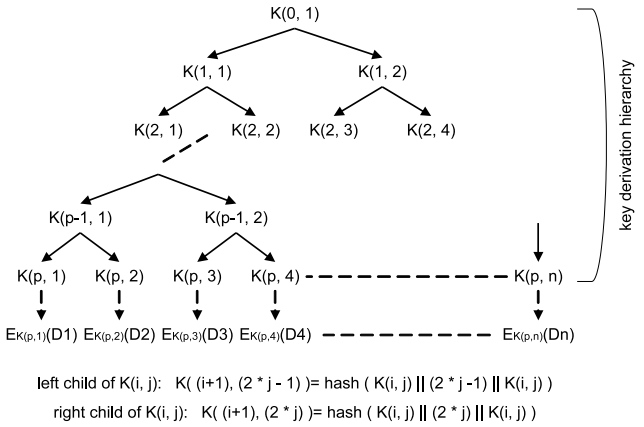

Figure 2: Key derivation hierarchy.

Now let us look at the key derivation procedure. The data owner chooses a public hash function $h()$. For any node $k_{i,j}$ in the hierarchy, its left child can be calculated as $k_{(i+1),(2*j-1)} = h(k_{i,j}||(2*j-1)||k_{i,j})$. Here we 'sandwich' the sequence number of the child node with the parent's key and then apply the hash function. We can calculate the right child of $k_{i,j}$ in a similar way. Through repeatedly applying this function, a node can calculate the secrets of all of its descendants. When we reach level p of the hierarchy, the hash results can be used as keys to encrypt the data blocks.

The safety of the approach comes from the one-way property of the hash function. While a node can easily derive its descendants in the hierarchy by applying the public hash function, it has to reverse the function to get the secrets of its siblings or ancestors.

For some applications, the hash results are not good enough to be used as encryption keys. For these cases, a more complicated method can be adopted. First, the owner will generate n encryption keys s_1 to s_n for the data blocks. When it outsources the data, for each block D_i it will store both $y_i = s_i - k_{p,i}$ and $E_{s_i}(D_i)$ on the service provider. During information access, an end user will use the hash hierarchy to calculate $k_{p,i}$ and recover s_i by combining this value with the stored meta-data. The cost of this method, however, is more storage and communication overhead during information access.

The system can adjust the parameters of the hierarchy to balance the storage and computation overhead during information access. For example, if the owner agrees to store multiple root keys, we can generate several independent key derivation hierarchies and each will be used to encrypt a part

of the outsourced data. As another example, by establishing an N-ary tree ($N > 2$) we can reduce the height of the hierarchy. However, this method is at the cost of reduction in key management flexibility.

We need to pay attention to several issues when we establish the key hierarchy and distribute the secrets during information access. First, when we choose the height of the hierarchy, we need to leave some room for the insertion and appending operations to the outsourced data. The details of such operations will be discussed in the next section. Second, we should not disclose encryption keys of the blocks that are temporarily missing from the outsourced data. For example, when $n = 7$ and the end user can access blocks D_5, D_6 and D_7, the owner should send $k_{2,3}$ and $k_{3,7}$ to the user instead of $k_{1,2}$. In this way, later when we append D_8 to the outsourced data and the user cannot access it, we do not need to revoke the secret.

We notice that the end users' access rights have a direct impact on the communication overhead of the proposed approach. For example, if one secret in the hierarchy can be used to derive all encryption keys of the data blocks that a user needs to access but not any other keys, the owner needs to send only this secret to the end user. On the other hand, if the end user can only access all data blocks with an odd index number, we will not be able to locate one or a few keys in the hierarchy to satisfy this read request. Under this condition, the number of returned keys will be linear to the number of requested blocks.

Two methods can be used to solve this problem and improve the efficiency of the owner. First, we can use the method described in [10]. The basic idea is to organize data blocks with similar access patterns into groups and give them sequential index numbers when they are outsourced. In this way, the end users have a higher probability to access data blocks with sequential index numbers and the owner can locate a few keys in the hierarchy to satisfy a read request. The second method is to construct multiple hierarchies over the data blocks. The details will be discussed in Section 7.2.

4.2 Data Access Procedure

In this part we describe the data access procedure in detail. To prevent revoked users from getting access to outsourced data through eavesdropping, we assume that the service provider will conduct over-encryption [13] when it sends data blocks to end users. To conduct this operation, we assume that the service provider and end users share a pseudo random bit sequence generator $P()$ [7, 20]. Given a seed *seed*, $P()$ can generate a long sequence of pseudo random bits. The cases in which the service provider refuses to offer this operation will be discussed in the next section.

We use $\mathcal{O}$ to represent the data owner, $\mathcal{S}$ to represent the service provider, and $\mathcal{U}$ to represent the end user. We assume that $\mathcal{O}$ shares the pairwise keys k_{OU} and k_{OS} with $\mathcal{U}$ and $\mathcal{S}$ respectively. With these assumptions, the data access procedure works as follows.

1. $\mathcal{U}$ will send a data access request to $\mathcal{O}$.

$$\mathcal{U} \rightarrow \mathcal{O}: \quad \{\mathcal{U}, \mathcal{O}, E_{k_{OU}}(\mathcal{U}, \mathcal{O}, request\ index, data\ block\ indexes, MAC\ code)\}$$

Since only $\mathcal{U}$ and $\mathcal{O}$ know k_{OU}, $\mathcal{O}$ will be able to authenticate the sender. The *request index* will be increased by 1 every time $\mathcal{U}$ sends out a request and it is used by $\mathcal{O}$ to defend against replay attacks. The request contains the index numbers of the data blocks that $\mathcal{U}$ wants to access. The Message Authentication Code (MAC) will protect the integrity of the packet.

2. When $\mathcal{O}$ receives this message, it will authenticate the sender and verify the integrity and freshness of the request. It will then examine its access control matrix and make sure that $\mathcal{U}$ is authorized to read all blocks in the request. If the request passes this check, the owner will determine the smallest set of keys $\mathcal{K}'$ in the hierarchy such that (1) $\mathcal{K}'$ can derive the keys that are used to encrypt the requested data blocks; and (2) $\mathcal{U}$ is authorized to know all keys that can be derived from $\mathcal{K}'$. $\mathcal{K}'$ can be determined by a greedy algorithm and we ignore its details here.

The owner will then generate the reply to the end user.

$$\mathcal{O} \rightarrow \mathcal{U}: \quad \{\mathcal{O}, \mathcal{U}, E_{k_{OU}}(\mathcal{O}, \mathcal{U}, request\ index, ACM\ index, seed\ for\ P(), \mathcal{K}', cert\ for\ \mathcal{S}, MAC\ code)\}$$

Here the request index is used to uniquely label this reply. The ACM index is used by $\mathcal{O}$ to label the freshness of the Access Control Matrix (ACM). This index will be increased by 1 every time $\mathcal{O}$ changes some end user's access rights. The updated ACM index will be sent to $\mathcal{S}$ by $\mathcal{O}$ to prevent those revoked users from using old certificates to access data blocks. The seed is a random number to initiate $P()$ so that $\mathcal{U}$ can decrypt the over-encryption conducted by $\mathcal{S}$. $\mathcal{U}$ will use $\mathcal{K}'$ to derive the data block encryption keys. The *cert* in the packet is a certificate for the service provider and it has the following format:

$$\{E_{k_{OS}}(\mathcal{U}, request\ index, ACM\ index, seed, indexes\ of\ data\ blocks, MAC\ code)\}$$

3. The user $\mathcal{U}$ will send $\{\mathcal{U}, \mathcal{S}, request\ index, cert\}$ to the service provider. When $\mathcal{S}$ receives this packet, it can verify that the *cert* is generated by $\mathcal{O}$ since only they know the secret k_{OS}. $\mathcal{S}$ will make sure that the user name and request index in *cert* match the values in the packet. If the ACM index in *cert* is smaller than the value that $\mathcal{S}$ receives from $\mathcal{O}$, some changes to the access control matrix have happened and $\mathcal{S}$ will notify $\mathcal{U}$ to get a new *cert*. Otherwise, the service provider will retrieve the encrypted data blocks and conduct the over-encryption as follows. Using *seed* as the initial state of $P()$, the function will generate a long sequence of pseudo random bits. $\mathcal{S}$ will use this bit sequence as one-time pad and conduct the xor operation to the encrypted blocks. The computation results will then be sent to $\mathcal{U}$.

4. When $\mathcal{U}$ receives the data blocks, it will use *seed* to generate the pseudo random bit sequence and use $\mathcal{K}'$ to derive the encryption keys. It will then recover the data blocks.

This approach adopts two methods to reduce the overhead on the data owner. First, the *cert* that we provide to the end user does not contain a timestamp. Therefore, if the access control matrix has not been changed and the ACM index value has not been updated, a user can reuse previous *cert*s to access the data blocks stored on the service provider. Second, during the whole data access procedure, the owner only needs to use the root key(s) to determine $\mathcal{K}'$ by calculating a group of hash functions. Since this computation

can be accomplished very efficiently, the data owner will not become the bottleneck in the application.

This approach adopts two methods to prevent revoked users from getting access to the outsourced data. First, when an end user $\mathcal{U}$ loses access to some data blocks, the access control matrix at $\mathcal{O}$ will be updated. This will lead to the increase of the ACM index and the value will be sent to $\mathcal{S}$ through a secure channel. In this way, if $\mathcal{U}$ presents the old *cert* to $\mathcal{S}$, it will be rejected since the ACM index value is invalid. However, $\mathcal{U}$ can still get access to the data blocks by eavesdropping on the traffic between $\mathcal{S}$ and other end users if it has kept a copy of the key set $\mathcal{K}'$. To defend against such attacks, we ask the service provider to conduct over-encryption before sending out the data blocks. Since for every data request the *seed* is dynamically generated by $\mathcal{O}$ and never transmitted in plaintext, $\mathcal{U}$ will not be able to regenerate the bit sequence of other end users. Therefore, unless $\mathcal{U}$ keeps a copy of the data blocks from previous access, it will not be able to get the information.

5. HANDLING DYNAMICS IN SYSTEM

Thus far, we have considered only static data and fixed access rights of end users. Although some applications, such as digital libraries, have relatively stable data contents, many scenarios for outsourced data storage require the system to support data dynamics. For example, in the DoE case discussed in Section 1, scientists may conduct new experiments and need to add new information to the outsourced data. At the same time, they may find that some data have been miscalculated and several data blocks on $\mathcal{S}$ need to be updated.

The proposed approach also needs to support changes to access rights of end users. For example, when DoE is collaborating with researchers in country X, it will temporarily authorize them to read more data. When the collaboration is terminated, the access rights will be revoked. Below we show how to amend the basic scheme to handle dynamic operations in the proposed system. The revised approach still tries to satisfy the requirements described in Section 3.2.

5.1 Dynamics in User Access Rights

Dynamics in user access rights can be represented as different combinations of two primitive operations: access right grant and revocation.

Grant Access Right

When an end user $\mathcal{U}$ is authorized to read a data block D_i, the owner will change its access control matrix and increase the value of ACM index. The next time that $\mathcal{U}$ submits a data access request, the owner will recalculate the key set $\mathcal{K}'$ based on the new access rights. The service provider and the end user do not need to change to adapt to this update.

Revoke Access Right

Access right revocation is a more complicated event and we need to discuss two mechanisms based on whether or not the service provider conducts over-encryption during data block transmission.

As we describe in Section 4.2, over-encryption conducted by the service provider can defend against eavesdroppers even when they have the data block encryption keys. Under this condition the owner will update the access control matrix and increase the ACM index by 1. It will repeatedly send the new ACM index to the service provider until it receives a confirmation. At this time the revoked user can no longer use its old *cert*. The owner will also calculate a new key set $\mathcal{K}'$ when it receives the next data access request from the revoked user.

If the service provider refuses to conduct over-encryption, the eavesdropper will be able to get access to data blocks if it keeps a copy of the encryption keys. Since the system design criteria require the owner to reduce the number of accesses to outsourced storage, we propose to adopt the lazy revocation method proposed in [17]. In lazy revocation, we assume that it is acceptable for the revoked user to read unmodified data blocks. However, it must not be able to read updated blocks. Lazy revocation trades re-encryption and data access overhead for a degree of security. The details of the method are as follows.

When the access right to data block D_i of the user $\mathcal{U}$ is revoked, the access control matrix in $\mathcal{O}$ will be updated and the ACM index increased. At the same time, $\mathcal{O}$ will label this data block to show that some user's access right has been revoked since its last content update. Before D_i is updated for the next time, the owner will not change the block on the outsourced storage. Since the ACM index value has been changed, $\mathcal{U}$ can no longer use its old *cert* to access D_i. However, when another user gets encrypted D_i through the network, $\mathcal{U}$ can eavesdrop on the traffic. Since the service provider refuses to conduct over-encryption, the data will be transmitted in the same format whoever the reader is. Therefore, should $\mathcal{U}$ have kept a copy of the encryption key, it will get access to D_i. This result, however, is the same as $\mathcal{U}$ has kept a copy of D_i before its access right is revoked.

When the owner needs to change the data block from D_i to D_i', it will check the label and find that some user's access right has been revoked. Therefore, it cannot encrypt the updated data block with the current key. To solve this problem, the owner will encrypt a control block with the secret $k_{p,i}$ and put it at the slot for D_i. The control block will contain a pointer to another block in which the updated data is stored. It will also contain enough information for the owner to derive the new encryption key. In this way, when a user receives this control block from the service provider, it will submit it to the owner. The owner will derive the new key and send it back to the user. At the same time, a new *cert* will be generated so that the user can get the new block from the service provider. A revoked user will be able to get access to the control block. However, the owner will not send the new encryption key and the *cert* to it. Therefore, the revoked user cannot get access to the updated data. More details of this method will be introduced when we discuss dynamics in the outsourced data.

The adoption of lazy revocation also explains the reason that we want to encrypt every data block with a different key. If we divide data blocks into groups based on the users that can access them, we can encrypt data in the same group with a single key. In this case, whenever a user's access right is revoked, the data block group needs to be fragmented and many blocks need to be reencrypted. Larger overhead will be caused on the data owner and service provider. At the same time, the requirement to reduce accesses to outsourced storage by the data owner will also be violated.

5.2 Dynamics in Outsourced Data

The data owner may need to conduct various operations on data blocks (e.g. update, delete, insert, append). Below we describe mechanisms to handle these changes.

Block Deletion

When a data block D_i is deleted from the outsourced data, the owner will use a special control block to replace D_i. The special block will be encrypted by $k_{p,i}$ and stored at the original slot for D_i on the service provider. At the same time, the owner will label its access control matrix to show that the block no longer exists. The end users can still access this control block but they will not get any useful information from the contents.

Block Update

We assume that the owner needs to modify the i-th block of the outsourced data from D_i to D'_i. Since in Section 5.1 we require the owner to maintain a label to show that whether or not some user's access right to this block has been revoked since its last update, below we describe two methods based on the value of the label.

If no user's access right to this data block has been revoked since its last update, the owner can update its value in the current storage place. The owner will first locate the slot in which D_i is currently stored and derive its encryption key. It will then use the key to encrypt D'_i and write the new value to the storage place. The end users will not be impacted by this operation and they will automatically get the new data when they access the block.

If some user's access right to D_i has been revoked since its last update and the service provider refuses to conduct over-encryption during data transmission, we cannot encrypt the new block D'_i with the current key. On the contrary, we will encrypt a control block with $k_{p,i}$ and write it to the i-th block of the outsourced data. The control block will contain the following information: (1) a pointer to the data block in which D'_i is currently stored; (2) information used by the data owner to derive the encryption key of D'_i; (3) information used by the data owner to verify the integrity of the control block. The owner will also use the new secret to encrypt D'_i and write it to the corresponding place in S.

When a user needs to access D'_i, it will get the encrypted control block from the service provider and submit it to the owner. The owner will verify the authenticity and integrity of the control block and derive the current encryption key. It will then return the key with a *cert* to the user through a secure channel so that the user can access D'_i from $\mathcal{S}$. A revoked user can get the control block but it will not get the new encryption key and the *cert*.

While there are different ways to implement the block update method, below we describe one approach. We assume that the outsourced data contain n blocks and $n < 2^p$. We have constructed a p-level key hierarchy with the root $k_{0,1}$ to determine the data block encryption keys. Now the data owner will choose two secret keys $k'_{0,1}$ and k_{verify}. The former is used to generate another p-level key hierarchy, and the latter is used to verify the integrity of the control blocks.

We use $k'_{0,1}$ to generate another p-level key hierarchy and the key derivation procedure is exactly the same as previously described. The keys in the new hierarchy will be represented as $k'_{i,j}$ so that they can be distinguished from the old ones. The new hierarchy has the following special properties. First, the i-th node in level p of the new hierarchy will have the index number $2^p + i$. In this way, we can construct a one-to-one mapping among the nodes in the old hierarchy and those in the new one. Second, the encryption key of the i-th data block in the new hierarchy will be $h(k'_{p,i}||x||k'_{p,i})$. Here $k'_{p,i}$ is the corresponding secret in the new hierarchy and x represents the number of times that the data block has been updated.

With this information, the block update operation will be conducted as follows. When the owner needs to update D_i, it will use $k_{p,i}$ to encrypt the control block and store it in the i-th block of the outsourced data. The control block will contain: (1) $(2^p + i)$ which is the index of the block in which D'_i is stored; (2) x which is the number of times that D_i has been updated; and (3) $h(k_{verify}||(2^p + i)||x||k_{verify})$ which is used to protect the integrity of the control block. The owner will encrypt D'_i with $h(k'_{p,i}||x||k'_{p,i})$ and store the result in the block with the index number $(2^p + i)$. Figure 3 illustrates the hierarchies and the update operations.

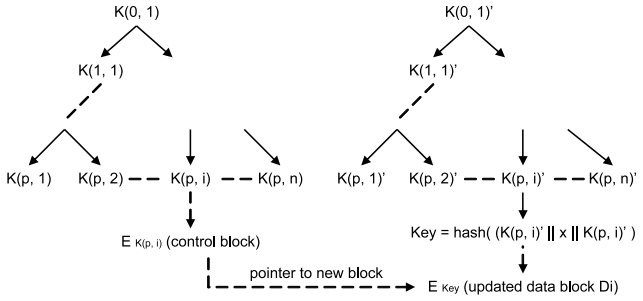

Figure 3: Handling updates to data blocks.

When a user $\mathcal{U}$ needs to access the updated data block D'_i, it will first get the encrypted control block from $\mathcal{S}$ and submit it to the data owner. The owner will use the secret k_{verify} to examine the integrity of the control block. It will then use $k'_{0,1}$ and x to derive the encryption key of D'_i. The owner will return the encryption key and a new *cert* to $\mathcal{U}$ through the secure communication channel between them. $\mathcal{U}$ will then get D'_i from the service provider.

This method has several properties. First, we store all meta-data in the control block on the service provider so that the data owner only needs to store two secrets $k'_{0,1}$ and k_{verify}. Second, since k_{verify} is known to only the owner, attackers cannot generate fake control blocks. Third, every time the data block D_i is updated, the value of x will be increased and the encryption key will be different. At the same time, the keys are safe since the owner never discloses $k'_{0,1}$ or $k'_{p,i}$.

The costs of this method include another group of hash calculation and a second round of communication between the owner and the user. Although the blocks in the new hierarchy have the index numbers from $(2^p + 1)$ to $(2^p + n)$, we have to clarify that we do not need to double the storage space on the service provider when only a few blocks are updated. Instead, we can store both the index numbers and the data block contents on $\mathcal{S}$.

Block Insertion and Appending

The data owner may need to generate new information and put it on the outsourced storage. Here we do not intentionally distinguish insertion from appending and follow the same procedure to handle the two operations. The data owner will locate an unused block index, derive the encryption key in the hierarchy using $k_{0,1}$, encrypt the data block, and store it on the service provider. One trick can be adopted to improve the efficiency of future data access operations. As we describe in Section 4.1, the data blocks that are always accessed together should be given sequential

block index numbers so that the owner can derive a smaller access key set $\mathcal{K}'$ for end users. Therefore, the owner can reserve some empty slots in the outsourced data. Later it can insert the new data blocks into these positions based on their access patterns.

6. ANALYSIS OF OVERHEAD: AN EXAMPLE

In this part, we follow the application description in Section 1 and analyze the computational, storage, and communication overhead of a data access operation. We assume that the size of the outsourced data is 10 PB and the data block size is 4 KB. Therefore, we have 2.5×10^{12} blocks. It is prohibitively expensive to store all encryption keys on the data owner. Using the method described in Section 4, we find that the height of the key hierarchy is $p = 42$. We assume that the user needs to retrieve 1 GB = 250,000 blocks of data from the server. Since the authors of [11, 26] find that scientific databases have very infrequent data update operations, we assume that 0.1% of data blocks have been updated and they have control blocks on the server. Previous research [2, 22] shows that the number of consecutive data blocks that are accessed in scientific applications ranges from 100 to 30,000. Therefore, in our analysis we assume that on average the data is accessed in chunks of 1,000 consecutive data blocks and the encryption keys of each chunk can be derived from one secret in the key hierarchy. We adopt 256-bit hash values and 64-bit block indexes in our system. We also assume that the owner, the end user, and the server all use computers with 1-GHz CPU.

Based on these assumptions, we can calculate the overhead of the operation. The computational overhead of the proposed approach comes from two aspects: key derivation using hash functions and over-encryption using a one-time key pad. When the data owner $\mathcal{O}$ receives the index numbers of the data blocks that $\mathcal{U}$ wants to access, it needs to derive the encryption keys based on $k_{0,1}$. Since the required data contains 250,000 / 1000 = 250 chunks and each chunk needs one key in the hierarchy, $\mathcal{O}$ needs to conduct $250*(42-log_2(1000)) \approx 8,000$ hash calculations. For the updated data blocks, $\mathcal{O}$ needs to conduct $250*43 = 10750$ hash calculations. Since the hash function needs about 20 machine cycles to process one byte [25], we need 1440 machine cycles to accomplish key derivation in one level. Therefore, the owner $\mathcal{O}$ will spend about $(8000 + 10750) * 1440$/1-GHz ≈ 0.027 *sec* on hash calculation. Following the same analysis, the end user $\mathcal{U}$ will spend about $250*2000*1440$/1-GHz ≈ 0.72 *sec* on hash calculation.

For the generation of and encryption with the one-time key pad, previous research shows that algorithms such as ISAAC [16] need 19 machine cycles to generate 32 bits of pseudo random number. At the same time, exclusive-or operation is usually more efficient than random number generation. Therefore, to conduct the over-encryption operation, the server $\mathcal{S}$ and the user $\mathcal{U}$ need about $2*1$ $GByte$ $*8/32*19$ $\approx 9.6*10^9$ machine cycles. That equals to about 10 *seconds* of calculation time and it can be easily hidden in the transmission time of the 1 GByte data.

The communication overhead of the proposed approach comes from the transmission of the data block index numbers, encryption keys in the hierarchy, control blocks, and updated data blocks. For each chunk of 1000 consecutive data blocks, we need to send only the index numbers of the first and the last blocks. The description in Section 5 shows that a control block contains a hash value, a block index number, and the number of times that the block has been updated. All these can be fit into 42 bytes. Since scientific databases usually have a very low rate of data updates [11, 26], we assume that 0.1% of data blocks are updated. Combining the information, we show the communication overhead of each party in Table 1.

computational overhead (in machine cycle)			
	owner $\mathcal{O}$	server $\mathcal{S}$	user $\mathcal{U}$
key derivation	$27M$	–	$720M$
one-time pad generation and over-encryption	–	$10G$	$10G$
communication overhead			
	owner $\mathcal{O}$	server $\mathcal{S}$	user $\mathcal{U}$
data blk index #	6KByte	–	10KByte
control blk	–	–	10.5KByte
keys in hierarchy	16KByte	–	–
updated data blk	–	1MByte	–

Table 1: Overhead of the proposed approach.
The table shows only the overhead of the proposed key management mechanism. It does not contain the transmission and decryption of the 1GB outsourced data since that overhead is independent of the adopted key management scheme.

The proposed approach introduces very limited storage overhead. The key derivation mechanism allows the owner $\mathcal{O}$ to store only the root keys of the hierarchies. The end user $\mathcal{U}$ does not need to pre-calculate and store all data block encryption keys. On the contrary, it can calculate the keys on the fly when it is conducting the data block decryption operations. The service provider $\mathcal{S}$ needs to store an extra copy of the updated data blocks. When the data update rate is very low in the application environment, the extra storage overhead at $\mathcal{S}$ is also low compared to the size of the outsourced data.

One problem that we need to consider is the lengthened data retrieval delay caused by the access to updated data blocks. We can shorten the response time from both the server and the data owner sides. At the server side, it has a one-to-one mapping among the index numbers of the original data blocks and those of the updated blocks. When the server receives a data access request to a control block, it can send the updated data block together with the control block to the user. This will not compromise the confidentiality of the information since the user still needs to get the new encryption key from the owner. At the data owner side, it can take advantage of the temporal locality of data access to shorten the response time. The owner can maintain a buffer of the mapping between the index numbers of updated data blocks and their new encryption keys. In this way, when the owner receives an access request to such a block, it can directly send both the old and the new encryption keys back without waiting for the control block. Please note that these two methods do not reduce the communication overhead of the proposed approach. It only allows the server and the owner to deliver information of the updated blocks to end users more efficiently. In this way, we avoid another round of request/reply and shorten the response time of the system.

7. DISCUSSION

In this section we discuss several problems of the proposed key management approach.

7.1 Comparison to CCS'05 Approach

In this part, we discuss the difference between our approach and the mechanism proposed by Atallah *et al.* in ACM CCS'05 [3]. First, we want to say that Atallah's paper provides a more generic approach to key management in access hierarchies. It has several features that our approach does not have: (1) separation between private key and actual encryption key; (2) support of downward and limited depth inheritance; and (3) support of shortcut edges in hierarchies.

However, when we zoom into the application scenarios investigated in this paper, our approach makes special adjustments to adapt to their properties. First, our approach has less communication and storage overhead for data retrieval from scientific databases when they have infrequent update operations. Using the approach in [3], we need to send metadata for every data block during information retrieval. Since the metadata represents the difference between a hash result and the real encryption key, its length will be $256/8 = 32$ Byte. If the user reads n blocks of outsourced data, the extra communication overhead will be $32n$ Bytes. On the contrary, in our approach we will maintain metadata only for those updated data blocks. Analysis in Section 6 shows that the control blocks have the length of 42 Bytes. If the block update rate is q, we need to transmit $42 * q * n$ Bytes for control blocks. When the data update rate q is smaller than $32/42 = 76\%$, our approach has less communication overhead. For storage overhead, the control block will take the space of the whole data block. Therefore, when the data update rate q is smaller than $32/4000 = 0.8\%$, our approach has less storage overhead on the service provider.

Second, our approach handles user revocation differently from [3]. When a user's access right changes (but the data blocks do not change), we want to avoid operations to the storage service provider. We adopt two methods: over-encryption and lazy revocation, to achieve this goal. In [3], the authors suggest two schemes, both of which will immediately change the encryption keys and metadata of the impacted data blocks. These methods will cause extra communication and computational overhead for data reencryption to achieve consistent data confidentiality.

7.2 Multi-hierarchy Key Management

In Section 4.1, we assume that the data owner uses only one hierarchy to derive the block encryption keys. To improve the efficiency of the proposed approach, we can construct multiple key derivation hierarchies simultaneously over the data blocks. During this procedure, we need to pay attention to several issues. First, since we store only one encrypted copy of each block on the service provider, we cannot directly use the hash results as the block encryption keys since each hierarchy has a different root key. Therefore, for each data block, we need to store the differences between the hash results and the real encryption key as metadata. Second, for each key derivation hierarchy, we need to generate a block index mapping function to establish a one-to-one mapping among the leaf nodes in the hierarchy and the index numbers of the outsourced blocks. In this way, data blocks have different organizations in different hierarchies and we can calculate the smallest $\mathcal{K}'$ for a data access request.

Building multiple key derivation hierarchies over the data blocks will not have a large impact on the data access and information update procedures. When the data block encryption keys change, we need to update the metadata for the corresponding entries. Another impact on the owner is that it needs to calculate the data access key set $\mathcal{K}'$ for multiple hierarchies to locate the one with the smallest size.

7.3 Security of the Approach

In previous sections we have described mechanisms to defend against eavesdroppers. In this part, we investigate the safety of the proposed approach over collusive attacks and replay attacks of the control blocks.

In collusive attacks, two or more revoked end users may put their stored keys together and try to derive a secret that is not the descendant of any key known to them. Following the proof in [3], we can show that the adversaries have to have a non-negligible advantage in breaking the hash function to accomplish this task. Therefore, the proposed approach is robust against collusive attacks if the hash function is considered safe.

In Section 5.2, we show that an end user needs to send the control block of the updated data to the owner to get the new encryption key. Here a user may send an old control block to the owner to get the encryption key of a previous version of the data block. In this way, an adversary that can access only the current data block will get a copy of the old encryption key. Should it have kept a copy of the encrypted data block by eavesdropping on the network traffic, it will compromise the backward secrecy of the system. To defend against the replay attack, the data owner must verify the freshness of the control block. For example, when the service provider sends the control block to the end user, it can encrypt the user name, the request index, and the hash of the control block with the secret key between $\mathcal{O}$ and $\mathcal{S}$. In this way, the owner will be able to verify whether or not this control block is the latest version.

7.4 Extending to Multi-owner Outsourced Data

While in this paper we consider only the simple case of outsourced data with a single owner, the proposed approach can be extended to the scenarios in which the data has multiple owners and each of them can change data blocks independently. In this part, we investigate techniques to accomplish this extension.

To maintain data consistency, we should have orderly execution of the update operations when multiple owners want to change the data contents. This can be achieved through a semaphore flag at the service provider $\mathcal{S}$. This problem has been extensively studied in Operating Systems and Distributed Databases for access to shared resources.

While each data owner will have its own key derivation hierarchy, we still want to store only one copy of each data block on the service provider. Therefore, for every leaf node in a key derivation hierarchy we need to store its difference from the real data block encryption key as public metadata. In this way, an end user can send a data access request to any data owner. It will then combine the returned keys in the hierarchy with the corresponding metadata to calculate the real encryption keys.

The data update operations are similar to the procedures described in Section 5. When a data owner wants to update a data block, it will store a control block to its original place.

The control block will be protected by a secret key shared among the data owners. The control block contains a pointer to the new storage place, the information used by the owners to derive the new data encryption key, and a hash value of the whole block to protect its integrity. To prevent an attacker from sending an old control block or a control block for another data block to the owner, the service provider will link a control block to a specific data access request when the contents are sent back to the end user.

8. CONCLUSIONS

In this paper we propose a mechanism to achieve secure and efficient access to outsourced data in owner-write-users-read applications. We assume that the outsourced data has a very large scale and we try to reduce the overhead at the data owner and service provider. We propose to encrypt every data block with a different key so that flexible cryptography-based access control can be achieved. Through the adoption of key derivation method, the owner needs to maintain only a few secrets. Analysis shows that the key derivation procedure based on hash functions will introduce very limited overhead. We propose to use over-encryption and/or lazy revocation to prevent revoked users from getting access to updated data blocks. We design mechanisms to handle both updates to outsourced data and changes in user access rights. We analyze the computational, storage, and communication overhead of the approach. We also investigate the scalability and safety of the approach.

Extensions to our approach include the following aspects. First, we plan to design a new scheme for key management based on this approach so that it can be applied to many-write-many-read applications. Second, we want to design dynamic mapping functions among keys in the hierarchy and index numbers of data blocks so that we can progressively reorganize the data blocks based on their access patterns. In this way, we can further reduce the number of keys that the owner sends to the end user. Finally, we plan to integrate existing approaches to access control, provable data possession, and key management for outsourced data to develop a new approach to secure Storage-as-a-Service.

9. REFERENCES

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. *SIGOPS Oper. Syst. Rev.*, 36(SI):1–14, 2002.

[2] G. Alvarez, E. Borowsky, S. Go, T. Romer, R. Becker-szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes. Minerva: an automated resource provisioning tool for large-scale storage systems. *ACM Transactions on Computer Systems*, 19:483–518, 2001.

[3] M. J. Atallah, M. Blanton, N. Fazio, and K. B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3):1–43, 2009.

[4] M. Blaze. Key management in an encrypting file system. In *Proceedings of the USENIX Summer Technical Conference*, pages 27–35, 1994.

[5] T. Chen, Y. Chung, and C. Tian. A novel key management scheme for dynamic access control in a user hierarchy. In *IEEE Annual International Computer Software and Applications Conference*, pages 396–401, 2004.

[6] H. Chien and J. Jan. New hierarchical assignment without public key cryptography. *Computers & Security*, 22(6):523–526, 2003.

[7] A. Chow, W. Coates, and D. Hopkins. A configurable asynchronous pseudorandom bit sequence generator. In *IEEE International Symposium on Asynchronous Circuits and Systems*, pages 143–152, 2007.

[8] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing. http://www.cloudsecurityalliance.org/, April 2009.

[9] E. Damiani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. *An Experimental Evaluation of Multi-Key Strategies for Data Outsourcing, IFIP International Federation for Information Processing, Volume 232, New Approaches for Security, Privacy and Trust in Complex Environments*, pages 385–396. Springer, 2007.

[10] E. Damiani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Key management for multi-user encrypted databases. In *Proceedings of the ACM workshop on Storage security and survivability*, pages 74–83, 2005.

[11] A. D'Atri and F. L. Ricci. Interpretation of statistical queries to relational databases. In *Proceedings of the international conference on Statistical and Scientific Database Management*, pages 246–258, 1988.

[12] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. A data outsourcing architecture combining cryptography and access control. In *Proceedings of the ACM workshop on Computer security architecture*, pages 63–69, 2007.

[13] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Over-encryption: management of access control evolution on outsourced data. In *Proceedings of the international conference on Very large data bases*, pages 123–134, 2007.

[14] E. Goh, H. Shacham, N. Modadugu, and D. Boneh. Sirius: Securing remote untrusted storage. In *Proceedings of the Internet Society (ISOC) Network and Distributed Systems Security (NDSS) Symposium*, pages 131–145, 2003.

[15] M. T. Goodrich, C. Papamanthou, R. Tamassia, and N. Triandopoulos. Athos: Efficient authentication of outsourced file systems. In *Proceedings of the international conference on Information Security*, pages 80–96, 2008.

[16] R. J. Jr. Isaac. In *Third International Workshop on Fast Software Encryption*, pages 41–49, 1996.

[17] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable secure file sharing on untrusted storage. In *Proceedings of the USENIX Conference on File and Storage Technologies*, pages 29–42, 2003.

[18] V. Kher and Y. Kim. Securing distributed storage: challenges, techniques, and systems. In *Proceedings of the ACM workshop on Storage security and survivability*, pages 9–25, 2005.

[19] J. Li, M. Krohn, D. Mazières, and D. Shasha. Secure untrusted data repository (sundr). In *Proceedings of the conference on Symposium on Opearting Systems Design & Implementation*, pages 121–136, 2004.

[20] P. Li, Z. Li, W. A. Halang, and G. Chen. A multiple pseudorandom-bit generator based on a spatiotemporal chaotic map. *Physics Letters A*, 349(6):467–473, 2006.

[21] C. Lin. Hierarchical key assignment without public-key cryptography. *Computers & Security*, 20(7):612–619, 2001.

[22] Z. Miled, Y. Liu, D. Powers, O. Bukhres, M. Bem, R. Jones, and R. Oppelt. An efficient implementation of a drug candidate database. *J. Chem. Inf. Comput. Sci.*, 43(1):25–35, 2003.

[23] E. Miller, D. Long, W. Freeman, and B. Reed. Strong security for distributed file systems. In *IEEE International Conference on Performance, Computing, and Communications*, pages 34–40, 2001.

[24] OpenCrowd. Opencrowd cloud taxonomy. http://www.opencrowd.com/views/cloud.php, 2009.

[25] B. Preneel, et, and al. Performance of optimized implementations of the nessie primitives. Deliverable 21 from the NESSIE IST FP5 project, 2003.

[26] P. J. Rhodes. *Granite: a scientific database model and implementation*. PhD thesis, University of New Hampshire, 2004. Adviser-Bergeron, R. Daniel and Adviser-Sparr, Ted M.

[27] H. Sakamoto. Data grid deployment for high energy physics in japan. *Computer Physics Communications*, 177(1-2):239–242, July 2007.

[28] A. Singh and L. Liu. Sharoes: A data sharing platform for outsourced enterprise storage environments. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 993–1002, 2008.

[29] W. Viana, B. Filho, and R. M. C. Castro. Pearl: a performance evaluator of cryptographic algorithms for mobile devices. In *The First International Workshop on Mobility Aware Technologies and Applications (MATA)*, 2004.

[30] B. Worthen. Inside the head of obama's cio. The Wall Street Journal Digits, March 5th, 2009.

[31] M. Xie, H. Wang, J. Yin, and X. Meng. Integrity auditing of outsourced data. In *Proceedings of the international conference on Very large data bases*, pages 782–793, 2007.

[32] S. Zhong. A practical key management scheme for access control in a user hierarchy. *Computers & Security*, 21(8):750–759, 2002.