Towards End-to-End Secure Content Storage and Delivery with Public Cloud

Huijun Xiong§, Xinwen Zhang†, Danfeng Yao§, Xiaoxin Wu†, Yonggang Wen‡
§ Virginia Tech, Blacksburg, VA, USA, {huijun, danfeng}@cs.vt.edu
†Security & Privacy Lab, Huawei Technologies, {xinwen.zhang, wuxiaoxin}@huawei.com
‡School of Computer Engineering, Nanyang Technological University, ygwen@ntu.edu.sg

ABSTRACT

Recent years have witnessed the trend of leveraging cloud-based services for large scale content storage, processing, and distribution. Security and privacy are among top concerns for the public cloud environments. Towards end-to-end content security, we propose and implement *CloudSeal*, a scheme for securely sharing and distributing content via the public cloud. CloudSeal ensures the confidentiality of content in the public cloud environments with flexible access control policies for subscribers and efficient content distribution via content delivery network.

CloudSeal seamlessly integrates symmetric encryption, proxy-based re-encryption, k-out-of-n secret sharing, and broadcast revocation mechanisms. These algorithms allow CloudSeal to cache the major part of a stored cipher content object in the delivery network for content distribution, while keeping the minor part in the cloud storage for key management. The separation of subscription-based key management and confidentiality-oriented proxy-based re-encryption policies uniquely enables flexible and scalable deployment of the solution as well as strong security for cached content in the network. We have implemented CloudSeal on Amazon Web Services, including EC2, S3, and CloudFront. Through experimental evaluation, we demonstrate the end-to-end efficiency and scalability of CloudSeal.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; K.6.5 [**Management of computing and Information Systems**]: Security and Protection—*Authentication, unauthorized access.*

General Terms

Algorithms, Security

Keywords

cloud computing, cloud storage, proxy-based re-encryption, security, confidentiality, content distribution

1. INTRODUCTION

Recent advance of Internet and information technology has shown two significant trends. First, media content has become the main Internet traffic. As predicated by Cisco, video streaming will consume approximately 90% of Internet traffic in 2015 [18]. Second, utilizing elastic cloud computing and storage resources has become the trend for enterprises and consumer-oriented commercial services. Large scale content processing, storage, and distribution via public cloud infrastructures become promising for quality-guaranteed and cost-efficient media streaming services. Despite the increasing usage of cloud in applications and services, security issues have been the top concerns for cloud computing [6, 22]. Among them, how to maintain the confidentiality and privacy of outsourced content in the public cloud remains a challenging task. The security requirement becomes more complex with flexible content processing and sharing among a large number of users through cloud-based applications and services.

Previous work has addressed such problems in conventional distributed environments [20, 34]. For large scale cloud-based content sharing and distribution services, there are new requirements beyond this. First, the content security should be realized by the content provider who uses public cloud services, instead of the cloud service provider [7, 28]. A content provider needs to encrypt her content with keys that are out of the reach of the cloud provider. Second, the access control policies should be flexible and distinguishable among users with different privileges to access the content. Each piece of content may be shared by different users or groups, and users may belong to multiple groups. Third, the number of redundant copies of the content cached in the content delivery network should be minimum in order to preserve efficiency of content distribution via the content delivery network. A user may earn benefits from the cache of encrypted content in the content delivery network of other users who have the same privilege. Multicast security [14] aims to address the confidentiality of content sharing with dynamic user groups. However, conventional multicast and broadcast involve only two types of entities: multicast/broadcast center and users. The content center belongs to the content provider or is fully trusted by the content provider. Their setting differs from our cloud-based model, which involves a semi-honest cloud provider to assist the content provider and the users. The earlier proxy-based encryption scheme for secure file systems [8] seems to work with semi-honest servers, but it fails to consider frequent key revocation problem, which is required by cloud-based data sharing systems. Therefore, we need a system with stronger content security guarantees and more flexible user and key management mechanisms.

In this paper, we propose CloudSeal, an end-to-end solution for secure content storage and delivery via the public cloud. By end-to-end, we mean that the content is encrypted at cloud-based storage and delivery channels. Only authorized end users or the content provider can decrypt it. CloudSeal ensures content confidentiality and content forward and backward security. CloudSeal applies dual encryption algorithms on the original plaintext of the content and uploads encrypted content to the cloud to protect content confidentiality. Then, CloudSeal seamlessly integrates proxy-based encryption and k-out-of-n secret sharing mechanisms to guarantee content forward and backward security. A proxy is employed in the cloud to transform encrypted content stored in the cloud storage to the delivery network or directly to the subscribers. The content provider updates re-encryption keys for legitimate groups and enforces shared secret keys among authorized subscribers, with which the content retrieved from the cloud can be decrypted.

Besides these security properties, CloudSeal further enables efficient content distribution and flexible content access control mechanisms. CloudSeal splits the ciphertext of the content stored in the cloud into two parts, so that the proxy only re-encrypts a very small part of ciphertext, and the large portion remains unchanged. The proxy guarantees two important properties of the content: 1) the content to be downloaded by the subscribers is always encrypted with the latest re-encryption key; 2) there is only *one* copy of the content stored in the content delivery network. These features enable the efficient cache mechanism during content distribution and achieve fast content distribution. For flexible content access control mechanisms, CloudSeal separates the distribution of the subscribers' decryption key from that of the content to enforce flexible authorization policies. Only authorized users can obtain the latest decryption key, and the content provider maintains the control of issuing new keys. We design k-out-of-n secret sharing and broadcast revocation protocols to renew the shared secret key in a scalable fashion.

We have implemented CloudSeal with Amazon Web Services (AWS), including EC2 (proxy service), S3 (cloud storage), and CloudFront (content delivery). With extensive experimental evaluation, we have demonstrated that CloudSeal achieves content distribution efficiency with the support of flexible user management.

The rest of the paper is organized as follows. We present the overview of CloudSeal and its design goals in the next section, and the details of its design in Section 3. Section 4 describes our prototype and evaluation results. We discuss related work in Section 5 and conclude in Section 6.

2. MODEL AND SYSTEM GOALS

Figure 1 shows a three-layer architecture for a typical content storage and distribution system over the public cloud infrastructure: a centralized *storage service* provided by a *cloud provider*, a *content delivery network (or service)* to accelerate content distribution over public network, and end

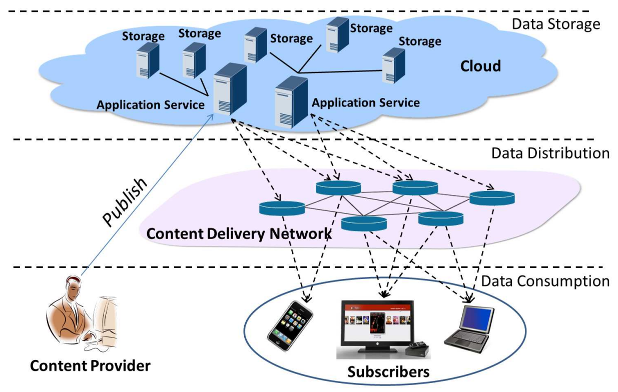

Figure 1: Schematic drawing of data flow in cloud-based storage and delivery services.

users with variant devices as *subscribers (content consumers or clients)*. A *cloud provider* provides two cloud-based services: storage and content delivery. A *content provider* utilizes these two services to store, share, and distribute her content to multiple subscribers. The content provider and subscribers can access content via a cloud-based *application service*, which reads and manages the content stored in the storage service via cloud storage APIs. The application service is an application deployed in the cloud by the content provider or a third party. The content provider can use multiple cloud-based services from different cloud service providers to host her application service, content storage service, and content delivery service. For example, Netflix [3] uses Amazon EC2 and S3 for content processing and storage, and uses multiple content delivery services such as Limelight, Level 3, and Akamia.

2.1 Trust Model and Assumption

CloudSeal trusts the content provider. The cloud service provider is *honest but curious*; that is, it follows the protocol and operations defined in CloudSeal, but it may actively attempt to gain the knowledge of the content. The content delivery service is also semi-trusted: it also may attempt to sniff content distributed and cached in the network, but it honestly performs all the operations and satisfies the quality of services, e.g., as specified in a service level agreement between the content provider and the delivery service provider. In addition, the cloud infrastructure (hardware and software) may be exploited by attackers who aim to obtain content [26].

CloudSeal aims to protect the content with large size and leverages public cloud for storage and content distribution. We assume that a subscriber does not store any cleartext and ciphertext of the content permanently in her local device. Instead, she downloads the content from the cloud and the content distribution network when she wants to access. We further assume that a subscriber does not re-disseminate decrypted content and her share of secrets to unauthorized parties.

2.2 Design Goals

We summarize the security objectives and system objectives of CloudSeal as follows.

- CloudSeal should ensure content confidentiality in the cloud storage and content delivery network even under the collusion between the cloud provider and the revoked subscribers.

- CloudSeal should support dynamic group-based user authorization, i.e., a user may choose to join or leave a group, or to be revoked from a group by the content provider at any time. Only authorized users are able to obtain the cleartext of protected content stored in cloud or cached in network at any system state.

- CloudSeal should support flexible security policies including forward and backward security.
  - For *forward security*, a user cannot access content published before she joins a group.
  - For *backward security*, a user cannot access content that is published after she leaves or is revoked from a group.

For forward and backward security, CloudSeal can be configured to support either or both. For example, a user may be allowed to access any movie that has been released before she subscribes, but cannot access any content after being revoked. For another example, a family content sharing application may allow a family friend to only access shared photos published during a certain period.

Beyond these security objectives, CloudSeal aims to achieve the following system and network performance requirements.

- CloudSeal should preserve the efficiency of content delivery network. In particular, it is desirable for the network to store a single copy of encrypted content at each system state.

- CloudSeal should support light-weight end storage cost. Only a small amount of storage should be required at the content provider side and the subscriber side to sustain user and key management of CloudSeal.

- CloudSeal should not affect user experiences at device side. The overhead of security mechanisms should be acceptable.

3. CLOUDSEAL SCHEME DETAILS

This section presents the overview, algorithms, and security analysis of CloudSeal in details. A preliminary scheme of CloudSeal is presented in [33].

3.1 Overview

Figure 2 shows the architecture of CloudSeal with three types of players: *cloud provider*, *content provider*, and *subscriber*.

- *Cloud Provider* provides two public cloud services: *storage service* for content storing and *content delivery* for content distribution. It also provides virtual infrastructure to host application services, which can be used by the content provider to manipulate the content stored in the cloud, or by the content subscribers to retrieve the content.

- *Content Provider* provides content to groups of subscribers, as well as user management. It uses cloud-based service from the cloud provider to store and distribute content.

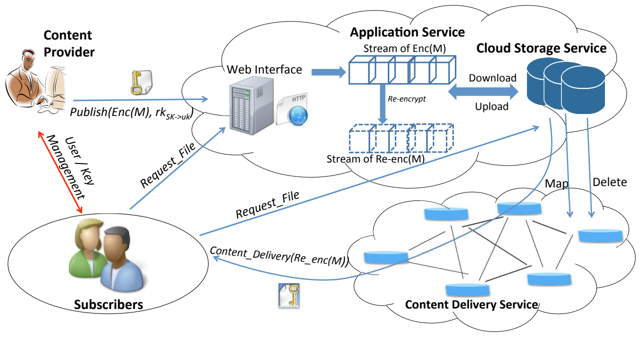

Figure 2: CloudSeal overview.

- *Subscriber* is able to access the content stored in the cloud if she successfully subscribes to the content provider. The subscriber can decrypt delivered content and consume it with local software.

The application service provides web interfaces for the content provider to publish and manipulate content in the cloud, as well as for subscribers to retrieve content from the cloud. The content can be directly accessed by consumers from the cloud storage service via storage APIs provided by the cloud service provider, or be cached in the delivery network once it has been accessed to save bandwidth and communication cost for the sake of high efficient content distribution.

To protect the content stored in the public cloud from being accessed by unauthorized parties, CloudSeal provides several cryptographic tools for the content provider to ensure that only authorized subscribers are able to obtain the decryption keys. The content provider preprocesses cleartext locally by calling CloudSeal `Enc` function and then publishes the processed content to the public cloud-based storage. `Enc` function performs dual encryption on the content with symmetric encryption as the first level of encryption and then proxy-based encryption as the second level of encryption. The dual encryption scheme enables CloudSeal to protect content confidentiality as well as outsource flexible access control policies. To delegate different access control mechanism, the content provider distributes corresponding re-encryption keys rk to the application service in the cloud. The application service transforms the ciphertext in the cloud by calling `Re_Enc` function with rk so that subscribers can decrypt them with the shared secret key uk. When an update happens in the user group, e.g., a user joins or leaves the group, the content provider updates the content re-encryption key rk stored in the application service to invalidate previous version of the content. Specifically, the content provider calls `Re_key` to generate a new delegation key for the application service. Then the application service produces new ciphertext by executing `Re_Enc` on the original cipher content in the cloud with the new re-encryption key rk. The small part of the resulting new cipher content is stored in the cloud storage service and the main part is cached in the delivery network to accelerate content distribution. During the re-encryption process, only a small part of the ciphertext of the content needs to be updated. The main part remains unchanged and can be persistently cached within the delivery network. When accessing a content, a client has to obtain all parts from cloud and distribution network in order to decrypt and render the content. This feature seamlessly achieves security objectives and efficient content distribution together.

For user management, CloudSeal has operations of `User Revocation` and `User Subscription`. CloudSeal supports a group of users' access to a set of encrypted content (or a channel) by sharing a secret key uk within the group. When a user joins the group, the content provider issues her shares of future secret keys as well as the current secret key. When a user leaves or is revoked from the group, the content provider broadcasts this user's share of the new decryption key to the entire group so that the remaining users are able to generate the new decryption key autonomously.

3.2 Preliminary

Bilinear Maps [10, 11]: Let $\mathbb{G}$, $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order r, we say $\hat{e}$ is an admissible bilinear map if: (1) computative actions in $\mathbb{G}$ and $\mathbb{G}_T$ are efficient; (2) for all $\alpha, \beta \in \mathbb{Z}_r$ of prime order r, $\hat{e}(g^\alpha, g^\beta) = \hat{e}(g,g)^{\alpha\beta}$; (3) $\hat{e}$ is non-degenerate, which means that not all pairs in $\mathbb{G} \times \mathbb{G}$ are sent to the identity in $\mathbb{G}_T$ by $\hat{e}$.

Secret Sharing [24, 29]: A k-out-of-n threshold secret sharing scheme is that a secret $S \in \mathbb{Z}_r$ shared by n users can be recovered, if the number of the secret shares exceeds the threshold k. The scheme utilizes a random polynomial P of degree $k-1$, where $P(x) \in \mathbb{Z}_r$ and $P(0) = S$. Given any k shares $< x_0, P(x_0) >, ..., < x_{k-1}, P(x_{k-1}) >$, one can use Lagrange interpolation formulas as follows to recover $P(0)$:

$$P(0) = \sum_{i=0}^{k-1} \lambda_i P(x_i), \text{ where } \lambda_i = \prod_{j \neq i} \frac{x_j}{x_j - x_i} \tag{1}$$

Proxy-based Re-encryption [8]: A proxy-based re-encryption algorithm transforms ciphertext c_{k1} to ciphertext c_{k2} with a key $rk_{k1 \to k2}$ without revealing the corresponding cleartext, where c_{k1} and c_{k2} can only be decrypted by different key $k1$ and $k2$, respectively. $rk_{k1 \to k2}$ is a re-key issued by another party, e.g., the originator of ciphertext c_{k1}.

3.3 CloudSeal Operations

CloudSeal operations involve in two planes: data plane and control plane. In the *data plane*, we describe the operations on the content, including *system setup*, *content publishing*, *proxy re-encryption*, and *content retrieving*. In the *control plane*, we present user management including *user revocation* – when a user leaves or is revoked from a group and *user subscription* – when a new user joins a group. Our scheme applies the dual encryption scheme to protect content confidentiality and uniquely bridges the proxy-based re-encryption scheme proposed by Ateniese *et al.* [8] and the secret sharing scheme in [24]. Table 1 shows the notation we use in this paper.

System Setup is called by the content provider to prepare the cryptographic system for content encryption and re-encryption. The content provider first chooses system public parameters *params*, namely $g \in \mathbb{G}$ and a bilinear map $\hat{e}$. It chooses a proxy secret key $SK \in \mathbb{Z}_r$ and public key $PK = g^{SK} \in \mathbb{G}$. The content provider keeps SK secret. The content provider chooses an integer k and a list L of polynomials of degree $k-1$ with coefficients randomly chosen from $\mathbb{Z}_r$, which are kept secret. The number of users who can be revoked at the same time is $k-1$. The content provider then chooses a random number from $\mathbb{Z}_r$ as the initial uk. This setup is performed by the content provider for each group of users.

Table 1: Notation.

Term	Notation
PK	content provider's public key
SK	content provider's secret key
uk, uk'	shared secret key for a group
$rk_{SK \to uk}$	re-encryption key
k	number of shares to recover uk
P	polynomial formula
x_i	user i's ID
$P(x_i)$	polynomial value of user i
M	original content
h	temporary secret of content provider

Content Publishing is run by the content provider to publish its content to the public cloud. The content provider performs the dual encryption scheme as follows. First she encrypts the content M with a symmetric data encryption key DEK to produce ciphertext $C(M, DEK)$. Then she further encrypts the content $C(M, DEK)$ with the secret key SK and *params* as shown in Algorithm 1. The resulting encrypted content has three components (u_{SK}, w, v), which are stored in the cloud-based storage service by the application service via cloud APIs. u_{SK} depends on the random secret h and content provider's secret key SK, w depends on the random secret h and the data encryption key DEK, and v depends on both h and the content. Usually, u_{SK} and w are much smaller than v.

Algorithm 1: *Enc(params, M, SK, DEK)*

```
1: Content provider chooses a random secret $h \in \mathbb{Z}_r$;
2: Content provider chooses symmetric data encryption
     key $DEK$ and encrypts the content $M$ to obtain
     ciphertext $C(M, DEK)$;
3: Let $Z$ denote $\hat{e}(g,g)$; Compute $u_{SK} = g^{SK \cdot h}$ and
     $Z^h = \hat{e}(g, g^h) = \hat{e}(g,g)^h \in \mathbb{G}_T$,
4: Content provider outputs ciphertext of content $M$:
     $(u_{SK}, w, v) = (g^{SK \cdot h}, DEK \cdot Z^h, C(M, DEK) \cdot Z^h)$.
```

Content Retrieving is for subscribers to access content stored in the public cloud. Two algorithms - *Re_Key* and *Re_Enc* - are involved in this process. Using *Re_Key* algorithm, the content provider generates a content re-encryption key $rk_{SK \to uk}$ with her secret key SK and the current decryption key uk. Details are shown as follows.

Algorithm 2: *Re_Key(params, SK, uk)*

```
1: Given $params, SK$ and $uk$, the content provider
     computes $rk_{SK \to uk} = g^{uk/SK}$.
```

Upon request, the application service obtains the newest $rk_{SK \to uk}$ from the content provider and re-encrypts the target cipher content (u_{SK}, w, v) with the following *Re_Enc* algorithm.

Algorithm 3: *Re_Enc((* u_{SK}, w, v *),* $rk_{SK \to uk}$ *, params)*

```
1: The proxy calculates $u_{uk} = \hat{e}(rk_{SK \to uk}, u_{SK})$
     $= \hat{e}(g^{uk/SK}, g^{SK \cdot h}) = \hat{e}(g,g)^{uk \cdot h} = Z^{uk \cdot h}$;
2: The proxy outputs re-encrypted content $(u_{uk}, w, v)$.
```

The re-encryption is only performed on u_{SK}. Because u_{SK} is independent of the content M, CloudSeal saves the processing time and storage I/O cost of the application service and storage service.

After this, the application service stores cipher content (u_{uk}, w, v) in the cloud storage service and allows the download. For each cipher content, w and v can be cached in the content delivery network, while u_{uk} cannot be cached. Since the size of u_{uk} is small, this operation does not affect the efficiency of content delivery.

When the system state is changed, e.g., a user joins or leaves or is revoked from the group, the shared secret key is updated from uk to uk'. Once the new secret key is updated to authorized users (explained next), the content provider generates the re-key $rk_{SK \to uk'}$ by running *Re_Key* algorithm and sends the re-key to the application service for content re-encryption with *Re_Enc* algorithm. The new cipher content is $(u_{uk'}, w, v)$. The user then downloads $u_{uk'}$ from the cloud storage, w and v from the content delivery network.

After a user obtains the encrypted content (u_{uk}, w, v), she follows Algorithm 4 below to decrypt the cipher with her current secret key uk. The user either obtains the secret key uk from the content provider when she first joins or computes it (described in User Revocation next).

```
Algorithm 4: Decrypt((u_uk, w, v), uk)
1: Given $u_{uk}$ and $uk$, the subscriber computes $u_{uk}^{1/uk} = Z^h$,
   where $u_{uk} = (Z^{uk \cdot h})$;
2: The subscriber calculates $DEK = w/Z^h$, and
   $C(DEK, M) = v/Z^h$;
3: The subscriber decrypts $C(DEK, M)$ with $DEK$;
4: The subscriber outputs original content $M$.
```

User Revocation happens when a subscriber leaves a group or is revoked by the content provider. It requires key revocation operations in the group. Our key revocation process is based on the k-out-of-n threshold secret sharing scheme. We consider the following two cases.

- *Case I:* There are $k-1$ users to be revoked at one time. The content provider revokes $k-1$ users with shares $P(x_1), P(x_2), ..., P(x_{k-1})$, respectively. The content provider broadcasts the shares of secrets and identities of these users $< x_1, P(x_1) >, < x_2, P(x_2) >, ..., < x_{k-1}, P(x_{k-1}) >$ to the entire group. Each user x in the group combines her share of secret $< x, P(x) >$ with these $k-1$ shares, to interpolate the new secret key $uk' = P(0)$. For example, $k = 2$, $P'(0)$ can be calculated by $\frac{x}{x-x_1}P'(x_1) + \frac{x_1}{x_1-x}P'(x)$ according to Equation 1. The content provider uses uk' as the new shared secret key to generate re-encryption key for non-revoked users.

- *Case II:* There are t users to be revoked, where $t < k-1$. The content provider performs the revocation by sending the t shares of secret and additional $k-t-1$ shares of the secret of polynomial P. These additional shares are values different from any existing users.

Polynomial P is then removed from the list L. If the list L is empty, the content provider adds new polynomials, as well as computes and distributes corresponding secret shares to current subscribers (for future interpolation purposes).

User Subscription happens when a user joins a group. Successful subscription authorizes a user's access to protected content. To prevent new users from accessing content published before they join (for forward security), the key revocation process is required to be executed as follows.

- Upon receiving join requests from t new users, the content provider obtains the first polynomial P' on list L, and calculates key $uk' = P'(0)$; uk' is sent to the new users in secure channels.

- The content provider assigns each new user a unique identity $x_i \in \mathbb{Z}_r$ and her share of secret from polynomial $P'(x_i)$, along with x_i's values from the other polynomials on list L. The content provider sends these polynomial values, except for $P'(x_i)$, to the new users respectively for future key updating through secure channels.

- The content provider broadcasts new users' share of secret $< x_i, P'(x_i)$ to the current group members for new key generation. If $t < k-1$, the content provider generates $k-t-1$ more share of the secret with P' and sends them to the entire group. These $k-t-1$ shares of the secret are different from any existing values. P' is removed from the list L.

For each current group member x_j, upon receiving $< x_1, P'(x_1) >, < x_2, P'(x_2) >, \ldots, < x_{k-1}, P'(x_{k-1})$ from the content provider, she calculates the new key with her share of secret $P'(x_j)$ for P' that was received when x_j joined earlier. This user can recover the new secret key $uk' = P'(0) = b$ by calculating $P'(0)$ with its share and received share.

3.4 Security Analysis

CloudSeal aims to protect content confidentiality and user access control (forward and backward security). We briefly discuss them next.

3.4.1 Content Confidentiality

CloudSeal performs two types of encryption algorithms to the content before outsourcing it. One is symmetric encryption algorithm to protect the confidentiality of the original content. The other is the *first-level encryption algorithm* in [8] executed on the resulting ciphertext from the symmetric encryption to enable flexible user access control of the ciphertext. Although we use the same secret parameter h for all blocks of a content to speed up the *first-level encryption algorithm*, CloudSeal is as strong as the symmetric encryption algorithm that is applied on the original plaintext of the content. Therefore, CloudSeal achieves the confidentiality of the encrypted content exposed in the public cloud. Furthermore, in any system state, the cloud service provider or an attacker cannot decrypt the cipher content with only re-encryption keys $rk_{SK \to uk}$.

CloudSeal leverages the dual encryption scheme and uses the same h for all blocks of a content to achieve security and performance at the same time. Several approaches may be alternatives. However they either fail to fulfill the security goals of CloudSeal, or cannot achieve the efficiency of encryption and content distribution for the practical usage. For example, one approach is to distribute DEK via a secure channel to individual users, instead of be encapsulated within the content. This solution cannot support forward and backward security objectives without invaliding cached content in the network and re-publishing content with a new DEK. Furthermore, any leakage of the DEK can compromise the confidentiality of the content permanently, unless the content provider invalidates the cache and re-encrypts and re-distributes it. Sole proxy-based re-encryption algorithm with different secret parameter h for each data block of a big file is too slow for encryption at the content provider side.

3.4.2 Forward and backward security

CloudSeal is designed to protect content forward and backward security. When a user joining or leaving event happens in a group, the content provider issues a new re-encryption key for this group to the application service, then requires the application service to alter the content to be delivered with the newest re-encryption key, and finally updates the entire group with the new group information. For a join event, the content provider sends her the latest decryption key and her shares of secrets for future key update. Consequently, a new user cannot decrypt the old content with the new decryption key, and a revoked user cannot decrypt new content with her old keys. Note that CloudSeal does not prevent a user from sharing decrypted content or decryption keys to unauthorized users. A digital rights management tool, such as the Microsoft DRM component in Netflix [25], can be used to solve this problem, which is out of the scope of CloudSeal.

3.4.3 Collusion Resistance

Collusion between delegatees and the proxy is one weakness of many proxy-based re-encryption algorithms. The goal of collusion resistance in such systems is to prevent the recovery of a delegator's secret keys by combination of the issued re-encryption key and delegated decryption keys. CloudSeal utilizes the proxy-based re-encryption scheme proposed in [8], which has been proven to be collusion resistant. This guarantees that the secret key of a content provider is secure even with a user or the cloud provider obtains both re-encryption key and the user's decryption key. With this algorithm, CloudSeal ensures that the entire outsourced content cannot be compromised and the content provider preserves the control of security policies by issuing of different re-encryption keys to different authorized groups of subscribers.

Besides security properties for content sharing and delivery via public cloud, CloudSeal uniquely achieves content distribution efficiency. When a group state changes, only a small part of the cipher content needs to be re-encrypted in cloud, while most of the content objects can be cached in the delivery network and shared by users. The separation of content operations (data plane) and user management operations (control plane) further enables flexible and scalable deployment of CloudSeal in the public cloud and network environment.

4. IMPLEMENTATION & EVALUATION

In this section, we first present the implementation of CloudSeal with Amazon Web Services (AWS), and then evaluate its system and network performance.

4.1 System Implementation

Content sharing applications with public cloud. We spent nontrivial efforts to build a simulated content sharing service based on three AWS services [1]: Elastic Compute Cloud (EC2) service to host an application service for subscribers, Simple Storage Service (S3) to store the cipher content for content providers, and Content Delivery Service (CloudFront) to distribute content. Amazon EC2 provides a virtual environment and allows developers to launch virtual machine instances with various operating system and applications. Amazon S3 is a cloud-based storage system, which stores data in *buckets* and allows write, read, and delete operations on them. Amazon CloudFront provides high-speed content delivery service by automatically caching content in the nearest edge locations and delivering the cache to end users. To achieve security enhancement of this service, we extend its functionality with algorithms and protocols aforementioned in CloudSeal.

We implement the application service as a website written by PHP and host it with an Apache server running in an Amazon Linux EC2 instance. In order to directly store and manipulate the content in S3, we create a `connection object` with Python library `boto` [2], which calls the function *boto.connect_s3(s3.key, s3.ID)* to establish connections between the EC2 instance and S3 storage. *s3.key* and *s3.ID* are the key and identifier number of Amazon S3 instances assigned when created.

Our content is stored in the buckets of Amazon S3, which assigns each bucket with a global unique name and each file a URL composing with bucket name and file name. We use this URL as a content object in the application service web page. For example, file *video1.mp4* is stored in bucket *bucket1*, the access URL to that file is `http://s3.amazonawa.com/bucket1/video1.mp4`. Applications can use HTTP or BitTorrent-like protocols to access data stored in Amazon S3. Our application service uses HTTP protocol. Users can request files by file names from the file list stored in the application service with `HTTP GET` request. For each published content, we create two buckets in S3: one is for the originally published cipher content $<u_{SK}, w, v>$ from the content provider, which is not updated once stored, and the other for the cipher content $<u_{uk}, w, v>$ to be delivered, which is updated once a user joins or leaves. Although both buckets can be made publicly readable to users, we set `private` permission to the original published one since only the content provider and the application service need to access them. The other content bucket is `publicly` readable to all users. To avoid storage redundancy, only u_{SK} of the cipher content is stored, as the other part $<w, v>$ is the same as the public bucket. This part is so small that we cache them on EC2 to facilitate the cryptographic operations.

CloudSeal with its cryptographic tools. We implement aforementioned data plane cryptographic algorithms (*Enc*, *Re_Key*, *Re_Enc*, and *Decrypt*) based on cryptographic functions from the OpenSSL library [4] and the pairing-based cryptographic library (PBC) [5] with independent native processes. We choose Advanced Encryption Standard (AES) with 16 bytes key as our symmetric encryption algorithm and implement the `CFB` mode of `AES` to randomize the ciphertext of each data block of a file. For pairing-based encryption, instead of mapping an arbitrary M to a certain field $\mathbb{G}_T$, we map the elements in $\mathbb{G}_T$ to byte array in C language and use `XOR` operation in our implementation to produce w and v. This approach accelerates the encryption operation for the content provider. For key updating, we choose random linear polynomial formula so that only one user can be revoked with a single re-encryption operation.

We further develop a web application as the administrative service to assist the content provider for the user and key management on EC2 instance. This service enforces the cryptographic tools running on the same EC2 instance as the application service. When a user signs up in a group with the administrative service, the service updates current decryption key of the group and assigns this key and a share of future decryption keys to the new user. It updates the entire group with the new user's share. When a user decides to leave the group, the user sends `Leave` message to the administrative service. The service again updates the current decryption key and distributes the share of secret of the revoked user. When an update happens in a group, the administrative service blocks any new file downloading activities in this group. Then it calculates the new key and distributes the revocation information to the remaining users, and then re-encrypts the content and stores it back on EC2 with the newest decryption key. After this, this group can start downloading files. In order to keep the up-to-date decryption key, the user periodically checks the administrative service to see whether or not there is a revocation by sending `Update_Checking` HTTP requests to the server. Once there is a revocation, remaining users send `GET_Update_Info` packets to get the newly revocation information and update her decryption key accordingly. As we integrate the administrative service into the HTTP server of the application service to measure the key and user management mechanisms, the user needs to actively pull the update information from the service, instead of passively receiving message from the service. We plan to realize the administrative service with an XMTP (XML MIME Transformation Protocol) server in the future, with which users can receive push notification from the server.

4.2 Experimental Evaluation

To confirm that CloudSeal achieves the performance objectives (cf. Section 2.2), we sought to answer the following questions to demonstrate that the security mechanisms from cryptographic algorithms in CloudSeal bring acceptable costs to end-to-end performance.

1. What is the overhead of cryptographic operations including content encryption by the content provider and the content decryption by the subscribers?

2. What is the overhead of the re-encryption operations including content re-encryption by the application service in the cloud? Will the application service become performance bottleneck due to the overhead?

3. What is the affect of content delivery network to the cloud-based content distribution?

We have conducted a number of experiments to evaluate CloudSeal in the system and cloud levels. We examine the host-based efficiency of the cryptographic algorithms with different pairing types and the user management costs for communication and storage. We further conduct experiments on Amazon public cloud environment to evaluate efficiency of different cloud services of CloudSeal with two EC2 types (small, and medium) and different data distribution mechanisms (with or without CloudFront).

4.2.1 *Efficiency of Algorithms and Protocols*

Computation Cost. To show the performance at the content provider side and the subscriber side with cryptographic operations, we conduct content encryption (by content provider) and decryption (by content consumer) tests

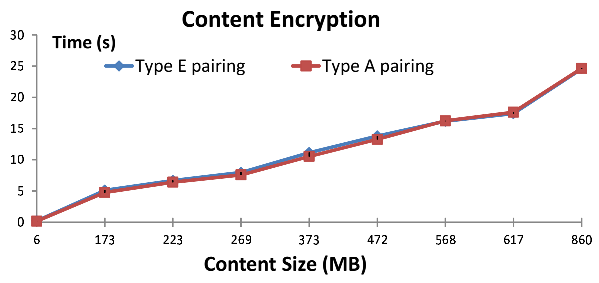

Figure 3: Overhead of encryption operations with different pairing types.

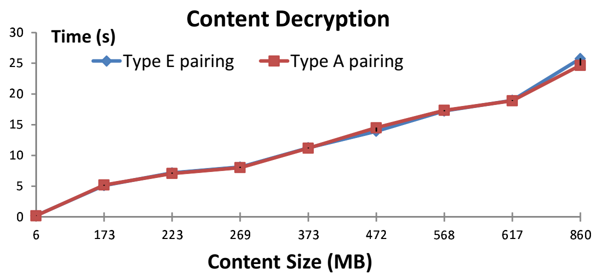

Figure 4: Overhead of decryption operations with different pairing types.

locally on a desktop with Intel Duo CPU 2.93GHz, 4GB RAM, SUMSUNG 7200RPM, and CentOS 2.6. The encryption time consists of symmetric encryption with CFB mode, pairing operation, and `XOR` operations. The decryption time includes key generation, symmetric decryption, pairing operations, and `XOR` operations. We choose two symmetric pairings from the pairing based library [5], including Type E pairing *e.param* and Type A pairing *a.param*, to examine the impact from different pairing types to CloudSeal according to the scheme of CloudSeal. The symmetric key length is 16 bytes. The length of pairing-based master key and decryption key is 20 bytes for both pairing types.

We put 9 different sizes of files in the desktop and run the encryption and decryption algorithms 20 times on each file to compute the average processing time. As shown in Figure 3 and Figure 4, encryption and decryption time increase along with the content size, while there is a tiny difference between the two symmetric pairing types. For content decryption at consumer side, it takes less than 30 seconds to decrypt a 800MB video file.

Communication Cost. We investigate the communication cost caused by a group update, e.g., a user joins or is revoked. Recall that the degree of the polynomials is the number of users who can be revoked simultaneously. For example, a degree 2 polynomial implies that the content provider can revoke two users at one time. Assume we choose a degree d polynomial and we need B bytes to send one user's revoking information, if we have r users remain in the group, each revocation causes $d * B * (r + d)$ bytes traffic in the network with our broadcasting mechanism, where $(r + d)$ is the total number of users in the group. Suppose there are 100 users in one group, d equals to 2, and B is 40 bytes according to our implementation, the total amount for one update in CloudSeal is around $8KB$.

Table 2: Re-encryption Time (Seconds) of 600MB Content on Different Amazon EC2 Instances.

Pairing	Key Length (Bytes)	Small EC2	Medium EC2
TypeE	20	0.00007425	0.000071
TypeA	20	0.00007925	0.000077

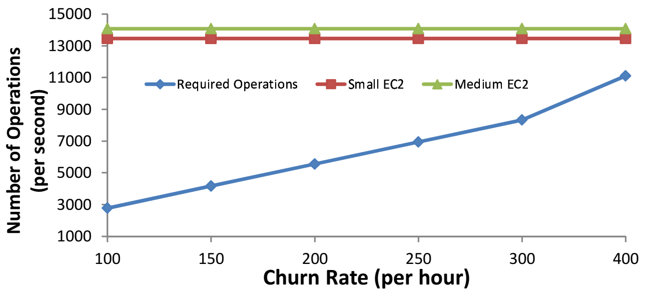

Figure 5: Number of required re-encryption operations with different churning rates

Storage Cost. We further look into the storage cost for the key management at both content provider side and subscriber side. In order to sustain system state, the content provider needs to keep current user secret key and a list of polynomials in terms of sets of coefficient for future system update. In our implementation, 20 bytes are enough to represent user secret key or each coefficient of polynomials. The average amount of storage cost for each group at content provider side is 6KB given 100 polynomials of degree 2. This cost is small enough to support a large number of groups of subscribers in the service. For subscribers, they only need to keep their own values of the list of polynomials with total amount of $2KB$ for 100 polynomials of degree 2.

4.2.2 *Application Service Performance*

We evaluate Amazon EC2 service on two different instance types: small and medium, located in AWS North California data center. The small EC2 instance consists of one core CPU with 1 ECU and 1.7GB memory, and the medium EC2 instance has two core CPU with 5 ECUs and 1.7GB memory. An ECU provides the equivalent CPU capacity of 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. We focus on examining the re-encryption algorithm efficiency on Amazon EC2, which is the only operation on cipher content by the application service. We run each experiment 20 times and report the average running time.

As Table 2 shows, the operation on the small EC2 instance is slightly slower than that on the medium instance, which implies that computation ability of CPU has positive impact on cloud performance for our application services. Comparing with previous encryption and decryption time shown in Figure 3 and Figure 4, the re-encryption time on same size of content is significantly shorter.

CloudSeal can serve multiple groups, each having a different set of authorized users. As a centralized component, the application service can be a bottleneck for performance, especially for the content re-encryption operations for all groups. Suppose there are 100 similar groups, each with maximum 100 subscribers and 1000 content objects. We adjust the churn rate of a group to estimate the required re-encryption operations of the application service. The churn rate varies from 100 users per hour to 400 users per hour, which indicates the number of subscribers who join or leave the group per hour. From Table 2, we can calculate that, with the small EC2 instance, the application service can handle at most $1/0.00007425 = 13468$ re-encryption operations per second and with the medium EC2 instance, the application service can perform $1/0.000071 = 14084$ operations per second. Figure 5 demonstrates that with either small EC2 instance or medium EC2 instance, the number of re-encryption operations per second performed by the application service is much larger than the required operations per second along with different churn rate in the system. This indicates that the application service is adequate in providing revocation induced content re-encryption operations for groups for large churn frequencies. Furthermore, with elastic computing resources, the content provider can allocate more computing instances for the application service in an on-demand manner. Therefore, re-encryption operation in cloud will not be a bottleneck to the system's performance. Besides, as elapsed time for re-encryption operation is unrelated with the size of content, frequent revocation will not add large overhead on our system.

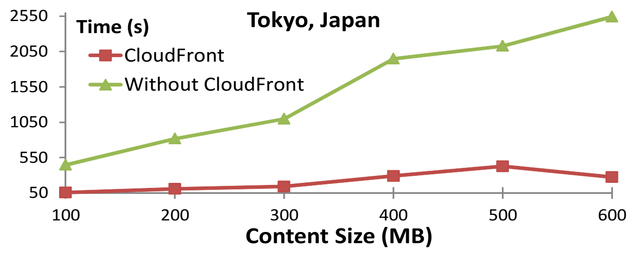

Figure 6: Downloading time for content delivery with storage center at Tokyo.

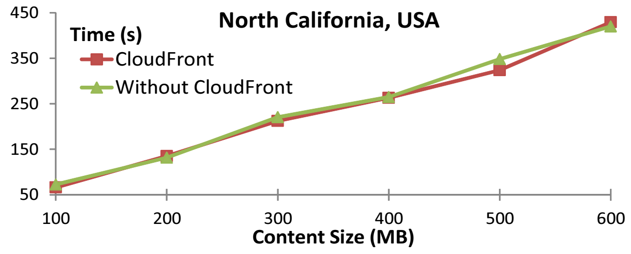

Figure 7: Downloading time for content delivery with storage center at N. California.

4.2.3 *Content Delivery Efficiency*

For content delivery network performance, we evaluate content delivery time with Amazon CloudFront. Four elements can affect the delivery efficiency: whether the CloudFront service is enabled, the locations of end consumers, deployed CloudFront edge servers [1], and the location of the content storage center. In our experiment, we store our content in two different Amazon S3 data centers: North California, USA and Tokyo, Japan. To initiate content delivery action, we develop a customized client with Python to continuously request files from one bucket stored in Amazon S3. We run 5 clients at North California, USA to leverage

[1]Our experiment was conducted in May 2011.

cache in local edge location. Each client requests 20 files and we measure the time of downloading all of them. As shown in Figures 6 and 7, without using CloudFront, the content delivery time in North California is much smaller than that in Tokyo. When the CloudFront is used, the delivery speed for content stored in Tokyo can be significantly improved by almost 10 times, while there is no obvious improvement for that in North California. We conjecture that this is because our client application is close to the North California S3 data center. Therefore the download speed does not change much when CloudFront is used.

Comparing the time for content delivery with the time of cryptographic operations (encryption, decryption, and re-encryption) shown in Figure 3, Figure 4, and Table 2, the cryptographic time is at least 40 times faster than the content delivery time. Therefore, the security mechanisms implemented in our prototype bring accesptable overhead.

In summary, our evaluation presents the efficiency of CloudSeal in content processing and content delivery. At subscriber side, CloudSeal brings acceptable decryption time. The overhead of CloudSeal does not affect user's experience. At the cloud side, the application service is not the bottleneck of the system and supports the efficient content re-encryption operations. Our system performs well when scaling up to a large number of subscribers. CloudSeal preserves the efficiency of content distribution of the content delivery network with a smaller overhead for cryptographic operations.

5. RELATED WORK

Several security solutions have been recently developed for securing data in cloud [9, 23, 27, 36]. With similar security concerns in outsourcing data to untrusted cloud service, authors in [16, 21, 31, 35, 37] have proposed several solutions. For example, Yu et al. [35] proposed an attribute based access control policy to securely outsource sensitive user data to the cloud. In their approach, data is encrypted by a symmetric key while the access to this symmetric key is controlled by KP-ABE algorithm [17]. To manage dynamic user groups, they delegate *rekey* operations to the cloud and let the cloud server update users' secret keys and re-encrypt data without revealing the underlying plaintext. CloudSeal is different from their approach in that: CloudSeal only allows a content provider to perform the `Re_Key` operation, and the proxy re-encryption is performed directly on part of the cipher content. Directly applying their approach to solve our problem of content delivery may not be practical, as their ciphertext is customized for different users. In comparison, as CloudSeal supports on-demanding data re-encryption operations, our design brings performance advantage for large scale content cached in content delivery network.

Secure storage system is an important application of proxy re-encryption algorithms [8, 19]. CloudSeal is based on the scheme proposed in [8], where the authors implemented an encrypted file storage system with an access control server in charge of data access according to proxy re-encryption schemes. When a client requests data from a block store, it firstly asks the access control server to re-encrypt the block with its public key and the system master key, such that it can decrypt the block with its own secret key. Access control server can deny re-encryption operation if the client is not authorized. In comparison, we consider user revocation problem in CloudSeal and also the real time efficiency of the system.

Secure multicast communication [12, 13, 14, 15, 32, 38] and conditional access systems [30] address similar security problems as ours in distributing content to dynamic user groups and key management. Proxy re-encryption algorithm and k-out-of-n mechanisms have been used to solve these problems. For example, researchers in paper [15] proposed proxy re-encryption based secure multicast mechanisms to achieve scalability and containment. Several proxies (routers), usually during the content transmission, transitively convert ciphertext data with re-encryption keys assigned when building a multicast network. When a user is to be revoked or a proxy (router) is off the network, corresponding proxy re-encryption keys and group secret keys need to be updated. The problem solved by CloudSeal is different from them due to the cache properties in content delivery network, which requires more efficient and flexible secure content delivery and user management mechanisms.

The security mechanism and supported access control policies of CloudSeal differ from what Netflix adopts [25] in several aspects. First, the goal of CloudSeal is to protect the security of outsourcing content for the content provider, rather than to prevent digital rights of the content on subscribers' device in Netflix [2]. Second, the access control of CloudSeal supports various groups of subscribers with different privileges for different shared content, while Netflix currently supports only one (unlimited) plan for their video streaming service. Third, CloudSeal encrypts the content when storing in public cloud and distributing via content delivery network. Netflix only encrypts the content at the edge of the content delivery network.

6. CONCLUSION AND FUTURE WORK

We designed and implemented CloudSeal, an end-to-end content confidentiality protection mechanism for large scale content storage and distribution systems over the public cloud infrastructure. By leveraging advanced cryptographic algorithms including symmetric encryption, proxy-based re-encryption, threshold secret sharing, and broadcast revocation, CloudSeal addresses unique challenges of efficient cipher content transformation, cipher content caching in the delivery network, and scalable user and key management. Through the prototype implemented on Amazon EC2, S3, and CloudFront, our experimental evaluation demonstrates that CloudSeal achieves the efficiency and avoids possible performance bottleneck. For future work, we plan to extend our current design to support an open service, where each user can publish content and delegate group membership control with our broadcast revocation library.

7. ACKNOWLEDGEMENTS

We thank the anonymous referees for providing valuable reviews. We especially thank Qingji Zheng for his constructive comments on the algorithms and Wei Zhu for his help in implementation.

[2]Netflix uses Microsoft DRM on subscriber side to ensure end-to-end content security and digital rights management, e.g., to prevent further dissemination of protected content by an end user.

8. REFERENCES

[1] Amazon Web Services. http://aws.amazon.com.

[2] boto: Python interface to amazon web services. http://code.google.com/p/boto/.

[3] Netflix on Amazon's Cloud. http://www.techflash.com/seattle/2010/05/netflix_on_amazon_cloud.html.

[4] OpenSSL Cryptography and SSL/TLS Tookit, http://www.openssl.org/.

[5] Pairing-based cryptography (pbc) library. http://crypto.stanford.edu/pbc/

[6] Cloud Computing, an IDC update, 2010.

[7] AWS Customer Agreement http://aws.amazon.com/agreement/, 2011.

[8] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. *ACM Trans. Inf. Syst. Secur.*, 9:1–30, February 2006.

[9] E. Bertino, F. Paci, R. Ferrini, and N. Shang. Privacy-preserving Digital Identity Management for Cloud Computing. *IEEE Data Eng. Bull.*, 2009.

[10] D. Boneh and M. K. Franklin. Identity-based Encryption from the Weil Pairing. In *CRYPTO '01.*

[11] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In *Proc. of ASIACRYPT '01.*

[12] B. Briscoe. MARKS: Multicast Key Management using Arbitratily Revealed Key Sequences. In *Proceedings of NGC'99.*

[13] B. Briscoe. Nark: Receiver-based Multicast Non-repudiation and Key Management. In *Proceedings of EC'99.*

[14] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. In *INFOCOM '99.*

[15] Y.-P. Chiu, C.-L. Lei, and C.-Y. Huang. Secure Multicast Using Proxy Encryption. In *Information and Communications Security*, Lecture Notes in Computer Science. 2005.

[16] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. Controlling Data in the Cloud: Outsourcing Computation without Outsourcing Control. In *Proceedings of CCSW '09.*

[17] 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.

[18] Cisco Inc. Cisco Visual Networking Index: Forecast and Methodology, 2010-2015. White paper, Cisco., 2011.

[19] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable Secure File Sharing on Untrusted Storage. In *Proceedings of FAST*, 2003.

[20] Y. Koglin, D. Yao, and E. Bertino. Secure Content Distribution by Parallel Processing from Cooperative Intermediaries. *IEEE Transactions on Parallel and Distributed Systems*, 2008.

[21] D. Lin and A. Squicciarini. Data Protection Models for Service Provisioning in the Cloud. In *Proceeding of ACM SACMAT '10.*

[22] Lockheed Martin, LM Cyber Security Alliance. Awareness, Trust and Security to Shape Government Cloud Adoption. White paper, Cisco, 2010.

[23] M. Nabeel, N. Shang, J. Zage, and E. Bertino. Mask: A System for Privacy-preserving Policy-based Access to Published Content. In *Proceedings of SIGMOD '10.*

[24] M. Naor and B. Pinkas. Efficient Trace and Revoke Schemes. In *Proceedings of the 4th International Conference on Financial Cryptography*, 2001.

[25] Pomelo, LLC Tech Memo. Analysis of Netflix's Security Framework for *Watch Instantly* Service, 2009.

[26] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, You, Get Off of My cloud! Exploring Information Leakage in Third-Party Compute Clouds. In *Proceedings of CCS*, 2009.

[27] R. Sandhu, R. Boppana, R. Krishnan, J. Reich, T. Wolff, and J. Zachry. Towards A Discipline of Mission-aware Cloud Computing. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, CCSW '10.

[28] Cloud Security Alliance. Security Guidance for Critical Areas of Focus in Cloud Computing V2.1, 2009. https://cloudsecurityalliance.org/csaguide.pdf.

[29] A. Shamir. How to Share A Secret. *Commun. ACM*, 22, November 1979.

[30] P. Traynor, K. R. B. Butler, W. Enck, and P. McDaniel. Realizing Massive-Scale Conditional Access Systems Through Attribute-Based Cryptosystems. In *NDSS*, 2008.

[31] W. Wang, Z. Li, R. Owens, and B. Bhargava. Secure and Efficient Access to Outsourced Data. In *Proceedings of CCSW '09.*

[32] C. K. Wong, M. Gouda, and S. S. Lam. Secure Group Communications Using Key Graphs. *IEEE/ACM Trans. Netw.*, 8, February 2000.

[33] H. Xiong, X. Zhang, W. Zhu, and D. Yao. Cloudseal: End-to-End Content Protection in Cloud-based Storage and Delivery Services. In *Proceedings of Securecomm*, 2011.

[34] D. Yao, Y. Koglin, E. Bertino, and R. Tamassia. Decentralized Authorization and Data Security in Web Content Delivery. In *Proc ACM Symp. on Applied Computing (SAC)*, 2007.

[35] S. Yu, C. Wang, K. Ren, and W. Lou. Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing. In *INFOCOM'10.*

[36] S. Zarandioon, D. Yao, and V. Ganapathy. K2C: Cryptographic Cloud Storage With Lazy Revocation and Anonymous Access. In *Proceedings of Securecomm*, 2011.

[37] L. Zhou, V. Varadharajan, and M. Hitchens. Enforcing role-based access control for secure data storage in the cloud. *The Computer Journal*, 2011.

[38] S. Zhu, C. Yao, D. Liu, S. Setia, and S. Jajodia. Efficient Security Mechanisms for Overlay Multicast based Content Delivery. *Comput. Commun.*, 30:793–806, February 2007.