CloudSky: A Controllable Data Self-Destruction System for Untrusted Cloud Storage Networks

The material in these slides mainly comes from the paper

Summarized by Adnan Alawneh
Outline

• Introduction
• Related Work
• CloudSky System Assumptions
• CloudSky Threat (Attack) Model
• Object-Based Storage Device (OSD)
• CloudSky Architecture
• Data self-destruction scheme overview
• Data sharing scheme
• Implementation
• Results
• Conclusion
Introduction

“Don’t ever say anything on e-mail or text messaging that you don’t want to come back and bite you” [1]

Solution

This paper proposes a self-destruction system, named CloudSky, which is able to enforce the security of user privacy over the untrusted cloud in a controllable way. CloudSky exploits a key control mechanism based on the attribute-based encryption (ABE) and takes advantage of active storage networks to allow the user to control the subjective life-cycle and the access control polices of the private data whose integrity is ensured by using HMAC to cope with untrusted environments.
Related Work

Vanish

- Sharing and protecting privacy.
- Secret key is divided and stored in a P2P system with distributed hash table (DHTs).
- Leverages Shamir’s secret sharing algorithm to restore the key.
  - How long the key can survive!
- Every 8 hours the DHT will refresh the nodes.

Related Work

Vanish

Related Work

**SafeVanish**
- Similar to Vanish, prevents hopping attack

**SeDas**
- Adopts active storage object (ASO) to control the deletion of the keys.
  - ASO has a time-to-live (ttl) value property.
  - The ttl value is used to trigger the self destruct operation.

**Plutus**
- A cryptographic file system to enable secure file sharing without placing much trust on the file servers
Ciphertext-Policy Attribute-Based Encryption (CP-ABE)

CloudSky System Assumptions:

1. *Time-limited value*
   The encrypted data is only valuable to the user for a limited period of time.

2. *Known timeout*
   When a data owner encrypts the data, he/she knows the approximate lifetime that the data could be used.

3. *Internet connectivity*
   Users are connected to the cloud by interacting with OSDs. Both users and OSDs have their own unique IDs.

4. *Dispensability under attack*
   Rather than risk exposure to an adversary, the data owner prefers the data to be destroyed, even if it is prematurely

CloudSky Threat (Attack) Model:

- How to think like an attacker!
- How to evaluate their goals, and
- How they might achieve those goals!

Key properties of Threat model:

1. **Trusted data owners/users.**
   - Users with legitimate access to the same files will trust each other.

2. **Collusion resilience.**
   - The model's data self-destruction scheme to be resilient to collusion attacks
   - If multiple users colludes, they may be able to decrypt a cipher text by combining their attributes even if each of the users cannot decrypt the cipher text alone. [2]

3. **Retroactive attacks on privacy.**
   - For example, email or SMS subpoenas typically come long after the user sends a particular sensitive email. Therefore, the system defends the user against future attacks against old copies of private data
   - The classes of adversaries we consider include: the user’s employer, the user’s ISP, the user’s web mail provider, and unrelated malicious nodes on the Internet.

4. **Compromise resistance.**
   - Compromising the storage device does not disclose the users’ data generated before the storage device is compromised.

**Object-Based Storage Device (OSD)**

**OSD-3 provides four security methods:**

- **NOSEC:** no security provided
- **CAPKEY** the integrity of the capability (command function that a command can request) included in each (command description block) CDB is validated
- **CMDRSP** the integrity of the CDB (including capability), status, and sense data for each command is validated.
- **ALLDATA** the integrity of all data between an application client and an OBSD in transit is validated

<table>
<thead>
<tr>
<th>Threat</th>
<th>NOSEC</th>
<th>CAPKEY</th>
<th>CMDRSP</th>
<th>ALLDATA</th>
<th>CloudSky</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forgery of credential</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Alteration of credential</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Alteration of capabilities</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Use of credential by unauthorized application client</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Replay of command or status</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Alteration of command or status</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Replay of data</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Alteration of data</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Inspection data</td>
<td>×</td>
<td>×</td>
<td>✓/✓*</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Inspection of command or status</td>
<td>×</td>
<td>×</td>
<td>✓/✓*</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

*Optionally, a secure channel may provide a data confidentiality guarantee that if a message is read, it can be understood only by authorized parties.*
CloudSky Architecture

The system has the following components:

1- Data owner/user
   Owner –
   • define (time attribute based) access policy
   • encrypt his own data with the policy before storing it
   • generation of key
   User –
   • must satisfy the access policy to access the file

2- Object-based storage devices
   – set of storage nodes, each can authenticate/authorize requests based on the user’s certificate and ACL stored on the node
   – ensure that the capability has not been tampered and the request is permissible
   – shares a secret key with security manager that is refreshed periodically

3- Metadata Server (MDS) – includes two managers
   Security Manager (SM) –
   • authorization, metadata maintenance, key management, as well as other information related to access control policies
   Policy/Storage Manager (P/SM) –
   • prepares policy-coordinated capabilities

4- Trusted authority (TA)
   – supplies and controls the encryption key for the encrypted data. It assembles the divided parts of the key from OSDS
Data self-destruction scheme overview

Lifetime of user data in the cloud

- **T1**: User encrypts data and uploads cipher text.
- **T2**: User downloads the cipher text and decrypts data. Note that cloud storage service may archive a pristine copy of the encrypted data during ($T_1 \sim T_2$).
- **T3**: The encryption key expire the active storage object is triggered and erase the encryption key in TA to self destruct the data. copy now is unreadable.
**Data sharing scheme**

The data sharing operation can be obtained by the following ABE-based algorithms:

**Setup (K):**

This algorithm takes as input a security parameter \( K \) and returns the public key \( PK \) as well as a system master secret key \( MK \). PK is used by data owners for data encryption key (DEK) encryption. MK is used to generate user secret keys and is known only to the trusted authority. Each user is assigned a set of attributes, an access structure that contains the authorized set of attributes.

The authority runs the Setup and Key Generation algorithms to generate system MK, PK, and user secret keys. [1]

**Data encryption key generation DEKGen(MK, U):**

This algorithm takes as input the master key \( MK \), and a set of users and it outputs a data encryption key (DEK) for each user.

**Attribute key generation SKGen(MK, Λ, U):**

Takes as input the master key \( MK \), a set of attributes, and a set of users and outputs a set of private attribute keys (SK) for each user that identifies with the attributes set.

The secret key \( SK \) enables the user to decrypt a message encrypted under an access structure iff the user’s set of attributes matches the access structure.

**Data owners encrypt sensed data DataEncrypt(DEK, D):**

Using a symmetric key algorithm, e.g., AES, the data encryption algorithm takes as input the data encryption key \( DEK \), and the data \( D \). It outputs a cipher text \( CD \).

**Encryption DEKEncrypt(PK, DEK, A):**

A randomized algorithm that takes as input the public key \( PK \), a data encryption key \( DEK \), and an access structure and outputs a cipher text \( CM \) such that only a user who possesses a set of attributes that satisfies the access structure will be able to decrypt the DEK message.

**Decryption DEKDncrypt(CM, SK, KΛ):**

Takes as input the cipher text \( CM \) which contains an access structure \( A \), a private key \( SK \), and a set of attribute group keys \( KΛ \) for a set of attributes \( Λ \). The decryption can be done iff the user’s set of attributes matches the access structure and the user’s attribute is not revoked.
• **Upload a file**

1. Data owner needs to select a unique ID, then
2. encrypt the data before uploading it, and finally
3. the owner has to define a set of attributes for encrypting DKE and also an access structure to control the access to the key.

• **Grant user access**

– For a user to have access, s/he needs a unique identity and a private key.
– The cipher text is sent to the TA when it is encrypted along with the owner’s signature and the access structure. The access structure is stored in the system while the cipher text is forwarded to the user where he can decrypt it with his/her private key to accept the access structure, shared key, and system public key.

• **Revoke a user – the data owner will**

– Define a minimal set of attributes so the leaving user’s access structure will not be satisfied and then update them.
– Next update the secret keys for all users except for the one to be revoked.
– Finally re-encrypt all the DEKs for all affected data files.

• **Download a file**

– Obtain the encrypted DEK for the requested file from TA and then call the decryption algorithm.

---

**Algorithm 1 Upload Algorithm**

1. procedure UploadFile(D, PK, DKE, A, TTL)  
   2. ‣ encrypt the input data with DKE  
   3. buffer ← DataEncrypt(DKE, D)  
   4. if connect(DataServer)=fail then  
      5. return fail  
   6. end if  
   7. ‣ create a file in the storage server  
      8. ‣ and write buffer into it  
   9. DataServer.file ← buffer  
   10. CM ← DKEEncrypt(PK, DKE, A)  
   11. ‣ k is count of data servers in CloudSky  
      12. ‣ use Shamir’s algorithm to get key shares  
   13. sharedkeys[1...k] ← ShamirSecShare(m, k, CM)  
   14. for ∀i ∈ [1...k] do  
      15. if connect(DataServer[i])=success then  
        16. create_object(sharedkeys[i], TTL)  
        17. else  
          18. for ∀j ∈ [1...i] do  
            19. delete key shares created before this one  
          20. end for  
        21. return fail  
      22. end if  
   23. end for  
   24. return success  
end procedure
• Integarity Check

- *CloudSky saves the HMAC of* the data to provide integrity check.
- *This check can be done at either file open stage or read stage.*
  - *Open Stage: HMAC of file must be saved in metadata and then compared after the file is fully read.*
  - *Read Stage: System needs to compute HMAC on per-block basis and save it for subsequent comparison*
- *CloudSky supports integrity in both stages.*

• Controllable Data Self-Destruction

- *When an owner calls the file upload operation, s/he include the system public key PK, the access structure, and the time-to-live (TTL) for the encrypted file*
- *CloudSky achieves data self-destruction by leveraging the capability of the active storage network where it has a time-to-live value property.*
- *Active storage object is constructed in CloudSky by using TTL along with the encrypted key shares generated by Shamir’s secrete sharing algorithm when the file is uploaded.*
- *TTL value is used to calculate the deadline such that (deadline = current time + ttl) which triggers the erasing of the encryption key on disk.*
• **Implementation** – *in the implementation*

  – The client node is implemented so that it supports two kernel modules
    – Cryptographic Object File System (COFS): compliant with virtual file system (VFS)
    – ISCSI initiator driver: provides iSCSI transport to access remote iSCSI targets over IP networks

  – MDS executes as a user level process that implements the functionalities of the SM and P/SM, while OSD implements the SCSI OSD command sets which are compliant to the T10-OSD standard specification.

  – OSD includes two kernel modules
    – The Object-Based File System (OBFS) and the iSCSI target driver.
    – The iSCSI target driver accepts and decapsulates iSCSI PDUs from the iSCSI initiator driver and presents the decapsulated SCSI OSD commands to OBFS. OBFS processes the SCSI OSD commands and manages the physical storage media.
• **Implementation** – continued …

– *CloudSky uses the crypto API from Linux kernel. The API provides a variety of encryption algorithm, such as AES, MD-5, SHA-1, SHA-2, etc.*

– *CloudSky uses the read and write operations provided by Linux.*

– *CloudSky safely deletes sensitive data.*

– *The global access list on MDS are stored in the form of LDAP directory.*

– *The experiments are conducted on 4 to 18 Linux hosts. Each host is connected to a RAID controller that is attached to 7 SATA disks (one for the OS and the rest for data).*

– *The performance overhead of CloudSky compared with OSD-3, which complies with T10 OSD- 3 security protocol.*
• **Results**

– Authentication and authorization overhead

• 512 files are used, each of 4KB size
• two configurations: one store a file in a single OSD, the other strips the file across 10 OSDs
• *CloudSky*’ write op is 1.031 (1 OSD) and 1.021 (10 OSD) times slower than OSD-3
• *CloudSky*’ read op is 1.026 (1 OSD) and 1.024 (1 OSD) times slower than OSD-3
• **Results**

– **Authentication and authorization overhead ... continued ...**

*Average Response Time*

– Intensive metadata request on the metadata server and the client nodes.
– Each server running 20 processes and each sending 100 file creation, 100 file read, and 100 file write request.
– Read and write requests are cached thus the average response time is much less than it is for create operation.
– When the number of requests are up to 4000, the capacity of the metadata server is saturated and therefore the average response time is increasing dramatically.

![Graphs showing MDS latency for different operations](image)
• **Results**

– *Throughput and scalability*

How would the performance degrade under high-bandwidth workload when CloudSky is added to a secure object-based storage system?

- Both OSD-3 and CloudSky bandwidth (throughput) increases with the transfer size.
• **Results**

  – **Throughput and scalability**

  *How would the performance degrade under high-bandwidth workload when CloudSky is added to a secure object-based storage system?*

  *The aggregate read throughput of all clients for the OSD-3 and CloudSky setups scales linearly with the number of clients/OSDs, which indicates that the MDS imposes very low overhead on a high-bandwidth workload and is not a bottleneck in the OSD-3 and CloudSky with*
• **Results**
  – **Idle CPU Time**

![Average percentile of idle CPU time on the metadata server](image)

The MDS’s idle CPU time decreases as the number of clients/OSDs increases. However, CloudSky imposes additional overhead.
• **Results**
  – encryption/decryption latency
• **Summary**

• *CloudSky integrate the ABE-based algorithms into a data self-destruction framework, and leverage active storage techniques* to achieve the controllability at various levels in an untrusted cloud storage network

• *The introduced overhead of CloudSky is small. Compared to the same system with OSD-3 security*