Secure and Efficient Access to Outsourced Data

Weichao Wang, Zhiwei Li, Rodney Owens, Bharat Bhargava

CCSW 2009: The ACM Cloud Computing Security Workshop

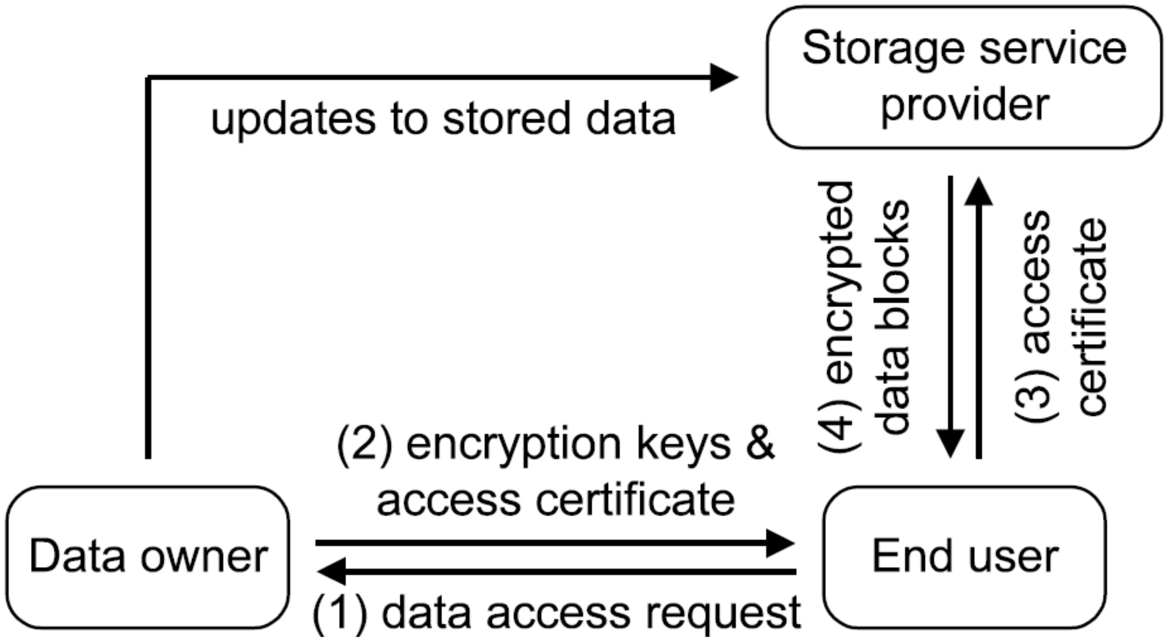
The Problem

- Providing secure and efficient access to outsourced data
  - An important component of cloud computing
  - Foundation for information management and other operations

- the security guidance published by Cloud Security Alliance
  - strong encryption and scalable key management
  - information lifecycle management
  - system availability and performance

Investigated Environment

- Owner-write-user-read Scenario
  - Data can be updated only by the original owner
  - Users read the information according to access rights
  - Example Application: LHC (Large Hadron Collider)



The Solution

- Fine grained access control to outsourced data
  - encrypt every data block with a different symmetric key
- Flexible and efficient management
  - adopt the key derivation method to reduce the number of secrets maintained
- Data isolation among end users
  - adopt over-encryption
  - lazy revocation
- Mechanisms to handle dynamics in both user access rights and outsourced data

Fine grained access control

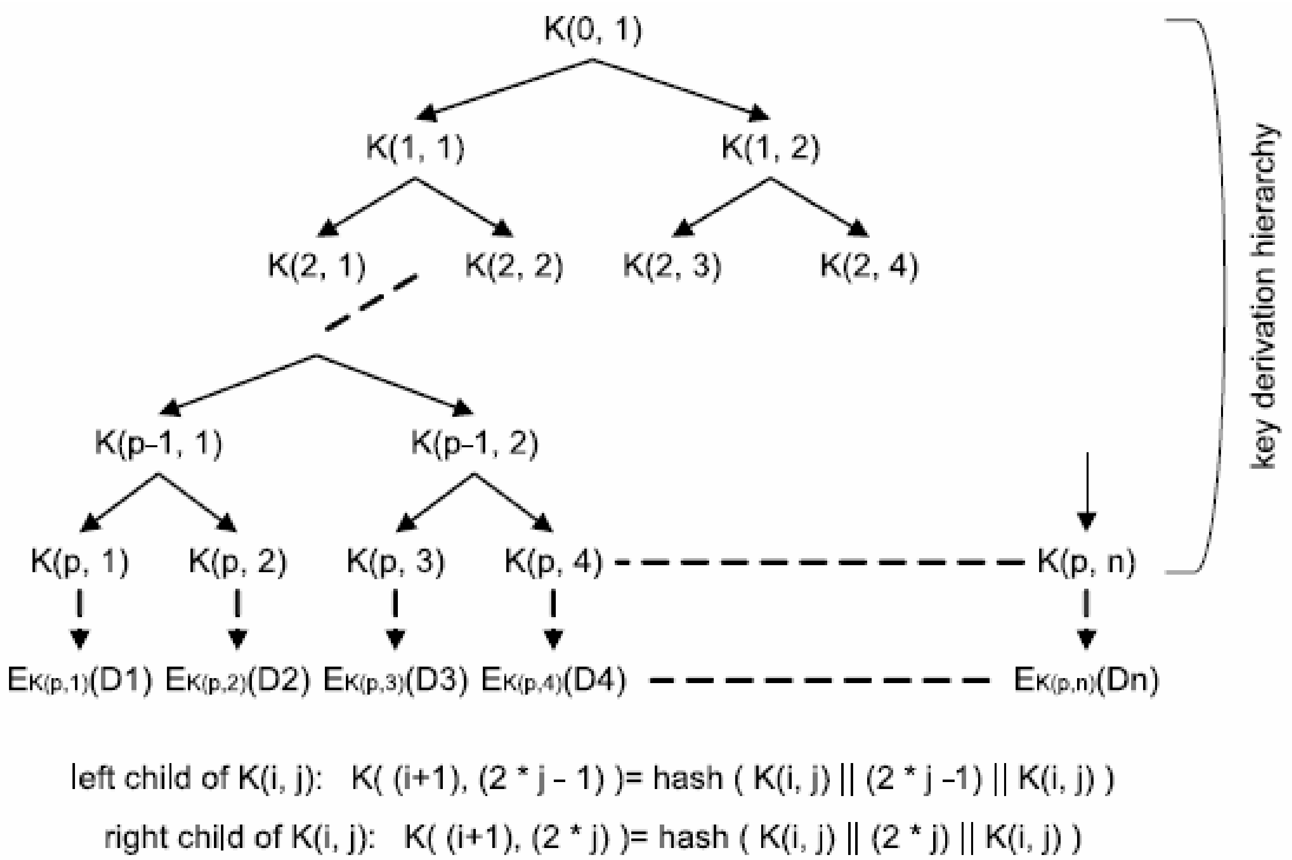
- Encrypt every data block with a different symmetric key
  - Data blocks $\{D_1, D_2, \cdots, D_n\}$
  - Encryption keys k_i $(i=1\ to\ n)$

- Worst case
  - Storage overhead linear to *n*
  - Communication overhead linear to *l*

Key-derivation-based data block encryption

- Key derivation method
  - 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
  - Calculation of one-way functions

Key derivation hierarchy



Issues of the key hierarchy

- Account for data updates
  - leave some room for the insertion and appending operations
- Only distribute necessary keys
  - we should not disclose keys of the blocks that are temporarily missing
- Impact of users' access rights on the communication overhead
  - organize data blocks with similar access patterns into groups

Data Access Procedure

1. (End user) sends a data access request to the data owner

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

2. (Data owner) authenticate the sender, verify the request, and determine the smallest key set

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

- *K'*
- *ACM index*
- *cert*

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

Data Access Procedure

3. (End user) sends $\{\mathcal{U}, \mathcal{S}, request\ index, cert\}$ to the service provider

4. (Service provider) verify the *cert*, check the user and ACM index, and retrieve data blocks and conduct the over-encryption

5. (End user) receive the data blocks, use seed and K' to derive keys, and then recover the data

Over-encryption

- Confidentiality of the outsourced data
  - Prevent revoked users from getting access to out-sourced data through eavesdropping
- *P()*: a pseudo random bit sequence generator
  - Shared between service provider and end users
- Given a *seed*, *P()* can generate a sequence of pseudo random bits
- Procedure
  - Use *seed* and P() generate a sequence of pseudo random bits
  - Use this bit sequence as one-time pad xor it to the encrypted block

Dynamics in User Access Rights

- Grant Access Right
  - Change access control matrix
  - Increase the value of ACM index
  - Service provider and the end user do not need to change

Dynamics in User Access Rights

- Revoke Access Right
  - Depends on whether or not the service provider conducts over-encryption

- If service provider conducts over-encryption
  - (Owner) updates the access control matrix and increase the ACM index
  - (Owner) send the new ACM index to the service provider until it receives acknowledgement

- If service provider refuses to conducts over-encryption
  - Adopt the lazy revocation method to prevent end users from reading updated blocks
  - trades re-encryption and data access overhead for a degree of security

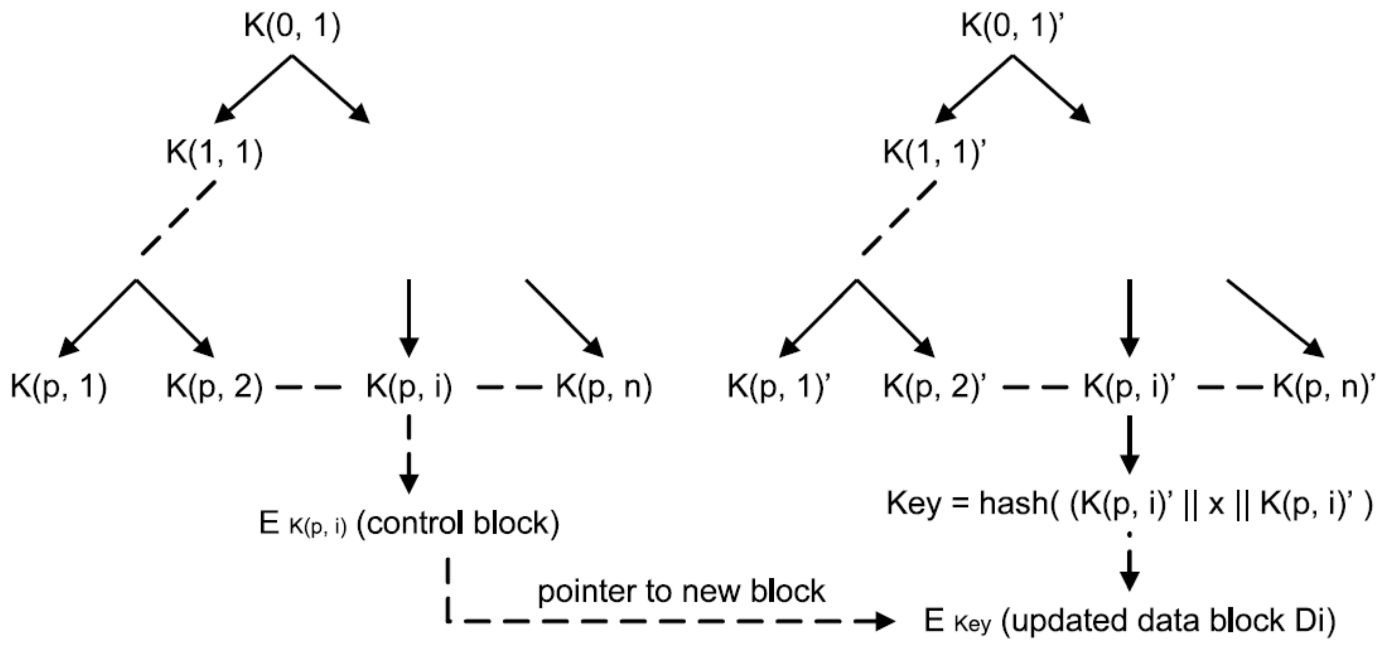
Dynamics in Outsourced Data

- Block Deletion
  - use a special control block to replace
  - label non-existence in the access control matrix

- Block Insertion /Appending
  - locate an unused block index
  - derive the encryption key
  - encrypt the data block
  - store it on the service provider
  - insert new data blocks based on their access patterns

Dynamics in Outsourced Data

- Block Update



Control block:
(1). Pointer to the new data block
(2). Information used to derive the encryption key of Di'
(3). Information to verify integrity

Overhead of the proposed approach

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	–

Outsourced data size: 10 PB
Data block size: 4 KB
Key hierarchy height: p = 42

User retrieve 1GB=250,000 blocks

Comparison to approach proposed by Atallah et al. (*CCS'05*)

- Their approach is more generic

- However, our approach
  - has less communication and storage overhead for data retrieval when they have infrequent update operations
  - handles user revocation without impacting service provider (over-encryption, lazy-revocation)

Conclusion

- Propose a mechanism to achieve secure and efficient access to outsourced data in owner-write-users-read applications.
- Analysis shows that the key derivation procedure based on hash functions will introduce very limited overhead.
- 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.

Future work

- Design a new scheme for key management for many-write-many-read applications
- Further reduce the number of keys by recognizing the access patterns of data blocks
- Develop a new approach to secure Storage-as-a-Service.