2013

# Data security and integrity in cloud computing

Miao Zhou
*University of Wollongong*


# Data Security and Integrity in Cloud Computing

A thesis submitted in fulfillment of the
requirements for the award of the degree

**Doctor of Philosophy**

from

UNIVERSITY OF WOLLONGONG

by

**Miao Zhou**

School of Computer Science and Software Engineering
October 2013

*Dedicated to*

*My family*

# Declaration

This is to certify that the work reported in this thesis was done by the author, unless specified otherwise, and that no part of it has been submitted in a thesis to any other university or similar institution.

Miao Zhou
October 26, 2013

# Abstract

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., network, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. During the last a few years, data security and integrity in cloud computing has emerged as a significantly important research area that has attracted increasing attention from both industry and academia. The virtual environment of cloud computing allows users to access computing power that exceeds what is contained within their own physical worlds. To enter this virtual environment, cloud users must transfer data throughout the cloud. Typically, cloud users know neither the exact location of their data nor the other sources of the data collectively stored with theirs. Consequently, several data security and integrity concerns have arisen, including key management, access control, searchable encryption techniques, remote integrity checks and proof of ownership in the cloud.

The first aspect of the work presented in this thesis is tree-based key management in cloud computing. Data encryption before outsourcing to the cloud is a common way to protect data privacy. Thus, key management is a challenging issue in cloud computing. It is the ability to correctly assign, monitor and secure keys which defines the level of operational security provided by any encryption implementation. The fundamental idea of this work is to design a secure and flexible key management mechanism for the outsourced data in cloud computing. In this thesis, an innovative tree-based key management scheme is proposed. The outsourced database remains private and secure, while some selected data and key nodes are shared with other parties in the cloud. Flexibility of key management is achieved and the security is proved in the standard model.

The second aspect of the work presented in this thesis is fine-grained access control. In order to secure the outsourced data in the cloud, designing efficient

and secure access control is a challenging issue. Unlike traditional access control in which the data users and storage servers are in the same trust domain, access control techniques are very different in cloud computing, as the cloud servers are not trusted by most cloud users. The key idea of this work is to attribute sets-based access control. This thesis points out that any access policy can be defined as a logical expression formula over different attribute sets. Logical expression indicates what kind of user is allowed to access the data. A fine-grained and efficient access control is proposed, based on logical expression.

The third aspect of the work presented in this thesis is efficient searchable encryption techniques in cloud computing. Because the data is usually encrypted before being outsourced to the cloud, searching the encrypted data in cloud computing has recently gained attention and led to the development of efficient searchable encryption techniques. The fundamental idea of this work is to reduce the search cost on encrypted data. In this thesis, a practical keyword searching mechanism is proposed. The solution is very simple. It enables efficient multi-user keyword searches and hides the private information in the search queries. The security is proved in the standard model.

The fourth aspect of the work presented in this thesis is public remote data integrity checks. As the clients store important data in remote cloud storage without a local copy, it is important to check the remote data integrity. Design of efficient remote integrity check protocols without downloading the data is a challenging issue in cloud computing. The key idea of this work is a public remote integrity check based on zero-knowledge proof. In this thesis, an innovative public remote integrity check scheme (PRIC) is proposed. No information of either the verified data or the homomorphic tags is leaked. In addition, the experiment result shows that PRIC is efficient, especially when the data size is large or the integrity check is frequent. The security of PRIC is proved in the random oracle model.

The last aspect of the work presented in this thesis is proof of multiparty ownership for encrypted data in the cloud. There are many applications of ownership sharing by different users and the design of the proof protocols of joint ownership is a challenging issue. Meanwhile, the design of proof-of-ownership mechanisms for encrypted data is even more difficult. This is because encryption of the same file by different users with random keys results in different ciphertexts, and the cloud server cannot store the same hash root value for ownership verification. In this thesis, a proof of multiparty ownership solution (PMOW) with encrypted data is proposed.

Every user can prove that he/she holds the plaintext of the encrypted file when the server stores one ciphertext only. In addition, a PMOW system is constructed. The security of PMOW is proved in the ideal cipher model.

The major contribution of this thesis is innovative and improved approaches to secure data in cloud computing. Using these approaches developed, a trustworthy cloud environment can be achieved.

# Acknowledgement

I am most grateful to my supervisor Prof. Yi Mu, for his support and guidance of this thesis. He has been providing invaluable suggestions and encouragement from the beginning of my research. This thesis would never been possible without his help and support.

I would like to thank my co-supervisors Prof. Willy Susilo and Dr. Man Ho Au, for their continuous guidance in the process of conducting this research. I would also like to thank Dr. Jun Yan, for his helpful discussions and suggestions during the research progress.

During my PhD study from July 2009 to present, I also received lots of advice and support from my colleagues and friends. The non-exhausted list includes: Minjie Zhang, Guilin Wang, Madeleine Cincotta, Jinguang Han, Fuchun Guo, Nan Li, Wei Wu, Xinyi Huang, Yong Yu, Yang Wang, Zhenfei Zhang, Bill Tourloupis, Thomas Plantard, Yafu Ji, Guomin Yang, Rui Zhang, Tsz Hon Yuen and Jiangshan Yu.

I am also grateful to University of Wollongong and Smart Services CRC, for their scholarships that help me to achieve my goals.

I would like to thank my parents and my husband Hao Gao, for their patience and love. Without them, this work would never be possible.

# Publications

- Miao Zhou, Yi Mu, Willy Susilo, Jun Yan and Liju Dong. Privacy enhanced data outsourcing in the cloud. *J. Network and Computer Applications*, 35(4):1367-1373, 2012.
- Miao Zhou, Yi Mu, Willy Susilo and Jun Yan. Piracy-preserved access control for cloud computing. In *IEEE TrustCom*, pages 83-90, 2011.
- Miao Zhou, Yi Mu, Willy Susilo and Man Ho Au. Privacy-enhanced keyword search in clouds. In *IEEE TrustCom*, pages 89-94, 2013.
- Miao Zhou, Yi Mu, Willy Susilo and Man Ho Au. Public remote integrity check for private data. (under review)
- Miao Zhou, Yi Mu, Willy Susilo and Man Ho Au. PMOW: proof of multiparty ownership for encrypted data in clouds. (under review)

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

There are a number of attempts to define cloud computing in various ways. Among definitions, the widely accepted one is proposed by [MG09] as follows: 'cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., network, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction'. There are five essential characteristics of the cloud model [MG09]:

- Rapid elasticity. The computing capabilities can be elastically provisioned and released. To the cloud users, the capabilities available for provisioning appear to be unlimited and can be assigned in any quantity at any time.
- Service on demand. The cloud users can use on-demand services such as server time and network storage, without requiring human interaction with each service provider.
- Broad network access. The cloud services are always available over the network and can be accessed by different user platforms.
- Location independence. The cloud provider's computing resources are pooled to serve multiple users using a multi-tenant model. It is location independence that the cloud users have no control or knowledge over the exact location of the provided resources.
- Measuring service. Cloud resource usage can be monitored, controlled and reported by the cloud providers.

The above advantages of cloud computing have dramatically changed the IT scene, as cloud computing offers cost savings and improvements to major operations. For example, cloud computing offers individuals and companies affordable

storage, professional maintenance and adjustable space. Many large technology companies (e.g., Amazon, Google, Microsoft) have built huge server centres to offer cloud computing with virtual applications, various services and business software with self-service interface so that cloud users can use on-demand resources with location independence [Erd09].

Such indirect control of the physical infrastructure, however, introduces vulnerabilities unknown in previous settings. Since resources are provided over the internet, the cloud is a single point of access for all the users. The cloud model for delivering computing power and processing, has raised several security questions. Ensuring security is more difficult when designing cloud architectures than in non-cloud contexts [GMCL09, RVR+07, BYV+09]. The security issues in cloud computing include: 1) data security, 2) identity and access control, 3) key management and 4) virtual machine security. Among these main security issues in the cloud, data security and integrity is believed to be the most difficult problem which could limit the use of cloud computing [RTSS09]. In fact, access control and key management are all issues involved in data security.

Data security in the cloud refers to data **confidentiality**, **integrity**, **availability** and **traceability** (CIAT), and these requirements pose major problems for cloud computing. Data confidentiality requires that information be available or disclosed only to authorized individuals, entities or IT processes. Data integrity ensures that the data is maintained in its original state and has not been intentionally or accidentally altered or deleted. Data availability ensures continuous access to data even in the event of a natural or man-made disaster or events such as fires or power outages. Data traceability means that the data and communications are genuine in a transaction and that both parties involved are who they claim to be. Specifically, to achieve the above requirements of CIAT, the critical security challenges of data security in the cloud can be mainly outlined as follows: **1) key management**, **2) access control**, **3) searchable encryption techniques**, **4) remote integrity check** and **5) proof of ownership**.

## 1.1 Motivations and Contributions

This thesis addresses the above critical data security challenges in the cloud. Security mechanisms are designed to ensure a trustworthy cloud environment.

**Key Management.** Data encryption before outsourcing to the cloud is a common and simple way to protect data privacy. Although the encryption algorithms are public, information encrypted under these algorithms is secure because the key used to encrypt the data remains secret. As a result, key management is a critical element in cloud computing. It is the ability to correctly assign, secure and monitor keys that defines the level of operational security provided by any encryption implementation.

The classical tree-based hierarchy schemes such as RFC2627 [WHA99]) and the scheme proposed by Wong *et al.* [WGL98] have been widely used in group key management. In RFC2627 [WHA99], the hierarchical tree approach is the recommended approach to address the multicast key management problem. Many key management methods of access hierarchies for data outsourcing have been proposed [WLOB09, DdVJ+03, BCdV+09, dVFJ+07a, ABFF09, PL01] based on this approach. These methods provide some useful solutions to minimize the number of cryptographic keys which have to be managed and stored. Aiming to provide secure and efficient access to outsourced data, Wang *et al.* [WLOB09] proposed a tree-based cryptographic key management scheme for cloud storage. Their tree-based key management structure is similar to a traditional one, where a single root node holds the master key that can be used to derive other node keys. Each node key can be used to derive the keys of its children in the tree hierarchy. With their scheme, a data block stored in the cloud can be deleted or updated by a party who holds either the specific decryption key or a node key corresponding to one of its parents. If there is an outsourcing server authorized to manage a node (not the root node) that has several child nodes, then the outsourced party is granted the node key, which can be used to derive all sub-keys for its child nodes. In other words, once a parent node in the tree is given, all the child nodes will be known. This is a common problem which exists in many tree-based key management schemes. Existing ones such as [WLOB09, DdVF+05, DdVF+07, dVFJ+08, dVFJ+07b, AFB05, YLZL01, KPT00, NNL01a, LNS03, KLKJ06, SL04, BSS11] can work perfectly, only if when all legitimate node users are authorized to access all the child nodes under the specific parent node.

Considering the above problem, an innovative tree-based key management scheme is proposed. With the proposed key management scheme, the outsourced database can remain private and secure, while some selected data and key nodes are shared with other parties. The proposed scheme is also a flexible key management solution that can be adapted and expanded for different scenarios (e.g., multiple branches

and multiple sub-keys). The security of the proposed scheme is proved in the standard model. The proposed tree-based outsourcing key management opens up an entirely new approach for secure and flexible key management.

**Access Control.** Unlike the traditional access control in which the data users and storage servers are in the same trusted domain, access control techniques are very different in cloud computing because the cloud servers are not seen as trustworthy by most cloud users, especially large enterprises and organizations. One possible method to enforce data access control without relying on cloud servers could be to encrypt data individually and disclose the corresponding decryption keys only to the privileged users, but that causes high performance costs. A fine-grained access control which is efficient and secure is important and necessary for cloud computing.

Several advancements have already been proposed [LZC+10, LWG11, CSK11, BLLS11, ZLS11, Lee12, ZHA+12, WeLD12, ASB+12, YLJ12, SWYW12, GM09, HYJZ09]. An existing feasible solution to achieve fine-grained access control of outsourced data in cloud computing is to encrypt the data through certain cryptographic primitives and only disclose the private keys to authorized users. Without the appropriate decryption keys, unauthorized users including the cloud providers, cannot decrypt the data. This solution has been widely used (such as [dVFJ+07b, KRS+03]) and most schemes using it are deployed by introducing a per file groups for efficiency. However, the complexity of these schemes [BKP09, HCM01, CHR09, TWZ09, YC10, LYRL10, TS11] is proportional to the system scale and the number of users. Additionally, this solution lacks scalability and flexibility, especially if the number of authorized users becomes large.

Vimercati *et al.* [dVFJ+07b] proposed an access control scheme for securing data stoppage on untrusted servers based on key derivation methods [AFB05]. In their scheme, the owner created corresponding public tokens to grant access for a user. With his secret key, the user was able to derive decryption keys for desired files. However, the complexity of operations of file creation and user grant/revocation is also linear to the number of users. Ateniese *et al.* [AFGH05] proposed a secure distributed storage scheme based on proxy re-encryption. Specifically, the data owner encrypted blocks of content with a master public key, which could only be decrypted by the master private key. The data owner then generated proxy re-encryption keys by using his master private key and the user's public key. With the proxy re-encryption keys, the semi-trusted proxies could convert the ciphertext into

another ciphertext for a specific user. Thus, they achieved access control. However, the main problem with this scheme is that user access privilege is not protected from the proxy.

Recently, ABE has been seen as an ideal technique for achieving flexible, scalable and fine-grained access control mechanisms in the cloud. Wang *et al.* [WLW10] proposed a hierarchical attribute-based encryption scheme to achieve fine-grained access control in cloud storage services by combining hierarchical identity-based encryption and ciphertext-policy attribute-based encryption (CP-ABE). In their scheme, they assumed that all attributes in one conjunctive clause were administered by the same domain master. However it is difficult to implement in practice so that the same attribute can be administrated by multiple domain masters according to specific policies. Another difficulty is that their scheme cannot support compound attributes efficiently and does not support multiple value assignments. Yu *et al.* [YWRL10] stretched out a scalable data access control scheme for cloud computing which combines techniques of key-policy attribute-based encryption (KP-ABE), proxy re-encryption and lazy re-encryption. In their scheme, the cloud servers were unable to learn the plaintexts though data re-encryption, however the encryptor in their scheme is not able to decide who can decrypt the encrypted data and has no choice but to trust the key issuer. In addition, the cloud servers must learn the whole user ID list and attribute list in exchange.

To tackle these problems, an encryption system is proposed to achieve practical, flexible and fine-grained access control on outsourced data. In particular, the problem of defining and assigning keys to users based on different attribute sets is concerned. The access policies are hidden as well as user information to the third-party cloud servers. Through the protocol executions, there is no leakage of users' privacy. The proposed scheme is partially based on the observation that, in practical application scenarios, each user can be associated with a set of attributes, which are meaningful in the access policy and data file context. The access policy can thus be defined as a logical expression formula over different attribute sets. The logical expression can indicate what kind of user is allowed to access the data file. As any access policy can be represented as such a logical expression formula, fine-grained access control can be achieved.

**Searchable Encryption Techniques.** As the data is usually encrypted before being outsourced to cloud servers, how to search the encrypted data in the cloud

has recently gained attention and led to the development of searchable encryption techniques. This problem is challenging however, because meeting performance, system usability and scalability requirements is extremely difficult.

In theory, the classical work of Goldreich *et al.* [GO96] on oblivious random-access machines (RAMs) could resolve the problem of doing (private) searches on remote encrypted data. They enabled a client to store only a constant amount of data in local storage. Meanwhile, the identities of the remote data files were hidden when the client accessed them. Oblivious RAM is often cited as a powerful tool, which can be used, for example, for search on encrypted data or for preventing cache attack. However, oblivious RAM is also commonly considered to be impractical due to its overhead. Suppose the client stores $n$ data files in remote storage, then each data request is replaced by $O(\log^4 n)$ or $O(\log^3 n)$ requests. Additionally, $O(n \cdot \log n)$ external memory is required in order to store the $n$ data files.

In an effort to reduce the round complexity associated with oblivious RAMs, Song *et al.* [SWP00] presented a solution for searchable encryption. After that, the question of how to do keyword searches on encrypted data efficiently was raised. In [SWP00], they achieved searchable encryption by constructing a special two-layered encryption for each word. Given a trapdoor, the server could strip the outer layer and assertain whether the inner layer was in the correct form. The limitations in this construction are as follows. First, it is not compatible with existing file encryption schemes and a specific encryption method must be used. Second, while the construction is proven to be a secure encryption scheme, it is not proven to be a secure searchable encryption scheme. Third, the distribution of the underlying plaintext is vulnerable to statistical attacks. Their approach may leak the locations of the keyword in a file. Finally, their searching time is linear in the length of the document collection.

The above limitations are addressed by Goh [jG03], Chang and Mitzenmacher [CM05] and also Curtmola *et al.* [CGKO06], etc. In [jG03], they built an index of keywords for each file using a Bloom filter with pseudo-random functions used as hash functions. One inherent problem with this Bloom filter-based approach is that Bloom filters can induce false positives, which would potentially cause mobile users to download extra files not containing the keyword. In [CM05], Chang and Mitzenmacher achieved the notion of security to IND2-CKA for chosen keyword attack, however their scheme cannot guarantee that the trapdoors do not leak any

information about the words being queried. In [CGKO06], they proposed a multi-user construction that is efficient on the server side, however every node in the link list has to be augmented with information about the file index of the next node.

In a different direction, Boneh, di Crescenzo, Ostvrosky and Persiano [BCOP04] and Boneh, Kuchilevitz, Ostvrosky and Skeith [BKOI07] studied the problem of how to search on data encrypted by a public-key cryptosystem. These schemes are motivated by an encrypted email system. Their constructions, however, have an overhead in search time that is proportional to the square root of the database size, which is far less efficient then the best private-key solutions. Boneh *et al.*'s approach [BCOP04] is known to be the seminal public key encryption scheme with keyword search (PEKS). It was observed in [BSNS05] that Boneh *et al.*'s scheme [BCOP04] requires a secure channel, which makes it impractical.

To reduce the search cost on encrypted data and design a practical searching mechanism in cloud computing, an efficient keyword search scheme for cloud computing is proposed. The proposed solution is very simple. It enables efficient multi-user keyword searches over outsourced data files in the cloud environment, without leaking any private information about either the data owner or users in the search query.

**Remote Integrity Check.** Storing data in remote cloud servers has become common practice. As clients store their important data in remote cloud servers without a local copy, it is important to check the remote data integrity (RIC). While it is easy to check data integrity after completely downloading the data, it is a large waste of communication bandwidth. Hence, designing efficient remote integrity check protocols without downloading the data is an important security issue in the cloud.

Early verification schemes [BEG$^+$94, BGG95, NR05] concentrated on the problem of data integrity on a local untrusted memory, i.e., memory checking. The challenging problem of data integrity verification without explicit knowledge of the full file was first proposed in broad generality by Blum *et al.* [BEG$^+$94], who explored the task of checking the correctness of a memory-management program efficiently. Naor and Rothblum [NR05] explored the problem of dynamic memory-checking in a range of settings. Clarke *et al.* [CSG$^+$05] focused on a trusted party which stores a small amount of state information, verifying the integrity of arbitrary blocks of external memory. These early verification schemes are the first to suggest checking data integrity, however they are not applicable for remote data integrity check

because they require the data to be transmitted in its entirety to the verifier.

The latest verification schemes concentrated on the problems of securing data integrity at remote servers and securing cloud storage applications. These schemes can be classified into *'Proof of retrievability'* (POR) schemes (e.g., [JJ07, BJO09, DVW09, SW08]) and *'Provable data possession'* (PDP) schemes (e.g., [ABC$^+$07, ABC$^+$11, EKPT09]). A POR scheme is a challenge-response protocol. In POR schemes, a cloud provider demonstrates the file retrievability (i.e., recoverability without any loss or corruption) to a client. PDP schemes are similar protocols which only detect a large amount of corruption in outsourced data. Several advances have already been proposed [SBMS07, OR05, DQS04, FB06, SBD$^+$04, SM06, JJ07, DVW09, ABC$^+$07, ABC$^+$11, WWL$^+$09, WCW$^+$09]. In Shah *et al.* [SBMS07], they introduced a third-party verifier who could delegate the periodic task of checking data integrity. In their scheme, they reduced the client's storage, communicational and computational cost. This simple solution, however, requires a third-party verifier to keep a lot of hash values of the data blocks. In Oprea *et al.* [OR05], they allowed a client to detect the modification of data blocks by a remote and untrusted server. Their protocol did not bring additional storage cost to the server and the client, but the entire file had to be retrieved during the verification executions. In addition, the communication complexity is linear in the file size. Deswarte *et al.* [DQS04] and Filho *et al.* [FB06] proposed techniques to verify data integrity using RSA-based hash functions. Their schemes allowed a verifier to perform multiple challenges using the same metadata. The limitation of their algorithms lies in the computational complexity at the server. The computation costs must exponentiate the entire data file. In addition, RSA over the entire file is extremely slow. As is showed in Filho *et at.* [FB06], it required 20 seconds per megabyte for 1024-bit keys on a GHz3.0 CPU. Yamamoto *et al.* [YOA07] presented an efficient scheme for large data integrity check. Their scheme was based on homomorphic hash functions. The advantage of their scheme is batch processing [CY07] for a homomorphic hash function. Similar techniques can also be found in Sebe *et al.* [SBD$^+$04]. In [SBD$^+$04], they presented a protocol based on the Diffie-Hellman problem in $\mathbb{Z}_N$, however the client has to store $N$ bits per data block ($N$ is the RSA modulus). The total storage cost on the client side is $O(n)$. Juels *et al.* [JJ07] proposed a scheme for proof of retrievability by using 'sentinels'. The sentinels (special blocks) were hidden among other blocks in file $F$. The verifier challenged the prover by specifying the positions of a collection of sentinels and asking the prover to return the associated

sentinel values. Their scheme is limited as they can handle only a limited number of queries and increase storage overhead on the server side. In Dodis *et al.* [DVW09], they improved the POR constructions. They built nearly optimal POR codes using hitting samplers and error-correcting codes. Ateniese *et al.* [ABC$^+$07, ABC$^+$11] described a proof of data possession (PDP) scheme that improved the response length of the simple hash scheme by using homomorphic verifiable tags. In their scheme, they constructed the homomorphic verifiable tags $\{T_i\}$ $(1 \leqslant i \leqslant n)$ for $n$ data block $\{m_i\}$. Later, the prover sent a linear combination of blocks $\sum_i a_i m_i$ (with arbitrary coefficients $\{a_i\}$) to the verifier. The verifier cloud be convinced if $\sum_i a_i m_i$ was correctly generated using an aggregate tag $T$ computed from $\{T_i\}$. They also proposed a variant of their PDP scheme to achieve public verifiability under a weaker security model. Erway *et al.* [EKPT09] introduced a framework and efficient constructions for dynamic provable data possession which extends Ateniese *et al.*'s model [ABC$^+$07] to support provable updates. Their constructions captured the dynamic operations such as insertion in the middle of a file, however they are not efficient when moving and deleting the entire files. Shacham and Waters [SW08] proposed an HTAG scheme which used a simple homomorphic MAC and a universal hash family to reduce the communication bits to a constant factor of $k$. They also achieved public verifiability in which the third party verifier could extract the data file through multiple challenges and responses. Most previous public RIC schemes, however, which include [SW08, ABC$^+$07, ABC$^+$11] do not provide security and this means a public verifier can learn the information of private data during an integrity check, since the cloud server might leak the data information.

It is noticed that Hao, Zhong and Yue's public RIC scheme [HZY11] provided a security feature against data leakage. They adopted the approach introduced by Sebe *et al.* [SFB$^+$08] to support data dynamics and privacy, but their scheme requires the verifier to be in possession of all the homomorphic tags used for the integrity check. Therefore, it increases costs for storage and communication. In addition, although they prove that the public verifier cannot learn the target data, the tags themselves can leak some information about the data.

To overcome these problems, a privacy-preserving RIC protocol which achieves public verifiability without disclosing any information is proposed. No information about the original data will be leaked. In fact, the verifier is only required to know the public key of the data owner. The experimental results indicate that the proposed scheme is efficient, especially when the data size is large or the integrity

check is frequent. The full proofs of security under the random oracle model are also given.

**Proof of Ownership.** Beyond storage correctness, proof of ownership (POW) is another security issue related to cloud data storage. Client-side deduplication allows an attacker to gain access to arbitrary-size files when he has small hash signatures of the files. To overcome such attacks, the technique of POW allows a user to efficiently prove to a cloud server about his ownership, rather than short information about the file such as a hash value.

Proof-of-ownership (POW) is closely related to two other similar problems: proof of retrievability (POR) and proof of data possession (PDP). POR schemes [JJ07], [SW08], [BJO09], [DVW09], [ZX11] are challenge-response protocols. In POR schemes, a cloud provider demonstrates the file retrievability (i.e., recoverability without any loss or corruption) to the client. PDP schemes [ABC$^+$07], [ABC$^+$11], [EKPT09], [SFB$^+$08], [FB06], [SBD$^+$04], [WWL$^+$09, WWRL10, WWR$^+$11, ZB12, YOA07] are similar protocols which only detect a large amount of corruption in outsourced data. The main difference between POW and POR/PDP is that the latter usually uses a pre-processing step on the client side while the former does not.

POW protocols are proposed for client-side data deduplication which enables the storage server to store a single copy of repeating data. Client-side data deduplication has become popular and important as it removes data redundancy and data replication, but it brings many data privacy and security issues for the user. Douceur *et al.* [DAB$^+$02] first studied the problem of deduplication in a multi-tenant system in which deduplication had to be reconciled with confidentiality. Their proposed convergent encryption enabled two users to produce a single ciphertext for deduplication. As there are many security problems with convergent encryption, Storer *et al.* [SGLM08] proposed a security model for secure data deduplication. Recently, Harnik *et al.* [HPSP10] formally identified the security problems of client-side deduplication as follows: 1) The first kind of attacks attempted to fool the storage server and abuse the storage system. A malicious user with the hash signature of a file could convince the cloud server that he owns the file. By accepting the hash value as a 'proxy' for the entire file, the cloud server allowed anyone who held the hash value to access the entire file. 2) The second kind of attacks targeted the privacy and confidentiality of users of the storage system. A malicious user could check whether another honest user had already outsourced a data file by trying to upload

it as well. 3) The third kind of attacks focused on subverting the intended use of a storage system. For example, two malicious users tried to use the cloud storage for a covert channel as they might not have a direct interaction channel. The two users first pre-agreed on two different files. Second, one malicious user outsourced one of the two files. Then the other user could detect which file had been deduplicated and output either 0 (for the fist file) or 1 (for the second file). In this way, two malicious users successfully exchange a bit of information without a direct transmitting channel.

To overcome such attacks, Halevi *et al.* [HHPSP11] introduced the notion of POW for client-side deduplication. In addition, they presented Merkle tree-based schemes to allow a user to efficiently prove his ownership to the server, rather than some short information. However, their scheme cannot be adopted for encrypted file scenario, because encryption of the same file by different users with random keys results in different ciphertexts. The server cannot store the same hash root value for the ownership verification. Some other schemes [RMW12], [NWZ12], [PS12], [ZX12] focused on improving the efficiency of POW and applying an encrypted file scenario. In [PS12], Pietro and Sorniottis proposed three correlative protocols to achieve an efficient POW for deduplication. The main idea of their protocols is to challenge random $K$ bits of file $F$. The probability that a malicious user is able to output the correct value of $K$ bits of the file where each bit is selected at a random position is negligible in security parameter $k$, but their scheme cannot be adopted for encrypted files. In addition, the client's files are totally revealed to the cloud server during the protocol executions. In [NWZ12], they presented a private POW scheme for encrypted files in cloud storage. Zheng *et al.* [ZX12] argued that the public verifiability offered by POR/PDP schemes could be naturally exploited to achieve POW, however by using POR/PDP schemes to achieve POW, their scheme brings the clients mass information because it stores all the verified tags. In addition, none of the above schemes can solve the POW problem for multiparty users because the protocol execution was only related to the file $F$ that has to be proven.

There are applications of ownership sharing by different users. Thus an innovative proof of a multiparty ownership solution is proposed with the encrypted data in the cloud. Every user can prove that he holds the plaintext of the encrypted file when the server stores one ciphertext only.

## 1.2 Summary of This Thesis

The definition and advantages of cloud computing are revised in this chapter. The security issues in cloud computing are outlined, especially the security challenges for data security in the cloud. In particular, the aims of this thesis are illustrated and the contributions are summarized .

In Chapter 2, the notations and definitions which are used throughout this thesis are covered. The preliminaries and review background materials are presented.

In Chapter 3, a privacy enhanced key management scheme in the cloud is given. It allows a data source to be accessed by multiple parties who hold different rights. The security of the database is remained, while some selected data sources can be securely shared with other parties.

In Chapter 4, a solution is presented to achieve flexible and fine-grained access control on outsourced data files. In particular, the problem of defining and assigning keys to users is concerned. The access policies and users' information are hidden from third-party cloud servers. The proposed scheme is partially based on the observation that, in practical application scenarios, each user can be associated with a set of attributes which are meaningful in the access policy and data file context. The access policy can thus be defined as a logical expression formula over different attribute sets. The logical expression can indicate what kind of user is allowed to access the data file. As any access policy can be represented as such a logical expression formula, fine-grained access control can be accomplished.

In Chapter 5, an efficient keyword search scheme for cloud computing is proposed. The proposed solution is very lightweight. It enables efficient multi-user keyword searches over outsourced data files in the cloud environment, without leaking any private information about either the data owner or users in the search query. The security requirements are formally defined and the proposed scheme is proven secure under a simple assumption in the standard model.

In Chapter 6, a privacy-preserving RIC protocol which achieves public verifiability without disclosing any information is proposed. No information about the original data will be leaked. In fact, the verifier is only required to know the public key of the data owner. The experimental results indicate that the proposed scheme is efficient, especially when the data size is large or the integrity check is frequent. The full proofs of security under the random oracle model is also given.

Chapter 7 deals with proof of ownership as an important data security issue in

cloud computing, as mentioned earlier. There are many applications of ownership sharing by different users. In this chapter, an innovative **PMOW** scheme for proof of multiparty ownership is proposed, for encrypted data in the cloud. Every user can prove that he holds the plaintext of the encrypted file when the server stores one ciphertext only. The proposed solution achieves CCA2 security and The full proof analysis is also given in the ideal cipher model.

Finally, the thesis is concluded in Chapter 8 with a summary of the thesis together with some possible future research directions.

# Chapter 2

# Background

In this chapter, the notations and definitions which are used throughout this thesis are covered. The aim of this chapter is to make this thesis self-contained. Background materials on the topic of hash function, pseudo-random function, number-theoretic problems, bilinear maps, Merkle-tree protocols and zero-knowledge knowledge-of-proof protocols will be presented.

## 2.1 Preliminaries

### 2.1.1 Miscellaneous Notations

**Notations.** Throughout this thesis, $\mathbb{N}$ denotes the set of natural numbers $\{1, 2, \cdots\}$ and by $\mathbb{Z}$ the set of integers $\{\cdots, -2, -1, 0, 1, 2, \cdots\}$. $\mathbb{Z}_p$ denotes the set $\{0, \cdots, p-1\}$ and $\mathbb{Z}_p^*$ the set of positive integers smaller than $p$ and relatively prime to $p$. That is,

$$\mathbb{Z}_p^* = \{n | 1 \leqslant n \leqslant p \ and \ gcd(n, p) = 1\}.$$

The notation $[1, k]$ denotes the set $\{1, \cdots, k\}$.

**Functions and Algorithms.** Let $f : X_1 \to X_2$ be the function $f$ with input $X_1$ and output $X_2$. Let $A$ denote an algorithm. Let $A(\cdot)$ to denote that $A$ has one input and $A(\cdot, \cdots, \cdot)$ to denote that $A$ has several inputs. $y \leftarrow A(x)$ denotes that $y$ is the output of algorithm $A$ on input $x$.

**Experiments.** Let $S$ be a probability distribution which is a graph, table or formula that gives the probability for each value of the random variable. $x \leftarrow S$ denotes the experiment of sampling an element $x$ from a probability distribution $S$. If $F$ is a finite set, then $x \leftarrow F$ denotes the experiment of sampling uniformly from the set

$F$. Semicolon is used to describe the ordered sequences of event that make up an experiment, e.g.,

$$u \leftarrow S;\ (v, w) \leftarrow A(u)$$

**Probabilities.** If $pre(.,.)$ denotes a predicate, then

$$\Pr[u \leftarrow S;\ (v, w) \leftarrow A(u) : pre(v, w)]$$

is the probability that the predicate $pre(v, w)$ is true after the ordered sequence of events $(u \leftarrow S;\ (v, w) \leftarrow A(u))$. The notation

$$\{u \leftarrow S; (v, w) \leftarrow A(u) : (v, w)\}$$

denotes the probability distribution over $\{v, w\}$ generated by the experiment $(u \leftarrow S;\ (v, w) \leftarrow A(u))$. Following standard notation,

$$\Pr[A|B]$$

denotes the probability of event $A$ conditioned on the event $B$. When the $\Pr[B] = 0$, then the conditional probability is not defined.

**Big-O Notation.** The standard asymptotic notation is used to describe the running time of algorithms. The expression $f(n) = O(g(n))$ means that there exist some positive constant $c$ and a positive integer $n_0$ such that $0 \leqslant f(n) \leqslant cg(n)$ for all $n \geqslant n_0$. Broadly speaking, it means that $g$ is the upper bound of $f$. If $f$ is bounder below by $g$, that is, $g(n) = O(f(n))$, then $f(n) = \Omega(g(n))$.

### 2.1.2 Complexity Problems on $\mathbb{Z}_p$

The security of many cryptosystems relies on the intractability of solving some hard problems. In the following, three hard problems are described in detail. Let $p$ be a large prime and $q$ a prime divisor of $p - 1$. Let $\mathbb{G}$ be a subgroup in $\mathbb{Z}_p^*$ with prime order $q$. Let $g$ be the generator of $\mathbb{G}$.

- **Discrete Logarithm Problem.** The discrete logarithm problem (DLP) [BL96] forms the basis in the security of many cryptosystems. DLP is restricted in cyclic group in this thesis.

  **Definition 2.1** *The Discrete Logarithm Problem in $\mathbb{G}$ is defined as follows: On input a tuple $(g, Y) \in \mathbb{G}^2$, output $x$ such that $Y = g^x$.*

Shoup [Sho97] derived a lower bound on any algorithms that solve DLP without exploiting any special properties of the encoding of the group element. Such algorithms are known as generic algorithms. Specifically, the lower bound is $\Omega(\sqrt{d})$, where $d$ is the largest prime dividing the order of the group. Indeed, such bound is met by the well-known Pollard's rho algorithm [Pol78] that works in arbitrary groups.

- **Computational Diffie-Hellman Problem.** If DLP in $G$ can be solved, the computation Diffie-Hellman problem (CDH) can also be solved although whether the converse is true or not is still an open problem.

  **Definition 2.2** *The Computational Diffie-Hellman Problem in* $\mathbb{G}$ *such that* $|\mathbb{G}| = p$ *is defined as follows: On input a tuple* $(g, g^x, g^y) \in \mathbb{G}^3$*, output* $g^{xy}$*.*

- **Decisional Diffie-Hellman Problem.** The decisional Diffie-Hellman problem (DDH) is the decisional version of the CDH problem. It was first formally introduced in [Bra93].

  **Definition 2.3** *The Decisional Diffie-Hellman Problem in* $\mathbb{G}$ *such that* $|\mathbb{G}| = p$ *is defined as follows: On input a tuple* $(g, g^x, g^y, g^z) \in \mathbb{G}^4$*, decide if* $g^z = g^{xy}$*.*

### 2.1.3 Bilinear Maps

The necessary facts about bilinear maps and groups are briefly reviewed, in the notation of [BLS04]: Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two (multiplicative) cyclic groups of prime order $p$, with an additional group $\mathbb{G}_T$ such that $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T|$. A bilinear map is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

1. Bilinear: for all $u \in \mathbb{G}_1, v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degenerate: for $g_1$ is a generator of $\mathbb{G}_1$ and $g_2$ is a generator of $\mathbb{G}_2$, $e(g_1, g_2) \neq 1$.

These properties imply another two properties: for any $u_1, u_2 \in \mathbb{G}_1$, $v \in \mathbb{G}_2$, $e(u_1 u_2, v) = e(u_1, v) \cdot e(u_2, v)$; and for any $u, v \in \mathbb{G}_2$, and $\phi$ is a computable isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$, with $\phi(g_2) = g_1$, then $e(\phi(u), v) = e(\phi(v), u)$.

## 2.2 Cryptographic Tools

### 2.2.1 Cryptographic Hash Functions

A hash function, $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ is an efficient one-way algorithms that maps an input of an arbitrary-length bit-string $x$ to an output $H(x)$ of fixed length $\lambda$. The cryptographic hash functions must be efficiently computable. The hash function is required to be *pre-image resistant*. That is, given a hash function $H$, it is computationally hard to find $x$ when given $y$, such that $y = H(x)$. The hash functions are also required to be *collision resistant* [Dam87]. That is, it is computationally hard to find a pair $x_0, x_1$ with $x_0 \neq x_1$, such that $H(x_0) = H(x_1)$.

### 2.2.2 Pseudo-random Functions

A random function is a machine that upon receiving input $x$ proceeds as follows. If it has not seen $x$ before, it chooses a value $y \leftarrow \{0,1\}^n$ and returns $y$; it then records that $f(x) = y$. If it has seen $x$ before, then it looks up $x$, and outputs the same value $y$ as before.

Intuitively, a pseudo-random function (PRF) is like a random function to any polynomial time adversary. It is originally defined by [GGM86] as an important cryptographic primitive. They are deterministic functions $f : \{0,1\}^n \rightarrow \{0,1\}^n$ which are computable in polynomial time and take two inputs $x, k \in \{0,1\}^n$ where $k$ is a hidden random seed. Naturally, a distribution of functions is pseudo-random if it satisfies the following requirements [NRR00]:

- Easy to sample: It is easy to sample a function according to the distribution.
- Easy to compute: Given such a function, it is easy to evaluate it at any given point.
- Pseudo-random: It is hard to tell apart a function sampled according to the pseudo-random distribution form a uniformly distributed function when the distinguisher is given access to the function as a black-box.

**Theorem 2.1** *If a pseudorandom generator exists, then pseudo-random functions exist.*

Pseudo-random functions have a wide range of applications, most notably in cryptography, but also in computational complexity and computational learning theory.

### 2.2.3 Public Key Encryption Scheme

In a public key encryption scheme, the public key is published in a secure repository, where anyone can use it to encrypt messages. The private key is kept by the recipient so that only he can decrypt massages which are sent to him.

**Definition 2.4** *(Public key encryption scheme) A triple of algorithms (**Gen**, **Enc**, **Dec**) is a public key encryption scheme if*

- $(pk, sk) \leftarrow \boldsymbol{Gen}(1^n)$ *is a polynomial time algorithm that produces a key pair* $(pk, sk)$.
- $c \leftarrow \boldsymbol{End}_{pk}(m)$ *is a polynomial time algorithm that given pk and* $m \in \{0,1\}^n$ *produces a ciphertext c.*
- $m \leftarrow \boldsymbol{Dec}_{sk}(c)$ *is a deterministic algorithm that given a ciphertext c and secret key sk produces a message* $m \in \{0,1\}^n \cup \perp$.
- *There exists a polynomial time algorithm A that on input* $(1^n, i)$*, outputs the* $i^{th}$ *n-bit message (if such a message exists) according to some order.*
- *For all* $n \in \mathbb{N}$, $m \in \{0,1\}^n$,

$$\Pr[(pk, sk) \leftarrow \boldsymbol{Gen}(1^n) : \boldsymbol{Dec}_{sk}(\boldsymbol{Enc}_{pk}(m)) = m] = 1.$$

The decryption algorithm is allowed to produce a special symbol $\perp$ when the input ciphertext is undecipherable. The security property for public key encryption can be defined as follows.

**Definition 2.5** *(Secure public key encryption) The public key encryption scheme (**Gen**, **End**, **Dec**) is said to be secure if for all non uniform polynomial time distinguisher D, there exists a negligible function* $\epsilon(\cdot)$ *such that for all* $n \in \mathbb{N}$, $m_0, m_1 \in \{0,1\}^n$, *D distinguishes between the following distributions with probability at most* $\epsilon(n)$*:*

- $\{(pk, sk) \leftarrow \boldsymbol{Gen}(1^n) : (pk, \boldsymbol{Enc}_{pk}(m_0))\}_n$.
- $\{(pk, sk) \leftarrow \boldsymbol{Gen}(1^n) : (pk, \boldsymbol{Enc}_{pk}(m_1))\}_n$.

### 2.2.4 Commitment Schemes

Commitment schemes are usually referred to as the digital equivalent of a "physical" locked box. They consist of two phases:

- Commit phase: Sender puts a value $x$ in a locked box.
- Reveal phase: Sender unlocks the box and reveals $x$.

It is required that before the reveal phase the value $x$ should remain hidden: this property is called *hiding*. Additionally, during the reveal phase, there should only exists a single value that the commitment can be revealed to: this property is called binding. In the following, a formalization of single-message commitments is provided, where both the commit and the reveal phases only consist of a single message sent from the committer to the receiver.

**Definition 2.6** *(Commitment) A polynomial time machine $Com$ is called a commitment scheme if there exists some polynomial $l(\cdot)$ such that the following two properties hold:*

- *Binding. For all $n \in \mathbb{N}$ and all $x_0, x_1 \in \{0,1\}^n$, $r_0$, $r_1 \in \{0,1\}^{l(n)}$, it holds that $Com(x_0, r_0) \neq Com(x_0, r_1)$.*
- *Hiding. For every non uniform polynomial time distinguisher $D$, there exists a negligible function $\epsilon$ such that for every $n \in \mathbb{N}$, $x_0, x_1 \in \{0,1\}^n$, $D$ distinguishes the following distributions with probability at most $\epsilon(n)$.*
  - $\{r \leftarrow \{0,1\}^{l(n)} : Com(x_0, r)\}$.
  - $\{r \leftarrow \{0,1\}^{l(n)} : Com(x_1, r)\}$.

***Theorem 2.2*** *If one-way permutations exist, then commitment schemes exist.*

### 2.2.5 Sequences of Games

Sequences of games [Sho04] is a tool for organizing the security proofs. To prove security using the sequence-of-games approach, it is needed to proceed as follows.

- Construct. Construct a sequence of games, Game 0, Game 1, $\cdots$, Game $n$, where Game 0 is the original attack game with respect to a given adversary and cryptographic primitive.

- Define. Let $S_0$ be the event $S$, and for $i = 1, \cdots, n$, the construction defines an event $S_i$ in Game $i$, usually in a way naturally related to the definition of $S$.

- Proof. The proof shows that $\Pr[S_i]$ is negligibly close to $\Pr[S_{i+1}]$ for $i = 0, \cdots, n-1$, and that $\Pr[S_n]$ is equal (or negligibly close) to the target probability.

From the above and fact that $n$ is a constant, it follows that $\Pr[S]$ is negligibly close to the target probability. Then the security is proved.

However, the above procession is the general framework of such a security proof. In constructing such proofs, it is desirable that the changes between successive games are very small, so that analyzing the change is as simple as possible. Shoup [Sho04] presented that the transitions between successive games can be restricted to one of three types: *indistinguishability*, *failure events* and *bridging steps*. Throughout this thesis, it is restricted on the type of transitions based on indistinguishability.

**Transitions Based on Indistinguishability [Sho04].** Suppose $D_1$ and $D_2$ are two computational indistinguishable distributions. To prove $|\Pr[S_i] - \Pr[S_{i+1}]|$ is negligible, one argues that there exist a distinguishing algorithm $A$ that interpolates between Game $i$ and Game $i+1$. When $A$ is given an element which is selected from distribution $D_1$, it outputs 1 with probability $\Pr[S_i]$. When $A$ is given an element which is selected from distribution $D_2$, it outputs 1 with probability $\Pr[S_{i+1}]$. Then the indistinguishability assumption implies that $|\Pr[S_i] - \Pr[S_{i+1}]|$ is negligible.

Typically, one can design the two games so that they could easily be rewritten as a single 'hybrid' game that takes an auxiliary input. One get Game $i$ if the auxiliary input is drawn from $D_1$ and get Game $i+1$ if the auxiliary input is drawn from $D_2$. The distinguisher algorithm $A$ then simply runs the hybrid game with its input and outputs 1 if the appropriate event occurs.

## 2.3 Zero-Knowledge Proof-of-Knowledge

**Proof of Knowledge.** In a proof of knowledge protocol[ASM10], the prover attempts to convince the verifier that it knows a certain quantity satisfying some kinds of relation with respect to a commonly known string. For example, given the publicly known value $y = g^x$, the prover attempts convince the verifier that it knows

$x$.

To define a proof of knowledge, the NP-relation $R$ is first defined as follows.

**Definition 2.7** *An NP-relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ is given by a deterministic algorithm $W(\cdot,\cdot)$ that runs in time polynomial in the length of its first input. The relation is:*

$$R = \{(x,w) : W(x,w) accepts\}.$$

*The associated NP-language $L_R = \{x : \exists w$ such that $W(x,w)$ accepts$\}$. The witness set for an $x \in \{0,1\}^*$ is $R(x) = \{w : W(x,w) = 1\}$.*

That is, the NP-relation $R$ is just the set of theorem-witness pairs $(x,w)$ that are accepted by the verifier $W$. It is easily seen that the language $L_R$ associated with the relation is an NP language. For example, for the discrete logarithm problem in $G$, the natural relation is $R = \{(x \in G, w \in \mathcal{Z}_p) : g^w = x\}$.

A proof of knowledge for a relation $R$ is now defined. The definition captures the intuition that form any (possible cheating) prover $P^*$ that is able to convince the verifier with good enough probability on a statement $x \in L_R$, there is a way to extract a valid witness for $x$ from $P^*$ with a related (not too small) probability.

**Definition 2.8** *An interactive proof system $(P,V)$ is a proof of knowledge with knowledge error $k \in [0,1]$ for an NP-relation $R$ if there exists an n.u.p.p.t oracle machine $K$ (the knowledge extractor) such that for any $x \in L_R$, and for any (possible unbounded) $P^*$ for which $p_x^* = \Pr[out_V[P^* \leftrightarrow V(x)] = 1] > k$, it has*

$$\Pr\left[K^{P^*}(x) \in R(x)\right] \geqslant poly(p_x^* - k).$$

That is, the probability that the knowledge extractor $K$ finds a valid witness for $x$ using its access to prover $P^*$ is at least polynomially related to the probability $p_x^*$ that $P^*$ convinces the honest verifier on $x$, unless some knowledge error.

**Zero-Knowledge Proof.** In a zero-knowledge proof protocol that was first introduced by Goldwasser, Micali and Rackoff [GMR89], a prover convinces a verifier that some statement is true without the verifier learning anything except the validity of the statement.

**Zero-Knowledge Proof-of-Knowledge (ZKPoK).** ZKPoK is a proof of knowledge protocol which is zero-knowledge. $\sum$-protocols are a special type of three-move

ZKPoK protocols, which can be converted into non-interactive Signature Proof of Knowledge (SPK) schemes or simply signature schemes [GMR88] that are secure in the random oracle (RO) model [BR93, CGH04]. Let $PK\{(x) : y = g^x\}$ denote a $\sum$-protocol that proves the knowledge of $x \in \mathbf{Z_p}$ such that $y = g^x$ for some $y \in \mathbb{G}$ [CS03]. The $x$ value on the left of the colon denotes variable whose knowledge is to be proven, while other values on denote publicly known value.

# Chapter 3

# Privacy Enhanced Data Outsourcing in the Cloud

How to secure outsourcing data in cloud computing is a challenging problem, since a public cloud environment cannot been considered to be trusted. The situation becomes even more challenging when outsourced data sources in a cloud environment are managed by multiple outsourcers who have different access levels. In this chapter, an efficient and innovative tree-based key management scheme is introduced, which allows a data source to be accessed by multiple parties who hold different rights. It is ensured that the database remains secure, while some selected data sources can be securely shared with other parties. The original scheme was presented at *Journal of Network and Computer Applications, 2012* [ZMS+12].

## 3.1 Introduction

Cloud Storage Services such as Microsoft's Azure storage and Amazon's S3 have gained popularity recently. While more and more enterprises store their private data on the cloud storages, which are generally managed by untrusted parties, secure and privacy have become major concerns. As a countermeasure, Microsoft has recently deployed a virtual private storage service [KL10]. Although the recent efforts in secure cloud computing, there are a number of unsolved security issues. One of such issues is the confidentiality and privacy of user data, while those data has to be shared/managed by multiple parties. This is also the issue to be addressed in this chapter.

The data stored in a cloud database is considered as data outsourcing, since they are managed by an external party. For security, those data are generally encrypted so that only authorized users can access them. Generally, outsourced data consist of many data blocks, hence the management of encryption keys is a major challenge

as mentioned in Chapter 1.

The classical tree-based hierarchy schemes such as RFC2627 [WHA99]) and the scheme proposed by Wong *et al.* [WGL98] have been widely used in group key management. In RFC2627 [WHA99], the hierarchical tree approach is the recommended approach to address the multicast key management problem. This approach provides for the following requisite features: 1)provides for the secure removal of a compromised user from the multicast group, 2)provides for transmission efficiency, 3)provides for storage efficiency. The hierarchical tree approach balances the costs of time, storage and number of required message transmissions, using a hierarchical system of auxiliary keys to facilitate distribution of new key. The result is that the storage requirement for each user and the transmissions required for key replacement are both logarithmic in the number of users, with no background transmissions required. This approach is robust against collusion of excluded users. Moreover, while the scheme is hierarchical in nature, no infrastructure is needed beyond a server (e.g., a root), though the presence of such elements could be used to advantage.

Many key management methods of access hierarchies for data outsourcing have been proposed based on this approach[WLOB09, DdVJ$^+$03, BCdV$^+$09, dVFJ$^+$07a, ABFF09, PL01]. These methods provide some useful solutions to minimize the number of cryptographic keys, which have to be managed and stored. Aiming to provide secure and efficient access to outsourced data, Wang *et al.*[WLOB09] proposed a tree-based cryptographic key management scheme for data storages in the cloud. They referred the scenario to as "owner-write-users-read". Their tree-based key management structure was similar to a traditional one, where a single root node held the master key that could be used to derive other node keys. Each node key could be used to derive the keys of its children in the hierarchy. With their scheme, a data block stored in the cloud can be updated by a party who holds either the specific decryption key or a node key corresponding to one of its parents.

If there is an outsourcing server authorized to manage a node (not the root node) that has several child nodes, then the outsourced party is granted the node key, which can be used to derive all sub-keys for its child nodes. In another word, once a parent node in the tree is given, all the child nodes wll be known. This is a common problem which exists in many tree-based key management schemes. Existing ones such as [WLOB09, DdVF$^+$05, DdVF$^+$07, dVFJ$^+$08, dVFJ$^+$07b, AFB05, YLZL01, KPT00, NNL01a, LNS03, KLKJ06, SL04, BSS11] can work perfectly, only if all legitimate users are authorized to access all the child nodes under the specific parent node.

Considering this problem, a practical application is suggested for private data management. It is named as OWUR/W (owner-write-users-read/write) applications, where a data source protected with a node key in a key management tree can be shared with or managed by another party without compromising the security of the data encrypted with its child nodes' keys. Additionally, data can be updated not only by the data owner, but also by other legitimate parties. It is found that this scenario is very useful in outsourcing management. Meanwhile, it is noticed that other existing schemes do not offer this feature.

Intuitively, it is necessary that the encrypted data block associated with a node can be decrypted by multiple decryption keys where one of them is associated with the tree and can be utilized to generate its keys children's keys, while other decryption keys are only used to decrypt the data block stored in the node. Let us assume two decryption keys $(d_1, d_2)$, assigned to a node, where one of them is associated with the tree (let us assume that $d_1$ is the key associated with the tree and $d_2$ is known to the manager only). Both decryption keys are associated with the unique encryption key, $e$. For a user, who is authorized to access only the data block stored in the node and should not have to access its children, the manager only grant $d_2$ to the user. With $d_2$, the user can decrypt the data block but can not generate the decryption keys of this node's children. It is believed that this method offers additional privacy protection to the outsourced data.

In this chapter, an innovative tree-based key management scheme is proposed. The proposed scheme can indeed capture the idea given above. It is also shown how apply the proposed scheme to protect outsourced data in cloud computing.

**Organization of This Chapter.** The rest of this chapter is organized as follows. In the next section, the data outsourcing model for cryptographic cloud storage is presented. In Section 3.3 , the key derivation hierarchy for key management in cloud storage is described. In Section 3.4, a concrete example is given and the main construction is presented, including the encryption method and the detailed algorithms. In Section 3.5, the data access procedure is analyzed. In Section 3.6, an extension of the proposed scheme is discussed. The security proof of the proposed scheme is given in Section 3.7 and Section 3.8 concludes this chapter.

## 3.2 The Model

In this section, the data outsourcing model is presented, following the application scenario presented in Section 3.1. An illustration of the proposed model is presented in Figure 3.1. The system consists of four major parties:

- The cloud Provider ($\mathcal{P}$), who provides third-party data storage services.
- The original data owner ($\mathcal{O}$), who holds the master (root) key and is responsible to set up the key management system.
- The sub-tree data manager ($\mathcal{M}$), who holds an authorized node key, which can be used to derive all decryption keys for its child nodes. Notice that it is assumed each node has two decryption keys: one can be used to derive all decryption keys of its children and the other can only be used to decrypt the encrypted data in the specific node.
- A user or another sub-tree data manager ($\mathcal{U}$), who probably use or share a data block at a node managed by $\mathcal{M}$ but does not hold the full administrative right of deriving the decryption keys of the children of this node.

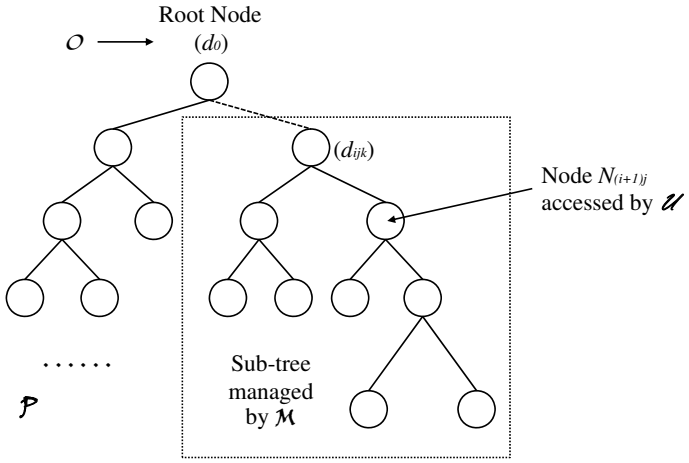


Figure 3.1: The data outsourcing model.

The encryption method is asymmetric, in the sense that the encryption key and decryption key(s) are different. For simplicity, it is assumed that the data stored in each node is encrypted by one encryption key associated with two decryption keys. One of these two decryption keys is used for the decryption of the database located

at the corresponding node and the generation of sub-keys for the child nodes, while another one can only be used for the decryption of the database at the same node.

Let $e_{ij}$ denote encryption keys and $d_{ijk}$ the corresponding decryption keys, respectively, where $i$ denotes the level of a tree, $j$ the index of nodes and $k$ the index of decryption keys. The root master key is denoted by $d_0$. Let $N_{ij}$ denote a node in the tree. Using a binary tree as an example, the proposed model can be described as follows:

- The original owner $\mathcal{O}$ generates a master (root) key, $d_0$, which can be used to derive all other decryption keys and encryption keys in a tree.
- Each node in the tree obtains a master decryption key $d_{ij1}$ and the secondary decryption key $d_{ij2}$, generated from the root key. The secondary decryption key $d_{ij2}$ is derived from $d_{ij1}$.
- The sub-tree data manager $\mathcal{M}$ obtains a key $d_{ij1}$ as its master key, which can be used to generate all node keys of the sub-tree, including the secondary keys.
- User $\mathcal{U}$ can request a secondary decryption key $d_{ij2}$ from $\mathcal{M}$ for accessing the encrypted data stored in node $N_{ij}$.

This key management scheme holds all features from a normal binary tree hierarchy and introduces the new secondary key, which enables flexibility in key management and additional privacy protection.

## 3.3 Key Derivation Hierarchy

Without loss of generality, it is assumed that the outsourced data contains $n$ blocks and $2^{(i-1)} \leqslant n \leqslant 2^i$ where $i$ denotes the level of the tree. Therefore, a complete binary tree from Node $N_0$ to $N_{ij}$ is constructed, where $i$ and $j$ denote the level of a tree and $j$ the node index, respectively.

Before defining the key derivation tree, Key Value is first defined.

**Definition 3.1 (Key Value)** *Except the root node $N_0$, any node $N_{ij}$ in the key derivation tree T, has a key value $K_{ij}$ of two decryption keys $d_{ij1}$ and $d_{ij2}$. These two decryption keys are also denoted as key pair ($d_{ij1}$,$d_{ij2}$). Such a key pair can generate the encryption key $e_{ij}$ for this node.*

The definition of Key Derivation Tree is given as follows. Notice that the encryption key associated with a key value is less important in the key derivation.

**Definition 3.2 (Key Derivation Tree)** *A key derivation tree, denoted T, is a tree $T = \langle N, K \rangle$, rooted at vertex $N_0$. Any node $N_{ij}$ except the leaves, can derive its child nodes of indices $i(2j-1)$ (for the left) and $i(2j)$ (for the right), for $i = 1, 2, ...$ and $j = 1, ..., 2i$, while its parent (if any) is found at index $(i-1)\lceil \frac{j}{2} \rceil$. $K_{ij} \subseteq K$, denotes the key value of each node $N_{ij}$, where the key value consists of a set of decryption keys corresponding to this node.*

To construct the key derivation tree, a cryptographic one-way hash function is chosen as the key generation function: $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$, which can be used to compute the decryption key of child nodes of any node $N_{ij}$, while hard to invert the key of $N_{ij}$. The key value $K_{ij}$ of node $N_{ij}$ is represented by $K_{ij} \leftarrow (d_{ij1}, d_{ij2})$ where $d_{ij1}$ denotes the master decryption key and $d_{ij2}$ denotes the secondary decryption key. The one-way hash function $H$ is also being used to compute $d_{ij2}$ from the input $d_{ij1}$: $d_{ij2} \leftarrow H(d_{ij1})$. The key derivation hierarchy is illustrated in Figure 3.2, where $i \geqslant 0, j \geqslant 1$.

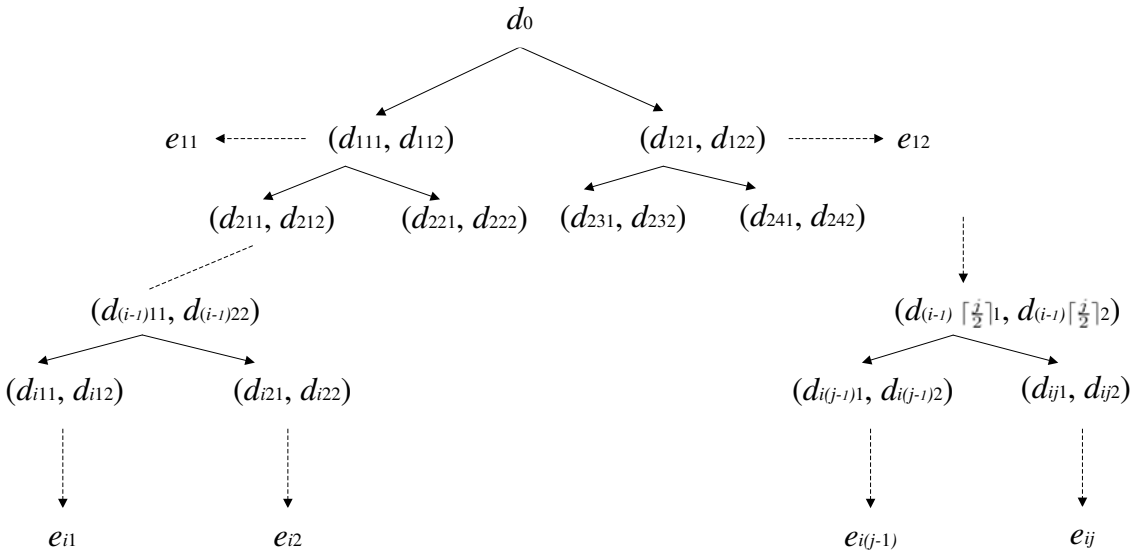


Figure 3.2: Key derivation hierarchy.

With the root key $d_0$ for node $N_0$, all the key pairs can be derived:

$$d_0 \rightarrow d_{111} \rightarrow d_{112}, \quad d_0 \rightarrow d_{121} \rightarrow d_{122}.$$

$$d_{111} \rightarrow d_{211} \rightarrow d_{212}, \quad d_{111} \rightarrow d_{221} \rightarrow d_{222}.$$
$$d_{121} \rightarrow d_{231} \rightarrow d_{232}, \quad d_{121} \rightarrow d_{241} \rightarrow d_{242}.$$
$$\ldots \qquad \ldots$$
$$d_{(i-1)\lceil \frac{j}{2} \rceil 1} \rightarrow d_{i(j-1)1} \rightarrow d_{i(j-1)2}, d_{(i-1)\lceil \frac{j}{2} \rceil 1} \rightarrow d_{ij1} \rightarrow d_{ij2}$$
$$\ldots \qquad \ldots$$

The encryption keys are generated from key pairs which contain the master decryption key and the secondary decryption key, for example,

$$(d_{111}, d_{112}) \rightarrow e_{11}, \quad (d_{121}, d_{122}) \rightarrow e_{12},$$
$$(d_{211}, d_{212}) \rightarrow e_{21}, \quad (d_{221}, d_{222}) \rightarrow e_{22},$$
$$\ldots \qquad \ldots$$
$$(d_{i(j-1)}, d_{i(j-1)2}) \rightarrow e_{i(j-1)}, \quad (d_{ij1}, d_{ij2}) \rightarrow e_{ij}$$
$$\ldots \qquad \ldots$$

To derive child keys, the original data owner $\mathcal{O}$ conducts the following computations. For a key pair ($d_{ij1}$,$d_{ij2}$) of node $(i, j)$, its child on the left can be calculated as

$$(d_{(i+1)(2j-1)1}, d_{(i+1)(2j-1)2}) = (H(d_{ij1} \| (2j-1)), H(H(d_{ij1} \| (2j-1)))).$$

and its child on the right can be calculated as

$$(d_{(i+1)(2j)1}, d_{(i+1)(2j)2}) = (H(d_{ij1} \| 2j), H(H(d_{ij1} \| 2j))).$$

Other sub-keys can be generated accordingly. In this way, the whole key derivation tree can be constructed. A concrete scheme will also be given in the following section.

## 3.4 The Concrete Scheme

Having demonstrated how the proposed scheme works, a concrete construction is now provided. The polynomial introduced in [MVN99] is borrowed and it is demonstrated how to apply it to the proposed key derivation tree.

### 3.4.1 The Polynomial Function

The security of this system relies on difficulty of computing discrete logarithm. The protocols are based on a polynomial function and a set of exponentials. Let $p$, $q$ be two large prime numbers such that $q|p-1$, and $g \in \mathbb{Z}_p^*$ be a generator of order $q$. Let $\{x_i\} \in_R \mathbb{Z}q$ for $i = 0, 1, 2..., n$ be a set of integers. The polynomial function of order $n$ is constructed as follows.

$$f(x) = \prod_{i=1}^{n}(x - x_i) \equiv \sum_{i=0}^{n} a_i x^i \mod q,$$

where $\{a_i\}$ are coefficients:

$$\begin{aligned} a_0 &= \prod_{j=1}^{n}(-x_j), \\ a_1 &= \sum_{i=1}^{n}\prod_{i\neq j}^{n}(-x_j), \\ &\cdots, \\ a_{n-2} &= \sum_{i\neq j}^{n}(-x_i)(-x_j), \\ a_{n-1} &= \sum_{i=1}^{n}(-x_j), \\ a_n &= 1. \end{aligned}$$

It is noted that $\sum_{i=0}^{n} a_i x_j^i = 0$. This property is important for the proposed scheme. Having the set $\{a_i\}$, the corresponding exponential functions can be constructed,

$$\{g^{a_0}, g^{a_1}, \cdots, g^{a_n}\} \equiv \{g_0, g_1, \cdots, g_n\} \mod p.$$

All elements here are computed under modulo $p$. For convenience, modulo $p$ will be omitted in the rest of this paper.

Now it is ready to construct an asymmetric-key system where the encryption key is the tuple $\{g_0, g_1, \cdots, g_n\}$ mapping to $n$ decryption keys $\{x_i\}$.

### 3.4.2 Key Derivation Tree and Data Encryption

Let us use a binary tree as an example and $(i, j)$ as an arbitrary node. Then the main construction contains four algorithms: *Key Generation*, *Encryption*, *Decryption* and *Key Derivation*.

### Key Generation

The decryption keys are denoted by $(d_{ij1}, d_{ij2})$, which correspond to $(x_1, x_2)$ in the 2-degree polynomial defined above, where $d_{ij2} = H(d_{ij1})$. For simplicity, it is denoted that $(d_{ij1}, d_{ij2}) = (d_1, d_2)$. The encryption key corresponding to $(d_1, d_2)$ is $e = (g_0, g_1, g_2)$, where $g_0 = g^{a_0} = g^{d_1 d_2}$, $g_1 = g^{a_1} = g^{-(d_1+d_2)}$, $g_2 = g^{a_2} = g$. For simplicity, the subscripts of $e_{ij}$ is omitted.

### Encryption

The encryption algorithm takes as input a message $M \in \{0,1\}^*$, the encryption key $e$, a random $k \in \mathbb{Z}_q$, and a generator $h \in \mathbb{Z}_p^*$, and outputs a ciphertext $(c_1, c_2)$, where

$$c_1 \leftarrow (h^k \cdot g_0^k, g_1^k, g_2^k), \qquad c_2 = M \cdot h^k.$$

Rewrite $c_1$ as $(b_1, b_2, b_3)$ for convenience. The encryption scheme is a variant of ElGamal encryption, which is proven to be secure under the assumption of Chosen Plaintext Attack when the group $\mathbb{Z}_p^*$ is properly selected. It can be easily converted into Chosen Ciphertext Security by Fujisaki-Okamoto transformation [FO99].

### Decryption

This algorithm takes as input the ciphertext $(c_1, c_2)$ and one of decryption keys $d_1$ and $d_2$, and outputs $M$. $h^k$ can be computed from $b_1 \cdot b_2^{d_i} \cdot b_3^{{d_i}^2}$, for $i \in \{1, 2\}$. Thus, $M$ can be computed as $M = c_2/h^k$.

### Key Derivation

This algorithm takes as input the master decryption key $d_{ij1}$ and a one way hash function $H : \{0,1\}^* \to \mathbb{Z}_q$. It outputs the two child nodes of key $d_{ij1}$. During the key derivation procedure, the left child node can be computed as

$$(d_{(i+1)(2j-1)1},\ d_{(i+1)(2j-1)2}) = (H(d_{ij1} \| (2j-1)),\ H(H(d_{ij1} \| (2j-1)))),$$

and the right child node can be computed as

$$(d_{(i+1)(2j)1},\ d_{(i+1)(2j)2}) = (H(d_{ij1} \| 2j),\ H(H(d_{ij1} \| 2j))).$$

By repeating this algorithm, the whole key derivation tree can be generated.

The proposed scheme also considers "Write" applications and allows the user to re-encrypt the data. This means that the encryption key $e$ should be given to

the user. The reader might think that the user could alter the encryption key by changing a new encryption key, as he can easily replace his existing decryption key with a different one. However, this action will fail if the data manager checks the correctness of the encryption regularly. An alternative solution is to use an RSA modulus and assume that two corresponding primes, which form the modulus, is only known to the manager. This change makes the encryption key inviolable.

## 3.5 Data Access Procedure

In this section, the data access procedure is described for four associated parties given in Section 3.2. Here some notations are first given.

- $\mathcal{M}$, $\mathcal{O}$, $\mathcal{P}$, $\mathcal{U}$: abbreviated names appear as the four major parties as given in the proposed model.
- $\mathcal{M} \rightarrow \mathcal{O}$: $m$. $\mathcal{M}$ sends message $m$ to $\mathcal{O}$.
- $k_{XY}$: a symmetric key shared between parties $X$ and $Y$.
- $T_X$: a timestamp generated by $X$;

Using $N_{ij}$ as example, the data block is encrypted with $e_{ij}$, which is corresponding to two decryption keys $(d_{ij1}, d_{ij2})$. The data access procedure is described in two phases, shown in Figure 3.3 and Figure 3.4.

### 3.5.1 The First Phase

In the first phase, Data Owner $\mathcal{O}$, Sub-tree Manager $\mathcal{M}$ and Cloud Provider $\mathcal{P}$ execute the following five steps.

1. $\mathcal{M}$ sends $\mathcal{O}$ a key request message: M_REQ, where M_REQ = $\{sub\_id, T_M,$ MAC$(k_{MO}, sub_id, T_M)\}$. MAC denotes the message authentication code with the key $k_{MO}$ shared by $\mathcal{M}$ and $\mathcal{O}$. After the original data owner $\mathcal{O}$ sets up the system, the first step in the data access procedure are run by $\mathcal{O}$ and its sub-tree data manager $\mathcal{M}$. $\mathcal{M}$ sends $\mathcal{O}$ an access request message M_REQ in order to obtain the sub-tree root key. The *sub_id* field in this message provides the index of the sub-tree root node. Upon receiving the M_REQ message, $\mathcal{O}$ performs data-integrity validation by checking the MAC of *sub_id* and timestamp.

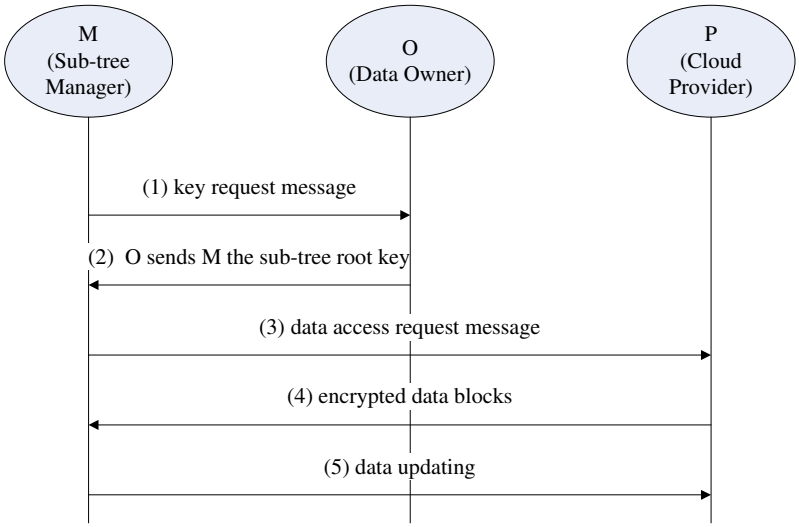


Figure 3.3: The first phase.

2. $\mathcal{O}$ sends $\mathcal{M}$ the sub-tree root key $d_{ij1}$ encrypted with $\mathsf{E}_{k_{MO}}$. Upon receiving $d_{ij1}$, $\mathcal{M}$ can use it to derive the second key and other keys of its child nodes and access the the encrypted data block as given in the next step.

3. $\mathcal{M}$ sends $\mathcal{P}$ an access request message: M_REQ, where M_REQ = $\{sub\_tree\_id$, $T_M$, MAC($k_{MP}$, $sub\_tree\_id$, $T_M$)$\}$

4. $\mathcal{M}$ accesses the stored data block.

5. With the sub-tree root key $d_{ij1}$ for $N_{ij}$, $\mathcal{M}$ can update all the sub-tree nodes. this sub-procedure is called as a write step because $\mathcal{M}$ can modify and more importantly, re-encrypt the sub-tree data. After that, $\mathcal{M}$ sends the re-encrypted data block back to $\mathcal{P}$.

### 3.5.2 The Second Phase

In the second phase, User $\mathcal{U}$, Data Owner $\mathcal{O}$ and Cloud Provider $\mathcal{P}$ execute the following five steps:

1. $\mathcal{U}$ sends $\mathcal{M}$ a key request message: U_REQ, where U_REQ = $\{req\_id$, $T_U$, MAC($k_{MU}$, $req\_id$, $T_U$)$\}$. This step runs between User $\mathcal{U}$ and the sub-tree data manager $\mathcal{M}$. Without any interaction with the original data owner $\mathcal{O}$, the user $\mathcal{U}$ starts the connection by sending a key request message U_REQ to the sub-tree manager $\mathcal{M}$. MAC is the message authentication code using the

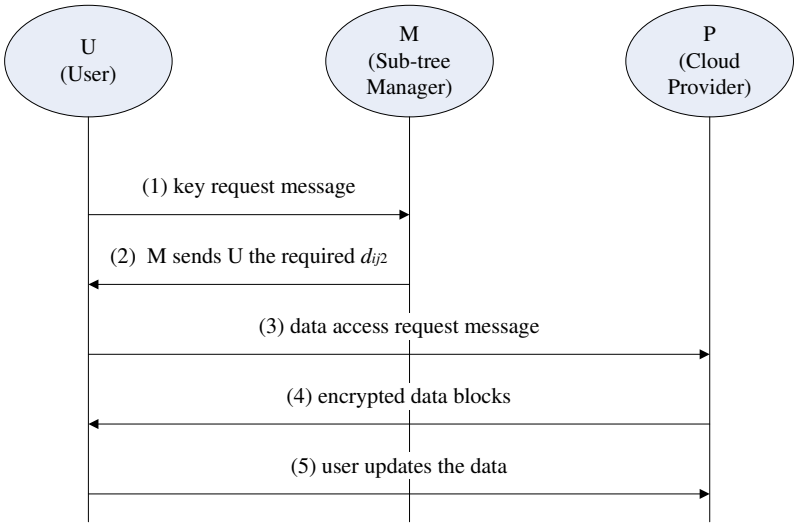


Figure 3.4: The second process.

key $k_{MU}$. The $req\_id$ field contains the node information for $\mathcal{U}$. $\mathcal{U}$ performs data-integrity validation with the MAC, upon receiving the U_REQ message.

2. $\mathcal{M}$ sends $\mathcal{U}$ the secondary decryption key $d_{ij2}$ with respect to the same node. $\mathcal{U}$ can use this decryption key to decrypt the same data block. If $\mathsf{U}$ is a legitimate user that can write the data, $\mathcal{M}$ also provides the encryptions key $e_{ij}$ corresponding to this node.

3. $\mathcal{U}$ sends $\mathcal{P}$ an access request: U_REQ $= \{req\_node\_id, T_U,$ MAC$(k_{UP}, req_node_id, T_U)\}$.

4. $\mathcal{U}$ accesses the encrypted data block.

5. Given the encryption key $e_{ij}$, $\mathcal{U}$ can update this specific data node. this sub-procedure is called as a user write step where a legitimate user can modify and re-encrypt the specific data block.

## 3.6 Generalization of Key Derivation Hierarchy

A flexible and efficient key management scheme should be adaptable and expandable for different application scenarios. For this consideration, the proposed scheme is expanded into multiple branches and multiple sub-keys respectively. Then a generalized key derivation hierarchy is given with a consideration of both cases.

### 3.6.1 Multiple Branches

The proposed scheme can be expanded easily for multiple branches ($m > 2$) as Figure 3.5.

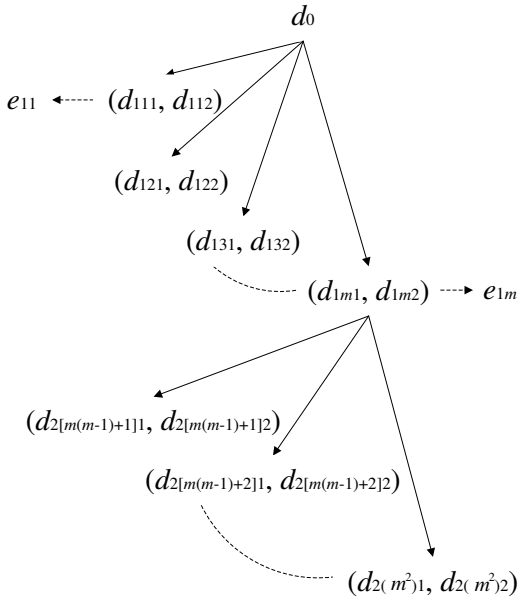


Figure 3.5: A key derivation tree with multiple branches.

Every parent node has $m$ child nodes which can be derived in a similar way. Taking parent node ($d_{1m1}$, $d_{1m2}$) as an example, its first child node ($d_{2[m(m-1)+1]1}$, $d_{2[m(m-1)+1]2}$) can be computed as ($H(d_{1m1}||[m(m_1) + 1])$, $H(H(d_{1m1}||[m(m_1) + 1]))$), and its last child node ($d_{2(m^2)1}$, $d_{2(m^2)2}$) can be computed as ($H(d_{1m1}||m^2$, $H(H(d_{1m1}||m^2)))$. This expansion lowers the level of derivation hierarchy, by increasing the number of nodes in each level.

### 3.6.2 Multiple Sub-keys

It can also be expanded to $n$ sub-keys ($n > 2$), where these sub-keys map to a single encryption key used to encryption the corresponding data block. In regard to node $N_{ij}$, this $n$ sub-keys are denoted by ($d_{ij1}$,$d_{ij2}$,...,$d_{ijn}$) and the encryption key by $e_{ij}$. Every decryption key can be used to decrypt the data block, while only the master decryption key $d_{ij1}$ can derive $n$ sub-keys in the tree.

Taking $n = 3$ as an example (shown in Figure 3.6), the decryption keys are

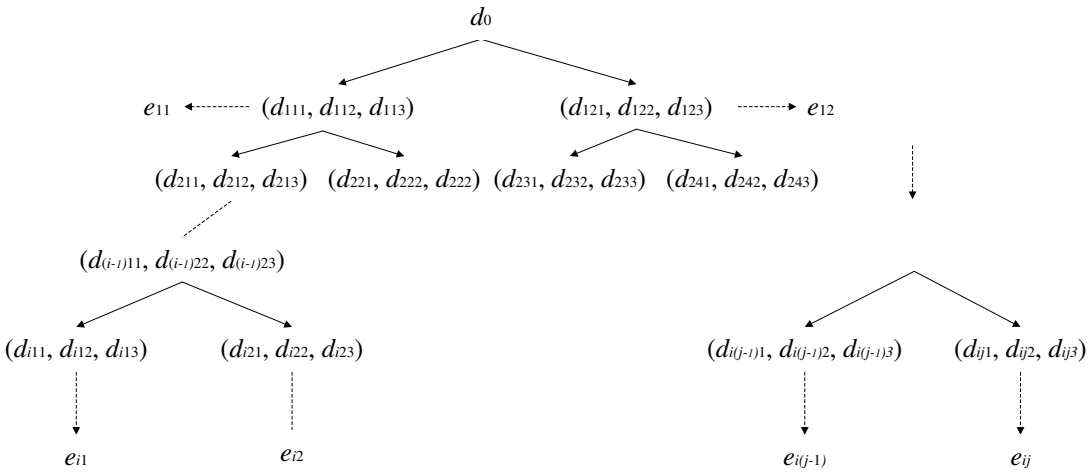


Figure 3.6: Take $n = 3$ as an example.

denoted by ($d_{ij1}$,$d_{ij2}$,$d_{ij3}$), which correspond to $(x_1, x_2, x_3)$ in the 3-degree polynomial defined in Section 3.4, where $d_{ij2} = H(d_{ij1}||1)$, $d_{ij3} = H(d_{ij1}||2)$. Denote $(d_{ij1}, d_{ij2}, d_{ij3})$ by $(d_1, d_2, d_3)$, the encryption key corresponding to $(d_1, d_2, d_3)$ is $e = (g_0, g_1, g_2, g_3)$, where $g_0 = g^{a_0} = g^{-d_1 d_2 d_3}$, $g_1 = g^{a_1} = g^{d_1 d_2 + d_1 d_3 + d_2 d_3}$, $g_2 = g^{a_2} = g^{-(d_1+d_2+d_3)}$, $g_3 = g^{a_3} = g$.

Combining both cases above, a generalized key derivation tree can be given. An $m$-branch $n$ sub-key derivation tree is presented in Figure 3.7.

## 3.7 Proof and Analysis

### 3.7.1 Security Natation

The informal definition of the security property is first given. A construction of encryption system must possess this security property.

**Confidentiality**. Entities (including the cloud server) other than the authorized users should not be able to learn anything about the underlying plaintext message.

**Definition 3.3** *(Security) A construction of the encryption system is secure if it holds confidentiality.*

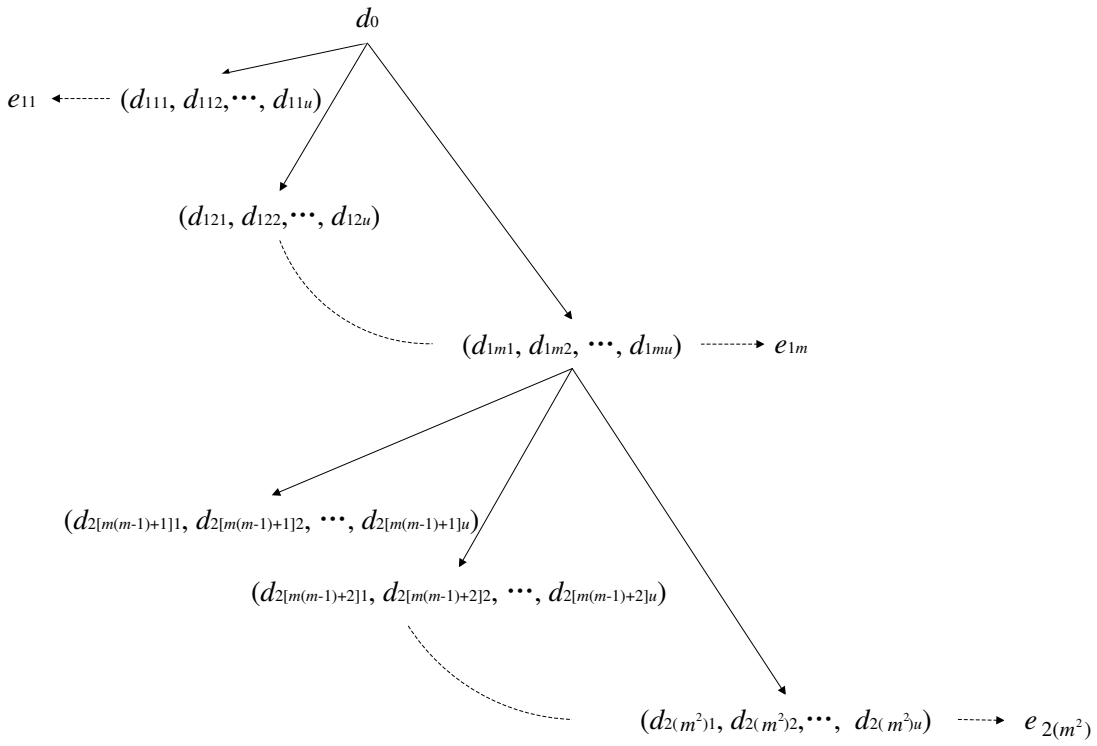


Figure 3.7: Expanding to M-Branch N-Tuple Hierarchy.

### 3.7.2 Security Model

The security requirement is formalized by the following security game. The adversary $\mathcal{A}$ models a malicious user or a cloud server which is *Byzantine*, i.e., can behave arbitrarily, however, cannot collude with authorized users. Let $\mathcal{C}$ be the challenger. Then the game runs as follows:

**Setup Phase** $\mathcal{C}$ sets up the encryption system and makes all public parameters in the system available to $\mathcal{A}$.

**Query Phase 1** The adversary $\mathcal{A}$ can make at most $q_s$ queries to an encryption oracle, providing some message $m_i$ during each query. $\mathcal{C}$ then computes the ciphertext $CT_{m_i}$ and sends it back to $\mathcal{A}$.

**Challenge Phase** The adversary $\mathcal{A}$ submits two messages $(M_0, M_1)$ of equal length. Then the Challenger $\mathcal{C}$ flips a fair coin $b \in_R \{0, 1\}$ and encrypts message $M_b$. The ciphertext is then passed to $\mathcal{A}$.

**Query Phase 2** $\mathcal{A}$ can issue queries as in Query Phase 1 except it cannot submit queries with input $M_0$ and $M_1$.

**Output Phase** Eventually $\mathcal{A}$ outputs a guess $b^*$ of $b$. $\mathcal{A}$ wins if and only if $b^* = b$.

The advantage of an adversary $\mathcal{A}$ in this game is defined as $\Pr[b^* = b] - 1/2$.

### 3.7.3 Proof of Security

**Theorem 3.1** *The encryption scheme is secure if DDH problem in $\mathbb{Z}_p^*$ is hard.*

*Proof:*

It will be shown that the proposed scheme is secure under the assumption of Chosen Plaintext Attack if the DDH problem is hard in $\mathbb{Z}_p^*$. Let $\mathcal{S}$ be a game simulator. Suppose there exist a polynomial-time adversary $\mathcal{A}$ that can attack the proposed scheme in the standard model with advantage $\epsilon/2$. The simulator $\mathcal{S}$ is given a DDH problem instance $(g, g^\alpha, g^\beta, R)$ from the DDH challenger generated in the following process. The challenger flips a fair binary coin $\mu$, outside of $\mathcal{S}$'s view. It sets the problem instance as $(g, g^\alpha, g^\beta, R)$ where $R = g^{\alpha\beta}$ if $\mu = 0$ or $R$ is a random group element if $\mu = 1$. The goal of $\mathcal{S}$ is to output the value $\mu$.

- **Setup Phase** The simulator $\mathcal{S}$ chooses public parameters. $\mathcal{S}$ generates the keys ($d_1$, $d_2$, $e$). $\mathcal{S}$ sets $h = g^k$ for some random value $k$.
- **Query Phase 1** The adversary $\mathcal{A}$ can make queries for encrypting the message $m$. $\mathcal{S}$ returns with the ciphertext $CT_m$.
- **Challenge Phase** $\mathcal{A}$ will submit two challenge messages $M_0$ and $M_1$ to $\mathcal{S}$. $\mathcal{S}$ flips a fair coin $b \in \{0, 1\}$ and sets $d_1 = \alpha$. The encryption key is then ($g^{a_0}$, $g^{a_1}$, $g^{a_2}$) such that each $a_j = u_j \cdot \alpha + v_j$ for some know $u_j$ and $v_j$ for $j = 0, 1, 2$. Thus, $g^{a_j}$ can be computed by $\mathcal{S}$ as $(g^\alpha)^{u_j} g^{v_j}$). $\mathcal{S}$ returns an encryption of $M_b$ as follows:

$$CT = \begin{cases} c_1 \leftarrow (g^\beta)^k \cdot R^{u_0} \cdot (g^\beta)^{v_0}, R^{u_1} \cdot (g^\beta)^{v_1}, R^{u_2} \cdot (g^\beta)^{v_2} \\ c_2 = M_b \cdot (g^\beta)^k \end{cases}$$

If $\mu = 0$, $R = g^{\alpha\beta}$. Then by inspection, the ciphertext is a valid ciphertext for message $M_b$, with the randomness $r = \beta$.

If $\mu = 1$, then $R$ is just a random group element. Thus the ciphertext is just a random element and the ciphertext contains no information about $M_b$.

- **Query Phase 2** $\mathcal{S}$ acts exactly as it did in Query Phase 1 except for messages $M_0$ and $M_1$.

- **Output Phase** $\mathcal{A}$ will submit $b^*$ of $b$. If $b^* = b$, the simulator will output $\mu^* = 0$ to indicate that it was given a valid DDH triple; otherwise, it will output $\mu^* = 1$ to indicate it was given a random element triple.

In the case where $\mu = 1$, the adversary gains no information about $b$. Therefore, $\Pr[b^* = b|\mu = 1] = \frac{1}{2}$. Since the simulator guesses $\mu^* = 1$ when $b^* \neq b$, $\Pr[\mu^* = \mu|\mu = 1] = \frac{1}{2}$. If $\mu = 0$ then the adversary sees an encryption of $M_b$. The adversary's advantage in this situation is $\epsilon$ by assumption. Therefore, $\Pr[b^* = b|\mu = 0] = \frac{1}{2} + \epsilon$. Since the simulator guesses $\mu^* = 0$ when $b^* = b$, $\Pr[\mu^* = \mu|\mu = 0] = \frac{1}{2} + \epsilon$. The overall advantage of the simulator in the DDH game is: $\frac{1}{2}\Pr[\mu^* = \mu|\mu = 0] + \frac{1}{2}\Pr[\mu^* = \mu|\mu = 1] - \frac{1}{2} = \frac{1}{2} \cdot (\frac{1}{2} + \epsilon) + \frac{1}{2} \cdot \frac{1}{2} - \frac{1}{2} = \frac{1}{2}\epsilon$.

## 3.8 Conclusion

Key management and access control are important for secure cloud computing. As a traditional approach, tree-based key management has attracted a lot of attention. It is found that a traditional tree-based approach had some drawbacks. In a traditional tree-based key management hierarchy, a node key holder can derive all the child node keys. In order to solve this problem and maintain the key management feature, in this chapter, OWUR/W applications are proposed for data sourcing. A secure and flexible tree-based key derivation hierarchy is presented. The proposed scheme allowed the outsourcing party to access the data block located at a specified node, while he could not access the data blocks encrypted with child keys. It is believed that the proposed tree-based outsourcing key management opens up an entirely new approach for secure and flexible key management.

# Chapter 4

# Piracy-Preserved Access Control for Cloud Computing

The problem of access control on outsourced data to 'honest but curious' cloud servers has received considerable attention, especially in scenarios involving potentially huge sets of data files, where re-encryption and re-transmission by the data owner may not be acceptable. Considering the user privacy and data security in cloud environment, in this chapter, a solution is proposed to achieve flexible and fine-grained access control on outsourced data files. In particular, the problem of defining and assigning keys to users is concerned. The access policies and users' information are hidden to the third-party cloud servers. The proposed scheme is partially based on the observation that, in practical application scenarios each user can be associated with a set of attributes which are meaningful in the access policy and data file context. The access policy can thus be defined as a logical expression formula over different attribute sets to reflect the scope of data files that the kind of users is allowed to access. As any access policy can be represented using a logical expression formula, fine-grained access control can be accomplished. The original scheme was presented at *Trustcomm, 2012* [ZMSY11].

## 4.1 Introduction

Cloud computing is a new computing infrastructure for hosting data and deploying services and has drawn extensive attention from both academia and industry. Cloud computing is regarded as an infrastructure for delivering computing power, where cloud users can use third-party resources through networks and save their investments significantly by migrating their businesses into the cloud. Due to its low cost, robustness and ubiquitous nature, cloud computing is changing the way entities store and manage their data. Outsourcing data to cloud servers can offer a number

of benefits to cloud users, including low capital expenditures, dynamic provisioning and economies of scale. While this infrastructure, exemplified by Microsoft's Azure Service Platform, Amazon Simple Storage Service (S3), Amazon's Elastic Compute Cloud (EC2), and Rackspace's Mosso has already provides well known examples, it has also encountered new security risks.

As more and more sensitive data are shared and stored in the cloud, data security and privacy have been considered as a thorniest problem that may impede the growth of cloud computing. Since all the resources are provided over the Internet, the cloud becomes a single point of access for all the users. Fine-grained data access control can play an important role on data security.

Recently, the problem of access management on outsourced data to cloud servers has received considerable attention and several advancements have already been proposed [LZC+10, LWG11, CSK11, BLLS11, ZLS11, Lee12, ZHA+12, WeLD12, ASB+12, YLJ12, SWYW12, GM09, HYJZ09]. An existing feasible solution to achieve fine-grained access control of outsourced data in cloud computing is to encrypt the data through certain cryptographic primitives and only disclose the private keys to authorized users. Without the appropriate decryption keys, unauthorized users including the cloud providers, cannot decrypt the data. This solution has been widely used (such as [dVFJ+07b, KRS+03]) and most schemes using it are deployed by introducing a per file group for efficiency. However, the complexity of these schemes [BKP09, HCM01, CHR09, TWZ09, YC10, LYRL10, TS11] is proportional to the system scale and the number of users. Recently, Yu *et al.* [YWRL10] stretched out a scalable data access control scheme for cloud computing, which combined techniques of key-policy attribute-based encryption, proxy re-encryption and lazy re-encryption. In their scheme, cloud servers were unable to learn the plaintext during data re-encryption. However, the cloud servers must learn the whole user ID list and attribute list in exchange.

Vimercati *et al.* [dVFJ+07b] proposed an encryption scheme for securing data stoppage on untrusted servers. Their scheme was based on key derivation methods [AFB05]. In their scheme, to grant access for a user, the owner created corresponding public tokens with his secret key. The user was able to derive decryption keys for desired files. Then the owner transmitted these public tokens to the semi-trusted server and delegates the task of token distribution to it. Given these public tokens, the server was not able to derive the decryption key of any file. This solution introduced a minimal number of secret key per user and a minimal number of encryption

key per each file. However, the complexity of operations of file creation and user grant/revocation is linear to the number of users. Goh *et al.* [GSMB03] proposed SiRiUS which was layered over existing file systems such as NFS, but provided end-to-end security. For the purpose of access control, SiRiUS attached each file with a meta data file that contained the file's access control list (ACL), each entry of which was the encryption of the file's file encryption key (FEK) using the public key of an authorized user. The extended version of SiRiUS used NNL broadcast encryption algorithm [NNL01b] to encrypt the FEK of each file. As the complexity of the user revocation solution in NNL is proportional to the number of revoked users, SiRiUS has the same complexity in terms of each meta data file's size and the encryption overhead, and thus is not flexible. Ateniese *et al.* [AFGH05] proposed a secure distributed storage scheme based on proxy re-encryption. Specifically, the data owner encrypted blocks of content with a master public key, which could only be decrypted by the master private key. The data owner then generated proxy re-encryption keys by using his master private key and the user's public key. With these proxy re-encryption keys, the semi-trusted proxies could convert the ciphertext into another ciphertext for a specific granted user. The main problem with this scheme is that user access privilege is not protected from the proxy.

Nowadays, ABE has been seen as an ideal technique for achieving flexible, scalable and fine-grained access control mechanisms in the cloud. Wang *et al.* [WLW10] proposed a hierarchical attribute-based encryption scheme to achieve fine-grained access control in cloud storage services by combining hierarchical identity-based encryption and ciphertext-policy attribute-based encryption (CP-ABE). In their scheme, they assumed that all attributes in one conjunctive clause were administered by the same domain master, however it is difficult to implement in practice so that the same attribute can be administered by multiple domain masters according to specific policies. Another difficulty is that their scheme can not support compound attributes efficiently and does not support multiple value assignments. Yu *et al.* [YWRL10] stretched out a scalable data access control scheme for cloud computing which combined techniques of key-policy attribute-based encryption (KP-ABE), proxy re-encryption and lazy re-encryption, where cloud servers were unable to learn the plaintext during data re-encryption, however the encryptor in their scheme is not able to decide who can decrypt the encrypted data, and has no choice but to trust the key issuer. In addition, the cloud servers must learn the whole user ID list and attribute list in exchange.

Considering the user privacy and data security in a cloud environment, in this chapter, an encryption system is proposed to achieve flexible and fine-grained access control on outsourced data. In particular, the problem of defining and assigning keys to users is concerned. The access policies and users' information are hidden to the third-party cloud servers. The proposed scheme is partially based on the observation that, in practical application scenarios, each user can be associated with a set of attributes, which are meaningful in the access policy and data file context. The access policy can thus be defined as a logical expression formula over different attribute sets to reflect the scope of data file that the kind of user is allowed to access. As any access policy can be represented as such a logical expression formula, fine-grained access control can be achieved. In this system, a policy hidden attribute-set based encryption and server re-encryption mechanism (SRM) are proposed to achieve as follows: 1) The cloud server can re-encrypt data files by given encryption keys from data owner, without learning the contents or requiring any information about the users from data owner, 2) data file creation/deletion does not require a system-wide data file update or re-keying, and 3) new user creation and user revocation do not affect other users and do not require other users to re-key their private key.

The contribution of this chapter can be summarized as threefold: 1) A two-tier encryption model is proposed to achieve flexible and fine-grained access control for outsourced data in clouds, 2) private data content and information about the users as well as the access policies are not leaked to the cloud provider, and 3) the proposed scheme is provable secure under the standard model.

**Organization of This Chapter.** The rest of this chapter is organized as follows. In Section 4.2, the design goals and main idea are introduced. The encryption model is also proposed. In Section 4.3, the base model is presented. In Section 4.4, the surface model is then described. The security proof and analysis of the proposed scheme are given in Section 4.5. Section 4.6 concludes this chapter.

## 4.2 Design Goal and Main Idea

### 4.2.1 Design Goal

The main design goal is to help the data owner achieve a flexible and fine-grained access control on the outsourced data in clouds. The aim is to prevent the cloud provider from learning the data contents and user information, and allow the data owner to define users who can get access to data files. Specifically, user's creation or revocation should not affect other users, namely other users do not need to update their secret keys. In addition, the proposed scheme also features policy-hiding and is secure against to the collusion attacks from malicious users.

### 4.2.2 Main Idea

Considering to achieve a flexible and fine-grained access control on the outsourced data in cloud environment, a functional encryption system which is named as encryption is proposed. This system proposes attribute-set-based encryption as a base tier and server re-encryption mechanism as a surface tier.

As any access structure can be represented as an access tree $T$, each data file is associated with an access structure from where different attribute sets can be generated. Take the example in Figure 4.1, which gives an instance of access structure and attribute sets that can be generated. It describes a data file that can be accessed by CS staff, CS students from class one, or CS students from class two. Specifically, the privileged users hold an attribute set as one of the following: (University, CS, Student, Class.1), (University, CS, Student, Class.2), or (University, CS, Staff). These attribute sets are used to generate the private keys of the privileged users. Each attribute in the attribute sets is given a chosen value $S_i$. A student in class one who holds an attribute set (University, CS, Student, Class.1) can decrypt this data file using his private key computed as $h(S_{uni} + S_{cs} + S_{student} + S_{class.1})$. Analogically, a professor in CS department decrypts the data file by using his secret key computed as $h(S_{uni} + S_{cs} + S_{staff})$. The construction of attribute-set based encryption allows different users to decrypt the data file with the corresponding secret keys, which does not encrypt access structure into a ciphertext.

However, this access control cannot be considered as flexible when the attribute-set based scheme runs alone. One challenging issue here, caused by user revocation, is to require a system-wide private keys update against the expedience of users.

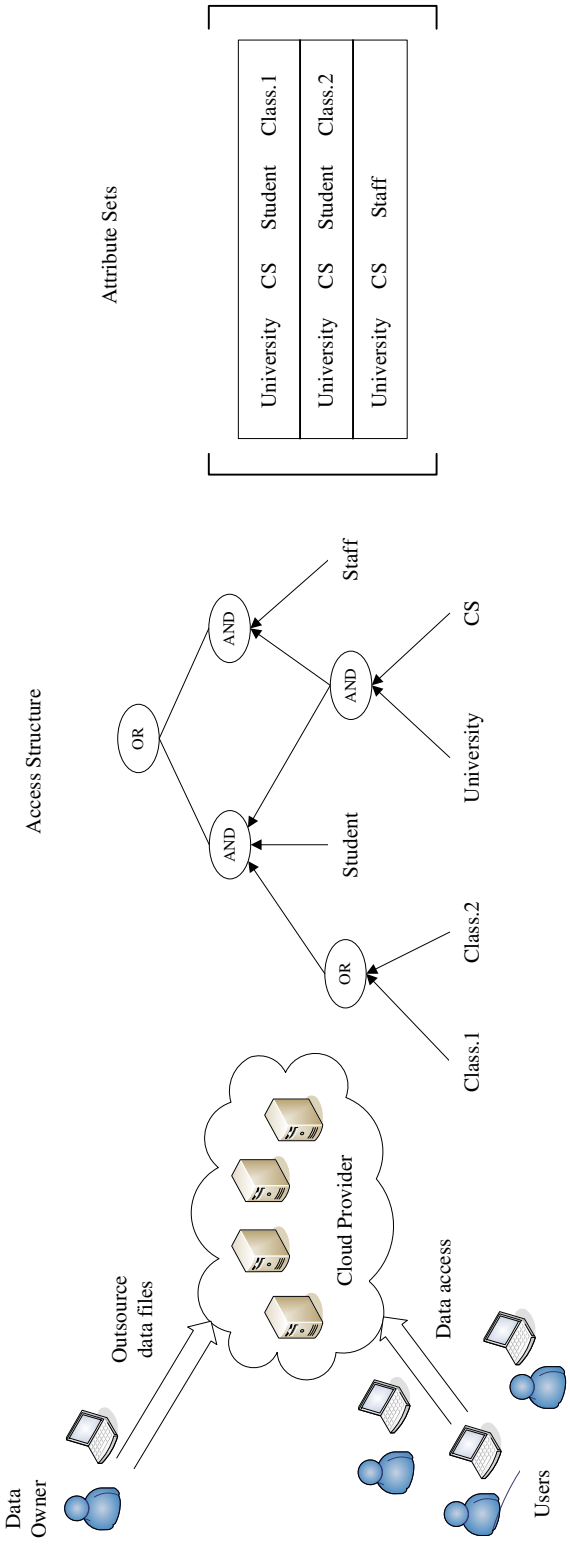


Figure 4.1: An example for the proposed scenario.

Thus, a server re-encryption mechanism is proposed. This mechanism combines with the attribute-set based scheme, to eliminate users from updating their private keys when a user joins or leaves. In addition, in scenarios involving potentially huge sets of data files of considerable size, re-encryption and re-transmission by data owner may not be acceptable. As the cloud servers are assumed to be more powerful, the task of data file re-encryption is done by the cloud server without disclosing file content and attribute list.

### 4.2.3 The Encryption System

It is assumed that the two-tier encryption system consists of the following three parties: the Data Owner who is also the cloud user, the Cloud Provider who provides cloud servers, and many data consumers that can be referred as users for brevity. The data owner encrypts the data files first before sends them into the cloud and builds a server re-encryption mechanism (SRM) that works as a second level dynamic password generator. Now each tier model of this system is introduced in details, as base model and surface model respectively.

- **Base Phase**: The data owner at local, before outsourcing data into the clouds, performs a attribute-set based encryption on the data files according to the access policies.
- **Surface Phase**: The cloud server performs the dynamic encryption operations over the encrypted data files, when receiving request messages from the data owner.

To access the data files stored in cloud, users download the ones of their interest from the cloud provider and decrypt them with their own decryption key. The data owner is not required to be always online unless there are necessary changes in the access structure that caused by the the user grant or revocation. As the servers in cloud are assumed to have abundant storage capacity and computation power, the task of data file re-encryption is transferred to the cloud servers without the leakage of data file contents and any information about the users, including the number of users and users' ID list. The server re-encryption mechanism, which run on the cloud servers, handles the data file re-encrypt task in a imperceptible way without requiring the users to re-key their decryption keys for re-encrypted data files.

Attribute Matrix

| University | CS | Student | Class.1 | Class.2 | Staff |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

Figure 4.2: Generate an access matrix.

## 4.3 Base Phase

### 4.3.1 Access Structure and Attribute Sets

Consistently with the data outsourcing scenario, it is assumed that there exists several attribute sets in the system and the data owner therefore defines access structures for users to access the outsourced data. These access structures are abstracted in a down-top manner to generate authorizations that can be modeled via an access matrix. Each row of the access matrix is set to one privilege attribute set for a specific user or users, with a generated secret key for corresponding data file.

**Definition 4.1 (Access Structure)**

*Any access structure can be represented as an access tree* $\mathbb{T}$*, where each interior node* $x$ *in the tree is a threshold gate with threshold value* $k_x$ *and* $num_x$ *children,* $0 < k_x \leq num_x$*. These threshold gates can capture the cases of* ***AND*** *and* ***OR****, as* $k_x = 1$ *for "****OR****" and* $k_x = num_x$ *for "****AND****". Each leaf node* $x$ *of the tree is described by an attribute and a threshold value* $k_x = 1$*.*

A few functions for $\mathbb{T}$ are first defined. The function $attr(x)$ and $node(i) = x$ is defined if and only if $x$ is a leaf node. $parent(x)$ returns the parent of a node $x$ while $attr(x) = i$ and $node(i) = x$ are used to associate the node $x$ with attribute $i$. $\mathbb{T}$ also defines an ordering for the children of every node, from 1 to $num$. The function $index(x)$ returns the ordering number associated with the node $x$. The index values are uniquely assigned to nodes in $\mathbb{T}$ in an arbitrary manner.

**Definition 4.2** *(Attribute Sets) Let $\mathbb{A}$ be the set of leaf nodes in the access structure $\mathbb{T}$, which contains n different attributes denoted as $a_1, a_2, a_3, ..., a_n$. Define a function*

$$F(\tilde{A}, \mathbb{T}) = \begin{cases} 1 & \text{when set } \tilde{A} \text{ satisfies the access structure } \mathbb{T}, \\ 0 & \text{otherwise} \end{cases}$$

*Then $\tilde{A}$ is an authorized attribute set if and only if $\tilde{A} \subseteq \mathbb{A}$ and $F(\tilde{A}, \mathbb{T}) = 1$. AS denotes the set of authorized attribute sets, that is, $AS = \{\tilde{A} | F(\tilde{A}, \mathbb{T}) = 1\}$. Let $m = |AS|$, then $AS = \{\tilde{A}_1, \tilde{A}_2, ..., \tilde{A}_m\}$ where $\tilde{A}_i \subseteq AS$ for $i = 1, 2, ..., m$.*

The set of attribute sets $AS$ can be presented by an $n \times m$ matrix, with $n$ and $m$ defined above. The $i$-th row of $Matrix(A)$ is a vector $(b_{i,1}, \ldots, b_{i,n})$ in $\{0,1\}^n$ representing the authorized attribute set $\tilde{A}_i$. $b_{i,j} = 1$ if the attribute $a_j$ is presented in $\tilde{A}_j$ and 0 otherwise. The $Matrix(A)$ then records the set of authorized attribute sets $AS = \{\tilde{A} | F(\tilde{A}, T) = 1\}$ that corresponds to different users. As an example, the attribute matrix shown in Figure 4.2 illustrates the attribute sets in Figure 4.1 correctly, and enforces the access structure represented by the access tree.

Given an attribute matrix $Matirx(A)$, secret keys to different users can be defined and assigned. It is assumed that each user is associated with a single key generated from the components of an attribute set, while communicated to him by the owner on a secure transmission channel. Given the values of the components in attribute set $\tilde{A}_i (1 \leq i \leq m)$, a secret key $SK_j$ is defined as,

$$SK_j = h(\sum_{a_i \in \tilde{A}_j} s_i), \quad 1 \leq i \leq n, \ 1 \leq j \leq m.$$

where $h$ is a deterministic cryptographic function, for example a secure one-way hash function, and $s_i$ denotes the random values assigned to the attributes $a_i$. After $m$ keys are generated from attribute sets $\tilde{A}_1, \tilde{A}_2, ..., \tilde{A}_m$, the key assignment function then associates with each user holding attribute set $\tilde{A}_i$ and the secret key $SK_i$.

**Definition 4.3** *(Key Assignment) Key assignment is a function G that releases $SK_i$ to each user $u \in U$ who holds the attribute set $\tilde{A}_i$.*

### 4.3.2 Definition of Attribute-set Based Encryption

The formal definition of the proposed attribute-set based encryption is given in this section. An attribute-set based encryption scheme consists of four probabilistic polynomial-time algorithms: Setup, KeyGen, Encryption and Decryption.

*Setup*$(\lambda, \mathbb{T}, \mathbb{A}) \rightarrow (SK_1, SK_2, ..., SK_m, Matrix(A))$. The setup algorithm takes as input the security parameter $\lambda$, the access structure $\mathbb{T}$ and the attribute universe description $\mathbb{A}$ and outputs $m$ different secret keys for $num_{x_{root}}$ attribute sets and an attribute matrix $Matrix(A)$.

*KeyGen*$(SK_1, SK_2, ..., SK_m) \rightarrow PK$. The key generation algorithm takes as input the different secret keys $(SK_1, SK_2, ..., SK_m)$ and outputs an encryption key $PK$.

*Encryption*$(PK, M) \rightarrow CT$. The encryption algorithm takes as input the encryption parameters $PK$, the message $M$, and an access formula $A$ over the universe attributes and outputs a ciphertext $CT$ such that only users whose private keys satisfy the access formula should be able to extract $M$.

*Decrypt*$(CT, SK_j) \rightarrow M$. The decryption algorithm takes as input a ciphertext $CT$, and one of the private keys $SK_j$ where $1 \leq j \leq m$ and outputs the plaintext $M$.

### 4.3.3 Main Construction

The main construction of the base phase is performed by the data owner, before outsourcing the data item into the cloud. It enforces policy hidden attribute-set based encryption on the data files according to the access policies. Here the polynomial function which is introduced in [MVN99] has been borrowed.

*Setup*$(\mathbb{T}, \mathbb{A}) \rightarrow (SK_1, SK_2, ..., SK_m, Matrix(A))$. In the basic construction, it chooses two large prime numbers $p$ and $q$ such that $q|p-1$, and chooses a generator element $g \in \mathbb{Z}_p^*$ of order $q$. For each attribute $a_i \in \mathbb{A}$, it chooses a random value $s_i \in \mathbb{Z}_p$. With the access structure $\mathbb{T}$, $x_{root}$ may have $num_{x_{root}}$ values the different attribute sets, that is, $m$ secret keys associated with attribute sets $\tilde{A}_1$ to $\tilde{A}_m$. By generating the $Matrix(A)$, these secret keys can be computed as $SK_j = h(\sum_{a_i \subseteq \tilde{A}_i} s_i a_i), \;\; 1 \leq i \leq n, \; 1 \leq j \leq m$.

*KeyGen*$(SK_1, SK_2, ..., SK_m) \rightarrow PK$. On input the secret keys, the key generation algorithm constructs a polynomial function as

$$f(x) = \prod_{j=1}^{m}(x - SK_j) \equiv \sum_{j=0}^{m} a_j x^j \mod q,$$

where $a_j$'s are coefficients. It generates the encryption key $PK = (g^{a_0}, g^{a_1}, ..., g^{a_m})$.

*Encryption*$(PK, M) \rightarrow CT$. The encryption algorithm then chooses random $r \in \mathbb{Z}_q$ and generators $h \in \mathbb{Z}_p^*$, and outputs a ciphertext $(c_1, c_2)$:

$$CT = \begin{cases} c_1 \leftarrow (h^r \cdot g_0^r, g_1^r, ..., g_m^r) \\ c_2 = M \cdot h^r \end{cases}$$

*Decrypt*$(CT, SK_j) \rightarrow M$. For each decryption key $SK_j$, the decryption algorithm computes

$$\begin{cases} h^r \leftarrow h^r \cdot g_0^r \cdot g_1^{SK_j r} \cdot ... \cdot g_m^{SK_j{}^m r} \\ M = c_2 / h^r \end{cases}$$

Thus $M$ can be computed.

## 4.4 Surface Phase

The surface phase is initialized by the data owner and performed by the cloud servers over the outsourced data files. It enforces the dynamic encryption operations over the encrypted data files, when receiving request messages from the data owner. The request messages contain new encryption keys for cloud servers as input. Combining with the base phase, the surface phase allows the server to conduct re-encryption for the users.

### 4.4.1 Server re-encryption Mechanism

The proposed server re-encryption mechanism (SRM) is a mechanism that runs by the cloud server, especially for new user creation or user revocation. This mechanism proceeds in rounds as a state transition diagram, shown as Figure 4.3. During each round, the server listens to the request from the data owner with an encrypted data file index $CT_i$ corresponding to a new public key $PK^*$, and then performs re-encryption on $CT_i$ with $PK^*$ and associate the re-encrypted ciphertext $CT^*$ with index $i$. Finally SRM updates the re-encrypted data $CT^*$ to replace the previous $CT$ and record this replacement in the system.

The main difference between the proposed server re-encryption mechanism and proxy re-encryption is as follows. In proxy re-encryption, it allows a proxy to transform a ciphertext computed under data owner's public-key into one that can be opened by user's secret key. In this case, the data owner could designate a proxy to re-encrypt her file into a format that the user can decrypt using his own secret key. The re-encryption key is generated by the user (decryptor) or generated by using the decryptor's public key. However in the proposed server re-encryption mechanism,

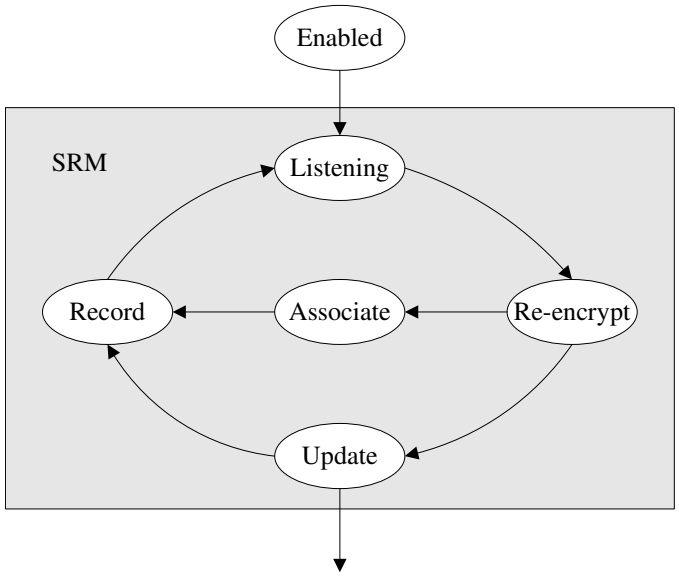


Figure 4.3: The state transition diagram of SRM.

the re-encryption key is assumed to be generated only by the data owner (encryptor) and no additional information is required.

Under the construction of SRM, the data owner only needs to generate server re-encryption keys and does not need to re-send a novel encrypted version of the data files. As for example, if the data file has a size of 10GByte and the request to the cloud server requires a 1MByte data packet, in terms of network traffic, compared to the transmission of the re-encrypted data file, the improvement is in the order of $10^7$.

### 4.4.2 Main Construction of SRM

The main construction of SRM is composed of three algorithms: Setup, Re-encrypt, and Decrypt. Noted that the Decrypt algorithm is not run on SRM, but be operated on the user's side. This algorithm is presented as part of the main construction only for completeness.

*Setup*$(\lambda, CT_i) \rightarrow \mathbb{I}$. The setup algorithm takes in the security parameters $\lambda$ and every index $CT_i$ of the outsourced data, generates an index list $\mathbb{I}$ as output.

*Re-encrypt*$(CT, PK^*) \rightarrow CT^*$. On receiving $PK^*$ as $((h^v \cdot (g_0^*)^v, (g_1^*)^v, ..., (g_m^*)^v), e)$, where $e = h^{-r} \cdot h^v$, output the re-encrypted ciphertext as $(c_1^*, c_2^*)$:

$$CT = \begin{cases} c_1^* \leftarrow (h^v \cdot (g_0^*)^v, (g_1^*)^v, ..., (g_m^*)^v) \\ c_2^* = c_2 \cdot e = M \cdot h^r \cdot h^{-r} \cdot h^v = M \cdot h^v \end{cases}$$

*Decrypt*$(CT^*, SK_j) \rightarrow M$. For each $SK_j$, the decryption algorithm computes $M$ as in the base phase.

## 4.5 Proof and Analysis

### 4.5.1 Security Notations

The informal definitions of the various security properties are first presented. A construction of encryption system must possess these security properties.

**Confidentiality**. Entities (including the cloud server) other than the intended recipients specified by the access structure should not be able to learn anything about the underlying plaintext message, even if users who are not intended recipients collude.

**Access-privacy**. The cloud server or any recipient should not be able to gain any knowledge of the access structure except that the recipient knows whether he satisfies the access policy. Also, colluded users (who do not meet the access structure) should be unable to gain any knowledge about the access policy.

**Definition 4.4** *(Security) A construction of the encryption system is secure if it holds confidentiality and access-privacy.*

### 4.5.2 Security Model

Formally speaking, the confidentiality of an encryption system is defined as follows.

**Definition 4.5** *(C-IND-CPA-RCA) An encryption system has Ciphertext Indistinguishability against Chosen Plaintext Attack and Restricted Collusion Attack (C-IND-CPA-RCA) if no PPT adversary $\mathcal{A}$ can win the following games against the Challenger $\mathcal{C}$ with probability non-negligibly greater than $1/2$. Meanwhile, none of the collusion of corrupted users collectively satisfies the target access structure throughout the games.*

Below a game-base approach is used to define the security formally. The adversary $\mathcal{A}$ models a malicious user or a cloud server which is *Byzantine*, i.e., can

behave arbitrarily, however, cannot collude with users that satisfies the target access structure. In the formalized game, the adversaries are trying to break the C-IND-CPA-RCA security.

Let $\mathcal{C}$ be the game simulator. Then the game 1 runs as follows:

**Setup Phase** $\mathcal{C}$ sets up the encryption system and makes all public parameters such as the attributes in the system available to $\mathcal{A}$.

**Probing Phase 1** The adversary $\mathcal{A}$ has the ability to arbitrarily: a) register a new user into the system, and decide the set of attributes possessed by the user being registered. For simplicity, it is assumed that a user immediately acquires all the credentials for his attributes upon registration, b) issue queries for attribute sets $\tilde{A}_i$ for many access structure $\mathbb{T}_j$ where $\tilde{A}_i \subseteq \mathbb{T}_j$ for all $j$, c) issue retrieval requests to the owner for encryption keys, and d) corrupt an honest user, thereby learning his secrets and acting on behalf of him.

**Challenge Phase** The adversary $\mathcal{A}$ submits two messages $(M_0, M_1)$ of equal length and a challenge access structure $\mathbb{T}$ of his choice under the restriction that, none of the collusion of corrupted users collectively satisfies the challenge access structure throughout the games. Then the Challenger $\mathcal{C}$ flips a fair coin $b \in_R \{0, 1\}$, generates $\gamma$ with $\tilde{A}_i$ and encrypts message $M_b$ with $\gamma$. The ciphertext is then passed to $\mathcal{A}$.

**Probing Phase 2** Probing phase 1 is repeated.

**Guess Phase** Eventually $\mathcal{A}$ outputs a guess $b^*$ of $b$. $\mathcal{A}$ wins if and only if $b^* = b$.

The advantage of an adversary $\mathcal{A}$ in this game is defined as $\Pr[b^* = b] - 1/2$.

Now the access-privacy of an encryption system is defined and formalized by using game 2.

**Definition 4.6** *(AP-IND-CPA-RCA) An encryption system has Access Privacy Indistinguishability against Chosen Plaintext Attack and Restricted Collusion Attack (AP-IND-CPA-RCA) if no PPT adversary $\mathcal{A}$ can win the following games against the Challenger $\mathcal{C}$ with probability non-negligibly greater than $1/2$. Meanwhile, all corrupted users satisfy none of the target access structures throughout the game and the target access structures have same number of satisfying attribute sets.*

Let $\mathcal{C}$ and $\mathcal{A}$ denote the Challenger and Adversary respectively, then game 2 runs as follows:

**Setup Phase** $\mathcal{C}$ sets up the encryption system and makes all public parameters such as the attributes in the system available to $\mathcal{A}$.

**Probing Phase 1** The adversary $\mathcal{A}$ has the ability to arbitrarily: a) register a new honest user into the system, and decide the set of attributes possessed by the user being registered. For simplicity, it is assumed that a user immediately acquires all the credentials for his attributes upon registration, b) make deposit and retrieval requests to the owner or cloud server, and c) corrupt an honest user, thereby learning his secrets and acting on behalf of him.

**Challenge Phase** The adversary $\mathcal{A}$ sends to $\mathcal{C}$ a message $M^*$ and two valid access structures $\mathbb{T}_1$ and $\mathbb{T}_2$ of his choice under the restriction same in the above definition. Then the Challenger $\mathcal{C}$ flips a fair coin $b \in_R \{0, 1\}$ and encrypts message $M$ under access structure $\mathbb{T}_b$. The resulting ciphertext $C^*$ is then passed to $\mathcal{A}$.

**Probing Phase 2** $\mathcal{A}$ may do whatever he is allowed to in Probing phase 1.

**Guess Phase** Eventually $\mathcal{A}$ outputs a guess $b^*$ of $b$. $\mathcal{A}$ wins if and only if $b^* = b$.

The advantage of an adversary $\mathcal{A}$ in this game is defined as $\Pr[b^* = b] - 1/2$.

### 4.5.3 Proof Of Security

The security of encryption is reduced to the hardness of the DDH assumption.

**Theorem 4.1** *If the DDH assumption holds for $\mathbb{Z}_p^*$, then the encryption system has Ciphertext Indistinguishability against Chosen Plaintext Attack and Restricted Collusion Attack (C-IND-CPA-RCA).*

*Proof:* Suppose there exists a polynomial-time adversary $\mathcal{A}$ that can attack the proposed scheme in the standard model with advantage $\epsilon$. A simulator $\mathcal{B}$ which can play the DDH game with advantage $\epsilon/2$ is built. The simulator $\mathcal{B}$ is given a DDH problem instance $(g, g^\alpha, g^\beta, R)$ from the DDH challenger generated in the following process. The challenger flips a fair binary coin $\mu$, outside of $\mathcal{B}$'s view. It sets the problem instance as $(g, g^\alpha, g^\beta, R)$ where $R = g^{\alpha\beta}$ if $\mu = 0$ or $R$ is a random group element if $\mu = 1$. The goal of $\mathcal{B}$ is to output the big $\mu$.

**Setup** The simulator $\mathcal{B}$ chooses public parameters $\mathbb{G}$ and $g$, and sets $h = g^k$ for some random value $k$.

**Probing 1** The adversary $\mathcal{A}$ adaptively makes requests for registering new users into the system with chosen attribute set $\tilde{A}_i$ for many access structure $\mathbb{T}_j$. $\mathcal{B}$ generates the encryption key and the decryption key according to the protocol.

**Challenge** The adversary $\mathcal{A}$ will submit two challenge messages $M_0$ and $M_1$ to the simulator and a challenge access structure $\mathbb{T}$. Define the polynomial $f(x) = \prod_{j=1}^{m}(x - SK_j)$ such that the $SK_j$'s are the user secret key satisfying the access structure $\mathbb{T}$. Without loss of generality, let $SK_0 = \alpha$ and suppose the rest of the $SK_j$'s are just random numbers. Rewrite $f(x) = \prod_{j=1}^{m}(x - SK_j) \equiv \sum_{j=0}^{m} a_j x^j \mod q$. The simulator can express each coefficient $a_j$ as $a_j = u_j \cdot \alpha + v_j$ for some known $u_j$, $v_j$ for $j = 0$ to $m$. $\mathcal{B}$ sets the encryption key as $(g^{a_j} = (g^{\alpha})^{u_j} g^{v_j})_{j=0}^{m}$. The simulation is perfect because due to the restriction of the game, $\mathcal{A}$ has no information about the $SK_j$'s.

The simulator flips a fair binary coin $b$, and returns an encryption of $M_b$ as follows:

$$CT = \begin{cases} c_1 \leftarrow (g^{\beta})^k \cdot R^{u_0} \cdot (g^{\beta})^{v_0}, R^{u_1} \cdot (g^{\beta})^{v_1}, ..., R^{u_m} \cdot (g^{\beta})^{v_m} \\ c_2 = M_b \cdot (g^{\beta})^k \end{cases}$$

If $\mu = 0$, $R = g^{\alpha\beta}$. Then by inspection, the ciphertext is a valid ciphertext for message $M_b$, with the randomness $r = \beta$.

If $\mu = 1$, then $R$ is just a random group element. Thus the ciphertext will be a random element of $\mathbb{G}$ from the adversary's viewpoint and the ciphertext contains no information about $M_b$.

**Probing 2** The simulator acts exactly as it did in phase 1.

**Guess** $\mathcal{A}$ will submit $b^*$ of $b$. If $b^* = b$, the simulator will output $\mu^* = 0$ to indicate that it was given a valid DDH triple; otherwise, it will output $\mu^* = 1$ to indicate it was given a random element triple.

In the case where $\mu = 1$, the adversary gains no information about $b$. Therefore, it has $\Pr[b^* = b|\mu = 1] = \frac{1}{2}$. Since the simulator guesses $\mu^* = 1$ when $b^* \neq b$, it has $\Pr[\mu^* = \mu|\mu = 1] = \frac{1}{2}$. If $\mu = 0$ then the adversary sees an encryption of $M_b$. The adversary's advantage in this situation is $\epsilon$ by assumption. Therefore, $\Pr[b^* = b|\mu = 0] = \frac{1}{2} + \epsilon$. Since the simulator guesses $\mu^* = 0$ when $b^* = b$, it has $\Pr[\mu^* = \mu|\mu = 0] = \frac{1}{2} + \epsilon$. The overall advantage of the simulator in the DDH game is:

$$\frac{1}{2}\Pr[\mu^* = \mu|\mu = 0] + \frac{1}{2}\Pr[\mu^* = \mu|\mu = 1] - \frac{1}{2} = \frac{1}{2} \cdot (\frac{1}{2} + \epsilon) + \frac{1}{2} \cdot \frac{1}{2} - \frac{1}{2} = \frac{1}{2}\epsilon.$$

**Theorem 4.2** *If the DDH assumption holds for $\mathbb{Z}_p^*$, then the encryption system has Access Privacy Indistinguishability against Chosen Ciphertext Attack and Restricted Collusion Attacks (AP-IND-CPA-RCA).*

*Proof:* Suppose there exists a polynomial-time adversary $\mathcal{A}$ that can attack the proposed scheme in the standard model with advantage $\epsilon$. A simulator $\mathcal{B}$ which can play the DDH game with advantage $\epsilon/2$ is built.

Again, $\mathcal{B}$ is given a DDH problem instance $(g, g^\alpha, g^\beta, R)$ and its goal is to determine if $R = g^{\alpha\beta}$ indicating $\mu = 0$ or $R$ is just a random group element indicating $\mu = 1$.

**Setup** The simulator $\mathcal{B}$ chooses public parameters $\mathbb{G}$ and $g$, and sets $h = g^k$ for some random value $k$.

**Probing 1** The adversary $\mathcal{A}$ adaptively makes requests for several access structures $\mathbb{T}_j$. $\mathcal{B}$ generates the encryption key and the decryption key according to the protocol.

**Challenge** The adversary $\mathcal{A}$ will submit a challenge message $M$ and two challenge access structures $\mathbb{T}_0$, $\mathbb{T}_1$ to the simulator. $SK_{j,b}$ for $j = 0$ to $m$ and $b \in \{0, 1\}$ is used to denote the set of secret keys for access structure $\mathbb{T}_b$. Since the two access structures are different, there exists at least one secret key that is not common to both access structure. Without loss of generality, let $SK_{0,0}$ be that secret key. The simulator $\mathcal{B}$ sets $SK_{0,0} = \alpha$ and randomly picks values for all other $SK_{j,b}$'s. The encryption key of $\mathbb{T}_0$ is then $(g^{a_{0,0}}, \ldots, g^{a_{m,0}})$ such that each $a_{j,0}$ is of the form $u_j\alpha + v_j$ for $j = 0$ to $m$. Thus, $g^{a_{j,0}}$ can be computed by $\mathcal{B}$ as $(g^\alpha)^{u_j} g^{v_j}$. The encryption key of $\mathbb{T}_1$ is $(g^{a_{0,1}}, \ldots, g^{a_{m,1}})$ such that each $a_{j,1}$ is known to the simulator.

It encrypts $M$ under $\mathbb{T}_0$ as follows.

$$CT = \begin{cases} c_1 \leftarrow (g^\beta)^k \cdot R^{u_0} \cdot (g^\beta)^{v_0}, R^{u_1} \cdot (g^\beta)^{v_1}, ..., R^{u_m} \cdot (g^\beta)^{v_m} \\ c_2 = M \cdot (g^\beta)^k \end{cases}$$

If $\mu = 0$, $R = g^{\alpha\beta}$. Then by inspection, the ciphertext is a valid ciphertext for message $M$ under the access structure $\mathbb{T}_0$. If $\mu = 1$, then $R$ is just a random element and thus the ciphertext contains no information about $T_b$.

**Probing 2** The simulator acts exactly as it did in phase 1.

**Guess $\mathcal{A}$** will submit $b^*$. If $b^* = 0$, the simulator will output $\mu^* = 0$ to indicate that it was given a valid DDH triple; otherwise, it will output $\mu^* = 1$ to indicate it was given a random element triple.

In the case where $\mu = 1$, the adversary gains no information about $\mathbb{T}_b$. Therefore, it has $\Pr[b^* = 0|\mu = 1] = \frac{1}{2}$. Since the simulator guesses $\mu^* = 1$ when $b^* \neq 0$, it has $\Pr[\mu^* = \mu|\mu = 1] = \frac{1}{2}$. The adversary's advantage in the case when $\mu = 0$ is $\epsilon$ by definition. Therefore, $\Pr[b^* = 0|\mu = 0] = \frac{1}{2} + \epsilon$. Since the simulator guesses $\mu^* = 0$ when $b^* = 0$, it has $\Pr[\mu^* = \mu|\mu = 0] = \frac{1}{2} + \epsilon$. The overall advantage of the simulator in the DDH game is:

$$\frac{1}{2}\Pr[\mu^* = \mu|\mu = 0] + \frac{1}{2}\Pr[\mu^* = \mu|\mu = 1] - \frac{1}{2} = \frac{1}{2} \cdot (\frac{1}{2} + \epsilon) + \frac{1}{2} \cdot \frac{1}{2} - \frac{1}{2} = \frac{1}{2}\epsilon.$$

## 4.6 Conclusion

There is an emerging trend towards data resourcing where data management is outsourced to clouds that provide storage capabilities and high-bandwidth distribution channels. In this chapter, an encryption scheme for a two-tier system is proposed to achieve flexible and fine-grained access control in the cloud. Most of the computation-intensive tasks are delegated to cloud servers without leaking private data. The security of the proposed scheme is proven in the standard model.

# Chapter 5

# Privacy-Enhanced Keyword Search in Clouds

The advent of cloud computing has dramatically changed the IT scene, as it offers cost savings and improvements to major operations. Nevertheless, the major obstacle relies on the effort on how to secure sensitive data files that are outsourced to the cloud environment. To ensure confidentiality, the sensitive data are usually encrypted prior to being outsourced. Nevertheless, effective data utilization remains a challenging task and there is a clear need for a secure and efficient searching mechanism over the encrypted data in the cloud, to increase the usability of the secure cloud environment. Unfortunately, existing works in the area of secure searching in the outsourcing scenario usually incur high computational complexity, which makes the approach impractical. In this chapter, an efficient keyword search scheme for cloud computing is proposed. The proposed solution is very simple, and it enables efficient multi-user keyword search over outsourced data files in the cloud environment, without leaking any private information about either the data owner or users in the search query. The security requirements are formally defined and the security of the proposed scheme is proven under a simple assumption in the standard model.

## 5.1 Introduction

Due to its low cost, robustness and flexibility, cloud computing changes the way entities manage their data and offers individuals and companies with affordable storage, professional maintenance and adjustable space. Among the four cloud computing deployed models that includes: public, private, community and hybrid, public cloud where the outsourced resources can be accessed by the general public has gain a dramatic growth. By using a public cloud, a variety of users could access or share information that are stored in the cloud, independent of their different locations.

Meanwhile, it also makes effective data utilization a challenging task as the outsourced data is usually in the encrypted form.

To set the scene, let us consider the following scenario. Let us consider a user Alice who uses a public cloud to store her personal data files such as family photos, blogs and working documents. To prevent the cloud server from learning the contents, she encrypts all her files prior to sending her data to the cloud. Once a while, she would like to access her files from different devices, such as a Boxee Box, an Apple TV or even her iPhone. Naturally, Alice would not remember all the contents that she has stored in the cloud. Therefore, there is a need for efficient searching over her outsourced files using the appropriate keywords. Since some of her mobile devices are only equipped with limited computational power, the searching mechanism should be very efficient, and it should ideally avoid using the relatively expensive techniques in public key cryptography, such as bilinear pairings.

In other some cases, she would also like to share some of the files with her family and friends. For example, data files with labels "family" and "friends" from Alice would be accessible by her family member and friends, respectively. However, this requires Alice to define whom her family and friends are and requires the cloud server to enforce access control. This requirement may burden the regular users with the required expensive operations, and it may also reveal some information with regards to Alice's social networks.

Thus it is necessary to look for a practical scheme which provides:

- an efficient data search, and
- a simple access control.

**Overview of The Proposed Approach.** It turns out that a simple and straightforward approach could fit Alice's requirements. Before outsourcing each data file, Alice attaches a "hidden index" $h$ related to a certain keyword $w$ to it. The hidden index $h$ is computed as $H(w)$ for a hash function $H$ using a keyword $w$. This can be easily extended to the multi-keyword case where each file $F$ is attached with several hidden indexes for the relevant keywords. In order to search all files related to a keyword $w^*$, Alice computes $h^* = H(w^*)$ and sends $h^*$ to the cloud, who returns all the data files attached with hidden index $h^*$. This idea can be used for simple access control as well. For instance, data files to be shared with Alice's friends could be attached be a hidden index with $w$ as "friends". Alice's friends could then access

those files using "friends" as the searching keyword. In this case a keyword plays the role of a password.

Unfortunately, this approach is inadequate in terms of the keyword search as well as a simple access control. Firstly, it leaks some information about the keyword to the cloud since the cloud can guess the keywords by testing whether the hash of it matches with the hidden index. Secondly, the keyword are human-memorable and is thus suspectable to the guessing attack. This is thus unsuitable even for a simple access control purpose.

One possible way to deal with these vulnerabilities is to generate a random value $t_i$ for each possible keyword $w_i$. Since $t_i$ is completely random, the scheme will not be vulnerable to the guessing attack. However, the drawback is apparent. Alice is required to build up a look up table which links every keyword to its corresponding random number. Firstly, the table could be large if Alice would set the file name as keyword (which is natural since this would allow her to search using the file name). Secondly, Alice has to keep an up-to-date copy of the table in all her devices.

Finally, the above issue is tackled by the use of the pseudo-random function (PRF). Instead of generating a random value $t_i$ for each possible keyword $w_i$, Alice computes $t_i$ as the output of the pseudo-random function with input $w_i$ and a secret seed $s$. The value of $s$ is kept secret and is stored in all Alice's devices. Indeed, the issue of storing the look up table has been reduced to just storing merely one secret seed $s$.

The proposed system is called simple privacy-enhanced keyword search in clouds (**SPEKS**) to emphasize its simplicity. The term privacy-enhanced is used to reflect the privacy guarantee about the proposed scheme. The cloud server can still tell if two data files share the same keyword. Exact security guarantee provided by the proposed system will be formalized in subsequent sections.

**Related Works.** Existing works close to the proposed scheme can be found in the areas of "searching with privacy" and "searching on private-key-encrypted data". In theory, the classical work of Goldreich and Ostrovsky [GO96] on oblivious RAMs could resolve the problem of doing (private) searches on remote encrypted data, where oblivious RAMs hided all information about the RAM use from a remote and potentially malicious server with a poly-logarithmic overhead in all parameters (including computation and communication). Although their scheme is asymptotically efficient and nearly optimal, it does not appear to be efficient in practice as large

constants are hidden in the big-$O$ notation.

In an effort to reduce the round complexity associated with oblivious RAMs, Song *et al.* [SWP00] presented a solution for searchable encryption and after that how to do keyword searches on encrypted data efficiently was raised. In [SWP00], they achieved searchable encryption by constructing a special two-layered encryption for each word. Given a trapdoor, the server could strip the outer layer and assertain whether the inner layer was in the correct form. The limitations in this construction are as follows. First, it is not compatible with existing file encryption schemes and a specific encryption method must be used. Second, while the construction is proven to be a secure encryption scheme, it is not proven to be a secure searchable encryption scheme. Third, the distribution of the underlying plaintexts is vulnerable to statistical attacks. Their approach may leak the locations of the keyword in a file. Finally, searching is linear in the length of the document collection.

The above limitations are addressed by Goh [jG03], Chang and Mitzenmacher [CM05] and also Curtmola, Garay, Kamara and Ostrovsky [CGKO06]. In [jG03], they built an index of keywords for each file using a Bloom filter with pseudo-random functions. One inherent problem with this Bloom-filter-based approach is that Bloom filters can induce false positives, which would potentially cause mobile users to download extra files not containing the keyword. In [CM05], Chang and Mitzenmacher achieved the notion of security to IND2-CKA for chosen keyword attack, however their scheme cannot guarantee that the trapdoors do not leak any information about the words being queried. In [CGKO06], they proposed a multi-user construction that was efficient on the server side. However, every node in the link list has to be augmented with information about the file index of the next node.

In a different direction, Boneh, di Crescenzo, Ostvrosky and Persiano [BCOP04] and Boneh, Kuchilevitz, Ostvrosky and Skeith [BKOI07] studied the problem of how to search on data encrypted by a public-key cryptosystem. These schemes are motivated by an encrypted email system. Their constructions, however, have an overhead in search time that is proportional to the square root of the database size, which is far less efficient than the best private-key solutions. Boneh *et al.*'s approach [BCOP04] was known to be the seminal public key encryption scheme with keyword search (PEKS). It was observed in [BSNS05] that Boneh *et al.*'s scheme [BCOP04] required a secure channel, which made it impractical. Hence, Baek, Susilo and Safavi-Naini [BSNS05] proposed the notion of secure-channel-free PEKS to improve this drawback. This work had been further extended and revised in the recent

literature, such as [RPSL09, RPSL10]. Byun *et al.* [BRPL06] suggested the notion of a *keyword-guessing attack* and showed that the existing schemes were insecure against this attack, given that the number of possible keywords was bounded by some polynomial. They provided an open problem on how to construct a PEKS with designated verifier that was secure against keyword-guessing attacks. This question was answered affirmatively in [RPSL10]. In order to realize the practicality of PEKS, the combination of a public key encryption scheme with PEKS to make a single integrated entity had been studied in [BSNS06].

**Organization of This Chapter.** The rest of this chapter is organized as follows. In the next section, the notion of **SPEKS** and its security requirements are formalized. In Section 5.3, the construction and security analysis are given. In Section 5.4, the performance of the proposed scheme is estimated. Section 5.5 concludes this chapter.

## 5.2 Model

In this section, the notion of **SPEKS** and its security requirements are formalized.

### 5.2.1 Syntax

**SPEKS** is a tuple of five algorithms, namely, ParamGen, KeyGen, IndexGen, TokenGen, Test, whose definition is given below.

- ParamGen. On input a security parameter $\lambda$, this algorithm outputs a system-wide parameter PARAM. PARAM is assumed to be an implicit to all the algorithms below.
- KeyGen. This algorithm outputs a secret key $s$.
- IndexGen. On input a keyword $w \in \{0,1\}^*$, a secret key $s$, this algorithm outputs a value $h$. The value $h$ is called a hidden index.
- TokenGen. On input a keyword $w \in \{0,1\}^*$, a secret key $s$, this algorithm outputs a value $\omega$. The value $\omega$ is called a hidden token.
- Test. On input a hidden index $h$, a hidden token $\omega$, this algorithm outputs 1 or 0.

As usual, correctness is of **SPEKS** is defined as follows.

**Definition 5.1** *(Correctness) For any $\lambda$ and any keyword $w$,* $\mathsf{Test}(\text{PARAM}, h, \omega) = 1$ *if there exists* $\mathsf{ParamGen}(\lambda) = \text{PARAM}$, $s = \mathsf{KeyGen}(\text{PARAM})$, $h = \mathsf{IndexGen}(\text{PARAM}, s, w)$ *and* $\omega = \mathsf{TokenGen}(\text{PARAM}, s, w)$.

### 5.2.2 Typical Use of SPEKS

Figure 5.1 briefly explain how a user Alice and the cloud server employ the algorithms in a typical scenario. Firstly, the cloud server invokes ParamGen to generate the parameters[1]. Alice invokes KeyGen to create her secret key $s$. Before outsourcing her data file $F$ to the cloud server, Alice would choose several suitable keywords, say, $w_1$, ..., $w_n$ and invokes $h_{F,i} = \mathsf{IndexGen}(s, w_i)$. She submits $F$ along with $h_{F,i}$ to the cloud. $F$ is possibly the encryption of Alice's data. How $F$ is generated is out of scope of this chapter.

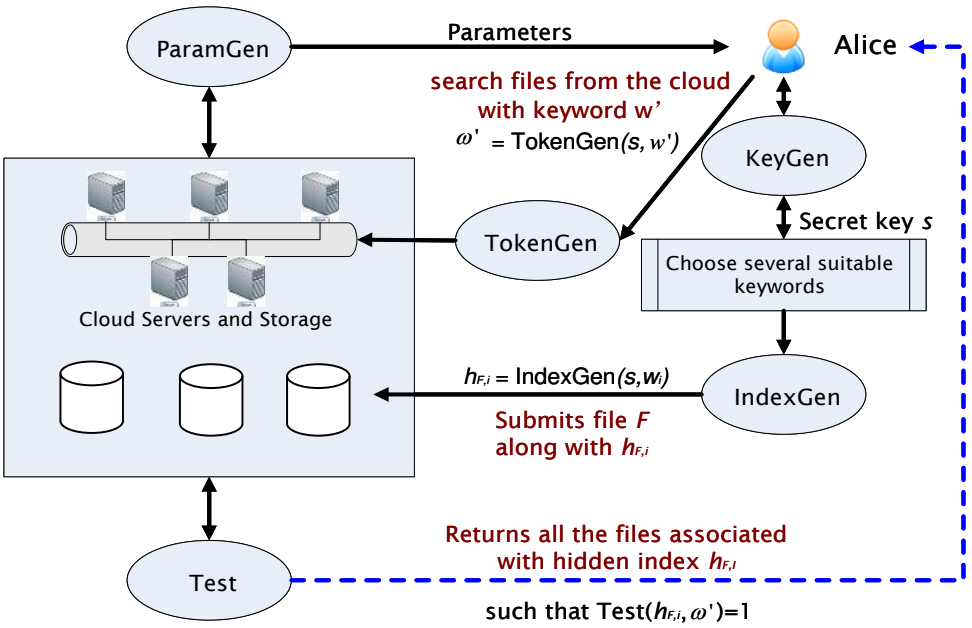


Figure 5.1: Typical use of **SPEKS**.

When Alice would like to search her files from the cloud with keyword $w^*$, she invokes $\omega^* = \mathsf{TokenGen}(s, w^*)$ and submits $\omega^*$ to the cloud. The cloud returns all the files associated with hidden index $h_{F,i}$ such that $\mathsf{Test}(h_{F,i}, \omega^*) = 1$. Alice could share her files with keyword $w^*$ to others by giving them the value $\omega^*$.

[1]It is noted that Alice could generate the parameters, but in this case, the cloud server is required to store a number of parameters since the cloud server needs to serve multiple users.

### 5.2.3 A SPEKS System

In this section, the proposed SPEKS system is described. Essentially, the scheme can be divided into two phases, namely *Setup* and *Search*. It should be highlighten that the cloud user's storage only consists of a small and constant value, in addition to the keywords.

- **Setup:** The cloud server $S$ runs the algorithm ParamGen to outputs a system-wide parameter PARAM. The cloud user $U$ runs the algorithm KeyGen to generate the secret key $s$. $U$ then runs the algorithm IndexGen and algorithm TokenGen to generate the hidden index $h$ and hidden token $\omega$. Then $U$ may now delete the original data files from his/her local storage.
- **Search:** When the cloud user $U$ requests to search for a hidden token $\omega$:

  1. $S$ computes $H(\omega)$.
  2. $S$ runs the algorithm Test, checks whether there is a hidden index $h$ equals with $H(\omega)$, and sends $U$ back the found data file.

  When a friend $F$ of $U$ requests to search for a keyword $w_i$:

  1. $F$ sends the keyword $w_i$ to $U$ and $U$ can return the hidden token $\omega$.
  2. $F$ sends $\omega$ to $S$.
  3. $S$ computes $H(\omega)$, checks whether there is a $h$ that matches with $H(\omega)$, and returns the corresponding data file to $F$.

### 5.2.4 Security Requirements

Two security requirements are identified for **SPEKS**. The first one regards privacy. Specifically, no one, not even the cloud server, should be able to obtain information about the underlying keyword when given the hidden token and hidden index. The second one concerns about the basic access control. No one should be able to compute the hidden token when given the hidden index. These two requirements are formalized by using security game played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

**Game Privacy.**

- **Setup** Challenger $\mathcal{C}$ invokes $\mathsf{ParamGen}(1^\lambda) = \text{PARAM}$, and $\mathsf{KeyGen}(\text{PARAM}) = s$. Here PARAM is given to the adversary $\mathcal{A}$.

- **Query Phase 1** $\mathcal{A}$ can issue two types of queries:

  1. Index Query: $\mathcal{A}$ submits a keyword $w$ and $\mathcal{C}$ replies with

  $$\mathsf{IndexGen}(\text{PARAM}, s, w).$$

  2. Token Query: $\mathcal{A}$ submits a keyword $w$ and $\mathcal{C}$ replies with

  $$\mathsf{TokenGen}(\text{PARAM}, s, w).$$

- **Challenge Phase** $\mathcal{A}$ submits two keywords $w_0$, $w_1$. $\mathcal{C}$ flips a fair coin $b \in \{0, 1\}$ and computes $h_b = \mathsf{IndexGen}(\text{PARAM}, s, w_b)$. $h_b$ is returned to $\mathcal{A}$ as the challenge.

- **Query Phase 2** $\mathcal{A}$ can issue the same type of queries as in Query Phase 1 except it cannot submit queries with input $w_0$ and $w_1$.

- **Output** $\mathcal{A}$ outputs a guess bit $b'$. $\mathcal{A}$ wins the game if $b = b'$.

The advantage of $\mathcal{A}$ in Game Privacy is defined as the probability that $\mathcal{A}$ wins minus 1/2.

**Game Authenticity.**

- **Setup** Challenger $\mathcal{C}$ invokes

  $$\mathsf{ParamGen}(1^\lambda) = \text{PARAM} \quad \text{and} \quad \mathsf{KeyGen}(\text{PARAM}) = s.$$

  PARAM is given to the adversary $\mathcal{A}$.

- **Query Phase 1** $\mathcal{A}$ can issue two types of queries:

  1. Index Query: $\mathcal{A}$ submits a keyword $w$ and $\mathcal{C}$ replies with

  $$\mathsf{IndexGen}(\text{PARAM}, s, w).$$

  2. Token Query: $\mathcal{A}$ submits a keyword $w$ and $\mathcal{C}$ replies with

  $$\mathsf{TokenGen}(\text{PARAM}, s, w).$$

- **Challenge Phase** $\mathcal{A}$ submits one keyword $w'$. $\mathcal{C}$ computes $h' = \mathsf{IndexGen}(\text{PARAM}, s, w')$. $h'$ is returned to $\mathcal{A}$ as the challenge.

- **Query Phase 2** $\mathcal{A}$ can issue the same type of queries as in Query Phase 1 except it cannot submit queries with input $w'$.

- **Output** $\mathcal{A}$ outputs a value $\omega'$. $\mathcal{A}$ wins the game if $\mathsf{Test}(\text{PARAM}, h', \omega') = 1$.

The advantage of $\mathcal{A}$ in Game Authenticity is defined as the probability that $\mathcal{A}$ wins.

**Definition 5.2** *(Security) A construction of* ***SPEKS*** *is secure if no PPT adversary* $\mathcal{A}$ *can win Game Privacy or Game Authenticity with non-negligible advantage.*

## 5.3 The Construction and Security Analysis

### 5.3.1 Main Construction

- ParamGen. On input $\lambda$, output a one-way hash function $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$.

- KeyGen. Randomly pick a pseudo-random function $\text{PRF} : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$ and a bitstring $s \in_R \{0,1\}^\lambda$ and output $s$.

- IndexGen. On input $w$ and $s$, output $h = H(\text{PRF}(s, w))$.

- TokenGen. On input $w$ and $s$, output $\omega = \text{PRF}(s, w)$.

- Test. On input a hidden index $h$ and a hidden token $\omega$, output 1 if and only if $h = H(\omega)$ and 0 otherwise.

### 5.3.2 Security Proof

**Theorem 5.1** *The construction of* ***SPEKS*** *is secure if the pseudo-random function* PRF *employed is secure and the hash function* $H$ *employed is one-way.*

*Proof:* The proof is divided into two parts. In the first part, it is shown that if there exists an adversary $\mathcal{A}$ that has non-negligible probability in wining Game Privacy, a simulator $\mathcal{S}$ can be constructed to distinguish a pseudo-random function PRF from a random function $R$. In the second part, it is shown that if there exists an adversary $\mathcal{A}$ that has non-negligible probability in wining Game Authenticity, a simulator $\mathcal{S}$ which breaks the one-way property of the hash function $H$ can be constructed.

**Privacy.**

- **Setup** $\mathcal{S}$ is given a function $F$ and its goal is to distinguish if $F$ is a random function or not. Suppose with probability $1/2$ $\mathcal{S}$ is given a truly random function. $\mathcal{S}$ can query the function $F$ adaptively. $\mathcal{S}$ chooses a one-way hash function $H$ and set $H$ as PARAM.
- **Query Phase 1** $\mathcal{S}$ answers the queries as follows.
  1. Index Query: $\mathcal{A}$ submits a keyword $w$ and $\mathcal{S}$ replies with $H(F(w))$ by querying $F$.
  2. Token Query: $\mathcal{A}$ submits a keyword $w$ and $\mathcal{S}$ replies with $F(w)$ by querying $F$.
- **Challenge Phase** $\mathcal{A}$ submits two keywords $w_0$, $w_1$. $\mathcal{S}$ flips a fair coin $b \in \{0, 1\}$ and computes $h_b = H(F(w))$. $h_b$ is returned to $\mathcal{A}$
- **Query Phase 2** $\mathcal{S}$ answer $\mathcal{A}$'s queries in the same way as in Query Phase 1.
- **Output** $\mathcal{A}$ outputs a guess bit $b'$.

If $\mathcal{A}$ guesses correctly, $\mathcal{S}$ concludes $F$ is not a random function. Otherwise, $\mathcal{S}$ concludes $F$ is a random function. Suppose $\mathcal{A}$ wins with probability $1/2+\epsilon$, probability that $\mathcal{S}$ distinguishes correctly is $1/2+\epsilon/2$. The reason is that if $F$ is a random function, probability that $\mathcal{A}$ wins is exactly $1/2$ since $h_b$ contains no information about $b$. On the other hand, if $F$ is not a random function, $\mathcal{A}$ can win with probability $1/2 + \epsilon$. Thus, $\mathcal{S}$ answers correctly with probability $1/2 + \epsilon/2$.

**Authenticity.** Here a simply game-hoping [Sho04, BP06] is used. In the first game, denoted as Game Authenticity Real, the behavior of simulator $\mathcal{S}$ is described below.

- **Setup** $\mathcal{S}$ chooses a pseudo-random function PRF with seed $s$ and a hash function $H$. $H$ is given to $\mathcal{A}$ as PARAM.
- **Query Phase 1** $\mathcal{S}$ answers the queries as follows.
  1. Index Query: $\mathcal{A}$ submits a keyword $w$ and $\mathcal{S}$ replies with $H(\text{PRF}(s, w))$.
  2. Token Query: $\mathcal{A}$ submits a keyword $w$ and $\mathcal{S}$ replies with $\text{PRF}(s, w)$.

- **Challenge Phase** $\mathcal{A}$ submits one keywords $w'$. $\mathcal{S}$ returns $h_b = H(\text{PRF}(s, w'))$ to $\mathcal{A}$.
- **Query Phase 2** $\mathcal{S}$ answers $\mathcal{A}$'s queries in the same way as in Query Phase 1.
- **Output** $\mathcal{A}$ outputs a guess bit $b'$.

In the second game, denoted as Game Authenticity Modified, the simulator $\mathcal{S}$'s behavior is defined as follows.

- **Setup** $\mathcal{S}$ is given a hash function $H$, a value $y$ and its goal is to compute a value $x$ such that $y = H(x)$. $\mathcal{S}$ gives $\mathcal{A}$ $H$ as PARAM.
- **Query Phase 1** For every keyword $w$ submitted by $\mathcal{A}$, $\mathcal{S}$ chooses a random value $r_w$ and maintains a list of tuples $(w, r_w)$. $\mathcal{S}$ then answers the queries as follows.
  1. Index Query: $\mathcal{A}$ submits a keyword $w$ and $\mathcal{S}$ replies with $H(r_w)$
  2. Token Query: $\mathcal{A}$ submits a keyword $w$ and $\mathcal{S}$ replies with $r_w$
- **Challenge Phase** $\mathcal{A}$ submits one keyword $w'$. $\mathcal{S}$ returns $y$ to $\mathcal{A}$.
- **Query Phase 2** $\mathcal{S}$ answers $\mathcal{A}$'s queries in the same way as in Query Phase 1.
- **Output** $\mathcal{A}$ outputs a value $\omega'$.

$\mathcal{A}$ wins if and only if $H(\omega') = y$. Thus, $\mathcal{S}$ outputs $x = \omega'$ as the pre-image of $y$. It remains to argue the advantage of $\mathcal{A}$ in Game Authenticity Real and Game Authenticity Modified are the same. A simple argument will do. If the advantage of $\mathcal{A}$ in both games are difference, it is straight-forward to use $\mathcal{A}$ to distinguishes PRF from a truly random function.

## 5.4 Performance

**Efficiency.** It is straightforward to see the construction of **SPEKS** is very efficient. Generation of a hidden index requires evaluation of one hash function and one pseudo-random function. Generation of a hidden token requires evaluation of one pesudo-random function. Testing if a hidden token matches with a hidden index requires evaluation of one hash function. The overall cost during $n$ protocol executions is $2T^n_{hash} + T^n_{prf}$, where $T^n_{hash}$ is the time cost of hashing $n$ values into the group

$G$ and $T^n_{prf}$ is the time cost of $n$ pseudo-random functions. The time complexity in the proposed scheme is $O(n)$. Since both hash function and pseudorandom function can be implemented efficiently by heuristic algorithms, all operations of **SPEKS** can be conducted efficiently. Indeed, they are computable even by low power hand-held devices. As for the storage, the data owner is required to store a single secret seed $s$ in the client side. In addition, while $O(n \cdot \log n)$ external memory is required in [GO96], **SPEKS** requires no external memory cost for storing the remote $n$ data files in the server side.

## 5.5 Conclusion

In this chapter, an efficient **SPEKS** was constructed. It was shown that **SPEKS** was suitable for keyword search in the cloud environment. Comparing with the existing keyword search schemes such as [LYCL11, WCRL12, ÖS12], the proposed construction is much more efficient on both sides of the data owner and the cloud server. In addition, the security of the proposed scheme has been proved in the standard model.

# Chapter 6

# Public Remote Integrity Check for Private Data

With Remote Integrity Check (RIC), the integrity of remotely stored data can be (publicly) verified without the need of retrieving the original data. Hao, Zhong and Yue [HZY11] (vol. 23, no. 9, IEEE Trans. Knowl. Data Eng.) proposed a privacy preserving scheme for RIC. Their scheme preserved private information during public verification; however, they require the verifier to be in possession of all the homomorphic tags, which are the outputs of a one-way homomorphic function on input the data, to the public. This requirement increases both the storage and communication cost. In addition, these tags might leak some information of the original data, although they proved that the verifier cannot obtain any information about the stored data during an execution. To overcome these problems, a privacy-preserving RIC protocol is proposed in this chapter. The proposed protocol achieves public verifiability without disclosing any information. It is ensured that no information about the original data will be leaked. In fact, the verifier is only required to know the public key of the data owner. The experimental results indicate that the proposed scheme is efficient especially when the data size is large or the integrity check is frequent. The full proofs of security is also given under the random oracle model.

## 6.1 Introduction

Data integrity is essential for cloud applications. Remotely verifying data in the cloud is referred to as the remote integrity check (RIC), where a verifier verifies the target cloud dataset without the need of retrieving the dataset. This scenario can be extended to achieve public verifiability. With a public RIC, anyone can be the verifier. This raises a concern on data security, since the information of the stored data should not be revealed to the public. Most of previous public RIC

schemes [SW08, ABC+07, ABC+11] do not provide security so that a public verifier can learn the information of private data during an integrity check, since the cloud server might leak the data information. It is noticed that Hao, Zhong and Yue's public RIC scheme [HZY11] provides a security feature against data leakage. They adopted the approach introduced by Sebe *et al.* [SFB+08] to support data dynamics and privacy. However, their scheme requires the verifier to be in possession of all the homomorphic tags used for the integrity check; therefore, it increases costs for storage and communication. In addition, although they have proved that the public verifier cannot learn the target dataset, the tags themselves could leak some information about the dataset.

In this chapter, an innovative public RIC scheme *without* disclosing any information of stored data in cloud storage is proposed. The proposed scheme ensures no information leakage of both the verified data and the homomorphic tags. The authenticator proposed by Shacham and Waters [SW08] is adopted to achieve public verifiability and protection of the data privacy. The proposed scheme also eliminates the knowledge requirement of integrity tags for verification.

**Comparison.**

A comparison between the proposed RIC scheme and some selected existing schemes is made in Table 6.1. The comparison is mainly based on the following requirements and properties:

- *Public verifiability.* Because data integrity check is a frequent task, a client may want to outsource this task to another party. With public verifiability, anyone can be a verifier.

- *Privacy-preserving.* During the remote integrity check, a public verifier should not obtain access to the client's data files. However, the homomorphic tags may disclose some information about the client's data files.

- *Without Data Disclosing.* This property is defined as the extension of privacy-preserving requirement. The third party verifier should not require any data information during the protocol execution. Additionally, the homomorphic tags should not disclose any data information to the public verifier.

- *Sampling.* As access the entire data file can be expensive in I/O costs, a flexible RIC should allow the server to access only small portions of the file

Table 6.1: Comparisons between some previous protocols and the proposed scheme. Here $n$ is the total block number, $c$ is the sampling block number, $l$ is the length of each block. $Communication^{(1)}$ and $Communication^{(2)}$ indicate the communication cost on verifier side and client side respectively. To achieve 80-bit security, the schemes [ABC$^+$07, ABC$^+$11, HZY11] use RSA cryptography where $l = 1024$ bit, while the schemes [SW08, WWL$^+$09, WWRL10] and the proposed scheme use elliptic curve cryptography with $|p| = 160bit$ where the block length $l = 160$ bit.

| | [ABC$^+$07],[ABC$^+$11] | [SW08] | [WWL$^+$09] | [WWRL10],[WCW$^+$09] | [HZY11] | Proposed Scheme |
|---|---|---|---|---|---|---|
| Public verifiability | ✓ | ✓ | × | ✓ | ✓ | ✓ |
| Privacy-preserving | × | × | × | ✓ | ✓ | ✓ |
| Without disclosing data | × | × | × | × | × | ✓ |
| Sampling | ✓ | ✓ | ✓ | ✓ | × | ✓ |
| Format independence | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Size of tags | $O(n \cdot l)$ | $O(n \cdot l)$ | $O(n \cdot l)$ | $O(n \cdot l)$ | $O(n \cdot l)$ | $O(n \cdot l)$ |
| Storage | $O(l)$ | $O(l)$ | $O(l)$ | $O(l)$ | $O(nl)$ | $O(l)$ |
| $Communication^{(1)}$ | $O(c \cdot l)$ | $O(c \cdot l)$ | $O(c \cdot l \cdot \log n)$ | $O(c \cdot l)$ | $O(l)$ | $O(c \cdot l)$ |
| $Communication^{(2)}$ | $O(l)$ | $O(l)$ | $O(l)$ | $O(l)$ | $O(n \cdot l)$ | $O(l)$ |

during the verification. This property can be also referred as 'sub-file access' or 'sub-linear authentication'.

- *Format independence.* Data format independence is a relevant feature in practical deployments. Encryption is an orthogonal issue and the outsourced data file may consist of encrypted data chunks.

- *Storage cost and Size of tags.* An efficient RIC protocol should have a minimal storage overhead to minimize costs, on both cloud server and verifier side. To reduce the storage cost, the size of tags should be small.

- *Communication cost.* Transmitting large amounts of data across the network can consume heavy bandwidths. An efficient RIC protocol should also introduce low communication cost for protocol execution.

**Main Contributions.** The main contributions of this chapter can be summarized as follows: 1) A remote integrity check protocol is proposed for outsourced data in cloud computing, which achieves public verifiability without disclosing any data information. Compared with the protocol proposed by Hao *et al.* [HZY11], the proposed scheme solves the issue that the verifier must be in possession of the tags that would leak information about the verified data. 2) Full proofs of security against arbitrary adversaries is given in the random oracle model. 3) The experimental results have shown that the proposed scheme is efficient especially when the size of the data file is large or the integrity check is frequent.

**Related Work.** Designing secure and efficient data verification protocols has attracted a lot of attention [OLR12]: works have varied from designing secure local memory and secure storage at remote servers to securing cloud storage applications. The needs in each field being specific, result in diverse kinds of such protocols.

Early verification schemes [BEG+94, BGG95, NR05] concentrated on the problem of data integrity on a local untrusted memory, i.e., memory checking. The challenging problem of data integrity verification without explicit knowledge of the full file was first proposed in broad generality by Blum *et al.* [BEG+94], who explored the task of checking the correctness of a memory-management program efficiently. Naor and Rothblum [NR05] explored the problem of dynamic memory-checking in a range of settings. Clarke *et al.* [CSG+05] focused on a trusted party which stored a small amount of state information, verifying the integrity of arbitrary blocks of

external memory. These early verification schemes are the first to suggest checking data integrity, however they are not applicable for remote data integrity checks because they require the data to be transmitted in its entirety to the verifier.

The latest verification schemes concentrated on the problems of securing data integrity at remote servers and securing clouds storage applications. These schemes can be classified into *'Proof of retrievability'* (POR) schemes (e.g., [JJ07, BJO09, DVW09, SW08]) and *'Provable data possession'* (PDP) schemes (e.g., [ABC$^+$07, ABC$^+$11, EKPT09]). A POR scheme is a challenge-response protocol. In POR schemes, a cloud provider demonstrates the file retrievability (i.e., recoverability without any loss or corruption) to a client. PDP schemes are similar protocols which only detect a large amount of corruption in outsourced data. For different application requirements, the latest verification schemes can also be classified into *static* schemes (e.g., [JJ07], [ABC$^+$07], [SW08]) and *dynamic* schemes (e.g., [EKPT09, SvDOJ11, CC12]). The static schemes consider static data, in which the client cannot modify the original data. In these schemes, the client can perform a limited set of updates only. The dynamic schemes support the full range of dynamic operations on the outsourced data, including modification, insertion and deletion. Most existing PDP and POR schemes are designed for static data which have infrequent modifications. In Erway *et al.* [EKPT09], dynamic PDP protocols were proposed. However, designing efficient dynamic POR schemes is still an open problem. Several papers [WWL$^+$09], [ZX11] claimed to construct dynamic POR protocols, but in fact only provided dynamic PDP schemes. Recently, Stefanov *et al.* [SvDOJ11] proposed Iris, a system that supported dynamic POR, including protection against small data corruption. However, Iris brings additional cost to the client because the parity data is required to be stored.

The main techniques for achieving secure and efficient remote data integrity check can be summarized as follows:

(1) *Use of a simple third-party verifier.* As the simple hash method requires the client to store numbers of hash values to do a remote integrity check, Shah *et al.* [SBMS07] introduced a third-party verifier who could delegate the periodic task of checking data integrity, in order to reduce the client's storage, communicational and computational cost. This simple solution, however, requires a third-party verifier to keep a lot of hash values of the data blocks.

(2) *Use of block ciphers.* Tweakable block ciphers were used in Oprea *et al.* [OR05] which allowed a client to detect the modification of data blocks by a remote

and untrusted server. Their protocol did not bring additional storage cost to the server and the client, but the entire file had to be retrieved during the verification executions and the communication complexity is linear in the file size.

(3) *Use of hash functions.* Deswarte *et al.* [DQS04] and Filho *et al.* [FB06] proposed RSA-based hash functions to verify remote data integrity. They allowed a verifier to perform multiple challenges by using the same metadata. The limitation of their algorithms lies in the computational complexity at the server, which must exponentiate the entire data file, that is accessing all the data blocks. In addition, RSA over the entire file is extremely slow. As is shown in Filho *et at.* [FB06], it requires 20 seconds per megabyte for 1024-bit keys on a GHz3.0 CPU. Yamamoto *et al.* [YOA07] presented an efficient scheme for large data integrity checks, based on homomorphic hash functions. The advantage of their scheme is batch processing [CY07] for a homomorphic hash function. Similar techniques can also be found in Sebe *et al.* [SBD$^+$04]. In [SBD$^+$04], they presented a protocol based on the Diffie-Hellman problem in $\mathbb{Z}_N$, however in their scheme, the client has to store $N$ bits per data block ($N$ is the RSA modulus) and the total storage cost on the client side is $O(n)$.

(4) *Use of erasure-coded data.* Schwarz and Miller [SM06] proposed a scheme that used $m/n$ erasure-correcting coding to safeguard the stored data. They used algebraic signatures, which are hash functions with algebraic properties, to do the verification. The algebraic properties ensured that the signature of the $k$ parity containers equaled the parity of the signatures of the $m$ data blocks. However, in their scheme, the file access, computation and communication complexity are all linear in the number of data blocks per challenge. Moreover, this scheme receives a less formal security analysis. Kotla *et al.* [KAD07] used a hierarchical erasure coding in which both the client and server computed forward error correcting codes. However, the server reveals the parameters of the encoding it uses to make storage safe from failures.

(5) *Use of "sentinels".* Juels *et al.* [JJ07] proposed a scheme for proof of retrievability using 'sentinels'. The sentinels (special blocks) were hidden among other blocks in file $F$. The verifier challenged the prover by specifying the positions of a collection of sentinels and asking the prover to return the associated sentinel values. Their scheme is limited, as they can handle only a limited number of queries and increase storage overhead on the server side. In addition, the client needs to store all the sentinels. Furthermore, it requires that the original $x$ can be recovered from

multiple challenges and responses. This is the main difference with the proposed scheme. In Dodis *et al.* [DVW09], they improved the POR constructions and built nearly optimal POR codes using hitting samplers and error-correcting codes.

(6) *Use of homomorphic verifiable tags.* Ateniese *et al.* [ABC$^+$07, ABC$^+$11] described a proof of data possession (PDP) scheme that improved the response length of the simple hash scheme using homomorphic verifiable tags. In their scheme, they constructed the homomorphic verifiable tags $T_i$ for each data block $m_i$. Later, the prover sent a linear combination of blocks $\sum_i a_i m_i$ (with arbitrary coefficients $\{a_i\}$) to the verifier. The verifier cloud be convinced if $\sum_i a_i m_i$ was correctly generated using an aggregate tag $T$ computed from $\{T_i\}$. They also proposed a variant of their PDP scheme to achieve public verifiability under a weaker security model. Erway *et al.* [EKPT09] introduced a framework and efficient constructions for dynamic provable data possession which extends Ateniese *et al.*'s model [ABC$^+$07] to support provable updates. Their constructions captured the dynamic operations such as insertion in the middle of a file, however they are not efficient when moving and deleting the entire files. Shacham and Waters [SW08] proposed two HTAG schemes which used a simple homomorphic MAC and a universal hash family to reduce the communication bits to a constant factor of $k$. One of their schemes supported private verifiability and the other supports public verifiability. In Xu and Chang [XC12], they improved the private verifiability scheme of Shacham and Waters'. The size of a response (or proof) was dominated by $s$ group elements where each was $\lambda$ bits long. Xu and Chang managed to aggregate these $s$ group elements into two group elements, leading to a reduction in proof size from $O(s \cdot \lambda)$ to $O(\lambda)$ bits. Some other schemes that also use homomorphic verifiable tags can be found in Chang and Xu [CX08], Wang *et al.* [WWL$^+$09], Wang *et al.* [WCW$^+$09], Wang *et al.* [WWRL10], Wang *et al.* [WWR$^+$11] and Zeng [Zen08].

**Organization of This Chapter.** The rest of this chapter is organized as follows. In Section 6.2, the framework and definitions are presented. In Section 6.3, the main construction of the remote integrity check scheme is presented. In Section 6.4, the full proofs of security is given in the random oracle model. In Section 6.5, the complexity of the proposed protocol is analyzed, in the aspects of communication, computation and storage costs. Experimental results are also given to show the performance. Section 6.6 concludes this chapter.

## 6.2 Framework

The framework for public remote integrity check in clouds is first described, as shown in Figure 6.1. The public RIC protocol checks that the cloud storage retains the outsourced data file $M$ which can be divided into chunks of data blocks $m_i$'s. Prior to outsourcing the data file, the data owner $\mathcal{O}$ pre-processes those $m_i$'s and generates several homomorphic verifiable tags $T_i$'s. Then the owner $\mathcal{O}$ outsources his file $M$ and all the tags $T_i$'s to cloud storage and deletes them in local database. The data format of $M$ is independent here and it could be an encrypted file. Later, the owner $\mathcal{O}$ outsources the periodic task of checking data integrity to some third party verifier $\mathcal{V}$. The third party verifier sends a challenge to the cloud server $\mathcal{S}$ and asks $\mathcal{S}$ to respond based on the stored data blocks as well as the tags. Upon successful completion of the protocol, $\mathcal{V}$ is convinced that the data file $M$ has not been altered or deleted by $\mathcal{S}$. To conduct the protocol, $\mathcal{V}$ is only required to know the public key of the data owner. In addition, no information about the data blocks nor the tags are revealed to $\mathcal{V}$. This framework is designed for the purpose of the data owner $\mathcal{O}$ who wants to outsource the periodic task of data integrity checking, but does not want to leak his/her data at the same time.

**The Main Difference with Simple TPV Solutions.** The main difference between the proposed RIC scheme and those remote integrity check schemes which use third-party verifier (TPV), is that the proposed solution has constant storage and communication cost by contrast of the multiple MACs that stored in the verifier's side. If one is to allow the TPV to store all the message authentication code for all the data blocks, then the simple solution that introduce a third-party verifier who can delegate the periodic task of checking data integrity, can reduce the data owner's storage, communicational and computational cost. However, this simple solution require the TPV to store numbers of MACs which is linear to the number of data chunks outsourced, and a large amount of bandwidth which is linear to the number of queried blocks in the verification interaction.

**Threat Model.** The cloud server $\mathcal{S}$ must answer the challenges from the third party verifier $\mathcal{V}$. Failing to do so represents a data loss. However the cloud server is not trusted, that is to say, the cloud server may try to convince the verifier that it possesses the file even though the file is totally or partially missing. The cloud server may have several motivations for misbehavior. For example, the server may

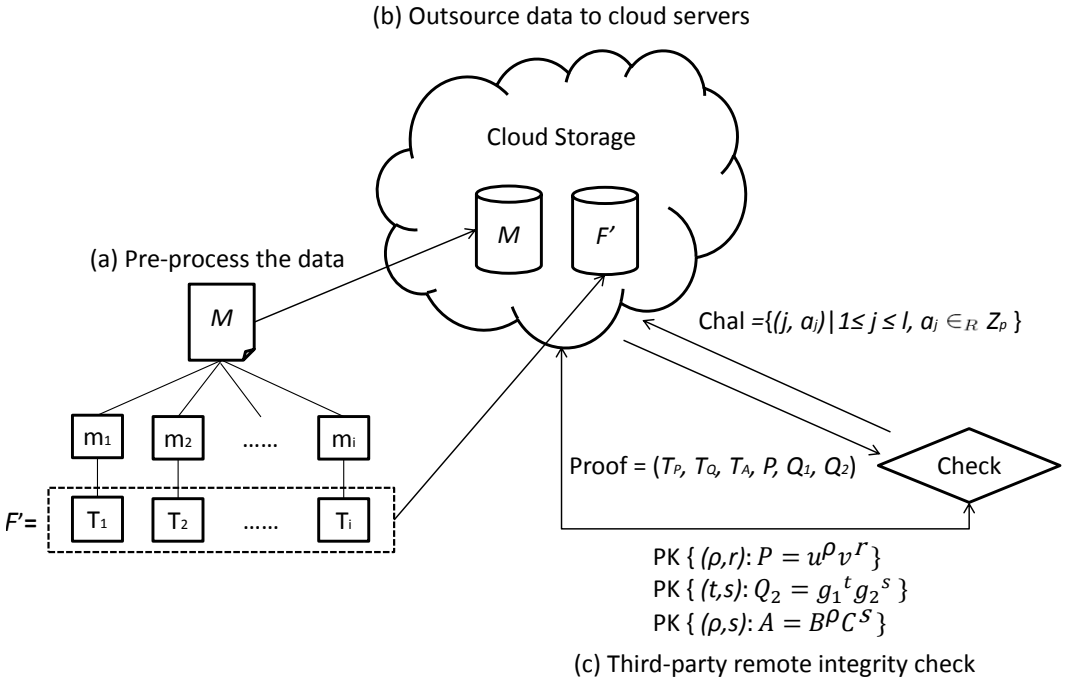


Figure 6.1: The framework of public RIC in cloud storage.

want to reclaim storage by discarding data that has not been or is rarely accessed, or hide some data loss incidents due to management errors and hardware failures. The third party verifier $\mathcal{V}$ must verify the integrity of the outsourced data. The third-party verifier may not be fully-trusted as well. That is, the verifier may try to retrieve the owner's data through the process of verification. A public remote integrity check protocol has to detect the cloud server's misbehavior when the server has deleted a fraction of the file. Moreover, it has to ensure that the third party verifier cannot learn any outsourced data through executions.

### 6.2.1 Formal Definitions

The syntax of the proposed public remote integrity check scheme is given as follows. In Section 6.4, the security definition will be given formally.

**Definition 6.1** *(Public Remote Integrity Check Scheme) A public RIC scheme consists of two polynomial-time algorithms (KeyGen, TagGen) and an interactive protocol (Proof) that captures the properties including data possession and public verification without disclosing data.*

**KeyGen.** This algorithm is run by the data owner. It takes a security parameter $k$ as input and outputs a pair of matching public and secret keys $(pk, sk)$.

**TagGen.** This algorithm is run by the data owner to generate the homomorphic tags for the data chunks. It takes as inputs a public key $pk$, a secret key$sk$ and a data block $m_i$, outputs the homomorphic tag $T_i$ for this data block $m_i$.

**Proof.** This is a pair of interactive algorithms $(\mathbf{Proof}_S, \mathbf{Proof}_V)$ executed by the cloud server and a public verifier respectively. The input of $\mathbf{Proof}_S$ is a public key $pk$, a set of data blocks and homomorphic tags $\{m_i, T_i\}$. The input of $\mathbf{Proof}_V$ is a public key $pk$. Upon completion of the protocol, $\mathbf{Proof}_V$ outputs 1 if the proof is accepted, and 0 otherwise.

**Definition 6.2** *(Correctness) A public remote integrity check scheme is correct if for all key pairs $(pk, sk)$ output by KeyGen, for all the tags $T_i$ output by TagGen with input $m_i, sk$, $\boldsymbol{Proof}_V$ with input $pk$ outputs 1 if it is interacting with $\boldsymbol{Proof}_S$ with input $pk, \{T_i, m_i\}$.*

## 6.3 The Proposed Scheme

Firstly, the main construction of the proposed scheme is described. The construction is based on the scheme from [SW08]. Zero-knowledge proof techniques are incorporated into their scheme so that the verifier does not learn anything about the data during the proof protocol. Secondly, how a public RIC system can be constructed is shown.

### 6.3.1 Main Construction

Let $e : G \times G \rightarrow G_T$ be a computable bilinear map with $|G| = |G_T| = p$ for some large prime $p$. Let $g$, $g_1$, $g_2$, $u$ and $v$ be generators of $G$. The scheme also employs a hash function $H : \{0,1\}^* \rightarrow G$, viewed as a random oracle. Details of

the construction of a hash function whose range is the group $G$ has been described in [BLS01]. The main construction of the proposed scheme is as follows:

**KeyGen.** Select a random $x \in \mathcal{Z}_p$, this algorithm generates the public key as $(g^x, g, g_1, g_2, u, v)$ . The secret key is $x$.

**TagGen**$((sk, m_i) \rightarrow T_i)$**.** Before generating the tags, the data file $M$ is pre-processed into $n$ data blocks $m_1, m_2, ..., m_n \in \mathcal{Z}_p$. In the following each data block will be referred as $m_i$ for $1 \leqslant i \leqslant n$. This algorithm then generates a homomorphic tag $T_i$ for data block $m_i$, where $T_i = (H(i) \cdot u^{m_i})^x$.

**Proof.** The pair of algorithms $\mathbf{Proof}_S, \mathbf{Proof}_V$ interacts as follows.

- $\mathbf{Proof}_V$ chooses a set of data blocks for challenge. Let $\mathcal{I} \subset \{1, \ldots, n\}$ be the index set for the blocks to be challenged. $\mathbf{Proof}_V$ randomly chooses $c, d \in_R \mathcal{Z}_p$, computes $D = g_1^c g_2^d$. $\mathbf{Proof}_V$ randomly picks $a_j \in_R \mathbb{Z}_p$ for $j \in \mathcal{I}$ and sends $\mathcal{I}, \{a_j\}_{j \in \mathcal{I}}, D$ to $\mathbf{Proof}_S$ as the challenge.

- $\mathbf{Proof}_S$ computes $T = \prod_{j \in \mathcal{I}} T_j^{a_j}$, $\rho = \sum_{j \in \mathcal{I}} a_j m_j$, $B = e(u, g^x)$ and $C = e(g_1, g)$. $\mathbf{Proof}_S$ then randomly chooses $r, s, t, t_\rho, t_r, t_t, t_s \in_R \mathbb{Z}_p$ and computes

$$T_P = u^{t_\rho} \cdot v^{t_r},\ T_Q = g_1^{t_t} \cdot g_2^{t_s},\ T_A = B^{t_\rho} \cdot C^{t_s},$$
$$P = u^{\rho} \cdot v^{r},\ Q_1 = T \cdot g_1^{s},\ Q_2 = g_1^{t} \cdot g_2^{s}.$$

  $\mathbf{Proof}_S$ sends $(T_P, T_Q, T_A, P, Q_1, Q_2)$ to $\mathbf{Proof}_V$.

- $\mathbf{Proof}_V$ sends $(c, d)$ to $\mathbf{Proof}_S$.

- $\mathbf{Proof}_S$ checks if $D = g_1^c g_2^d$ and computes

$$z_\rho = t_\rho - c \cdot \rho, \quad z_r = t_r - c \cdot r,$$
$$z_t = t_t - c \cdot t, \quad z_s = t_s - c \cdot s$$

  and sends $(z_\rho, z_r, z_t, z_s)$ to $\mathbf{Proof}_V$.

- $PV$ outputs 1 if and only if all the following equations hold

$$T_P \stackrel{?}{=} P^c \cdot u^{z_\rho} \cdot v^{z_r} \tag{6.1a}$$

$$T_Q \stackrel{?}{=} Q_2^c \cdot g_1^{z_t} \cdot g_2^{z_s} \tag{6.1b}$$

$$T_A \stackrel{?}{=} \left[ \frac{e(Q_1, g)}{e(\prod_{j=1}^{l} H(j)^{a_j}, g^x)} \right]^c \cdot B^{z_\rho} \cdot C^{z_s}, j \in \mathcal{I} \tag{6.1c}$$

  and 0 otherwise.

### 6.3.2 A Brief Explanation of The Proposed Protocol

Note that a tuple $(T, \rho)$ satisfying the equation

$$e(T, g) = e(\prod_{j \in \mathcal{I}} (H(j)^{a_j}) u^{\rho}, v)$$

is a proof-of-retrievability of the messages $\{m_j\}_{j \in \mathcal{I}}$ according to [SW08]. Instead of requiring the cloud server to send $(T, \rho)$ to the verifier (which leaks information about the data), the protocol $\mathbf{Proof}_S$ is modified so that it is a zero-knowledge proof-of-knowledge of the tuple $(T, \rho)$ satisfying the above equation. Indeed, the message flow $\wp = \big(D, (T_P, T_Q, T_A, P, Q_1, Q_2), (c, d), (z_\rho, z_r, z_t, z_s)\big)$ can be viewed as a zero-knowledge proof-of-knowledge of such a tuple. Intuitively, the proposed protocol is secure if [SW08] is a secure proof-of-retrievability and that $\wp$ is a zero-knowledge proof-of-knowledge protocol with soundness.

### 6.3.3 A Public RIC System

Now a public RIC system can be constructed as Figure 6.2, from the above scheme in three phases: *Setup*, *Challenge* and *Proof*. It should be emphasized that the cloud server does not send back to the verifier any of the data file blocks and not even their sum. Additionally, the third-party verifier checks the integrity of the outsourced file without having the secret key. In particular, the third-party verifier does not retrieve any information about the data even when the challenge only consists of a single block.

## 6.4 Security Analysis

Two security requirements are identified for the proposed scheme. The first one regards ***data possession***. Specifically, an adversary cannot successfully construct a valid proof without possessing all the blocks corresponding to a given challenge, unless it guesses all the missing blocks. The second one is to capture ***public verification without disclosing data***. No third party verifier should be able to retrieve any data or tag information in probabilistic polynomial time. To formalize this notion, it is required that for any verifier, a PPT simulator can be constructed to simulate the view of the verifier without the knowledge of the messages and homomorphic tags.

- **Setup:** The data owner $\mathcal{O}$ invokes the algorithm $KeyGen$ to generate a pair of matching public and secret keys $(pk, sk)$. $\mathcal{O}$ then invokes $TagGen$ for all data blocks $(1 \leqslant i \leqslant n)$ and sends all the homomorphic tags $T_i$'s to the cloud server $\mathcal{S}$, along with $pk$ and $m_i$'s. $\mathcal{O}$ may now delete the original data file $M$ and all the tags $T_i$'s from his local storage.

- **Challenge:** The third-party verifier $\mathcal{V}$ requests remote integrity check for $l$ distinct blocks of the data file $M$ (with $1 \leqslant l \leqslant n$):

  1. $\mathcal{V}$ generates a challenge $chal = \{(j, a_j) | 1 \leqslant j \leqslant l, a_j \in_R \mathbb{Z}_p\}$ and sends $chal$ to $S$.
  2. $\mathcal{S}$ computes $(T_P, T_Q, T_A, P, Q_1, Q_2)$ as follows: $T_P = u^{t_\rho} \cdot v^{t_r}$, $T_Q = g_1^{t_t} \cdot g_2^{t_s}$, $T_A = B^{t_\rho} \cdot C^{t_s}$, $P = u^{\rho} \cdot v^{r}$, $Q_1 = T \cdot g_1^{s}$, $Q_2 = g_1^{t} \cdot g_2^{s}$ where $T = \prod_{j \in \mathcal{I}} T_j^{a_j}$, $\rho = \sum_{j \in \mathcal{I}} a_j m_j$, $B = e(u, g^x)$ and $C = e(g_1, g)$. $(T_P, T_Q, T_A, P, Q_1, Q_2)$ is returned to $\mathcal{V}$.
  3. $\mathcal{V}$ generates a random challenge $c$.
  4. $\mathcal{S}$ computes $z_\rho = t_\rho - c \cdot \rho$, $z_r = t_r - c \cdot r$, $z_t = t_t - c \cdot t$, $z_s = t_s - c \cdot s$, and returns $(z_\rho, z_r, z_t, z_s)$ to $\mathcal{V}$.

- **Proof:** $\mathcal{V}$ checks the integrity of the outsourced data without either having the data or the secret key $sk$:

  1. $\mathcal{V}$ computes $A = \frac{e(Q_1, g)}{e(\prod_{j=1}^{l} H(j)^{a_j}, g^x)}$.
  2. $\mathcal{V}$ checks the validity of the proof by verifying whether the three equations 6.1$a$, 6.1$b$ and 6.1$c$ hold.

Figure 6.2: A Public RIC System

### 6.4.1 Data Possession

This requirement is formalized by using a security game played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. A public RIC protocol is sound if any cloud server that convinces an honest verifier is in possession of the data file $M$. Following the definition in [HZY11], it means that there exists an extractor algorithm that can extract the data file $M$ when it is interacting with the cloud server in the public RIC protocol.

**Data Possession:**

In the DP-game, the challenger $\mathcal{C}$ denotes the third-party verifier and the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ denotes a cheating cloud server. The game is divided into two phases, the learning phase and the proof phase.

- **Setup:** The challenger $\mathcal{C}$ runs $KeyGen$ to generate a keypair $(pk, sk)$, and sends $pk$ to the adversary $\mathcal{A}$.
- **Query:** The adversary $\mathcal{A}$ can make at most $q_s$ queries to a tag oracle, providing some file block $m_i$ during each query. $\mathcal{C}$ then computes all the homomorphic tags $T_i$ and sends them back to $\mathcal{A}$, along with all the $m_i$'s. The adversary stores all the $m_i$'s and the corresponding tags $T_i$'s.
- **Execute:** The adversary $\mathcal{A}$ can undertake at most $q_e$ executions of public RIC protocol for any file $M$ that he has made a tag query, with specifying the corresponding $T_i$'s.
- **Proof:** The adversary $\mathcal{A}$ outputs the description of a prover machine $P$ and the set of messages $\mathcal{M} \subset \{1, \ldots, q_s\}$ to be challenged. $\mathcal{C}$ interacts with $P$ in the proof protocol with $\mathcal{M}$ being the set of messages to be challenged.

Informally speaking, the data possession property requires that if $P$ can pass through the verification protocol with $\mathcal{C}$, $P$ should be in possession of all messages $m_i$ for $i \in \mathcal{M}$. This is formalized as follows.

**Definition 6.3** *(Soundness) A public remote integrity check scheme is sound if there exists a knowledge extractor $\mathcal{K}$ that can recover $\{m_i | i \in \mathcal{M}\}$ from the prover machine $P$ with non-negligible probability.*

**Theorem 6.1** *The construction of a public RIC system captures data possession if Waters public-verification scheme is secure in the random oracle model.*

In the following it shows how to construct the extractor $\mathcal{K}$. Note that $\mathcal{K}$ will only be used in proofs. It is an algorithm that runs in probabilistic polynomial time on input $P$, $pk$ and the index set $\mathcal{M}$. The output of $\mathcal{K}$ is the set of messages $\{m_i | i \in \mathcal{M}\}$

The knowledge extractor $\mathcal{K}$ will work in the following way: If the prover $P$ responds correctly to an initial challenge $c$, then $\mathcal{K}$ will rewind $P$ and issue a different challenge $c'$ for which $P$ also responds correctly. Note that this requires $\mathcal{K}$ to have the discrete logarithm of $g_2$ to base $g_1$ so that it can issue two different challenges for the same value $D$. If $\mathcal{K}$ can find two such challenges $c$ and $c'$, then $\mathcal{K}$ has the following equations:

$$\begin{aligned}
T_P &= P^c \cdot u^{z_\rho} \cdot v^{z_r} = P^{c'} \cdot u^{z_\rho{}'} \cdot v^{z_r{}'}, \\
T_Q &= Q_2^c \cdot g_1^{z_t} \cdot g_2^{z_s} = Q_2^{c'} \cdot g_1^{z_t{}'} \cdot g_2^{z_s{}'}, \\
T_A &= A^c \cdot B^{z_\rho} \cdot C^{z_s} = A^{c'} \cdot B^{z_\rho{}'} \cdot C^{z_s{}'}.
\end{aligned}$$

Now it can get

$$\begin{aligned}
\frac{P^c}{P^{c'}} &= \frac{u^{z_\rho} \cdot v^{z_r}}{u^{z_\rho{}'} \cdot v^{z_r{}'}}, \\
\frac{Q_2^c}{Q_2^{c'}} &= \frac{g_1^{z_t} \cdot g_2^{z_s}}{g_1^{z_t{}'} \cdot g_2^{z_s{}'}}, \\
\frac{A^c}{A^{c'}} &= \frac{B^{z_\rho} \cdot C^{z_s}}{B^{z_\rho{}'} \cdot C^{z_s{}'}}.
\end{aligned}$$

If denotes $\triangle c = c' - c$, $\triangle z_\rho = z_\rho{}' - z_\rho$, $\triangle z_r = z_r{}' - z_r$, $\triangle z_t = z_t{}' - z_t$ and $\triangle z_s = z_s{}' - z_s$, from the above equations it can have

$$\begin{aligned}
P &= u^{-\frac{\triangle z_\rho}{\triangle c}} v^{-\frac{\triangle z_r}{\triangle c}}, \\
Q_2 &= g_1^{-\frac{\triangle z_t}{\triangle c}} g_2^{-\frac{\triangle z_s}{\triangle c}}, \\
A &= B^{-\frac{\triangle z_\rho}{\triangle c}} C^{-\frac{\triangle z_s}{\triangle c}}.
\end{aligned}$$

Then, it has

$$\frac{e(Q_1, g)}{e(\prod_{j=1}^{l} H(j)^{a_j}, g^x)} = e(u^{-\frac{\triangle z_\rho}{\triangle c}}, g^x) \cdot e(g_1, g)^{-\frac{\triangle z_s}{\triangle c}}.$$

That is,

$$e(Q_1 \cdot g_1^{\frac{\triangle z_s}{\triangle c}}, g) = e(\prod_{j=1}^{l} H(j)^{a_j}, g^x) \cdot e(u^{-\frac{\triangle z_\rho}{\triangle c}}, g^x).$$

Thus $\mathcal{K}$ can compute $T = Q_1 \cdot g_1^{\frac{\triangle z_s}{\triangle c}}$ and $\rho = -\triangle z_\rho / \triangle c$. In [SW08], it has been proven under the CDH assumption that $\rho$ must be correctly formed. That is, note that $\rho$ is a linear equation of the form $a_1 m_1 + a_2 m_2 + \cdots + a_l m_l$. Then by choosing independent coefficients $a_1, a_2, \cdots, a_l$ in $l$ executions of the extraction above on the same blocks $m_1, m_2, \cdots, m_l$, $\mathcal{K}$ can obtain $l$ independent linear equations in the variables $m_1, m_2, \cdots m_l$. Thus $\mathcal{K}$ can solve these equations to obtain the data file $M$ that contain blocks $m_1, m_2, \cdots, m_l$.

The extraction is required to be succeed (with all but negligible probability) from the prover (adversary) that causes the verifier (challenger) to accept with a non-negligible probability $\epsilon$. Then the extractor $\mathcal{K}$ can recover enough blocks to reconstruct the original $M$ from such an adversary will take $O(n/\epsilon)$ interactions.

### 6.4.2 Public Verification without Disclosing Data

To capture the requirement of public verification without disclosing data, it is required that there exists a PPT simulator $S$ which is capable of simulating the view of any verifier given that the public parameters and the public key of the user as input. In other words, the RIC protocol is zero-knowledge.

**Theorem 6.2** *The proposed public remote integrity check protocol is a RIC protocol is zero-knowledge.*

*Proof:* In order to prove that the public RIC protocol is a zero-knowledge (ZK) proof system, one needs to prove three properties: completeness, soundness and zero-knowledge.

**Completeness.** Suppose the honest prover posses the secret $M$ just as required in *GenProof*. It is straightforward that the prover can always convince the verifier to accept the proof. Thus the perfect completeness condition follows.

**Soundness.** To show soundness (namely that even an arbitrarily malicious prover $P^*$ can not convince the verifier $V$ to accept a false statement with more than a negligible probability), how to extract a witness $M$ from two different runs of the proposed protocol is first demonstrated.

Assume there are two different transcripts of the proposed protocol, the extractor $K$ can be constructed in the following way that as described in section 6.4.1.

Having $O(n/\epsilon)$ interactions, the extractor $K$ can recover enough blocks to reconstruct the original $M$ from a malicious prover $P^*$. In such a case, $T = Q_1 \cdot g^{\frac{\triangle z_s}{\triangle c}}$ and $\rho = -\triangle z_\rho / \triangle c$ can be efficiently computed from the transcripts. Thus it can be concluded that for any first round message $(T_P, T_Q, T_A, P, Q_1, Q_2)$ that $P^*$ may send to $V$, there is at most one possible challenge $c$ which $P^*$ could answer correctly. Then $P^*$ can effectively compute a witness as above. However $P^*$ must decided upon its first round message before seeing the challenge and therefore will receive the "correct" challenge $c$ with probability at most $\frac{1}{p}$ which is negligible.

**Honest Verifier Zero-knowledge.** To show that the proposed protocol is zero-knowledge, it is necessary to describe an efficient simulator $S$ that interacts with any verifier, and produces an interaction that is indistinguishable from the verifier's interaction with the prover.

- The verifier sends the values $\mathcal{I}, \{a_i\}, D$ to $S$.
- $S$ randomly picks and sends $T'_P$, $T'_Q$, $T'_A$, $P'$, $Q'_1$, $Q'_2 \in_R G$ to the verifier.
- The verifier returns $(c, d)$ to $S$.
- $S$ validates $D = g_1^c g_2^d$ and rewinds the verifier to the point when it receives the tuples $(\mathcal{I}, \{a_i\}, D)$.
- $S$ randomly picks $z_\rho, z_r, z_t, z_s \in_R \mathbb{Z}_p$, $P, Q_1, Q_2 \in_R G$ and computes the following values:

$$
\begin{aligned}
T_P &= P^c u^{z_\rho} v^{z_r} \\
T_Q &= Q_2^c \cdot g_1^{z_t} \cdot g_2^{z_s} \\
T_A &= \left[ \frac{e(Q_1, g)}{e(\prod_{j=1}^{l} H(j)^{a_j}, g^x)} \right]^c \cdot B^{z_\rho} \cdot C^{z_s}
\end{aligned}
$$

- $S$ sends $T_P, T_Q, T_A$ to the verifier.
- The verifier returns $(c, d)$ to $S$. Under the discrete logarithm assumption, the values $(c, d)$ will be the same with the first run.
- $S$ returns $z_\rho, z_r, z_t, z_s$ to the verifier.

Note that the distribution of $P$, $Q_1$, $Q_2$ is identical to the real one computable by the prover since for any value of $\rho$ and $T$, there exists random numbers $r, s, t$ such that $P = u^{\rho}v^{r}$, $Q_1 = Tg^{s}$ and $Q_2 = g_1^{t}g_2^{s}$. Similarly, for any values of $\rho, t, r, s$ there exists random numbers $t_{\rho}, t_t, t_r, t_s$ such that $z_{\rho} = t_{\rho} - c\rho$, $z_t = t_t - ct$, $z_r = t_r - cr$ and $z_s = t_s - ts$. Thus, the view of the verifier interacting with the simulator $S$ is identical to that of the real prover.

## 6.5 Performance Analysis

### 6.5.1 Complexity Analysis

In this section, the complexity of the proposed scheme is analyzed, in the aspects of communication, computation and storage costs. In particular, the proposed scheme is compared with the one proposed in [HZY11].

**Communication Costs.** Unlike [HZY11], the proposed scheme does not only consider the communication costs in Proof phase, but also consider the communication cost in Setup phase as the communications on the client side only occur in Setup phase. As mentioned in Table 1, the client side communication cost is $O(1)$ in the proposed protocol, while is $O(n)$ in [HZY11] as it requires the client to releases all the tags to every possible verifier. The verifier side communication cost is $O(c)$ in the proposed protocol while is $O(1)$ in [HZY11]. As the sampling block number $c$ is much more smaller than the total block number $n$, for example, $c = 460$ or 300, the sum cost $O(1) + O(c)$ in the proposed scheme is lower than $O(1) + O(n)$ in [HZY11].

**Storage Costs.** The storage cost mainly occurs on the server side as all the data chunks and homomorphic tags have been outsourced to the cloud storage. Separate analysis of the storage cost is given on the client side, the server side and the verifier side. A detailed comparison with [HZY11] is drawn .

- **Client side.** In [HZY11], the client needs to store the public key and the private key. The storage cost is $2|N| + |p| + |q|$ bits ($|N| = 1024$ bits). In the proposed scheme, the client also needs to store the pair of matching public and secret keys $(pk, sk)$, however with the lower storage cost as $7|p|$ bits (e.g., $|p| = 160$ bits).

- **Verifier side.** The verifier needs to store the public key $pk$ in the proposed

scheme. The storage cost is only $6|p|$ bits. However in [HZY11], the verifier is required to store all the RSA-based tags. The storage cost of the tags is linear to the total number of blocks, that upper bounded by $\lceil |m|/l \rceil \cdot |N|$ bits. Here $|m|$ denotes the total length of the outsourced data and $l$ denotes the length of each data block.

- **Server side.** The cloud server needs to store all the data chunks and the tags. The storage cost on the server side is $|m| + \lceil |m|/l \rceil \cdot |p|$ bits in the proposed scheme while it is higher in [HZY11] with $|m| + \lceil |m|/l \rceil \cdot |N|$ bits.

**Computation Costs.** Separate analysis of the computation costs is also given on the client side, the server side and the verifier side.

- **Client side.** The computation cost on the client side only occurs in Setup phase, for generating the homomorphic tags. It is $O(n)$ in [HZY11]. As modulo operations are far more efficient than the modular exponentiations, only the latter is considered as [HZY11]. The computation cost of the client is $2\lceil |m|/l \rceil T_{exp}(|p|, p) + T_{hash}^{\lceil |m|/l \rceil} + T_{multi}^{\lceil |m|/l \rceil}$, where $T_{exp}(len, num)$ is the time cost for computing a modular exponentiation with a $len$-bit long exponent modular $num$, $T_{hash}^{n}$ is the time cost of hashing $n$ values into the group $G$ and $T_{mult}^{n}$ is the time cost of $n$ multiplications. The time complexity in the proposed scheme is also $O(n)$.

- **Verifier side.** During the verification in the proposed scheme, the verifier needs to check whether the three equations hold. It contains $(9 + c)$ modular exponentiations, four pairings and $(c-1+2+2+2)$ multiplications in group $G$. The computation cost in verifier side is $(9+c)T_{exp}(|p|, p)+T_{pairing}^{4}+T_{mult}^{c+5}$, where $c$ is the sampling block number and $T_{pairing}^{n}$ is the time cost of $n$ pairings. The time complexity in the proposed scheme is $O(c)$ while it is $O(2^n)$ in [HZY11].

- **Server side.** In the proposed scheme, the main computation cost on the server side occurs for generating the proof, as the server needs to compute $(11 + c)$ modular exponentiations, two pairings, $(c + 5)$ multiplications and $(c - 1)$ additions. The computation cost for generating the response is much lower, which only contains four additions and four multiplications in group $G$. Thus the sum cost for computation on the server side is $(11 + c)T_{exp}(|p|, p) + T_{pairing}^{2} + T_{mult}^{c+9} + T_{add}^{c+3}$, where $T_{add}^{n}$ is the time cost of $n$ additions in group

$G$. The time complexity in the proposed scheme is $O(c)$, while it is $O(2^n)$ in [HZY11].

### 6.5.2 Performance Analysis

The performance of the proposed scheme is now assessed. This analysis focuses on the extra computation cost that is introduced by the proposal. The experiment is conducted by using Turbo C on a Windows 7 system with an Intel Core Duo processor running at 2.53GHz and 4.00GB of RAM. All the algorithms use the Pairing-Based Cryptography (PBC) library version 0.5.12. The elliptic curve utilized in the experiment is an MNT curve that has a 160-bit group order, with base field size of 512 bits and the embedding degree $k = 2$. All experimental results present the mean of 20 trials.

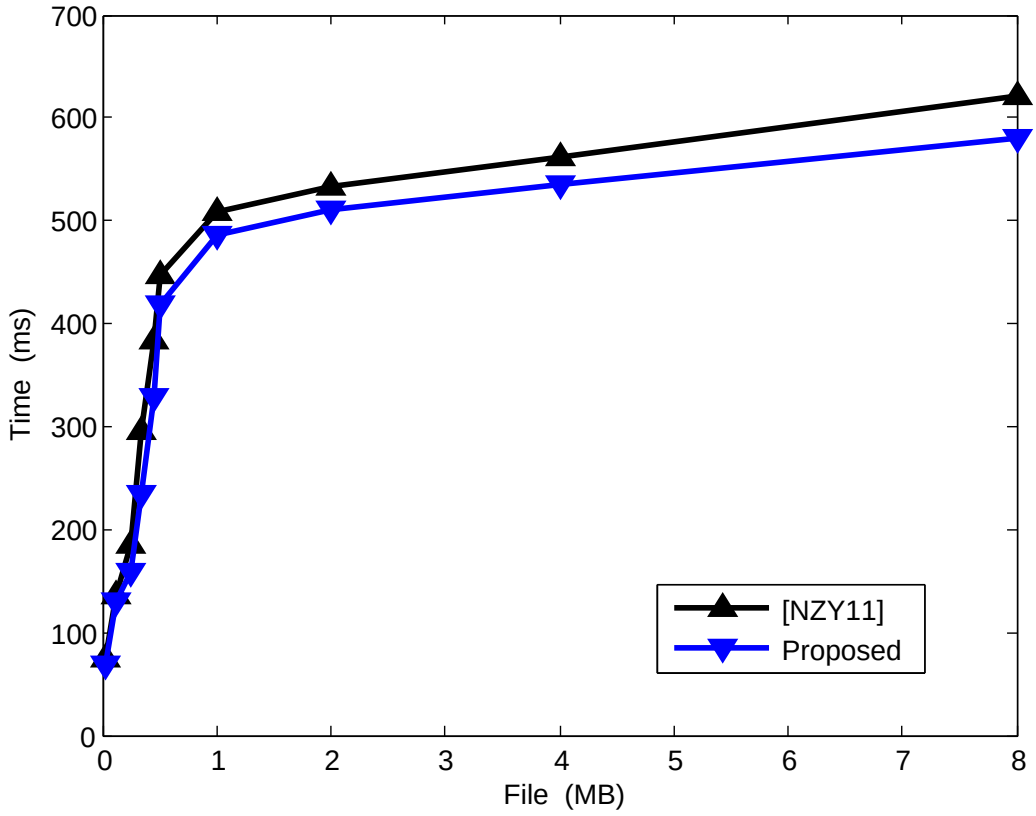


Figure 6.3: Comparison on computation time cost for verifying the proof on verifier side, between the proposed scheme and [HZY11].

The analysis begins by estimating the computation cost on client side. From

Table 7.1, it can be seen that the processing time increases linearly with the data block number $n$. Compared with the protocol [HZY11] in which is also implemented by Turbo C and Miracl library on the same platform, the process executions of the proposed scheme cost much. The main reason is that, for a same file $F$, the block number is $\lceil \frac{F}{1024} \rceil$ in [HZY11] where the length of each data block is chosen to be 1024 bits, and the block number is $\lceil \frac{F}{160} \rceil$ in the proposed scheme.

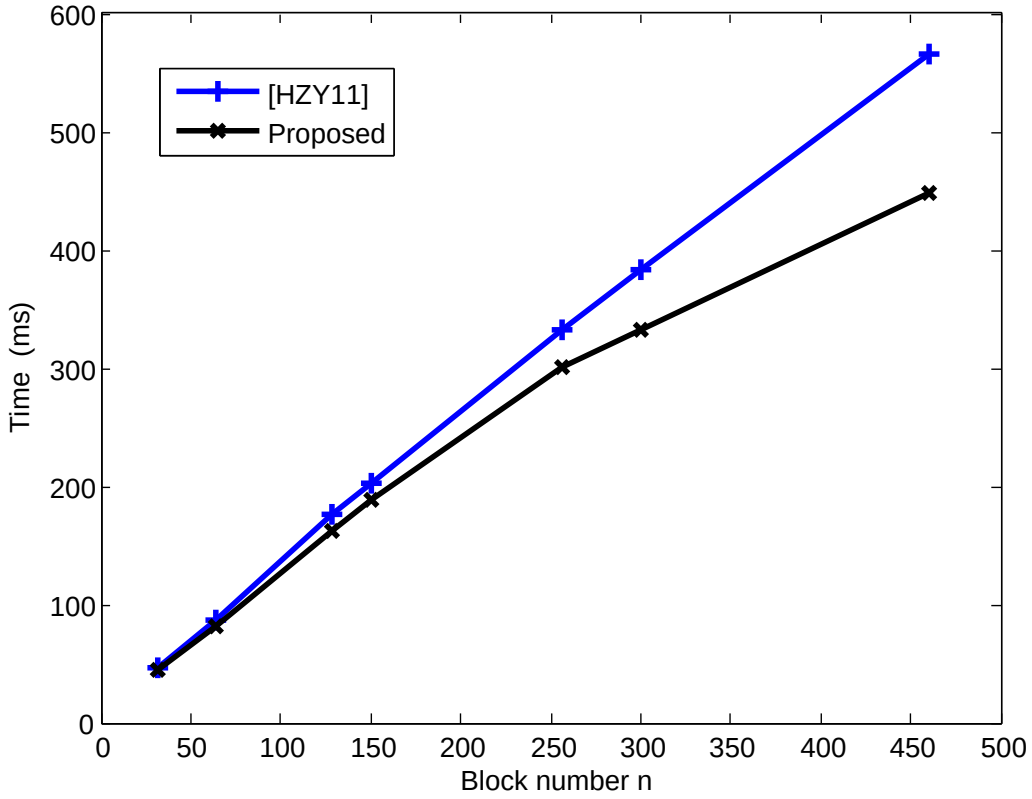


Figure 6.4: Comparison on computation time cost for generating the proof on server side, between the proposed scheme and [HZY11].

The experimental results show that the computation costs are fewer on both of verifier side and server side in the proposed scheme, comparing with the proposed one in [HZY11]. For a fair comparison, the cost of processing all data blocks is compared, though the proposed scheme supports sampling in which a much smaller number of blocks is challenged, for a slightly degraded soundness guarantee (to achieve an accuracy of 99%, a sample of 460 blocks is often sufficient). As shown in Figure 6.3 and Figure 6.4, the processing time of the proposed scheme is slightly better than [HZY11] even without sampling. If $c = 460$ blocks are sampled for 99

percent accuracy, the processing time would keep at around 69ms, regardless of the actual file size. This indicates that for data files which have large file size or frequent integrity check, the proposed scheme is highly efficient.

Table 6.2: The processing time for generating the homomorphic tags on client side. The length of each data block $|m_i|$ is set to be 160 bits.

| File Size (KB) | The Proposed Scheme | |
|---|---|---|
| | sampling block number | processing time (ms) |
| 1.25 | 64 | 725 |
| 2.5 | l28 | 1443 |
| 5 | 256 | 2798 |
| 10 | 512 | 5518 |

## 6.6 Conclusion

In this chapter, a public RIC scheme is presented for privacy-preserving remote integrity check in cloud storage. The proposed scheme achieves data possession and protection against malicious third party verifiers. It ensures no leakage of both the verified data and the tags during the protocol executions. The security of the proposed scheme has been proven in the random oracle model. The performance analysis and experimental results show that the proposed scheme is efficient, especially when the size of the data file is large or the integrity check is frequent.

# Chapter 7

# PMOW: Proof of Multiparty Ownership for Encrypted Data in Clouds

The notion of proof of ownership (POW) in clouds was introduced by Halevi *et al.* [HHPSP11]. In their scheme, a user could efficiently prove that he holds the file, rather than some short information. However, their scheme could not support either multiple users application or encrypted file scenario. It is observed that there are applications where different users share the ownership of the outsourced file. Thus in this chapter, an innovative **PMOW** scheme is proposed for proof of multiparty ownership with the encrypted data in cloud. Every user can prove he holds the plaintext of the encrypted file when the server stores one ciphertext only. The proposed solution achieves CCA2 security and Full proof analysis is also given under the ideal cipher model.

## 7.1 Introduction

Currently, cloud storage is becoming more popular. A promising technology called *deduplication* has been applied to reduce the cost of the cloud storage. Deduplication is essentially a data compression technique for elimination of redundant data. A typical flavor of deduplication detects duplicate data blocks within the storage device and deduplicates them by placing pointers rather than storing multiple copies at various places within the disk.

Client-side deduplication in cloud storage attempts to save the bandwidth of uploading copies for the existing files to the server. It enables the cloud server to store a single copy of repeating data. In a typical cloud storage with client-side deduplication, e.g. Dropbox [1], Wuala and MozyHome, the user sends to the server

[1]It is noticed that Dropbox disabled the deduplication across different users (in Feb 2012), probably due to recent vulnerabilities discovered in their original cross-user client-side deduplication

a hash of the file before uploading. Then the server checks whether this hash value is already in the database. If so, the server claims that the file is already in the cloud storage and is not needed to upload. Recently, Harnik *et al.* [HPSP10] addressed the security problems of client-side deduplication. A malicious user who gets the hash signature of a file can convince the cloud server about his ownership. By accepting the hash value as a "proxy" for the entire file, the cloud server allows anyone who gets the hash value to access the entire file. To overcome such attacks, Halevi *et al.* [HHPSP11] introduced the notion of proofs-of-ownership (POW). In addition, they present Merkle-tree based schemes to allow an user efficiently prove about the user's ownership, rather than some short information. The POW protocol works by first encoding the file $F$ using an encoding function $E$ which is resilient to erasure of up to $\alpha$ fraction of the bits, and then building a Merkle-tree over the encoded file $F' = E(F)$. Let $H$ be a collision resistant hash function with output length of $\lambda$ bits. Let $MT_{H,\lambda}(F')$ be the binary Merkle-tree over buffer $F'$ by using $\lambda$-bit leaves and the hash function $H$. On $M$-bit input file $F \in \{0,1\}^M$, the server who is the verifier computes the encoded $F' = E(F)$ and builds the Merkle-tree $MT_{H,\lambda}(F')$. After that, the server stores the root node $H_{root}$ of the Merkle-tree. During the protocol execution of POW, the server randomly chooses $u$ leaf indexes, $l_1, l_2, \cdots, l_u$, where $u$ is the smallest integer such that $(1-\alpha)^u < \epsilon$ ($\epsilon$ is the desired soundness bound). The server then enquires the sibling-paths of all the leaves from the prover, and accepts if all are valid with respect to $MT_{H,\lambda}(F')$.

However, the state-of-the-art work by Halevi *et al.* [HHPSP11] cannot be adopted directly for a joint ownership where multiple users share the ownership of the input file $F$, because the protocol execution is related to the proven file only. Additionally, their scheme cannot be adopted for encrypted file scenario, because encryption of the same file by different users with random keys results in different ciphertexts. The server cannot store the same hash root value for ownership verification.

There are applications of ownership sharing by different users. For example, multiple members in a workgroup share the ownership of some working documents in clouds. No one can delete the files individually before the files are not downloaded by every member. These working documents are usually be encrypted before outsourcing, for security and privacy considerations. The cloud provider who has access to the storage infrastructure, can neither be considered as a fully trusted

method. This also indicates the importance and urgency in the study of security in client-side deduplication.

Table 7.1: Comparisons between previous schemes and the proposed scheme.

| | [NWZ12] | [PS12] | [ZX12] | [XCZ13] | Proposed Scheme |
|---|---|---|---|---|---|
| Proof of ownership | ✓ | ✓ | ✓ | ✓ | ✓ |
| Privacy | ✓ | × | ✓ | ✓ | ✓ |
| Encrypted files | ✓ | × | ✓ | × | ✓ |
| Joint ownership | × | × | × | × | ✓ |

party nor a resistance to attacks. Thus in this chapter, a solution for proof of a multiparty ownership with the encrypted data is designed. Every user can prove that he holds the plaintext of the encrypted file when the server stores one ciphertext only. The proposed protocol is called as proof of multiparty ownership (**PMOW**) to emphasize its simplicity.

**Main Contributions.** The main contributions of this chapter can be illustrated as following: 1) **PMOW** is proposed. This protocol helps different users prove the joint ownership of encrypted files for client-side deduplication in clouds. 2) CCA2 security against arbitrary adversaries is achieved under the ideal cipher model.

**Comparison.**

A comparison is made between the proposed PMOW scheme and some selected existing schemes in Table 7.1. The comparison is mainly based on the following requirements and properties:

- *Proof of ownership.* A malicious user who gets some partial information of a file cannot convince the verifier about his ownership.
- *Privacy.* During a proof of ownership execution, only the privileged users should obtain access to the client's data files. In other words, the verifier who has access to the storage infrastructure, can neither be considered as a fully trusted party nor a resistance to attacks.
- *Encrypted files.* The proof of ownership protocol should be applied to encrypted files scenario.
- *Joint ownership.* The proof of ownership protocol should support the applications of ownership sharing by different users.

**Related Work.** Proof-of-ownership (POW) is closely related to two other similar problems: proof of retrievability (POR) and proof data possession (PDP). POR schemes [JJ07], [SW08], [BJO09], [DVW09], [ZX11] are challenge-response protocols. In POR schemes, a cloud provider demonstrates the file retrievability (i.e., recoverability without any loss or corruption) to a client. PDP schemes [ABC+07], [ABC+11], [EKPT09], [SFB+08], [FB06], [SBD+04], [WWL+09, WWRL10, WWR+11, ZB12, YOA07] are similar protocols which only detect a large amount of corruption in outsourced data. The main difference between POW and POR/PDP is that the latter usually use a pre-processing step on the client side, while the former does not.

POW protocols are proposed for client-side data deduplication that enables the storage server to store a single copy of repeating data. Cilent-side data deduplication has become popular and important as it removes data redundancy and data replication, but it brings many data privacy and security issues for the user. Douceur *et al.* [DAB+02] first studied the problem of deduplication in a multi-tenant system in which deduplication had to be reconciled with confidentiality. Their proposed convergent encryption enabled two users to produce a single ciphertext for deduplication. As there are many security problems of convergent encryption, Storer *et al.* [SGLM08] proposed a security model to secure data deduplication.

Recently, Harnik *et al.* [HPSP10] formally identified the security problems of client-side deduplication as follows: 1) The first kind of attacks attempted to fool the storage server and abuse the storage system. A malicious user with the hash signature of a file could convince the cloud server about his ownership. By accepting the hash value as a 'proxy' for the entire file, the cloud server allowed anyone who held the hash value to access the entire file. 2) The second kind of attacks targeted the privacy and confidentiality of users of the storage system. A malicious user could check whether another honest user had already outsourced a data file by trying to upload it as well. 3) The third kind of attacks focused on subverting the intended use of a storage system. For example, two malicious users tried to use the cloud storage for a convert channel as they might not have a direct interaction channel. These two users first pre-agreed on two different files. Second, one malicious user outsourced one of the two files. Then the other user could detect which file had been deduplicated and output either 0 (for the fist file) or 1 (for the second file). In this way, two malicious users successfully exchange a bit of information without a direct transmitting channel.

To overcome such attacks, Halevi *et al.* [HHPSP11] introduced the notion of

POW for client-side deduplication. In addition, they presented Merkle tree-based schemes to allow a user to efficiently prove his ownership to the server, rather than short information about the file such as a hash value. However, the state-of-the-art work [HHPSP11] cannot be adopted directly for multiple users who share the ownership of the input file $F$. In addition, their scheme cannot be adopted for encrypted file scenarios, as mentioned before. Some other schemes [RMW12], [NWZ12], [PS12], [ZX12], [XCZ13] focused on improving efficiency of POW, or applying an encrypted file scenario. In [PS12], Pietro and Sorniottis proposed three correlative protocols to achieve an efficient POW for deduplication. The main idea of their protocols is to challenge random $K$ bits of file $F$. The probability that a malicious user is able to output the correct value of $K$ bits of the file, where each bit selected at a random position, is negligible in the security parameter $k$. However, their scheme cannot be adopted for encrypted file scenarios. In addition, the client's files are totally revealed to the cloud server during the protocol executions. Zheng *et al.* [ZX12] argued that the public verifiability offered by POR/PDP schemes could be naturally exploited to achieve POW, however by using POR/PDP schemes to achieve POW, their scheme brings the clients mass information because it stores all the verified tags. In [NWZ12], they presented a private POW scheme for encrypted files in cloud storage. In [XCZ13], they proposed a secure client-side deduplication scheme to protect data confidentiality (and some partial information) against both outside adversaries and honest-but-curious cloud storage server, while Halevi *et al.* [HHPSP11] trusted cloud storage server in data confidentiality. However, none of the above schemes can solve the POW problem for multiparty users because the protocol executions are only related to the file that has to be proven.

## 7.2 Technique Preliminaries

This section starts by describing the framework for proof of multiparty ownership in cloud storage, as shown in Figure 7.1. The **PMOW** protocol checks whether every user has the ownership of the file $F$. Prior to outsource the data file, the client $\mathcal{C}$ encrypts the file, chooses a random symmetric key $k$ and generates several ciphertexts $CT_i(k)$ for users $U_i$, $1 \leqslant i \leqslant n$. Then the client $\mathcal{C}$ outsources the ciphertexts $(CT_F, \{CT_i(k)\})$ to cloud server $\mathcal{V}$ who is the verifier in the protocol. $\mathcal{V}$ uses an erasure-code $R$ to pre-process the input buffer $CT_F$ and divides $R(CT_F)$ into blocks $m_j$, $1 \leqslant j \leqslant m$. These blocks are hashed in pairs until the root node $H_{root}$

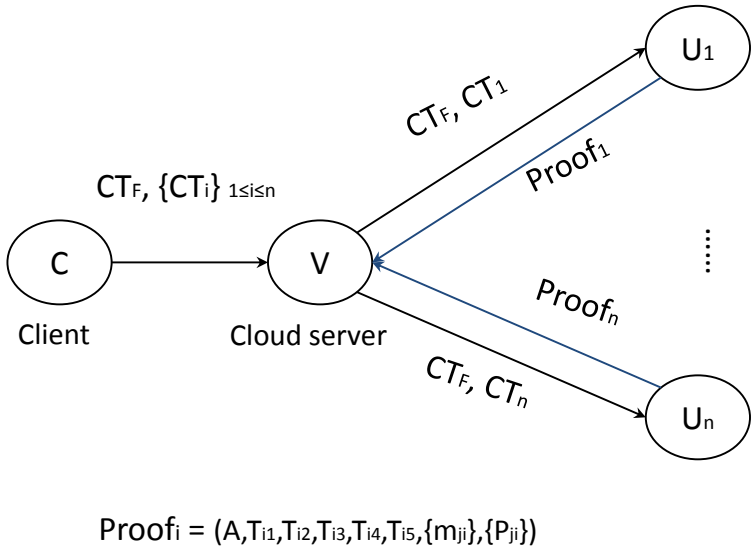


Figure 7.1: The framework for proof of multiparty ownership in cloud.

is reached, as described in Figure 7.2. By generating such a Merkle-tree $MT_{H,\lambda}$, $\mathcal{V}$ can compute a short verification information $v$ which is the root node of the tree. Later at a time, the multiple users $\mathcal{U}_i$ who share the file ownership download the ciphertexts and decrypt the symmetric key $k$ and the file $F$ ulteriorly. $\mathcal{V}$ sends a challenge to every user in the set and asks a proof based on the file $F$. Upon successfully completion of the protocol, the cloud server $\mathcal{V}$ is convinced that every user $U_i$ $(1 \leqslant i \leqslant n)$ has the file $F$. To conduct the protocol, $\mathcal{V}$ only needs to know the short verification information $v$. In addition, no information about $F$ is revealed to $\mathcal{V}$.

**Theat Model.** The multiple users $\mathcal{U}_i$ must answer the challenges from the cloud server (verifier) $\mathcal{V}$. If the users are failing to do so, it represents a failing joint proof of ownership. However the users are not trusted, that is to say, some of the users may try to convince the verifier alone with their own proofs. The motivations can be various for their misbehavior. For example, a malicious user wants to tamper with the working group's documents by giving the verifier an forged ownership proof which should be generated by all the group members. Additionally, the cloud server $\mathcal{V}$ must verify the multiparty ownership of the stored encrypted file. However the cloud server may not be fully trusted as well. That is to say, the cloud server may

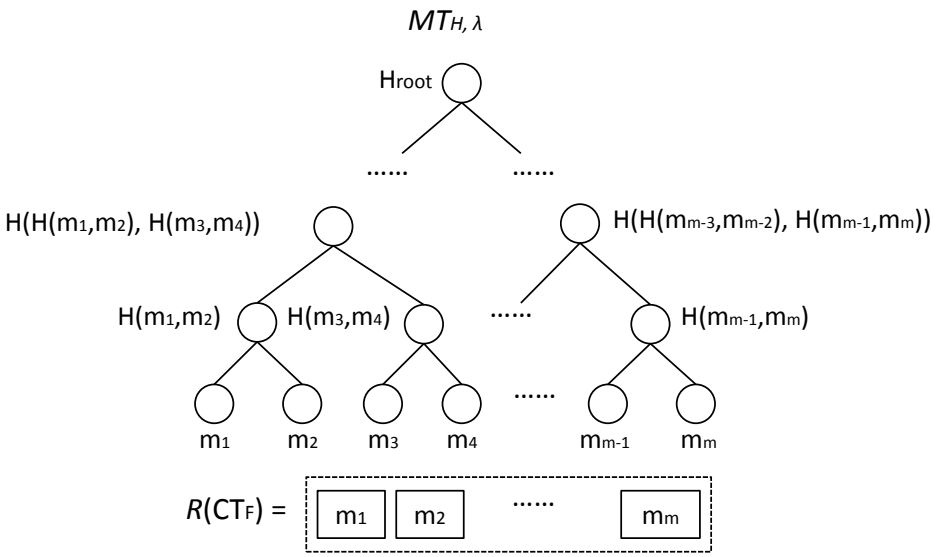


Figure 7.2: Build the Merkle-Tree.

try to retrieve the client $\mathcal{C}$'s file through the proof protocol executions. A **PMOW** protocol must detect the misbehavior of the users when someone cheats with the forged joint proof of ownership. Moreover, it has to ensure that the cloud server cannot learn any outsourced files through all the protocol executions.

### 7.2.1 Preliminaries

In this section, some notations are defined and the cryptographic tools which are used as building blocks in **PMOW** construction are reviewed.

***Notations.*** Let $\lambda$ be a security parameter, a function $\epsilon(\lambda)$ is negligible if it is smaller than $\lambda^{-const}$ for any *const* and sufficiently large $\lambda$. Let $p$ be a large prime and $G$ be a group of prime order $p$. Let $H(\cdot)$ be a collision-resistant hash function that maps binary strings of arbitrary length to binary strings of a fixed length, i.e., $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$. The scheme also employs an erasure-code $R(\cdot) : \{0,1\}^F \rightarrow \{0,1\}^{F'}$, resilient to erasure of $\alpha$ fraction of the bits (for some constant $\alpha > 0$). Namely, from any $(1-\alpha)M'$ bits of $R(F)$ it is possible to recover the original $F$. Let *Enc*, *Dec* be the symmetric encryption/decryption algorithms. Denote $MT_{H,\lambda}$ (with respect to hash function $H$) as the Merkle-tree and $MTP_H(v, m, u)$ as the

Merkle-tree protocol where the verifier knows the root value $v$ and number of leaves $m$, and asks to see $u$ leaves of the tree and their sibling paths.

***Merkle-tree Lemma.*** A Merkle tree can provide a commitment to a large input file. With this commitment, it is later possible to verify individual blocks of the file without giving the entire input buffer. The same method is used to construct the Merkle tree in [HHPSP11] . Firstly, the large input file is divided into blocks. Secondly, group these blocks in pairs and use a collision-resistant hash function to hash each pair. Then the hash values are grouped again in pairs and each pair is further hashed until reach the root hash value. At last it constructs a binary tree with the leaves corresponding to the file blocks and the root corresponding to the last remaining hash value. In the following the notations in [HHPSP11] are used. $MT_{H,\lambda}$ (with respect to hash function $H$) denotes the Merkle-tree and $MTP_H(v, m, u)$ denotes the Merkle-tree protocol where the verifier knows the root value $v$ and number of leaves $m$, and asks to see $u$ leaves of the tree and their sibling paths (for a leaf node $l \in MT_{H,\lambda}$, the sibling path of $l$ consists of all the sibling nodes on the path from $l$ to the root). The following lemma shows that every prover that passes the Merkle-tree protocol with high enough probability can be converted into an extractor that extracts most of the leaves of the tree. The security of Merkle-tree lemma has been proved in [HHPSP11].

**Lemma 7.1** *There exists a black-box extractor $\mathcal{K}$ with oracle access to a Merkle-tree prover, that has the following properties:*

- *For every prover $P$ and $v \in \{0,1\}^*$, $m, u \in \mathbb{N}$ and $\beta \in [0,1]$, $\mathcal{K}^P(v, m, u, \beta)$ makes at most $\frac{u^2 s(\log(s)+1)}{\beta}$ calls to its prover oracle $P$.*
- *Fix any hash function $h$ and input buffer $F$ with $m$ leaves of $\lambda$-bits each, and let $v$ be the root value of $MT_{h,\lambda}$. Also fix some $u \in \mathbb{N}$ and a prover (that may depend on $h$, $F$ and $u$) $P^* = P^*(h, F, u)$.*

*Then if $P^*$ has probability at least $(1-\alpha)^u+\beta$ of convincing the verifier in $MTP_h(v, m, u)$ (for some $\alpha$, $\beta \in (0,1]$), then with probability at least $\frac{1}{4}$ ( over its internal randomness) the extractor $\mathcal{K}^{P^*}(v, m, u, \beta)$ outputs values for at least a $(1-\alpha)$-fraction of the leaves of the tree, together with valid sibling paths for all the leaves.*

### 7.2.2 Formal Definitions

The syntax of the proposed proof of multiparty ownership (**PMOW**) scheme is given as follows.

**Definition 7.1** *(Proof of Multiparty Ownership Scheme) A proof of multiparty ownership scheme consists of four polynomial-time algorithms ($KeyGen_C$, $KeyGen_U$, $FilePrep_C$, $FilePrep_U$) and an interactive multiparty protocol (Proof) that captures the properties including strong proof-of-ownership.*

**KeyGen**$_C$. This algorithm is run by the client who outsource the file. This algorithm takes as input a security parameter $\lambda$ and gives as output the client's key pair ($CPK$, $CSK$).

**KeyGen**$_U$. This algorithm is run by the user. **KeyGen**$_U$ takes as input the security parameter $\lambda$ and outputs a pair of matching public and secret keys ($PK$, $SK$).

**FilePrep**$_C$. This algorithm is run by the client to prepare the file before outsourcing. It takes as input ($F$,$PK_{U_i}$,$CSK$) and gives as output the ciphertext of $CT_F$.

**FilePrep**$_V$. This algorithm is run by the cloud server to prepare the verification information. It takes as input $CT_F$ and gives as output the verification information $v$ on file $CT_F$.

**Proof.** This is a tuple of multiparty interactive algorithms ($\{$**Proof**$_{U_i}\}$, **Proof**$_V$) executed by the users and the cloud server (verifier) respectively. The input of **Proof**$_{U_i}$ is ($CT_F$, $CPK$, $SK_{U_i}$). The input of **Proof**$_V$ is ($v$, $PK_{U_i}$, $CT_F$). Upon completion of the protocol, **Proof**$_V$ outputs 1 if the proof is accepted, and 0 otherwise.

**Definition 7.2** *(Correctness) A proof of multiparty ownership scheme is correct if for ($CPK$, $CSK$) output by **KeyGen**$_C$, for all key pairs ($PK$, $SK$) output by **KeyGen**$_U$, for all $CT_F$ output by **FilePrep**$_C$ with input ($F$,$PK_{U_i}$,$CSK$), for all verification information $v$ output by **FilePrep**$_V$ with input $CT_F$, **Proof**$_V$ with input ($CT_F$, $PK_{U_i}$) outputs 1 if it is interacting with all the **Proof**$_{U_i}$ with input ($CT_F$, $CPK$, $SK_{U_i}$).*

## 7.3 The Proposed Scheme

Firstly, a short review of Cramer and Shoup's CCA2 encryption scheme [CS98] and Desai's CCA2 Unbalanced Feistel Encryption (UFE) scheme [Des00] is given. Secondly, the proof protocol between the cloud server (verifier) and the multiple users (prover) is described. Thirdly, the main construction of the proposed scheme is presented. The proposed construction is based on the original scheme in [HHPSP11] and Cramer and Shoup's CCA2 encryption scheme is used to encrypt $CSK$.

### 7.3.1 Cramer and Shoup's CCA2 encrytion scheme

*KeyGeneration.* Choose random elements $g_1$, $g_2 \in G$ and $x_1$, $x_2$, $y_1$, $y_2$, $z \in \mathbf{Z}_p$. Choose a hash function $H$ from the family of universal one-way hash functions. Then the private key is $(x_1, x_2, y_1, y_2, z)$, the public key is $(g_1, g_2, e, d, h, H)$ where $e = g_1^{x_1} \cdot g_2^{x_2}$, $d = g_1^{y_1} \cdot g_2^{y_2}$, $h = g_1^z$.

*Encryption.* Given a symmetric key $k \in G$, the encryption algorithm chooses $r \in_R \mathbf{Z}_p$ and generates the ciphertext $CT = (c_1, c_2, c_3, c_4)$ where $c_1 = g_1^r$, $c_2 = g_2^r$, $c_3 = h^r \cdot k$ and $c_4 = (g_1^{x_1} g_2^{x_2})^r \cdot (g_1^{y_1} g_2^{y_2})^{r \cdot H(c_1, c_2, c_3)}$.

*Decryption.* Given a ciphertext $CT$, this algorithm computes $\omega = H(c_1, c_2, c_3)$ and test if $c_1^{x_1 + y_1 \omega} \cdot c_2^{x_2 + y_2 \omega}$ equals to $c_4$. If yes, this algorithm outputs $k = c_3 / c_1^z$.

### 7.3.2 Desai's CCA2 UFE scheme

Desai's UFE scheme is a CCA2 symmetric encryption scheme which is based on a variable-length input pseudorandom function (VI-PRF) and a variable-length output pseudorandom function (VO-PRF). A VI-PRF is a function that takes inputs of any pre-specified length or of variable length and outputs some fixed length. Some efficient constructions of VI-PRFs are the CBC-MAC variant analyzed by Petrank and Rackoff [PR00] and the "three-way" variants of Black and Rogaway [BR00]. A VO-PRF is a function that takes inputs a fixed-length part and a part specifying the length of the required output, outputs some pre-specified length or variable length. A simple and efficient VO-PRF has been presented in [Des00]. Denote $\mathcal{K}_{vi-prf}$ and $\mathcal{K}_{vo-prf}$ are the randomized key generation algorithms for VI-PRF and VO-PRF respectively. Let $I : \mathcal{K}_{vi-prf} \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be a VI-PRF. Let $G : \mathcal{K}_{vo-prf} \times \{0,1\}^* \times 1^* \rightarrow \{0,1\}^*$ be a VO-PRF. The UFE scheme is as following.

*KeyGeneration.* This algorithm specifies a key $k = k_1 \| k_2$ ($|k_1| = |k_2|$) where $k_1 \leftarrow \mathcal{K}_{vi-prf}$ and $k_2 \leftarrow \mathcal{K}_{vo-prf}$.

*Encryption.* Given a file $F$, this algorithm chooses a random $r \leftarrow \{0,1\}^{\lambda}$, computes $C = F \oplus G_{k_2}(r)$ and $\sigma = r \oplus I_{k_1}(C)$, and outputs a ciphertext $CT = C \| \sigma$.

*Decryption.* Given a ciphertext $CT$, this algorithm firstly parses $CT$ as $C \| \sigma$ where $|\sigma| = \lambda$ and then computes $r = \sigma \oplus I_{k_1}(C)$. The output of this algorithm is $F = C \oplus G_{k_2}(r)$.

### 7.3.3 Main Construction

The main construction of the proposed scheme is as follows.

**KeyGen**$_C$. Takes the security parameter $\lambda$ as input, this algorithm outputs the symmetric key $k = k_1 \| k_2$.

**KeyGen**$_U$. This algorithm takes as input the security parameter $\lambda$ and gives as output ($pk_i$, $sk_i$) for user set $\mathcal{I} \subset \{1, \cdots, n\}$. For each user $i \in \mathcal{I}$, this algorithm chooses random elements $g_1$, $g_2 \in G$ and $x_{i1}$, $x_{i2}$, $y_{i1}$, $y_{i2}$, $z_i \in \mathbf{Z}_p$, computes

$$e_i = {g_1}^{x_{i1}} \cdot {g_2}^{x_{i2}},\ d_i = {g_1}^{y_{i1}} \cdot {g_2}^{y_{i2}},\ h_i = {g_1}^{z_i}.$$

Then the public key $pk_i = (g_1, g_2, e_i, d_i, h_i, H)$ and the secret key $sk_i = (x_{i1}, x_{i2}, y_{i1}, y_{i2}, z_i)$.

**FilePrep**$_C(\{pk_i\}_{i\in\mathcal{I}}, k, F) \rightarrow (CT_F, \{CT_i\}_{i\in\mathcal{I}})$. This algorithm takes as input the file $F$, the symmetric key $k$ and all $pk_i$ for $i \in \mathcal{I}$, it outputs the ciphertext of file $F$ as $CT_F = C \| \sigma$ where $C = F \oplus G_{k_2}(r_1)$ and $\sigma = r_1 \oplus I_{k_1}(C)$ for $r_1 \leftarrow \{0,1\}^{\lambda}$. It also outputs the ciphertexts $CT_i$ as $CT_i = (C_{i1}, C_{i2}, C_{i3}, C_{i4})$ where $C_{i1} = g_1^{r_2}$, $C_{i2} = g_2^{r_2}$, $C_{i3} = h_i^{r_3} \cdot k$, and $C_{i4} = ({g_1}^{x_{i1}} {g_2}^{x_{i2}})^{r_2} \cdot ({g_1}^{y_{i1}} {g_2}^{y_{i2}})^{r_2 \cdot H(C_{i1}, C_{i2}, C_{i3})}$ for $r_2 \in_R \mathbf{Z}_p$.

**FilePrep**$_V$ ($CT_F \rightarrow v$) Before generating the verification information $v$, the input buffer $CT_F$ is firstly pre-processed into $R(CT_F)$by using the erasure-code $R(\cdot)$. Divides $R(CT_F)$ into $m = \lceil \frac{|R(CT_F)|}{\lambda} \rceil$ blocks. In the following, each block is referred as $m_j$ for $1 \leqslant j \leqslant m$. This algorithm then construct a binary Merkle-tree $MT_{H,\lambda,i}$ (with respect to hash function $H$) on the data blocks $m_j$ as $m$ leaf nodes. It groups the hashed leaves in pairs and continue to hash each pair until reach the root node $H_{root}$. At last, this root node is stored as the verification information $v$.

**Proof.** The tuple of algorithms $\{\mathbf{Proof}_{U_i}\}_{i\in\mathcal{I}}$, $\mathbf{Proof}_V$ interact as follows.

- $\mathbf{Proof}_V$ sends the ciphertexts $CT_F$, $CT_i$ to $\mathbf{Proof}_{U_i}$.
- $\mathbf{Proof}_{U_i}$ computes $\omega = H(C_{i1}, C_{i2}, C_{i3})$ and tests if

$$C_{i1}^{x_{i1}+y_{i1}\cdot\omega} \cdot C_{i2}^{x_{i2}+y_{i2}\cdot\omega} \stackrel{?}{=} C_{i4}.$$

  If yes, then computes

$$k = k_1 \| k_2 = \frac{C_{i3}}{C_{i1}^{z_i}}, \;\; r_1 = \sigma \oplus I_{k_1}(C), \;\; F = C \oplus G_{k_2}(r_1).$$

- $\mathbf{Proof}_V$ runs $MTP_H(v, m, u)$ and chooses random $u$ leaf nodes, where $u$ is the smallest integer such that $(1-\alpha)^u < \epsilon$ for desired soundness bound $\epsilon$. $\mathbf{Proof}_V$ also randomly chooses $c, d \in_R \mathbf{Z}_p$ and computes $D = g_1^c \cdot g_2^d$. Let $\mathcal{L} \subset \{1, \cdots, u\}$ be the index set of the leaf nodes. $\mathbf{Proof}_V$ sends $\mathcal{L}$ and $D$ to $\mathbf{Proof}_{U_i}$ as the challenge.
- $\mathbf{Proof}_{U_i}$ first computes $A = k \cdot {y_i}^{t_i}$ for some random $t_i \in \mathbf{Z}_p$, $y_i \in G$. Then $\mathbf{Proof}_{U_i}$ chooses $\rho_{z_i}$,$\rho_{t_i}$, $\rho_{x_{i1}}$, $\rho_{x_{i2}}$, $\rho_{y_{i1}}$, $\rho_{y_{i2}} \in_R \mathbf{Z}_p$ and computes

$$\begin{aligned}
T_{i1} &= C_{i1}^{\rho_{z_i}} \cdot {y_i}^{-\rho_{t_i}}, \\
T_{i2} &= g_1^{\rho_{z_i}}, \\
T_{i3} &= C_{i1}^{\rho_{x_{i1}}} \cdot (C_{i1}^{\omega})^{\rho_{y_{i1}}} \cdot C_{i2}^{\rho_{x_{i2}}} \cdot (C_{i2}^{\omega})^{\rho_{y_{i2}}}, \\
T_{i4} &= g_1^{\rho_{x_{i1}}} \cdot g_2^{\rho_{x_{i2}}}, \\
T_{i5} &= g_1^{\rho_{y_{i1}}} \cdot g_2^{\rho_{y_{i2}}}.
\end{aligned}$$

  $\mathbf{Proof}_{U_i}$ next constructs the Merkle-tree $MT_{H,\lambda}$ which is corresponding with $\{m_j\}_{1\leqslant j\leqslant m}$. Denote $\mathcal{P}_j$ the set of all the sibling nodes on the path from leaf $m_j$ to root $H_{root}$ in $MT_{H,\lambda}$. After that, $\mathbf{Proof}_{U_i}$ sends $\mathbf{Proof}_V$ the proof ($A$, $T_{i1}$, $T_{i2}$, $T_{i3}$, $T_{i4}$, $T_{i5}$, $\{m_{ji}\}_{j\in\mathcal{L},i\in\mathcal{I}}$, $\{\mathcal{P}_{ji}\}_{j\in\mathcal{L},i\in\mathcal{I}}$).
- $\mathbf{Proof}_V$ randomly sends the challenge $c, d$ to $\mathbf{Proof}_{U_i}$.
- $\mathbf{Proof}_{U_i}$ checks if $D = g_1^c \cdot g_2^d$ and computes

$$\begin{aligned}
z_{z_i} &= \rho_{z_i} - c \cdot z_i, \qquad z_{t_i} = \rho_{t_i} - c \cdot t_i, \\
z_{x_{i1}} &= \rho_{x_{i1}} - c \cdot x_{i1}, \quad z_{x_{i2}} = \rho_{x_{i2}} - c \cdot x_{i2}, \\
z_{y_{i1}} &= \rho_{y_{i1}} - c \cdot y_{i1}, \quad z_{y_{i2}} = \rho_{y_{i2}} - c \cdot y_{i2}.
\end{aligned}$$

  and sends ($z_{z_i}$, $z_{t_i}$, $z_{x_{i1}}$, $z_{x_{i2}}$, $z_{y_{i1}}$,$z_{y_{i2}}$) to $\mathbf{Proof}_V$.

- **Proof**$_V$ generates $\{H_{root}\}_{i\in\mathcal{I}}$ according to $(\{l_{ji}\}_{j\in\mathcal{L},i\in\mathcal{I}}, \{\mathcal{P}_{ji}\}_{j\in\mathcal{L},i\in\mathcal{I}})$ and computes $v' = H(\{H_{root}\}_{i\in\mathcal{I}})$. It outputs 1 if the following equations hold

$$
\begin{aligned}
&v' \stackrel{?}{=} v,\\
&T_{i1} \stackrel{?}{=} [\frac{C_{i3}}{A}]^c \cdot C_{i1}^{z_{z_i}} \cdot (y^{-1})^{z_{t_i}},\\
&T_{i2} \stackrel{?}{=} h_i^c \cdot g_1^{z_{z_i}},\\
&T_{i3} \stackrel{?}{=} C_{i4}^c \cdot C_{i1}^{z_{x_{i1}}} \cdot (C_{i1}^{\omega})^{z_{y_{i1}}} \cdot C_{i2}^{z_{x_{i2}}} \cdot (C_{i2}^{\omega})^{z_{y_{i2}}},\\
&T_{i4} \stackrel{?}{=} e_i^c \cdot g_1^{z_{x_{i1}}} \cdot g_2^{z_{x_{i2}}},\\
&T_{i5} \stackrel{?}{=} d_i^c \cdot g_1^{z_{y_{i1}}} \cdot g_2^{z_{y_{i2}}}.
\end{aligned}
$$

  and 0 otherwise.

## 7.4 Security Requirement

There are two security requirements that a construction of **PMOW** scheme must possess. One concerns about *joint proof of ownership*. Specifically, in a **PMOW** scheme with joint proof of ownership, a valid proof can only be generated by all privilege users who have the whole file $F$. Another one regards indistinguishability. In a **PMOW** scheme with indistinguishability, no cloud server should be able to learn the verified file during the proof executions.

### 7.4.1 Joint proof of ownership

This requirement is formalized by using security game played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. In the PMOW-game, the challenger $\mathcal{C}$ denotes the cloud server and the adversary$\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ denotes $n$ cheating users. $\mathcal{A}_1$ controls all the users and can make arbitrary queries to the challenger. $\mathcal{A}_2(st_i)$ presents a user $i$ with input the state information $st_i$. The game is divided into two phases, the learning phase and the proof phase. In the learning phase, $\mathcal{A}_1$ generates arbitrary state information after querying the challenger; in the proof phase, every user $\mathcal{A}_2(st_i)$ generates his proof without any interaction.

**PMOW-game**

- **Setup:** The challenger $\mathcal{C}$ runs **KeyGen**$_C$ to generate a key pair (CPK, CSK). The challenger $\mathcal{C}$ also runs **KeyGen**$_U$ to generate $n$ key pairs (PK$^*$,SK$^*$). The adversary $\mathcal{A}$ is given (CPK, PK$^*$,SK$^*$).

- **Phase 1:** For any file $F$, the adversary $\mathcal{A}_1$ can make arbitrary queries to the challenger $\mathcal{C}$ before the proof protocol begins. For (PK$^*$, SK$^*$), the challenger generates the verification information v and returns the ciphertext of $CT_F$ and $CSK$. The adversary $\mathcal{A}_1$ outputs arbitrary state information $(st_1, st_2, \cdots, st_n)$.

- **Phase 2:** The adversary $\mathcal{A}_2$ takes input the state information $st_i, 1 \leqslant i \leqslant n$ individually. Without any interaction, $\mathcal{A}_2(st_1)$, $\mathcal{A}_2(st_2)$, $\cdots$, $\mathcal{A}_2(st_n)$ output a prover machine $P$ and a joint proof $v'$.

Informally speaking, the joint proof of ownership property requires that if $P$ can pass through the proof protocol with $\mathcal{C}$, $\mathcal{A}_2(st_i)$ should be in possession of the file $CT_F$ and symmetric key $k$. This can be formalized as follows.

**Definition 7.3** *(Soundness) A **PMOW** scheme is sound if there exist a knowledge extractor $\mathcal{K} = (\mathcal{K}_1, \mathcal{K}_2)$ that can recover $CT_F$ and $CSK$ from every $\mathcal{A}_2(st_i)$ $(1 \leqslant i \leqslant n)$ with non-negligible probability.*

**Theorem 7.2** *The construction of a **PMOW** scheme captures the joint proof of ownership if Halevis POW scheme is secure in the random oracle model.*

*Proof.* The proof is divided into two parts. In the first part, the construction of a **PMOW** scheme is proven to be a proof of ownership protocol with soundness $(1-\alpha)^u$. Assume that there is an adversary $\mathcal{A}_2(st_i)$ that breaks the joint proof-of-ownership property. Specifically, it is assumed that the erasure code can correct erasure of up to $\alpha$-$fraction$ of the input. It is also assumed that the adversary $\mathcal{A}_2(st_i)$ runs an most $q_k$ proof protocols and succeeds in convincing the cloud server with probability better than $t(1-\alpha)^u + \beta$ ($t$ denotes the bits of min entropy of any input distribution $D$) for a non-negligible $\beta$.

Now it is shown how to construct the extractor $\mathcal{K}_1$. Note that $\mathcal{K}_1$ will only be used in proofs. It is an extractor from Merkle-tree lemma (which says that every prover that passes the Merkle-tree protocol with high enough probability can be converted into an extractor that extracts most of the leaves of the tree). If a prover $P^*$ has probability at least $(1-\alpha)^u + \beta$ of convincing the verifier in the Merkle-tree protocol (for some $\alpha, \beta \in (0, 1]$), then with probability at least $1/4$ (over its internal randomness) the extractor $\mathcal{K}$ outputs values for at least a $(1-\alpha)$-fraction

of the leaves of the tree, together with valid sibling paths for all these leaves. In [HHPSP11], the security of Merkle-tree lemma has been proven.

A simulator $S$ is created. $S$ gets as input a random hash function $H \in \mathcal{H}$. The simulator $S$ chooses the encrypted file $CT_F \in \mathcal{D}$ and runs the PMOW game with the challenger. Whenever the adversary $\mathcal{A}_2(st_i)$ begins a new interaction with the challenger $\mathcal{C}$, the simulator $S$ uses the extractor $\mathcal{K}_1$ to extract a $(1-\alpha)$-fraction of the leaves of the tree. If these $(1-\alpha)$ fraction of the leaves differ from the corresponding leaves in $CT_F$, then the simulator $S$ extracts a collision for the hash function $h$ form the Merkle tree, since both the paths in the real tree $MT_{H,\lambda}(F)$ and the paths given by the adversary $\mathcal{A}_2(st_i)$ (who is the prover) are valid.

As it is also assumed that the adversary (the same as the extractor) is missing at least slackness $s$ bits of min-entropy about encrypted file $CT_F$, then the probability of the extractor $\mathcal{K}_1$ outputting $(1-\alpha)$-fraction of $CT_F$ is at most $2^{-s}$. It is assumed that the adversary wins the game in at least one of the $k$ executions of the protocol with probability at least $(1-\alpha)^u + \beta/k$. Hence with probability at least $\beta/2k$ over the choice of $H \in \mathcal{H}$, this function $H$ still leaves the adversary with probability greater than $(1-\alpha)^u + \beta/2k$ of convincing the verifier in the protocol execution. For such hash function $H$, the extractor $\mathcal{K}_1$ will have probability of at least $1/4$ to output $(1-\alpha)$-fraction of the leaves. Hence for such function $h$, collisions can be found with probability at least $1/4 - 2^{-s}$, and the overall collision probability is at least $\beta/2k \cdot (1/4 - 2^{-s}) \approx \beta/8k$.

Thus, whenever the adversary $\mathcal{A}_2(st_i)$ begins a successful interaction with the challenger $\mathcal{C}$, the extractor $\mathcal{K}_1$ can extract $\frac{1-\alpha}{m}$ leaves of $l_{ij}$ for every $\mathcal{A}_2(st_i)$, $1 \leqslant i \leqslant n$. That is to say, the extractor $\mathcal{K}_1$ can output enough leaves to extract $CT_F$ from every $\mathcal{A}_2(st_i)$ will take $\frac{m^2}{(1-\alpha)\cdot\beta}$ interactions.

In the second part, the construction of a **PMOW** scheme is proven to be a joint proof of ownership protocol. Now it is shown how to construct the extractor $\mathcal{K}_2$. As the same as $\mathcal{K}_1$, $\mathcal{K}_2$ will only be used in proofs too. It is an algorithm that runs in probabilistic polynomial time on input the prover machine $P$, $CPK$ and $PK_{U_i}$. The output of the extractor $\mathcal{K}_2$ is the symmetric key $k$ and $SK_{U_i}$.

The knowledge extractor $\mathcal{K}_2$ will work in the following way: If the prover machine $P$ responds correctly to an initial challenge $c$, then $\mathcal{K}_2$ rewinds $P$ and issues a different challenge $c'$ for which $P$ also responds correctly. If the extractor $\mathcal{K}_2$ can find two such challenges $c$ and $c'$, then $\mathcal{K}_2$ has the following equations:

$$T_{i1} = [\frac{C_{i3}}{A}]^c \cdot C_{i1}^{z_{z_i}} \cdot (y^{-1})^{z_{t_i}} = [\frac{C_{i3}}{A}]^{c'} \cdot C_{i1}^{z'_{z_i}} \cdot (y^{-1})^{z'_{t_i}},$$
$$T_{i2} = h_i^c \cdot g_1^{z_{z_i}} = h_i^{c'} \cdot g_1^{z'_{z_i}},$$
$$T_{i3} = C_{i4}^c \cdot C_{i1}^{z_{x_{i1}}} \cdot (C_{i1}^{\omega})^{z_{y_{i1}}} \cdot C_{i2}^{z_{x_{i2}}} \cdot (C_{i2}^{\omega})^{z_{y_{i2}}} = C_{i4}^{c'} \cdot C_{i1}^{z'_{x_{i1}}} \cdot (C_{i1}^{\omega})^{z'_{y_{i1}}} \cdot C_{i2}^{z'_{x_{i2}}} \cdot (C_{i2}^{\omega})^{z'_{y_{i2}}},$$
$$T_{i4} = e_i^c \cdot g_1^{z_{x_{i1}}} \cdot g_2^{z_{x_{i2}}} = e_i^{c'} \cdot g_1^{z'_{x_{i1}}} \cdot g_2^{z'_{x_{i2}}},$$
$$T_{i5} = d_i^c \cdot g_1^{z_{y_{i1}}} \cdot g_2^{z_{y_{i2}}} = d_i^{c'} \cdot g_1^{z'_{y_{i1}}} \cdot g_2^{z'_{y_{i2}}}.$$

If denotes $\triangle c = c' - c$, $\triangle z_{z_i} = z'_{z_i} - z_{z_i}$, $\triangle z_{t_i} = z'_{t_i} - z_{t_i}$, $\triangle z_{x_{i1}} = z'_{x_{i1}} - z_{x_{i1}}$, $\triangle z_{x_{i2}} = z'_{x_{i2}} - z_{x_{i2}}$, $\triangle z_{y_{i1}} = z'_{y_{i1}} - y_{x_{i1}}$, $\triangle z_{y_{i2}} = z'_{y_{i2}} - z_{y_{i2}}$, from the above equations it can have

$$[\frac{C_{i3}}{A}]^{\triangle c} = C_{i1}^{-\triangle z_{z_i}} \cdot (y^{-1})^{-\triangle z_{t_i}},$$
$$h_i^{\triangle c} = g_1^{-\triangle z_{z_i}},$$
$$C_{i4}^{-\triangle c} = C_{i1}^{\triangle z_{x_{i1}}} \cdot (C_{i1}^{\omega})^{\triangle z_{y_{i1}}} \cdot C_{i2}^{\triangle z_{x_{i2}}} \cdot (C_{i2}^{\omega})^{\triangle z_{y_{i2}}},$$
$$e_i^{-\triangle c} = g_1^{\triangle z_{x_{i1}}} \cdot g_2^{\triangle z_{x_{i2}}},$$
$$d_i^{-\triangle c} = g_1^{\triangle z_{y_{i1}}} \cdot g_2^{\triangle z_{y_{i2}}}.$$

Then it has

$$\frac{C_{i3}}{C_{i1}^{\triangle z_{z_i}}} = A \cdot y^{-\frac{\triangle z_{t_i}}{\triangle c}},$$
$$h_i = g_i^{-\frac{\triangle z_{z_i}}{\triangle c}},$$
$$C_{i4} = C_{i1}^{-\frac{\triangle z_{x_{i1}}}{\triangle c}} \cdot (C_{i1}^{\omega})^{-\frac{\triangle z_{y_{i1}}}{\triangle c}} \cdot C_{i2}^{-\frac{\triangle z_{x_{i2}}}{\triangle c}} \cdot (C_{i2}^{\omega})^{-\frac{\triangle z_{y_{i2}}}{\triangle c}},$$
$$e_i = g_1^{-\frac{\triangle z_{x_{i1}}}{\triangle c}} \cdot g_2^{-\frac{\triangle z_{x_{i2}}}{\triangle c}},$$
$$d_i = g_1^{-\frac{\triangle z_{y_{i1}}}{\triangle c}} \cdot g_2^{-\frac{\triangle z_{y_{i2}}}{\triangle c}}.$$

Thus the knowledge extractor $\mathcal{K}_2$ can compute $k = A \cdot y^{-\frac{\triangle z_{z_i}}{\triangle c}}$, $z_i = -\triangle z_{z_i}/\triangle c$, $x_{i1} = -\triangle z_{x_{i1}}/\triangle c$, $x_{i2} = -\triangle z_{x_{i2}}/\triangle c$, $y_{i1} = -\triangle z_{y_{i1}}/\triangle c$ and $y_{i2} = -\triangle z_{y_{i2}}/\triangle c$. In other words, the extractor $\mathcal{K}_2$ can obtain $k$ and $SK_{U_i}$ from the above equations.

The extraction is required to be succeed (with all but negligible probability) from the adversary $\mathcal{A}_2(st_i)$ (prover) that causes the challenger $\mathcal{C}$ (verifier) to accept with a non-negligible probability $\beta$. Then the knowledge extractor $\mathcal{K}_2$ can recover the symmetric key $k$ from such an adversary will take $O(n/\beta)$ interactions.

### 7.4.2 Indistinguishability under chosen ciphertext attack

The indistinguishability against adaptive chosen ciphertext attack is defined as the following game between the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$. In the Indistinguishability-game, the challenger $\mathcal{C}$ denotes the client and honest users and the adversary $\mathcal{A}$ denotes a cheating cloud server who wants to learn the file that are stored in clouds.

**Indistinguishability-game**

- **Setup:** The challenger $\mathcal{C}$ takes a sufficiently large security parameter and runs $\mathbf{KeyGen}_C$ to generate a key pair (CPK, CSK). The challenger $\mathcal{C}$ also runs $\mathbf{KeyGen}_U$ to generate $n$ key pairs ($\mathrm{PK}_i$, $\mathrm{SK}_i$) for user set $\{U_i\}_{i\in\mathcal{I}}$. The adversary $\mathcal{A}$ is given (CPK, $\mathrm{PK}_i$).

- **Query Phase 1:** The adversary $\mathcal{A}$ can perform a polynomially bounded number of following queries in an adaptive manner:

  - FilePrep Query: $\mathcal{A}$ submits a file $F$. $\mathcal{C}$ chooses a random symmetric key $k$ and replies with

    $$\mathrm{FilePrep}(F, \{U_i\}_{i\in\mathcal{I}}) = (CT_F, \{CT_i(k)\}_{i\in\mathcal{I}}).$$

  - Reveal Query: $\mathcal{A}$ submits the ciphertexts of $CT'_F$ and $\{CT'_i(k)\}_{i\in\mathcal{I}}$. $\mathcal{C}$ replies with file $F$.

  - Proof Query: $\mathcal{A}$ submits the ciphertexts of $CT_F$ and $\{CT_i(k)\}_{i\in\mathcal{I}}$, and the user set $\{U_i\}_{i\in\mathcal{I}}$. $\mathcal{C}$ replies with $Proof$.

- **Challenge Phase:** $\mathcal{A}$ gives two messages $F_0^*$ and $F_1^*$ to the challenger $\mathcal{C}$. $\mathcal{C}$ randomly picks a bit $b$ and sends $(CT^*_{F_b}, \{CT_i^*(k)\}_{i\in\mathcal{I}})$ to $\mathcal{A}$ as returns.

- **Query Phase 2:** The adversary $\mathcal{A}$ can issue the same type of queries as query phase 1, except it cannot submit reveal queries with input $(CT^*_{F_b}, \{CT_i^*(k)\}_{i\in\mathcal{I}})$.

- **Guess Phase:** $\mathcal{A}$ outputs a guess $b^*$.

The advantage of an adversary $\mathcal{A}$ in the Indistinguishability-game is defined to be

$$Adv_{\mathcal{A}} = Pr[b^* = b] - \tfrac{1}{2}.$$

**Definition 7.4** *(Indistinguishability) A construction of **PMOW** is IND-CCA secure if no PPT adversary $\mathcal{A}$ can win Indistinguishability-game with non-negligible advantage.*

**Theorem 7.3** *The construction of a **PMOW** is secure if Cramer and Shoups scheme is CCA2 secure.*

*Proof.* Assume there exist an adversary $\mathcal{A}$ that can win the Indistinguishability game with non-negligible probability $\epsilon$. Then a simulator $\mathcal{S}$ which interacts with the adversary $\mathcal{A}$ can be constructed. In order to simplify the proof, an ideal cipher **Enc** is used, instead of Desais CCA2 symmetric encryption scheme. $\mathcal{S}$ runs in the following way:

- **Setup.** $\mathcal{S}$ receives the public keys $PK_i$ of the encryption schemes. Based on the public keys $PK_i$, $\mathcal{S}$ chooses a hash function $H(\cdot)$ and an erasure code $R(\cdot)$. Then $\mathcal{S}$ sends all the public parameters to the adversary $\mathcal{A}$.
- **Queries Simulation.** $\mathcal{S}$ simulates all the queries as follows.
  - FilePrep Query: $\mathcal{S}$ maintains an ideal cipher table of triples $(k, F, CT_F)$ of answered queries $CT_F = \mathbf{Enc}_k(F)$. The ideal cipher table is referred as "Enc-table". If there exists a tuple $(k, F, CT_F)$ in the Enc-table, $\mathcal{S}$ returns $CT_F$. Else, $\mathcal{S}$ computes the value $CT_F = \mathbf{Enc}(k, F)$ and records $(k, F, CT_F)$ in the Enc-table. Upon receiving the fileprep query for a file $F$, $\mathcal{S}$ chooses a random key $k$, searches the Enc-table and outputs the ciphertexts $(CT_F, CT_i(k))$ where $CT_F = \mathbf{Enc}_k(F)$ and $\{CT_i(k)\}_{i\in\mathcal{I}} = \{(C_{i1}, C_{i2}, C_{i3}, C_{i4})\}_{i\in\mathcal{I}}$ where $C_{i1} = g_1^{r_2}$, $C_{i2} = g_2^{r_2}$, $C_{i3} = h_i^{r_3} \cdot k$, and $C_{i4} = (g_1{}^{x_{i1}} g_2{}^{x_{i2}})^{r_2} \cdot (g_1{}^{y_{i1}} g_2{}^{y_{i2}})^{r_2 \cdot H(C_{i1}, C_{i2}, C_{i3})}$ for $r_2 \in_R \mathbf{Z}_p$.
  - Reveal Query: Define $DO_1$ as the decryption oracle of Cramer and Shoups scheme and $DO_2$ as the decryption oracle of ideal cipher **Enc**. If there exists a tuple $(k, F, CT_F)$ in the Enc-table, $DO_2$ answers $F$. Otherwise, $DO_2$ picks a random $F \leftarrow \{0,1\}^\lambda$, records $(k, F, CT_F)$ in the Enc-table and returns $F$ while making sure that no collision is created for $\mathbf{Enc}^{-1}(k, \cdot)$. Upon receiving the reveal query for the ciphertexts of $CT'_F$ and $\{CT'_i(k)\}_{i\in\mathcal{I}}$. $\mathcal{S}$ uses $DO_1$ and $DO_2$ to output the file $F$.

– Proof Query: Upon receiving the proof query for the ciphertexts of $CT_F$ and $\{CT_i(k)\}_{i\in\mathcal{I}}$, and the user set $\{U_i\}_{i\in\mathcal{I}}$. $\mathcal{S}$ produces an interaction that is indistinguishable form the adversary's interaction with the provers. In other words, $\mathcal{S}$ plays the role of a full zero-knowledge simulator for proof queries.

* $\mathcal{A}$ sends the values $\mathcal{I}$, $u$, $D$ to $\mathcal{S}$.
* $\mathcal{S}$ randomly picks and sends $T'_{i1}$, $T'_{i2}$, $T'_{i3}$, $T'_{i4}$, $T'_{i5}$, $A'$, $C'_{i1}$, $C'_{i2}$, $C'_{i3}$, $C'_{i4} \in_R \mathbf{Z}_p$ to $\mathcal{A}$.
* $\mathcal{A}$ returns $c, d$ to $\mathcal{S}$.
* $\mathcal{S}$ validates $D = g_1^c \cdot g_2^d$ and rewinds the verifier to the point when it receives the tuple $(\mathcal{I}, u, D)$.
* $\mathcal{S}$ randomly picks $z_{z_i}$, $z_{t_i}$, $z_{x_{i1}}$, $z_{x_{i2}}$, $z_{y_{i1}}$, $z_{y_{i2}}$, $A$, $C_{i1}$, $C_{i2}$, $C_{i3}$, $C_{i4} \in_R \mathbf{Z}_p$ and computes the following values for $\omega = H(C_{i1}, C_{i2}, C_{i3})$:

$$
\begin{aligned}
v' &= v, \\
T_{i1} &= [\frac{C_{i3}}{A}]^c \cdot C_{i1}^{z_{z_i}} \cdot (y^{-1})^{z_{t_i}}, \\
T_{i2} &= h_i^c \cdot g_1^{z_{z_i}}, \\
T_{i3} &= C_{i4}^c \cdot C_{i1}^{z_{x_{i1}}} \cdot (C_{i1}^{\omega})^{z_{y_{i1}}} \cdot C_{i2}^{z_{x_{i2}}} \cdot (C_{i2}^{\omega})^{z_{y_{i2}}}, \\
T_{i4} &= e_i^c \cdot g_1^{z_{x_{i1}}} \cdot g_2^{z_{x_{i2}}}, \\
T_{i5} &= d_i^c \cdot g_1^{z_{y_{i1}}} \cdot g_2^{z_{y_{i2}}}.
\end{aligned}
$$

To the adversary $\mathcal{A}$ all the distributions given by $\mathcal{S}$ are indistinguishable from the real one generated by the provers.

- **Challenge.** $\mathcal{A}$ submits two messages $F_0^*$ and $F_1^*$. $\mathcal{S}$ chooses two random keys $k_0^*, k_1^*$, flips a fair coin $b \in \{0, 1\}$, and computes $CT^* = \mathbf{Enc}(k_b, F_b)$ and $\{CT_i^*(k)\} = Enc_{pk_i}(k_b)$ for $i \in \mathcal{I}$.

Suppose there exists an adversary $\mathcal{A}$ which can win the Indistinguishability game with non-negligible probability $\frac{1}{2} + \epsilon$, in the following it is shown that the simulator $\mathcal{S}$ can breaks the CCA2 security of Cramer and Shoups encryption scheme $Enc$.

The adversary $\mathcal{A}$ is given the ciphertexts of $CT^* = \mathbf{Enc}(k_b, F_b)$ and $\{CT_i^*(k)\} = Enc_{pk_i}(k_b)$ for $i \in \mathcal{I}$. If $\mathcal{A}$ can win the game with non-negligible probability $\frac{1}{2} + \epsilon$, the simulator $\mathcal{S}$ can distinguish the ideal cipher $textbf{Enc}$ from a random funtion correctly with a probability $\frac{1}{2} + \frac{\epsilon}{2}$. The reason is that if **Enc** is a random function,

- **Setup.** The client $\mathcal{C}$ invokes the algorithm $\mathbf{KeyGen}_C$ to generate a symmetric key $k$ and also invokes the algorithm $\mathbf{KeyGen}_U$ to generate $n$ pairs of matching public and secret keys $(pk_i,\ sk_i)$ for user set $\mathcal{I} \subset \{1, \cdots, n\}$. $\mathcal{C}$ then invokes $\mathbf{FilePrep}_C$ for file $F$ and sends the ciphertexts $(CT_F,\ CT_i)$ to the cloud server $\mathcal{V}$. $\mathcal{V}$ invokes $\mathbf{FilePrep}_V$ to construct the binary Merkle-tree $MT_{H,\lambda}$ and generate the verification information $v$. Note that $\mathcal{V}$ only access the verification information $v$ during the protocol executions.
- **Proof.** $\mathcal{V}$ verifies the proof of multiparty ownership by invoking the interactive algorithm **Proof** with the multiple users $\mathcal{U}_i$.

Figure 7.3: A PMOW System

probability that $\mathcal{A}$ wins is exactly $1/2$ since $CT^*$ contains no information about $b$. On the other hand, if **Enc** is not a random function, $\mathcal{A}$ can win with probability $\frac{1}{2} + \epsilon$. Then the simulator $\mathcal{S}$ can answer correctly with probability $\frac{1}{2} + \frac{\epsilon}{2}$. Thus the simulator $\mathcal{S}$ can break the CCA2 security of Cramer and Shoups encryption scheme with non-negligible probability $\frac{1}{2} + \frac{\epsilon}{2}$.

## 7.5 Complexity Analysis

In this section, a **PMOW** system is first given. Then the complexity of the proposed scheme is analyzed, in the aspects of communication, computation and storage costs.

### 7.5.1 A PMOW System

A **PMOW** system can be constructed as in Figure 7.3, from the proposed scheme in two phases: *Setup* and *Proof*. It should be emphasized that the cloud server only accesses the small verification information $v$, rather than the stored encrypted file when verifying the joint proof. Moreover, the cloud server does not retrieve any data during the proof executions.

### 7.5.2 Complexity Analysis

**Communication Costs.** On the client side, the communications only occur in Setup phase and the cost is $O(n)$ as the client needs to send the cloud server the ciphertexts of the symmetric key for all the users. The communications only occur

in Proof phase on both of verifier and user side, and the costs are $O(n)$ and $O(u)$ respectively.

**Storage Costs.** The additional storage costs which are brought by the proposed scheme are considered. The storage cost mainly occurs on the verifier side as all the ciphertexts are outsourced to cloud storage. Separate analysis of the storage cost is given on the client side, the verifier side and the user side.

- **Client side.** The client has to store the symmetric key for the file. In addition, he has to store the matching public and private keys for the multiple users. The total cost is $2|k| + 11n \cdot |p|$ bits ($k$ is the key length for VO-PRF and VI-PRF, e.g., $|k| = 128bit$).
- **Verifier side.** The cloud server has to store the verification information $v$, and all the public keys for the users. In addition, he has to store all the ciphertexts and the symmetric key. Denote $|M|$ as the total length of the outsourced file $F$. Then the cost is $|M| + 2|\lambda| + 10n \cdot |p|$ bits.
- **User side.** Every user needs to store the corresponding public keys, private keys and the file $F$. The total storage cost for $n$ users is $|M| + 11n \cdot |p|$ bits.

**Computation Costs.** Separate analysis of the computation costs is also given on the client side, the verifier side and the user side.

- **Client side.** The computation cost on the client side only occurs in Setup phase, for encrypting the input file $F$ and the symmetric key $k$ for the user set $\mathcal{I} \subset \{1, \cdots, n\}$. The computation cost of the client is $19nT_{exp}(|p|, p) + T^n_{hash} + T^{7n}_{multi} + T^2_{xor} + T^2_{prf}$, where $T_{exp}(len, num)$ is the time cost for computing a modular exponentiation with a $len$-bit long exponent modular $num$, $T^n_{hash}$ is the time cost of $n$ hashing operations, $T^n_{multi}$ is the time cost of $n$ multiplications, $T^n_{xor}$ is the time cost of $n$ xor operations and $T^n_{prf}$ is the time cost of $n$ pseudorandom functions (either $VO - PRF$ or $VI - PRF$). The time complexity on the client side is $O(n)$.
- **Verifier side.** The cloud server needs to generate the verification information $v$ and check whether the six equations hold. During the whole protocol execution, the cloud server has to do $(m - 1)$ hashing operations, $16n$ modular

exponentiations and $14n$ multiplications. The computation cost on verifier side is $16nT_{exp}(|p|, p) + T^(m-1)_{hash} + T^{14n}_{multi}$. The time complexity on the verifier side is $O(n)$.

- **User side.** The main computation cost on the user side occurs for generating the joint proof. For every user $i \subset \mathcal{I}$, he needs to compute $m - 1 + 1$ hash, nineteen multiplications, eight additions, fifteen modular exponentiations, two xor operations and two pseudorandom functions. Thus the computation cost on user side is $15T_{exp}(|p|, p) + T^m_{hash} + T^1 9_{multi} + T^8_{add} + T^2_{xor} + T^2_{prf}$ where $T^n_{add}$ is the time cost of $n$ additions. As $m = \lceil \frac{|R(CT_F)|}{\lambda} \rceil$, the time complexity on the user side is $O(1)$.

## 7.6 Conclusion

In this chapter, the solution of proof-of-multiparty-ownership (**PMOW**) is proposed. Multiple users can prove to a cloud server about their joint ownership, when they do not send the file to the server actually. Additionally, the cloud server stores one ciphertext only. In client-side deduplication, the proposed proposal can counter the attacks as follows: 1) Malicious users cheat on the cloud server about their joint ownership, when some users do not have the file, 2) the untrusted cloud server reveals the client's file, 3) when an attacker has a short information of the file, he fools the cloud server about his ownership. Two definitions for security are defined and CCA2 security is achieved under the ideal cipher model. The full proof analysis and complexity analysis of the proposed scheme are also presented.

# Chapter 8

# Conclusions

## 8.1 Summary of The Contributions

The contributions of this thesis can be briefly concluded as follows.

### 8.1.1 Key Management in Cloud Computing

Data encryption before outsourcing to the cloud is a common and simple way to protect data privacy. Although the encryption algorithms are public, information encrypted under these algorithms is secure because the key used to encrypt the data remains secret. As a result, key management is a critical element in cloud computing. It is the ability to correctly assign, secure and monitor keys that defines the level of operational security provided by any encryption implementation.

In Chapter 3, it is found that a traditional tree-based approach has some drawbacks. In a traditional tree-based key management hierarchy, a node key holder can derive all the child keys. In order to solve this problem and maintain the key management feature, in this chapter OWUR/W applications for data sourcing were proposed and a secure and flexible tree-based key derivation hierarchy was presented. The proposed tree-based key derivation hierarchy allowed the outsourcing party to access the data block located at a specified node, while not being able to access the data blocks encrypted with child keys. It is believed that the proposed tree-based outsourcing key management opens up an entirely new approach for secure and flexible key management.

### 8.1.2 Access Control in Cloud Computing

Unlike the traditional access control in which the data users and storage servers are in the same trusted domain, access control techniques are very different in cloud

computing as the cloud servers are not considered trustworthy by most cloud users, especially large enterprises and organizations. One possible method to enforce data access control without relying on cloud servers could be to encrypt data and disclose the corresponding decryption keys only to the privileged users, but that causes high performance costs. A fine-grained access control which is efficient and secure is necessary for cloud computing.

In Chapter 4, an encryption scheme for a two-tier system was presented to achieve flexible and fine-grained access control in the cloud. Most of the computation-intensive tasks were delegated to cloud servers without leaking private data. The security of the proposed scheme was also proven in the standard model.

### 8.1.3 Searchable Encryption Techniques in Cloud Computing

As the data is usually encrypted before being outsourced to cloud servers, how to search the encrypted data in the cloud has recently gained attention and led to the development of searchable encryption techniques. This problem is challenging, however, because meeting performance, system usability and scalability requirements is extremely difficult.

In Chapter 5, an efficient **SPEKS** was constructed. **SPEKS** is suitable for keyword search in the cloud environment. Compared with the existing keyword search schemes such as [LYCL11, WCRL12, ÖS12], the proposed construction is much more efficient from the point of view of both the data owner and the cloud servers. In addition, the security of the proposed scheme had been proved in the standard model.

### 8.1.4 Remote Integrity Check

Storing data in the remote cloud servers has become common. As the clients store their important data in remote cloud servers without a local copy, it is important to check remote data integrity (RIC). While it is easy to check data integrity after completely downloading the data, it is a large waste of communication bandwidth. Hence, designing efficient remote integrity check protocols without downloading the data is an important security issue in the cloud.

In Chapter 6, a privacy-preserving RIC protocol was proposed. The proposed

protocol achieved public verifiability without disclosing any information. It is ensured that no information about the original data would be leaked. In fact, the verifier was only required to know the public key of the data owner. The experimental results indicated that the proposed scheme is efficient especially when the size of the data file is large or the integrity check is frequent. The full proofs of security was also given under the random oracle model.

### 8.1.5 Proof of Ownership

Beyond storage integrity, proof of ownership (POW) is another security issue related to cloud data storage. Client-side deduplication allows an attacker to gain access to arbitrary-size files when he has small hash signatures of these files. To overcome such attacks, the technique of POW allows a user efficiently prove to a cloud server about his ownership, rather than some short information (i.e. a short hash value of the file).

In Chapter 7, an innovative **PMOW** scheme for proof of multiparty ownership with the encrypted data was proposed. Every user can prove to the server that he holds the plaintext of the encrypted file when the server stores one ciphertext only. The proposed solution achieved CCA2 security and the full proof analysis was given in the ideal cipher model.

## 8.2 Future Work

How to design efficient dynamic POR schemes is still an open problem. In addition, how to prevent malicious cloud users from abusing cloud resources is still a security issue (i.e., malicious data hosting or bonnet command and control). One way to solve this problem is to monitor the cloud usage more strictly, however this is inevitably in conflict with legal users' privacy rights. Further research is needed.

# Bibliography


[ABC+07] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Xiaodong Song. Provable data possession at untrusted sto/Users/hg/Library/Containers/com.apple.TextEdit/Data/Desktop/Thesis by Miao/MyThesis/Thesis.bblres. In *ACM Conference on Computer and Communications Security*, pages 598–609, 2007.

[ABC+11] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary N. J. Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Trans. Inf. Syst. Secur.*, 14:1–34, 2011.

[ABFF09] Mikhail J. Atallah, Marina Blanton, Nelly Fazio, and Keith B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3), 2009.

[AFB05] Mikhail J. Atallah, Keith B. Frikken, and Marina Blanton. Dynamic and efficient key management for access hierarchies. In *ACM Conference on Computer and Communications Security*, pages 190–202, 2005.

[AFGH05] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *IACR Cryptology ePrint Archive*, 2005:28, 2005.

[ASB+12] Abdulrahman Almutairi, Muhammad I. Sarfraz, Saleh Basalamah, Walid G. Aref, and Arif Ghafoor. A distributed access control architecture for cloud computing. *IEEE Software*, 29(2):36–44, 2012.

[ASM10] Man Ho Au, Willy Susilo, and Yi Mu. Proof-of-knowledge of representation of committed value and its applications. In *ACISP*, pages 352–369, 2010.

[BCdV+09] Carlo Blundo, Stelvio Cimato, Sabrina De Capitani di Vimercati, Alfredo De Santis, Sara Foresti, Stefano Paraboschi, and Pierangela Samarati. Efficient key management for enforcing access control in outsourced scenarios. In *SEC*, pages 364–375, 2009.

[BCOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004.

[BEG+94] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. *Algorithmica*, 12(2/3):225–244, 1994.

[BGG95] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography and application to virus protection. In *STOC*, pages 45–56, 1995.

[BJO09] Kevin D. Bowers, Ari Juels, and Alina Oprea. Hail: a high-availability and integrity layer for cloud storage. In *ACM Conference on Computer and Communications Security*, pages 187–198, 2009.

[BKOI07] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. Public key encryption that allows pir queries. In *CRYPTO*, pages 50–67, 2007.

[BKP09] Rakeshbabu Bobba, Himanshu Khurana, and Manoj Prabhakaran. Attribute-sets: A practically motivated enhancement to attribute-based encryption. In *ESORICS*, pages 587–604, 2009.

[BL96] Dan Boneh and Richard J. Lipton. Algorithms for black-box fields and their application to cryptography (extended abstract). In *CRYPTO*, pages 283–297, 1996.

[BLLS11] Mrinmoy Barua, Xiaohui Liang, Rongxing Lu, and Xuemin Shen. Espac: Enabling security and patient-centric access control for ehealth in cloud computing. *IJSN*, 6(2/3):67–76, 2011.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT*, pages 514–532, 2001.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptology*, 17(4):297–319, 2004.

[BP06] Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. In *CRYPTO*, pages 537–554, 2006.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR00] John Black and Phillip Rogaway. Cbc macs for arbitrary-length messages: The three-key constructions. In *CRYPTO*, pages 197–215, 2000.

[Bra93] Stefan Brands. An efficient off-line electronic cash system based on the representation problem. Technical report, CWI, 1993.

[BRPL06] Jin Wook Byun, Hyun Suk Rhee, Hyun-A Park, and Dong Hoon Lee. Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In *Secure Data Management*, pages 75–83, 2006.

[BSNS05] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Public key encryption with keyword search revisited. *IACR Cryptology ePrint Archive*, 2005:191, 2005.

[BSNS06] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. On the integration of public key data encryption and public key encryption with keyword search. In *ISC*, pages 217–232, 2006.

[BSS11] Atanu Basu, Indranil Sengupta, and Jamuna Kanta Sing. Secured cloud storage scheme using ecc based key management in user hierarchy. In *ICISS*, pages 175–189, 2011.

[BYV+09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Comp. Syst.*, 25(6):599–616, 2009.

[CC12] Bo Chen and Reza Curtmola. Robust dynamic provable data possession. In *ICDCS Workshops*, pages 515–525, 2012.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM Conference on Computer and Communications Security*, pages 79–88, 2006.

[CHR09] Danwei Chen, Xiuli Huang, and Xunyi Ren. Access control of cloud service based on ucon. In *CloudCom*, pages 559–564, 2009.

[CM05] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, pages 442–455, 2005.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, pages 13–25, 1998.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, pages 126–144, 2003.

[CSG+05] Dwaine E. Clarke, G. Edward Suh, Blaise Gassend, Ajay Sudan, Marten van Dijk, and Srinivas Devadas. Towards constant bandwidth overhead integrity checking of untrusted data. In *IEEE Symposium on Security and Privacy*, pages 139–153, 2005.

[CSK11] ByungRae Cha, JaeHyun Seo, and JongWon Kim. Design of attribute-based access control in cloud computing environment. In *ICITCS*, pages 41–50, 2011.

[CX08] Ee Chien Chang and Jia Xu. Remote integrity check with dishonest storage server. In *ESORICS*, pages 223–237, 2008.

[CY07] Koji Chida and Go Yamamoto. Batch processing of interactive proofs. In *CT-RSA*, pages 196–207, 2007.

[DAB+02] John R. Douceur, Atul Adya, William J. Bolosky, Dan Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *ICDCS*, pages 617–624, 2002.

[Dam87] Ivan Damgård. Collision free hash functions and public key signature schemes. In *EUROCRYPT*, pages 203–216, 1987.

[DdVF+05] Ernesto Damiani, Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Key management for multi-user encrypted databases. In *StorageSS*, pages 74–83, 2005.

[DdVF+07] Ernesto Damiani, Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. An experimental evaluation of multi-key strategies for data outsourcing. In *SEC*, pages 385–396, 2007.

[DdVJ+03] Ernesto Damiani, Sabrina De Capitani di Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. In *ACM Conference on Computer and Communications Security*, pages 93–102, 2003.

[Des00] Anand Desai. New paradigms for constructing symmetric encryption schemes secure against chosen-ciphertext attack. In *CRYPTO*, pages 394–412, 2000.

[DQS04] Yves Deswarte, Jean Jacques Quisquater, and Ayda Saidane. Remote integrity checking. In Sushil Jajodia and Leon Strous, editors, *Integrity and Internal Control in Information Systems VI*, volume 140 of *IFIP International Federation for Information Processing*, pages 1–11. Springer Boston, 2004.

[dVFJ+07a] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. A data outsourcing architecture combining cryptography and access control. In *CSAW*, pages 63–69, 2007.

[dVFJ+07b] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Over-encryption: Management

of access control evolution on outsourced data. In *VLDB*, pages 123–134, 2007.

[dVFJ+08] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, Gerardo Pelosi, and Pierangela Samarati. Preserving confidentiality of security policies in data outsourcing. In *WPES*, pages 75–84, 2008.

[DVW09] Yevgeniy Dodis, Salil P. Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *TCC*, pages 109–127, 2009.

[EKPT09] C. Christopher Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. In *ACM Conference on Computer and Communications Security*, pages 213–222, 2009.

[Erd09] Hakan Erdogmus. Cloud computing: Does nirvana hide behind the nebula? *IEEE Software*, 26(2):4–6, 2009.

[FB06] Décio Luiz Gazzoni Filho and Paulo Sérgio Licciardi Messeder Barreto. Demonstrating data possession and uncheatable data transfer. *IACR Cryptology ePrint Archive*, pages 150–159, 2006.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO*, pages 537–554, 1999.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GM09] Antonios Gouglidis and Ioannis Mavridis. On the definition of access control requirements for grid and cloud computing systems. In *GridNets*, pages 19–26, 2009.

[GMCL09] Luis Miguel Vaquero Gonzalez, Luis Rodero Merino, Juan Caceres, and Maik A. Lindner. A break in the clouds: towards a cloud definition. *Computer Communication Review*, 39(1):50–55, 2009.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, 1996.

[GSMB03] Eu-Jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. Sirius: Securing remote untrusted storage. In *NDSS*, 2003.

[HCM01] Hugh Harney, Andrea Colgrove, and Patrick Drew McDaniel. Principles of policy in secure groups. In *NDSS*, 2001.

[HHPSP11] Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Proofs of ownership in remote storage systems. In *ACM Conference on Computer and Communications Security*, pages 491–500, 2011.

[HPSP10] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security & Privacy*, 8(6):40–47, 2010.

[HYJZ09] Luokai Hu, Shi Ying, Xiangyang Jia, and Kai Zhao. Towards an approach of semantic access control for cloud computing. In *CloudCom*, pages 145–156, 2009.

[HZY11] Zhuo Hao, Sheng Zhong, and Nenghai Yu. A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability. *IEEE Trans. Knowl. Data Eng.*, 23(9):1432–1437, 2011.

[jG03] Eu jin Goh. Secure indexes. *IACR Technical Report 2003/216*, 2003.

[JJ07] Ari Juels and Burton S. Kaliski Jr. Pors: proofs of retrievability for large files. In *ACM Conference on Computer and Communications Security*, pages 584–597, 2007.

[KAD07] Ramakrishna Kotla, Lorenzo Alvisi, and Michael Dahlin. Safestore: A durable and practical storage system. In *USENIX Annual Technical Conference*, pages 129–142, 2007.

[KL10] Seny Kamara and Kristin Lauter. Cryptographic cloud storage. In *Financial Cryptography Workshops*, pages 136–149, 2010.

[KLKJ06] Deuk-Whee Kwak, Seungjoo Lee, JongWon Kim, and Eunjin Jung. An efficient key tree management algorithm for lkh group key management. In *ICOIN*, pages 703–712, 2006.

[KPT00] Yongdae Kim, Adrian Perrig, and Gene Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *ACM Conference on Computer and Communications Security*, pages 235–244, 2000.

[KRS+03] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus: Scalable secure file sharing on untrusted storage. In *FAST*, 2003.

[Lee12] Ruby B. Lee. Hardware-enhanced access control for cloud computing. In *SACMAT*, pages 1–2, 2012.

[LNS03] Donggang Liu, Peng Ning, and Kun Sun. Efficient self-healing group key distribution with revocation capability. In *ACM Conference on Computer and Communications Security*, pages 231–240, 2003.

[LWG11] Jun Liu, Zhiguo Wan, and Ming Gu. Hierarchical attribute-set based encryption for scalable, flexible and fine-grained access control in cloud computing. In *ISPEC*, pages 98–107, 2011.

[LYCL11] Ming Li, Shucheng Yu, Ning Cao, and Wenjing Lou. Authorized private keyword search over encrypted data in cloud computing. In *ICDCS*, pages 383–392, 2011.

[LYRL10] Ming Li, Shucheng Yu, Kui Ren, and Wenjing Lou. Securing personal health records in cloud computing: Patient-centric and fine-grained data access control in multi-owner settings. In *SecureComm*, pages 89–106, 2010.

[LZC+10] Jin Li, Gansen Zhao, Xiaofeng Chen, Dongqing Xie, Chunming Rong, Wenjun Li, Lianzhang Tang, and Yong Tang. Fine-grained data access control systems with user accountability in cloud computing. In *CloudCom*, pages 89–96, 2010.

[MG09] Peter Mell and Timothy Grance. The nist definition of cloud computing. Technical report, National Institute of Standards and Technology, Information Technology Laboratory, 2009.

[MVN99] Yi Mu, Vijay Varadharajan, and Khanh Quoc Nguyen. Delegated decryption. In *IMA Int. Conf.*, pages 258–269, 1999.

[NNL01a] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In *CRYPTO*, pages 41–62, 2001.

[NNL01b] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In *CRYPTO*, pages 41–62, 2001.

[NR05] Moni Naor and Guy N. Rothblum. The complexity of online memory checking. In *FOCS*, pages 573–584, 2005.

[NRR00] Moni Naor, Omer Reingold, and Alon Rosen. Pseudo-random functions and factoring (extended abstract). In *STOC*, pages 11–20, 2000.

[NWZ12] Wee Keong Ng, Yonggang Wen, and Huafei Zhu. Private data deduplication protocols in cloud storage. In *SAC*, pages 441–446, 2012.

[OLR12] Nouha Oualha, Jean Leneutre, and Yves Roudier. Verifying remote data integrity in peer-to-peer data storage: A comprehensive survey of protocols. *Peer-to-Peer Networking and Applications*, 5(3):231–243, 2012.

[OR05] Alina Oprea and Michael K. Reiter. Space-efficient block storage integrity. In *NDSS*, 2005.

[ÖS12] Cengiz Örencik and Erkay Savas. Efficient and secure ranked multi-keyword search on encrypted cloud data. In *EDBT/ICDT Workshops*, pages 186–195, 2012.

[PL01] Chang-Seop Park and Dong Hoon Lee. Secure and efficient key management for dynamic multicast groups. *Operating Systems Review*, 35(4):32–38, 2001.

[Pol78] J. M. Pollard. Monte carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.

[PR00] Erez Petrank and Charles Rackoff. Cbc mac for real-time data sources. *J. Cryptology*, 13(3):315–338, 2000.

[PS12] Roberto Di Pietro and Alessandro Sorniotti. Boosting efficiency and security in proof of ownership for deduplication. In *ASIACCS*, 2012.

[RMW12] Fatema Rashid, Ali Miri, and Isaac Woungang. A secure data deduplication framework for cloud environments. In *PST*, pages 81–87, 2012.

[RPSL09] Hyun Sook Rhee, Jong Hwan Park, Willy Susilo, and Dong Hoon Lee. Improved searchable public key encryption with designated tester. In *ASIACCS*, pages 376–379, 2009.

[RPSL10] Hyun Sook Rhee, Jong Hwan Park, Willy Susilo, and Dong Hoon Lee. Trapdoor security in a searchable public-key encryption scheme with a designated tester. *Journal of Systems and Software*, 83(5):763–771, 2010.

[RTSS09] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *ACM Conference on Computer and Communications Security*, pages 199–212, 2009.

[RVR+07] Barath Raghavan, Kashi Venkatesh Vishwanath, Sriram Ramabhadran, Ken Yocum, and Alex C. Snoeren. Cloud control with distributed rate limiting. In *SIGCOMM*, pages 337–348, 2007.

[SBD+04] F. Sebe, A. M. Balleste, Y. Deswarte, J. D. Ferrer, and J.J.Quisquater. Time-bounded remote file integrity checking. Technical report, LAAS, 2004.

[SBMS07] Mehul A. Shah, Mary Baker, Jeffrey C. Mogul, and Ram Swaminathan. Auditing to keep online storage services honest. In *HotOS*, 2007.

[SFB+08] Francesc Sebé, Josep Domingo Ferrer, Antoni Martínez Ballesté, Yves Deswarte, and Jean Jacques Quisquater. Efficient remote data possession checking in critical information infrastructures. *IEEE Trans. Knowl. Data Eng.*, 20(8):1034–1038, 2008.

[SGLM08] Mark W. Storer, Kevin M. Greenan, Darrell D. E. Long, and Ethan L. Miller. Secure data deduplication. In *StorageSS*, pages 1–10, 2008.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004.

[SL04] Yan Sun and K. J. Ray Liu. Scalable hierarchical access control in secure group communications. In *INFOCOM*, 2004.

[SM06] Thomas J. E. Schwarz and Ethan L. Miller. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *ICDCS*, pages 12–22, 2006.

[SvDOJ11] Emil Stefanov, Marten van Dijk, Alina Oprea, and Ari Juels. Iris: A scalable cloud file system with efficient integrity checks. *IACR Cryptology ePrint Archive*, 2011:585, 2011.

[SW08] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *ASIACRYPT*, pages 90–107, 2008.

[SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[SWYW12] Lili Sun, Hua Wang, Jianming Yong, and Guoxin Wu. Semantic access control for cloud computing based on e-healthcare. In *CSCWD*, pages 512–518, 2012.

[TS11] Wei-Tek Tsai and Qihong Shao. Role-based access-control using reference ontology in clouds. In *ISADS*, pages 121–128, 2011.

[TWZ09] XiuXia Tian, Xiaoling Wang, and Aoying Zhou. Dsp re-encryption: A flexible mechanism for access control enforcement management in daas. In *IEEE CLOUD*, pages 25–32, 2009.

[WCRL12] Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE Trans. Parallel Distrib. Syst.*, 23(8):1467–1479, 2012.

[WCW+09] Cong Wang, Sherman S. M. Chow, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for secure cloud storage. *IACR Cryptology ePrint Archive*, pages 579–590, 2009.

[WeLD12] Zhiguo Wan, Jun e Liu, and Robert H. Deng. Hasbe: A hierarchical attribute-based solution for flexible and scalable access control in cloud computing. *IEEE Transactions on Information Forensics and Security*, 7(2):743–754, 2012.

[WGL98] Chung Kei Wong, Mohamed G. Gouda, and Simon S. Lam. Secure group communications using key graphs. In *SIGCOMM*, pages 68–79, 1998.

[WHA99] D. Waller, E. Harder, and R. Agee. Key management for multicast: Issues and architectures. Technical report, RFC2627, 1999.

[WLOB09] Weichao Wang, Zhiwei Li, Rodney Owens, and Bharat K. Bhargava. Secure and efficient access to outsourced data. In *CCSW*, pages 55–66, 2009.

[WLW10] Guojun Wang, Qin Liu, and Jie Wu. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *ACM Conference on Computer and Communications Security*, pages 735–737, 2010.

[WWL+09] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *ESORICS*, pages 355–370, 2009.

[WWR+11] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou, and Jin Li. Enabling public auditability and data dynamics for storage security in cloud computing. *IEEE Trans. Parallel Distrib. Syst.*, 22(5):847–859, 2011.

[WWRL10] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *INFOCOM*, pages 525–533, 2010.

[XC12] Jia Xu and Ee-Chien Chang. Towards efficient proofs of retrievability. In *ASIACCS*, pages 79–80, 2012.

[XCZ13] Jia Xu, Ee-Chien Chang, and Jianying Zhou. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. In *ASIACCS*, pages 195–206, 2013.

[YC10] Jong P. Yoon and Zhixiong Chen. Using privilege chain for access control and trustiness of resources in cloud computing. In *NDT (1)*, pages 358–368, 2010.

[YLJ12] Ran Yang, Chuang Lin, and Yixin Jiang. Enforcing scalable and dynamic hierarchical access control in cloud computing. In *ICC*, pages 923–927, 2012.

[YLZL01] Yang Richard Yang, Xiaozhou (Steve) Li, X. Brian Zhang, and Simon S. Lam. Reliable group rekeying: a performance analysis. In *SIGCOMM*, pages 27–38, 2001.

[YOA07] Go Yamamoto, Satoshi Oda, and Kazumaro Aoki. Fast integrity for large data. In *SPEED*, pages 21–32, 2007.

[YWRL10] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *INFOCOM*, pages 534–542, 2010.

[ZB12] Yihua Zhang and Marina Blanton. Efficient dynamic provable possession of remote data via update trees. *IACR Cryptology ePrint Archive*, 2012:291, 2012.

[Zen08] Ke Zeng. Publicly verifiable remote data integrity. In *ICICS*, pages 419–434, 2008.

[ZHA+12] Yan Zhu, Hongxin Hu, Gail-Joon Ahn, Dijiang Huang, and Shan-Biao Wang. Towards temporal access control in cloud computing. In *INFOCOM*, pages 2576–2580, 2012.

[ZLS11] Tianyi Zhu, Weidong Liu, and Jiaxing Song. An efficient role based access control system for cloud computing. In *CIT*, pages 97–102, 2011.

[ZMS+12] Miao Zhou, Yi Mu, Willy Susilo, Jun Yan, and Liju Dong. Privacy enhanced data outsourcing in the cloud. *J. Network and Computer Applications*, 35(4):1367–1373, 2012.

[ZMSY11] Miao Zhou, Yi Mu, Willy Susilo, and Jun Yan. Piracy-preserved access control for cloud computing. In *IEEE TrustCom*, pages 83–90, 2011.

[ZX11] Qingji Zheng and Shouhuai Xu. Fair and dynamic proofs of retrievability. In *Proceedings of the 1st ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 237–248, 2011.

[ZX12] Qingji Zheng and Shouhuai Xu. Secure and efficient proof of storage with deduplication. In *Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 1–12, 2012.