2016

Secure data storage and retrieval in cloud computing

Rongmao Chen
University of Wollongong

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Secure Data Storage and Retrieval in Cloud Computing

Rongmao Chen

Supervisor:
Professor Yi Mu
Co-supervisor:
Dr. Guomin Yang

This thesis is presented as part of the requirements for the conferral of the degree:

Doctor of Philosophy

The University of Wollongong
School of Computing and Information Technology

June 27, 2016

Abstract

Nowadays cloud computing has been widely recognised as one of the most influential information technologies because of its unprecedented advantages. In spite of its widely recognised social and economic benefits, in cloud computing customers lose the direct control of their data and completely rely on the cloud to manage their data and computation, which raises significant security and privacy concerns and is one of the major barriers to the adoption of public cloud by many organisations and individuals. Therefore, it is desirable to apply practical security approaches to address the security risks for the wide adoption of cloud computing.

In this thesis, we carry out the study on the secure data storage and retrieval in cloud computing. Data storage outsourcing is one of the important cloud applications where both individuals and enterprises can store their data remotely on the cloud to relieve the storage management burden. Aside from eliminating the local storage management, storing data into the cloud requires that the data can be efficiently and securely retrieved for flexible utilization. To provide strong security guarantees for data storage and retrieval, in this thesis, we have made the following contributions.

Firstly, we give a formal treatment on Merkle Hash Tree for secure dynamic cloud auditing. We first revisit a well-known authentication structure named Merkle Hash Tree (MHT) and demonstrate how to extend its basic version to a sequence-enforced version that allows position checking. In order to support efficient and verifiable dynamic data operations, we further propose a variant of MHT, named rank-based MHT (rMHT) that can be used to support verifiable dynamic data auditing. We also review a data auditing protocol named Oruta and showed that Oruta is vulnerable to replace and replay attacks. We then employ the proposed rMHT to fix the security problems in Oruta without sacrificing any desirable features of the protocol.

Secondly, we formalize a new primitive called Block-Level Message-Locked Encryption (BL-MLE) for deduplication of large files to achieve space-efficient storage in cloud. We also present a concrete BL-MLE scheme that can efficiently realize our design ideas. We demonstrate that our proposed scheme can achieve significant savings in space and bandwidth. Moreover, we show that our BL-MLE scheme can be easily extended to achieve efficient data auditing, which makes our scheme multi-purpose for secure cloud storage.

Thirdly, we propose two different solutions towards an inherent vulnerability of the conventional Public Key Encryption with Keyword Search (PEKS) systems

under inside keyword guessing attack (KGA). The first solution is named Dual-Server Public Key Encryption with Keyword Search (DS-PEKS). We introduce a new Smooth Projective Hash Function (SPHF) that enables a generic DS-PEKS construction. An efficient instantiation is also given to illustrate the feasibility of the generic construction. As the second solution, we provide a more practical treatment on this security issue by formalizing a new PEKS system named Server-Aided Public Key Encryption with Keyword Search (SA-PEKS). We introduce a universal transformation from any conventional PEKS scheme to a secure SA-PEKS scheme, along with the first concrete SA-PEKS scheme. Moreover, we also show how to securely implement the client-KS (keyword server) protocol with a rate-limiting mechanism against on-line KGA.

Lastly, we introduce a new leakage-resilient security model for authenticated key exchange protocols to achieve strongly secure data transmission in cloud computing. Our model is the first to allow the adversary to obtain challenge-dependent leakage on both long-term and ephemeral secret keys, and hence are strong yet meaningful compared with the previous models. We also present a generic framework to construct efficient one-round AKE protocol that is secure under the proposed security model, as well as an efficient instantiation of the general framework under a standard assumption.

Acknowledgement

The research work presented in this thesis would never be possible without the support of many individuals.

First of all, I would like to express my sincere gratitude to my supervisor, Professor *Yi Mu*. Professor *Mu* is an excellent supervisor who has offered me directions yet freedom for me to explore different areas. Many thanks to him for the continuous support of my research, for his guidance, patience, motivation and immense knowledge. I would like to thank my co-supervisor, Dr. *Guomin Yang*, who had a tremendous influence on my research as well. Dr. *Yang* provided me with valuable comments on many of my works. All the discussions with him are very helpful in the development of this thesis.

Dr. *Fuchun Guo* deserves special appreciations. I had many interesting discussions with Dr. *Guo*. I really appreciate his valuable training that helped me build up my knowledge in cryptography. I would like to also give my special thanks to Professor *Willy Susilo* for his indispensable contribution to our joint research and his helpful advice.

I also thank *Man Ho Au, Xinyi Huang, Jinguang Han, Qiong Huang, Xiaofen Wang, Yong Yu, Mingwu Zhang,* and *Futai Zhang* for their invaluable advice on my research. Discussions with them are always stimulating and rewarding. I am fortunate to have great colleagues and friends during my PhD study in Wollongong. Many thanks for their support and kindness. The non-exhausted list includes *Hui Cui, Zhimin Gao, Clementine Gritti, Jongkil Kim, Peng Jiang, Yinhao Jiang, Jianchang Lai, Nan Li, Weiwei Liu, Tran Viet Xuan Phuong, Shams Ud Din Qazi, Fatemeh Rezaeibagha, Jean-Marc Robert, Yangguang Tian, Yang Wang, Kefeng Wang, Huai Wu, Jiannan Wei, Shengmin Xu, Zhenfei Zhang, Siwei Zhang, Zhongyuan Yao, Hongyun Zhang, Jianjia Zhang, Yan Zhao*, etc. Even assuming that I did not forget people to whom that I want to express my appreciations, the list of people I am indebted to is far more extensive. It certainly includes all the authors of many papers appeared in the reference list. It also includes the authors of many papers that I have read during my PhD study.

Finally, my love and gratitude to my parents *Shuilin, Tiantu* and my sister *Xiaoping* for their patience, encouragement and love. Without them, all my achievements would never be possible.

[This page is intentionally left blank]

Publications

This thesis is based on the following presented or published papers, which were finished when I was in pursuit of the PhD degree in University of Wollongong.

1. Rongmao Chen, Yi Mu, Guomin Yang and Fuchun Guo. BL-MLE: Block-Level Message-Locked Encryption for Secure Large File Deduplication. *IEEE Transactions on Information Forensics and Security*, 10(12): 2643-2652, 2015.

2. Rongmao Chen, Yi Mu, Guomin Yang, Fuchun Guo and Xiaofen Wang. A New General Framework for Secure Public Key Encryption with Keyword Search. *Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings*, 59-76, 2015.

3. Rongmao Chen, Yi Mu, Guomin Yang, Fuchun Guo and Xiaofen Wang. Dual-Server Public-Key Encryption with Keyword Search for Secure Cloud Storage. *IEEE Transactions on Information Forensics and Security*, 11(4): 789-798, 2016.

4. Rongmao Chen, Yi Mu, Guomin Yang, Fuchun Guo, Xinyi Huang and Xiaofen Wang. Server-Aided Public Key Encryption with Keyword Search. *IEEE Transactions on Information Forensics and Security*, accepted on 17 DEC 2015.

5. Rongmao Chen, Yi Mu, Guomin Yang, Willy Susilo and Fuchun Guo. Strongly Leakage-Resilient Authenticated Key Exchange. *CT-RSA 2016*, 19-36.

Papers which are under review:

6. Rongmao Chen, Yi Mu, Guomin Yang, Fuchun Guo,Yong Yu and Xinyi Huang. A Formal Treatment on Merkle Hash Tree for Secure Dynamic Cloud Auditing. (under review)

I am thankful to have opportunities to collaborate with others in other areas of computer and communications security. The contributions are listed below and they are beyond the scope of this thesis.

1. Rongmao Chen, Yi Mu, Guomin Yang, Willy Susilo and Fuchun Guo. Strong Authenticated Key Exchange with Auxiliary Inputs. (under review)

2. Rongmao Chen, Yi Mu, Willy Susilo, Guomin Yang, Fuchun Guo and Mingwu Zhang. One-Round Strong Oblivious Signature-Based Envelope. *21st Australasian Conference on Information Security and Privacy (ACISP)*, 2016: accepted.

3. Jianchang Lai, Yi Mu, Fuchun Guo Willy Susilo and Rongmao Chen. Anonymous Identity-Based Broadcast Encryption with Revocation for File Sharing. *21st Australasian Conference on Information Security and Privacy (ACISP)*, 2016: accepted.

4. Willy Susilo, Rongmao Chen, Fuchun Guo, Guoming Yang, Yi Mu and and Yang-Wai Chow. Recipient Revocable Identity-Based Broadcast Encryption: How to Revoke Some Recipients in IBBE without Knowledge of the Plaintext. *11th ACM Asia Conference on Computer and Communications Security*, AsiaCCS 2016: 201-210.

5. Xiaofen Wang, Yi Mu, Rongmao Chen. One-Round Privacy-Preserving Meeting Location Determination for Smartphone Applications. *IEEE Transactions on Information Forensics and Security*, 11(8): 1712-1721, 2016.

6. Xiaofen Wang, Yi Mu, Rongmao Chen, Guozhen Xiao. Secure Channel Free ID-Based Searchable Encryption for a Peer-to-Peer Group. *Journal of Computer and Science Technology*, accepted on 26 JAN. 2016.

7. Xiaofen Wang, Yi Mu, Rongmao Chen. Privacy-Preserving Data Search and Sharing Protocol for Social Networks through Wireless Applications. *Concurrency and Computation: Practice and Experience*, accepted on 11 Apr. 2016.

List of Notations

The following notations are used throughout this thesis. Some special notations will be defined when they are first used.

ℓ	A security parameter;
1^ℓ	The string of ℓ ones;
$\forall$	For all;
$\exists$	There exists;
$\mathbb{Z}$	The set of integers;
$\mathbb{Z}_p$	The set consists of the integers modulo p;
$\mathbb{Z}_p^*$	The multiple group of integers modulo p;
$\epsilon(\ell)$	A negligible function on ℓ;
$a\|\|b$	The concatenation of the string a and the string b;
$\Pr[\mathbf{A}]$	The probability of the event $\mathbf{A}$ occurring;
$a \xleftarrow{\$} \mathbf{A}$	a is selected from $\mathbf{A}$ uniformly at random if $\mathbf{A}$ is a finite set;
$a \xleftarrow{R} \mathbf{A}$	a is selected from $\mathbf{A}$ randomly if $\mathbf{A}$ is a finite set;
$X \overset{s}{\equiv} Y$	Distributions X and Y are perfectly indistinguishable.
$X \overset{c}{\equiv} Y$	Distributions X and Y are computationally indistinguishable.

List of Abbreviations

The following abbreviations are used throughout this thesis. Some special abbreviations will be defined when they are first used.

CSP	Cloud Service Provider;
PKC	Public-Key Cryptography;
PPT	Probabilistic Polynomial Time;
DL	Discrete Logarithm;
CDH	Computational Diffie-Hellman;
DDH	Decisional Diffie-Hellman;
PKE	Public Key Encryption;
IND-CCA2	Indistinguishability against Adaptive Chosen Ciphertext Attacks;
IND-CPA	Indistinguishability against Adaptive Chose Plaintext Attacks;
EU-CMA	Existentially Unforgeable under Chosen-message Attacks;
SEU-CMA	Strongly Existentially Unforgeable under Chosen-message Attacks;
MHT	Merkle Hash Tree;
rMHT	Rank-based Merkle Hash Tree;
MLE	Message-Locked Encryption;
PEKS	Public Key Encryption with Keyword Search;
SPHF	Smooth Projective Hash Function;
PoR	Proof of Retrievability;
PDP	Provable Data Possession;
AKE	Authenticated Key Exchange;

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Cloud computing has been widely recognised as one of the most influential information technologies because of its unprecedented advantages. With resource virtualization, the cloud can provide on-demand self-service, ubiquitous network access, rapid resource elasticity and usage-based pricing, which thus make cloud services as convenient as the daily-life utilities such as electricity, water, and gas. In spite of its widely recognised economic benefits, cloud computing removes customers' direct control over the systems that manage their data, which raises significant security and privacy concerns and is one of the major barriers to its adoption. To well address the security risks, it is desirable to apply cryptographic approaches to achieve the security goals as cryptographic primitives can provide the vital security properties required by a cloud system. Nevertheless, cryptographic approaches introduce additional costs (e.g., computation overhead and storage overhead) for the cloud system and thus might significantly reduce the economic benefits of the cloud. Therefore, it is a critical challenge to propose practical cryptographic approaches for providing security guarantees in cloud computing.

1.1 Background

The advancement in networking technologies and ever-growing need for computing resources have promoted a fundamental paradigm shift in how people deploy and deliver computing services: computing outsourcing. This new computing model, commonly referred to as Cloud Computing, is a service model that offers customers on-demand network access to a large shared pool of computing resources, which is provided by a cloud service provider (CSP). Cloud computing consists of various types of services. The first type is Infrastructure as a Service (IaaS), where a customer makes use of the CSP's computing, storage or networking infrastructure. Another type is Platform as a Service (PaaS), where customers leverage the resources from the CSP to run custom applications. The last one is Software as a Service (SaaS), where customers use software that is run on the CSP's infrastructure.

Cloud infrastructures can be categorized as either private or public. Private cloud means that the infrastructure is implemented locally and under the control of the customer while the infrastructure in a public cloud is owned and managed remotely by a CSP and hence outside the customer's control. By now the economic benefits of cloud computing have been widely recognized, especially by using the

public cloud. Moving data and computing services to the public cloud alike infrastructures promises to provide unprecedented benefits like ubiquitous network access, rapid resource elasticity, minimal management overhead, etc. For most customers, the ability to utilize and pay for resources on demand is also the strong incentive for migration to the public cloud.

In spite of the benefits mentioned above, the public cloud deprives customers' direct control over the systems that manage their data and applications [YCN+10, ZL12, KK05], raising significant security and privacy risks. On one hand, although the cloud infrastructures are much powerful and reliable than personal devices, a broad range of both internal and external threats still exist, including hardware failures, software bugs, power outages, server misconfiguration, etc. On the other hand, there exist incentives for the CSP to behave unfaithfully towards the customers to increase the profit margin by reducing cost. For example, the server may have the incentive to reduce the cost by shifting users' data to a slower but cheaper storage or even deliberately deleting some data blocks that are seldom or never used. Moreover, The CSP may even attempt to hide service failure incidents so as to maintain a reputation. It may not store the files correctly due to various reasons such as system crash [BGPS07, Mil10] but intends to cover data loss events for reputation consideration.

To date, although individual customers have been willing to trade privacy for the convenience of services (e.g., Dropbox [Dro] and Google Drive [Goo]), this is not the case for enterprises and government organizations where the data is mission-critical (e.g., medical records, financial records or high impact business data). So even though the public cloud has enormous promise, many potential customers will be reluctant to outsource their data unless the issues of security and privacy are well addressed.

Therefore, in order to improve the adoption of cloud computing, it is desirable for the public cloud to provide security guarantees to customers' data. By now cryptographic approaches have been proposed as solutions for achieving the security goals in cloud computing. The reason is that compared to legal mechanisms or physical security, cryptographic approaches can provide customers with a certain degree of control on their outsourced data and also the security guarantees provided by cryptography. More specifically, cryptographic primitives provide the vital security properties of a secure system, namely, confidentiality, integrity, authenticity, and non-repudiation.

1. Confidentiality ensures that the information is inaccessible to unauthorized users or systems.

2. Integrity means the data in transit or in storage can not be modified by unau-

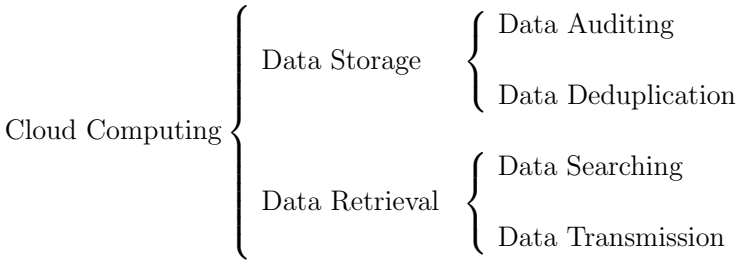


Figure 1.1: Problem Description

thorized users or systems.

3. Authenticity is presented to ensure that parties involved in a communication are really what they claim to be.

4. Non-repudiation refers to one user can not deny at a later stage his previous operation on any data.

1.2 Problem Statement

Although cryptographic approaches can achieve the security goals for the cloud system, it might significantly reduce the efficiency of the cloud system and hence makes deployment of traditional data utilization service difficult. For example, the traditional encryption of data in the cloud makes it inefficient to exploit the data redundancy when the server performs deduplication to save storage space. Moreover, the encrypted data cannot be searched in the traditional way due to the protection of the data privacy and hence results in additional costs for the user and the server.

Therefore, it is desirable to build cryptographic approaches to achieve the security goals without introducing significant overhead for the cloud system. In this thesis, we mainly focus on the data storage and retrieval in cloud computing. To give a clearer picture, we describe the problems in Figure 1.1.

1.2.1 Data Storage in Cloud Computing

According to the analysis of the International Data Corporation (IDC), the volume of data in the world will reach 40 trillion gigabytes in 2020 [GD12]. One of the representative cloud applications is data storage outsourcing [Dro, Goo, Bit] where both individuals and enterprises store their data remotely on the cloud to relieve the storage management burden and achieve much more flexible data utilization. Regarding this typical application, we investigate the studies on the following two issues.

Data Auditing. Despite the appealing benefits brought by cloud storage (and cloud computing in general), it is still debatable for its widespread adoption. The main reason is the data security concern as users no longer physically possess their data [YCN+10,ZL12,KK05]. Specifically, the server may have the incentive to reduce the cost by shifting users' data to a slower but cheaper storage or even deliberately deleting some data blocks that are seldom or never used. Moreover, the server may not store the files correctly due to various reasons such as system crash [BGPS07, Mil10] but intends to cover data loss events for reputation consideration. Hence, although data storage outsourcing is economically attractive for long-term large-scale data storage management, the widespread adoption of the cloud storage can be impeded without the guarantees on data integrity and availability. Therefore, a cloud storage server should be able to offer a customer the guarantee that it is indeed storing all of the customer's data which is available. Conventional cryptographic primitives cannot be adopted directly for remote data integrity checking as users lose the direct control of their data, and downloading all the data for integrity checking is not practical due to the high bandwidth cost. In order to enable the server to provide the users with such guarantee, there have been various proposed data auditing protocols that allow a verifier (customer or the third party) to verify that a server indeed stores the file correctly.

Data Deduplication. To achieve optimal usage of storage resources, many CSPs perform deduplication, which exploits data redundancy and avoids storing duplicated data from multiple users. It has been reported that deduplication can achieve up to 90-95% and 68% savings on storage space for backup systems [Ope] and standard file systems [MB12], respectively. Cost savings make deduplication widely used by a number of commercial cloud storage services such as Bitcasa [Bit], Dropbox [Dro] and Google Drive [Goo]. However, as mentioned above, customers may not entirely trust the CSP in the reality. In order to protect data privacy, files may be encrypted first before being uploaded to the server. This brings a new challenge for deduplication since different users will use different keys to perform encryption, which would make two identical files completely different after being encrypted. Although searchable encryption [BBO07, BCOP04, SWP00, YTHW10] can support equality testing of encrypted data, cloud storage providers still cannot perform any deduplication. The reason is that if a user (say Bob) does not store his encrypted file on the server due to deduplication, e.g., another user Alice has stored the same file (also in encrypted form) in the server, then Bob could not retrieve the original file later since he cannot decrypt Alice's file. Therefore, new cryptographic primitives should be proposed for efficient and secure deduplication for encrypted data in cloud storage.

1.2.2 Data Retrieval in Cloud Computing

Aside from eliminating the local storage management, storing data into the cloud serves no purpose unless they can be efficiently and securely retrieved for utilization. In this thesis, we also aim at the secure data searching and transmission for cloud data retrieval.

Data Searching. In reality, end users may not entirely trust the CSP and may prefer to encrypt their data before uploading them to the cloud server in order to protect the data privacy. This usually makes the data utilization more difficult than the traditional storage where data is kept in the absence of encryption. One trivial solution is downloading all the data and decrypting locally. Nevertheless, this solution might cause lots of transmission costs and storage space whenever users query data and thus is clearly impractical. Another typical solution is to set up keywords for each encrypted document and a user can search the encrypted documents with specific keywords they wish to query. However, the pre-set keywords result in storage overhead and privacy issues. On one hand, apart from the encrypted data, the cloud has to store the keywords and hence cost additional space. On the other hand, the keywords attached to the encrypted data might leak information about the corresponding encrypted data and destroy the data privacy. Therefore, the keywords should be encrypted to protect their privacy. Another privacy issue is the query privacy. That is when the user searches the data in the cloud, the query should not leak any information to the cloud or other untrusted parties either. Moreover, the response to a search query should not have a long delay, i.e., the computation overhead of the server while matching encrypted data for the query should be low otherwise the CSP cannot provide high-quality service which can reduce its attraction to the customers. This necessitates the need for developing efficient and secure searching techniques over encrypted cloud data of massive scale. Such techniques should enable critical search functionalities that have long been enjoyed in modern search engine over unencrypted data, like Google, Bing, etc. The adequacy of such techniques is essential to the long-term success of the cloud services and the ultimate privacy protection of both individuals and organizations.

Data Transmission. In cloud computing, the data owner and cloud servers are very likely to be in two different domains. Especially, the data resources are not physically under the full control of the owner. Therefore, the CSP needs to transmit the data through the network to the user upon the data retrieval request. In the real application, data transmission is usually through a public network since it is much cheaper and more convenient than using the private channel which offers better security guarantees. However, the communication channel in practice over a public network can be easily attacked by a malicious attacker and hence is insecure

by default for message transmission. An AKE protocol enables a secure channel to be established among a set of communicating parties by first allowing them to agree on a cryptographically strong secret key, and then applying efficient symmetric key tools to ensure the data confidentiality and authenticity. Many practical AKE protocols such as the ISO protocol (a.k.a. SIG-DH) [ISO, CK01] and the Internet Key Exchange protocol (a.k.a. SIGMA) [Kra03] have been proposed and deployed for secure network communication. However, it has been shown that AKE protocols proven secure in the traditional model could be completely insecure in the presence of side-channel attacks (a.k.a. leakage attacks). For example, an attacker can launch a memory attack [HSH+08, AGV09] to learn partial information about the static secret key, and also obtain partial information about the chosen randomness (e.g., via poorly implemented PRNGs [Mar,SF,Zet]). Therefore, designing AKE protocols, which are secure against various types of leakage attacks, is of practical significance.

1.3 Contributions

The contributions of this thesis can be summarized as follows.

1. We give a formal treatment on Merkle Hash Tree for secure dynamic cloud auditing. We first revisit a well-known authentication structure named Merkle Hash Tree (MHT) and demonstrate how to extend its basic version to a sequence-enforced version that allows position checking. In order to support efficient and verifiable dynamic data operations, we further propose a variant of MHT, named rank-based MHT (rMHT) that can be used to support verifiable dynamic data auditing. We also review a cloud storage data auditing protocol named Oruta and showed that the protocol is vulnerable to replace and replay attacks. We then employ the proposed rMHT to fix the security problems in Oruta without sacrificing any desirable features of the protocol. It is of independent interest to find other security applications for rMHT.

2. We formalize a new primitive called Block-Level Message-Locked Encryption for deduplication of large files to achieve space-efficient storage in the cloud. We also present a concrete BL-MLE scheme that can efficiently realize our design ideas. We show that our proposed scheme can achieve significant savings in space and bandwidth. Moreover, we show that our BL-MLE scheme can be easily modified to achieve efficient data auditing, which makes our scheme multi-purpose for secure cloud storage.

3. We propose two different solutions towards the inherent vulnerability the conventional Public Key Encryption with Keyword Search (PEKS) systems under

inside keyword guessing attack (KGA). The first solution is a new framework named Dual-Server Public Key Encryption with Keyword Search (DS-PEKS). A new Smooth Projective Hash Function (SPHF) is introduced and used to construct a generic DS-PEKS scheme. We also show an efficient instantiation of the new SPHF based on the Diffie-Hellman problem, which gives an efficient DS-PEKS scheme without pairings. As the second solution, we provide a practical treatment on this security issue by formalizing a new PEKS system named Server-Aided Public Key Encryption with Keyword Search (SA-PEKS). We introduce a universal transformation from any PEKS scheme to a secure SA-PEKS scheme, along with the first instantiation of the SA-PEKS scheme. We also show how to securely implement the client-KS (keyword server) protocol with a rate-limiting mechanism against on-line KGA. The experimental results show that our proposed scheme achieves much better efficiency while providing resistance against both off-line and on-line KGAs.

4. We introduce a new leakage-resilient security model for Authenticated Key Exchange (AKE) protocols to achieve strongly secure data transmission in cloud computing. Our model is the first to allow the adversary to obtain challenge-dependent leakage on both long-term and ephemeral secret keys, and hence are strong yet meaningful compared with the previous models. We also present a generic framework to construct efficient one-round AKE protocol that is secure under the proposed security model, as well as an efficient instantiation of the general framework under the DDH assumption. Our framework ensures the session key is private and authentic even if the adversary learns a large fraction of both the long-term secret key and ephemeral secret key and provides qualitatively stronger privacy guarantees than existing AKE protocols constructed in prior and concurrent works, since such protocols necessarily become insecure if the adversary can perform leakage attacks during the execution of the target session.

1.4 Thesis Organization

The remainder of this thesis is organized as follows.

In Chapter 2, we introduce the preliminaries that will be used throughout this thesis, including miscellaneous notations, foundations of algebra, complexity assumptions and cryptographic tools. The aim of this chapter is to make this thesis self-contained.

In Chapter 3, we give a formal treatment on a well-known data structure named, Merkle Hash Tree (MHT), which has been widely used in data auditing protocol.

We introduce a new *rank-based Merkle Hash Tree* that can support verifiable dynamic data operations. We then employ the proposed rank-based MHT to fix the security flaws (more precisely, replace attack and replay attack) in a recently proposed dynamic data auditing protocol named Oruta [WLL14] without sacrificing any desirable features of the protocol.

In Chapter 4, we propose a new approach to achieving more efficient deduplication for (encrypted) large files in the cloud storage. Our then show that our approach, named *Block-Level Message-Locked Encryption* (BL-MLE), can achieve file-level and block-level deduplication, block key management, and proof of ownership simultaneously using a small set of metadata.

In Chapter 5, we investigate the security of a well-known cryptographic primitive, namely Public Key Encryption with Keyword Search (PEKS). We propose a new PEKS framework named Dual-Server Public Key Encryption with Keyword Search (DS-PEKS) to address the inherent insecurity of traditional PEKS systems. We define a new variant of the Smooth Projective Hash Functions (SPHFs) referred to as linear and homomorphic SPHF (LH-SPHF) and show a generic construction of secure DS-PEKS from LH-SPHF. we also provide an efficient instantiation of the general framework.

In Chapter 6, we provide a practical and applicable treatment on the security vulnerability of the conventional PEKS system by formalizing a new PEKS system named Server-Aided Public Key Encryption with Keyword Search (SA-PEKS). We then introduce a universal transformation from any PEKS scheme to a secure SA-PEKS scheme using the deterministic blind signature, followed with an instantiation. Moreover, we describe how to securely implement the client-KS protocol and evaluate the performance of our solution in experiments.

In Chapter 7, we revisit the security modelling of leakage-resilient AKE protocols, and then introduce a new strong yet meaningful security model. We propose a general framework for the construction of AKE protocols secure under the proposed model. We also present a practical instantiation of the general framework and show that the instantiation is efficient in terms of the communication and computation overhead and captures more general leakage attacks.

Chapter 8 concludes this thesis.

Chapter 2

Preliminaries

In this chapter, we introduce the preliminaries that will be used throughout this thesis, including miscellaneous notations, foundations of algebra, complexity assumptions and cryptographic tools. The aim of this chapter is to make this thesis self-contained. More details of cryptography theory can be found in the following books [KL07].

2.1 Miscellaneous Notions

By ℓ, we denote a security parameter. By 1^ℓ, we denote the string of ℓ ones. We say that a function $\epsilon : \mathbb{Z} \to \mathbb{R}$ is negligible if $\forall\ k \in \mathbb{Z}$, $\exists\ z \in \mathbb{Z}$ such that $\epsilon(\ell) \leq \frac{1}{\ell^k}$ for all $\ell > z$. Unless otherwise specified, by ϵ, we always denote a negligible function.

For a finite set Ω, $\omega \xleftarrow{\$} \Omega$ denotes that ω is selected uniformly at random from Ω. $\omega \xleftarrow{R} \Omega$ denotes that ω is selected randomly from Ω.

Statistical Indistinguishability. Let X and Y be two random variables over a finite domain Ω, the *statistical distance* between X and Y is defined as $\mathsf{SD}(X, Y) = 1/2 \sum_{\omega \in \Omega} |\Pr[X = \omega] - \Pr[Y = \omega]|$.

Definition 2.1 *We say that X and Y are ϵ-statistically indistinguishable if $\mathsf{SD}(X, Y) \leq \epsilon$ and for simplicity we denote it by $X \overset{s}{\equiv}_\epsilon Y$. If $\epsilon = 0$, we say that X and Y are perfectly indistinguishable and denote it by $X \overset{s}{\equiv} Y$.*

Computational Indistinguishability. Let $\mathcal{V}_1$ and $\mathcal{V}_2$ be two probability distribution over a finite set Ω where $|\Omega| \geq 2^\ell$. We then define a distinguisher $\widetilde{\mathcal{D}}$ as follows. In the game, $\widetilde{\mathcal{D}}$ takes as input $\mathcal{V}_1$ and $\mathcal{V}_2$, the challenger flips a coin $\gamma \xleftarrow{\$} \{0, 1\}$. $\widetilde{\mathcal{D}}$ is then given an element $v_1 \xleftarrow{\$} \mathcal{V}_1$ if $\gamma = 1$, otherwise an element $v_2 \xleftarrow{\$} \mathcal{V}_2$. Finally, $\widetilde{\mathcal{D}}$ outputs a bit $\gamma' \in \{0, 1\}$ as its guess on γ. We define the advantage of $\widetilde{\mathcal{D}}$ in this game as $\mathsf{Adv}_{\widetilde{\mathcal{D}}}^{\mathcal{V}_1, \mathcal{V}_2}(\ell) = \Pr[\gamma' = \gamma] - 1/2$.

Definition 2.2 *We say that $\mathcal{V}_1$ and $\mathcal{V}_2$ are computationally indistinguishable if for any polynomial-time distinguisher $\mathcal{D}$, $\mathsf{Adv}_{\widetilde{\mathcal{D}}}^{\mathcal{V}_1, \mathcal{V}_2}(k)$ is negligible, and we denote it by $\mathcal{V}_1 \overset{c}{\equiv} \mathcal{V}_2$.*

Unless otherwise specified, by indistinguishability, we mean that it is computationally indistinguishable.

2.2 Foundations of Algebra

In this section, we review the basic algebra knowledge: group and cyclic group.

Definition 2.3 (Group) *A group $(\mathbb{G}, *)$ is a set $\mathbb{G}$ equipped with an operation $*$ with the following properties:*

Closure. $\forall\ g, h \in \mathbb{G},\ g * h \in \mathbb{G}$;

Associativity. $\forall\ g, h, \mu \in \mathbb{G},\ (g * h) * \mu = g * (h * \mu)$;

Identity. $\exists\ 1_{\mathbb{G}} \in \mathbb{G}$, *the identity of* $(\mathbb{G}, *)$, *such that* $\forall\ g \in \mathbb{G},\ 1_{\mathbb{G}} * g = g * 1_{\mathbb{G}} = g$;

Inverse. $\forall\ g \in \mathbb{G}, \exists\ g^{-1} \in \mathbb{G}$ *called the inverse of* g *such that* $g * g^{-1} = g^{-1} * g = 1_{\mathbb{G}}$.

For simplicity, a group $(\mathbb{G}, *)$ is often denoted as $\mathbb{G}$ when the operation $*$ is clear. The number of the elements in $\mathbb{G}$ is called the order of $\mathbb{G}$ and denoted as $|\mathbb{G}|$. A group $\mathbb{G}$ is a finite group if $|\mathbb{G}|$ is finite; otherwise, it is an infinite group. A group $\mathbb{G}$ is an *Abelian* group if $\forall\ g, h \in \mathbb{G},\ g * h = h * g$.

Definition 2.4 (Order of Group Element) *Suppose that $g \in \mathbb{G}$, the order of g in $\mathbb{G}$ is the least $i \in \mathbb{Z}^+$ such that $g^i = 1_{\mathbb{G}}$. If for all $i \in \mathbb{Z}^+$, $g^i \neq 1_{\mathbb{G}}$, the order of g is infinite. The order of g is denoted as $ord(g)$.*

Especially, if any element in a group $\mathbb{G}$ can be expressed by a special element in $\mathbb{G}$, $\mathbb{G}$ is called as a cyclic group. The formal definition of a cyclic group is as follows.

Definition 2.5 (Cyclic Group) *A group $\mathbb{G}$ is a cyclic group if there exists $g \in \mathbb{G}$, for all $h \in \mathbb{G}$, there exists $i \in \mathbb{Z}$ such that $h = g^i$. The element g is called as a generator of the group $\mathbb{G}$. $\mathbb{G}$ is said to be generated by g and denoted as $\mathbb{G} = \langle g \rangle$.*

2.3 Bilinear Groups

In this section, we review the knowledge related to the bilinear group. A pairing is a bilinear mapping from a group element to a group element.

Definition 2.6 (Bilinear Map) . *Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_\tau$ be three cyclic groups with the order p. Let g and h be the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. A bilinear map (pairing) is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\tau$ satisfying the following properties :*

Bilinearity. $\forall\ x \in \mathbb{G}_1, y \in \mathbb{G}_2$ *and* $a, b \in \mathbb{Z}_p,\ e(x^a, y^b) = e(x, y)^{ab}$.

Non-degeneracy. $e(g, h) \neq 1_{\mathbb{G}_\tau}$ *where* $1_{\mathbb{G}_\tau}$ *is the identity of the group* $\mathbb{G}_\tau$.

Computability. $\forall\ x \in \mathbb{G}_1$ *and* $y \in \mathbb{G}_2$, *there exists an efficient algorithm to compute* $e(x, y)$.

Definition 2.7 (Bilinear Groups [GPS08]) $\mathbb{G}_1, \mathbb{G}_2$, *and* $\mathbb{G}_\tau$ *constitute a bilinear group if there exists a bilinear map* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\tau$, *where* $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_\tau| = p$.

2.4 Complexity Assumptions

In this section, we review the complexity assumptions used throughout this thesis.

2.4.1 Discrete Logarithm Assumption

The discrete logarithm (DL) assumption [Odl85] in a finite field is one of the basic assumptions in cryptography research. The DL assumption is defined as follows.

Definition 2.8 (Discrete Logarithm (DL) Assumption [Odl85]) *Let* $\mathbb{G}$ *be a group with prime order* p *and* g *is the generator. Given* $(g, h) \in \mathbb{G}^2$, *we say that the discrete logarithm assumption holds on* $\mathbb{G}$ *if no PPT adversary* $\mathcal{A}$ *can compute* $a \in \mathbb{Z}_p$ *such that* $h = g^a$ *with the advantage*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{DL}} = \Pr\left[h = g^a | \mathcal{A}(p, g, h, \mathbb{G}) \to a\right] \geq \epsilon(\ell)$$

where the probability is taken over the random choice of $h \in \mathbb{G}$ *and the bits consumed by the adversary* $\mathcal{A}$.

2.4.2 Computational Diffie-Hellman Assumption

This assumption is proposed by Diffie and Hellman [DH76].

Definition 2.9 (Computational Diffie-Hellman (CDH) Assumption [DH76]) *Let* $\mathbb{G}$ *be a group with prime order* p *and* g *is the generator. Given* (g, g^a, g^b), *we say that the computational Diffie-Hellman assumption holds on* $\mathbb{G}$ *if no PPT adversary* $\mathcal{A}$ *can compute* g^{ab} *with the advantage*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{CDH}} = \Pr\left[\mathcal{A}(g, g^a, g^b) \to g^{ab}\right] \geq \epsilon(\ell)$$

where the probability is taken over the random choices of $a, b \xleftarrow{R} \mathbb{Z}_p$ *and the bits consumed by the adversary* $\mathcal{A}$.

2.4.3 Decisional Diffie-Hellman Assumption

Boneh [Bon98] surveyed the various applications of decisional Diffie-Hellman assumption and demonstrated some results regarding it security.

Definition 2.10 (Decisional Diffie-Hellman (DDH) Assumption [Bon98]) *Let $\mathbb{G}$ be a group with prime order p and g is the generator. Given (g, g^a, g^b), we say that the decisional Diffie-Hellman assumption holds on $\mathbb{G}$ if no PPT adversary $\mathcal{A}$ can distinguish (g^a, g^b, g^{ab}) from (g^a, g^b, g^c) with the advantage*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{DDH}} = \left|\Pr[\mathcal{A}(g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(g^a, g^b, g^c) = 1]\right| \geq \epsilon(\ell)$$

where the probability is taken over the random choices $a, b, c \xleftarrow{R} \mathbb{Z}_p$ and the bits consumed by the adversary $\mathcal{A}$.

2.5 Cryptographic Tools

In this section, we introduce some useful cryptographic tools.

2.5.1 Hash Function

Carter and Wegman [CW79] introduced the universal classes of hash functions. Roughly speaking, a hash function $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\ell$ is a deterministic function which takes as input an arbitrary-length string as input and returns a constant-size string as output.

Hash functions provide the following two main properties:

1. One-wayness. Given a value y, no PPT algorithm can find a value x such that $y = \mathcal{H}(x)$ with non-negligible probability.

2. Collusion Resistance. No PPT algorithm can find $x \neq y$ such that $\mathcal{H}(x) = \mathcal{H}(y)$ with non-negligible probability.

Hash function is an important cryptographical primitive and has been used as a building block to design encryption scheme [FOPS01], digital signature scheme [BR93b], message authentication code (MAC) scheme [BCK96], etc.

2.5.2 Random Oracle Model

Bellare and Rogaway [BR93b] introduced the notion of *random oracle model*, where all parties have access to a public random oracle, which hence provides a bridge between cryptographic theory and cryptographic practice. This paradigm yields protocols much more efficient than standard ones while enjoying many advantages of provable security [KL07].

Precisely speaking, despite the public existence, a randomly-chosen hash function $\mathcal{H}$ that can be evaluated only by querying an oracle, can be thought of as a

magic box that returns $\mathcal{H}(x)$ when given input x. Choosing a function $\mathcal{H}(x) \in \{0,1\}^\ell$ uniformly at random is generating random outputs for $\mathcal{H}$ "on-the-fly" as needed. Specifically, imagine that the function is defined by a table of pairs $\{(x_i, y_i)\}$ that is initially empty. When the oracle receives a query x it first checks whether $x = x_i$ for some pair (x_i, y_i) in the table; if so, the corresponding y_i is returned. Otherwise, a random string $y \in \{0,1\}^\ell$ is chosen, the answer y is returned, and the oracle stores (x, y) in its table so the same output can be returned if the same input is ever queried again. A distinctive feature of the random oracle model is that, if an adversary $\mathcal{A}$ has not explicitly queried the oracle on some point x, then the value of $\mathcal{H}(x)$ is completely random (at least as far as $\mathcal{A}$ is concerned).

Say we are trying to prove the security of some scheme in the random oracle model. We will often construct a *reduction* showing how any adversary $\mathcal{A}$ breaking the security of the scheme (in the random oracle model) can be used to violate some cryptographic assumption. As part of the reduction, the random oracle that $\mathcal{A}$ interacts with must be simulated as part of the reduction. That is: $\mathcal{A}$ will submit queries to and receive answers from what it believes to be the oracle, but what is actually the reduction itself. This gives the reduction a lot of power. As part of the reduction, we may choose values for the output of $\mathcal{H}$ "on-the-fly" (as long as these values are correctly distributed, i.e., uniformly random). The reduction gets to "see" the queries that $\mathcal{A}$ makes to the random oracle.

However, when the random oracle H is instantiated with a concrete hash function, the advantages above for reduction will no longer exist. Therefore, a scheme which is proven to be secure in the random oracle model does not necessarily imply that it is secure in the standard model [CGH98].

Unless otherwise specified, by saying a scheme is secure, we mean that it is secure in the standard model in this thesis.

2.5.3 Public-Key Encryption

Diffie and Hellman [DH76] introduced new research directions in cryptography called *public-key cryptography* (PKC) where two parties can communicate over public channels without compromising the security of the system.

Syntax. The formal definition of a PKE scheme is as follows [DH76].

$\mathsf{Setup}(1^\ell)$. The setup algorithm takes as input 1^ℓ and outputs the public parameters $params$.

$\mathsf{KeyGen}(1^\ell)$. The key generation algorithm takes as input 1^ℓ and outputs a secret-public pair (SK, PK).

$\mathsf{Enc}(params, PK, M)$. The encryption algorithm takes as input the public parameters $params$, the public key PK and a message M, and outputs a ciphertext CT.

$\mathsf{Dec}(params, SK, CT)$. The decryption algorithm takes as input the public parameters $params$, the secret key SK and the ciphertext CT, and outputs the message M.

The correctness property of a PKE scheme requires that,

$$\Pr\left[\begin{array}{l|l} \mathsf{Dec}(params, SK, CT) \rightarrow M & \begin{array}{l}\mathsf{Setup}(1^\ell) \rightarrow params; \\ \mathsf{KeyGen}(1^\ell) \rightarrow (SK, PK); \\ \mathsf{Enc}(params, PK, M) \rightarrow CT\end{array}\end{array}\right] = 1$$

Security Model. The standard notion of the security for a PKE scheme is called *indistinguishability against adaptive chosen ciphertext attacks* (IND-CCA2) [RS92]. This model is defined by the following game executed between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Setup. $\mathcal{C}$ runs $\mathsf{Setup}(1^\ell)$ to generate the public parameters $params$ and runs $\mathsf{KeyGen}(1^\ell)$ to generate the secret-public key pair (SK, PK). It then sends the public parameters $params$ with the public key PK to $\mathcal{A}$.

Phase 1. $\mathcal{A}$ can adaptively query the decryption oracle. $\mathcal{A}$ submits a ciphertext CT to $\mathcal{C}$, where $CT = \mathsf{Enc}(param, PK, M)$. $\mathcal{C}$ runs $\mathsf{Dec}(params, SK, CT)$ and responds $\mathcal{A}$ with M. This query can be made multiple times.

Challenger. $\mathcal{A}$ submits two messages M_0 and M_1 with equal length. $\mathcal{C}$ randomly selects M_b and computes $CT^* = \mathsf{Enc}(params, PK, M_b)$, where $b \in \{0, 1\}$. $\mathcal{C}$ responds $\mathcal{A}$ with CT^*.

Phase 2. $\mathcal{A}$ can adaptively query the decryption oracle. $\mathcal{A}$ submits a ciphertext CT to $\mathcal{C}$, where the only restrict is $CT \neq CT^*$. Phase 1 is repeated. This query can be made multiple times.

Guess. $\mathcal{A}$ outputs his guess b' on b. $\mathcal{A}$ wins the game if $b' = b$.

Definition 2.11 (IND-CCA2) *We say that a public-key encryption scheme is $(T, q, \epsilon(\ell))$-indistinguishable against adaptive chosen ciphertext attacks (IND-CCA2) if no PPT adversary $\mathcal{A}$ making q decryption queries can win the game with the advantage*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CCA2}} = \left|\Pr[b' = b] - \frac{1}{2}\right| \geq \epsilon(\ell)$$

in the above model.

Another security notion for public-key encryption is called *indistinguishability against adaptive chosen plaintex attacks* (IND-CPA). In this model, the adversary $\mathcal{A}$ is not allowed to query the decryption oracle. The formal definition of this model is as follows.

Definition 2.12 (IND-CPA) *We say that a public-key encryption scheme is* $(T, \epsilon(\ell))$*-indistinguishable against adaptive chosen plaintex attacks (*IND-CPA*) if no PPT adversary* $\mathcal{A}$ *who is restricted to query the decryption oracle can win the game with the advantage*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}} = \left| \Pr[b' = b] - \frac{1}{2} \right| \geq \epsilon(\ell)$$

in the above model.

2.5.4 Digital Signature

Digital signature was proposed by Diffie and Hellman [DH76]. It is the electronic version of a handwritten signature. A valid digital signature can convince a verifier that it was generated by a known party for a public message. Especially, a digital signature can provide non-repudiation property, namely a signer cannot deny he has generated the signature.

Syntax. [GMR88] A digital signature scheme formally consists of the following four algorithms:

- $\mathsf{Setup}(1^\ell)$. The setup algorithm takes as input 1^ℓ and outputs the public parameters $params$.
- $\mathsf{KeyGen}(1^\ell)$. The key generation algorithm takes as input 1^ℓ and outputs a secret-public key pair (SK, PK).
- $\mathsf{Sign}(params, SK, M)$. The signature algorithm takes as input the public parameters $params$, the secret ky SK and a message M, and outputs a signature σ on M.
- $\mathsf{Verify}(params, M, PK, \sigma)$. The verification algorithm takes as input the public parameters $params$, the message M, the public key PK and the signature σ, and outputs $True$ if $\mathsf{Sign}(params, M, SK) \rightarrow \sigma$; otherwise, it outputs $False$.

The correctness requires that for any $\mathsf{Sign}(params, SK, M) \rightarrow \sigma$,

$$\Pr\left[\mathsf{Verify}(params, M, PK, \sigma) \rightarrow True\right] \geq 1 - \epsilon(\ell)$$

and

$$\Pr\left[\mathsf{Verify}(params, M, PK, \sigma) \rightarrow False\right] < \epsilon(\ell).$$

Security Model. The security model of a digital signature scheme is formally defined by the following game executed between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Setup. $\mathcal{C}$ runs $\mathsf{Setup}(1^\ell)$ to generate the public parameters $params$ and runs $\mathsf{KeyGen}(1^\ell)$ to generate a secret-public pair (SK, PK) and sends $params, PK$ to $\mathcal{A}$.

Query. $\mathcal{A}$ can adaptively query the signature oracle. $\mathcal{A}$ adaptively sends messages $\{M_1, M_2, \cdots, M_q\}$ to $\mathcal{C}$. $\mathcal{C}$ runs $\mathsf{Sign}(params, SK, M_i)$ to generate a signature σ_i on M_i and responds $\mathcal{A}$ with σ_i, for $i = 1, 2, \cdots, q$.

Output. $\mathcal{A}$ outputs a message-signature pair (M^*, σ^*).

The traditional security of a digital signature is called *existential unforgeability under adaptive chosen message attacks* (EU-CMA) [GMR88] which is defined as follows.

Definition 2.13 (EU-CMA) *We say that a digital signature scheme is $(T, q, \epsilon(\ell))$-existentially unforgeable against adaptive chosen message attacks (*EU-CMA*) if no PPT adversary $\mathcal{A}$ can win the game with the advantage*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EU\text{-}CMA}} = \Pr\left[\begin{array}{c}\mathsf{Verify}(params, M^*, PK, \sigma^*) \rightarrow True \\ \bigwedge M^* \notin \{M_1, M_2, \cdots, M_q\}\end{array}\right] \geq \epsilon(\ell)$$

in the above model.

An, Dodis and Rabin [ADR02] proposed a stronger definition for the security of digital signature schemes called *strongly existential unforgeability under an adaptive chosen message attack* (SEU-CMA) as follows.

Definition 2.14 (SEU-CMA) *We say that a digital signature scheme is $(T, q, \epsilon(\ell))$-strongly existentially unforeable against adaptive chosen message attacks (*SEU-CMA*) if no PPT adversary $\mathcal{A}$ can win the game with the advantage*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SEU\text{-}CMA}} = \Pr\left[\begin{array}{c}\mathsf{Verify}(params, M^*, PK, \sigma^*) \rightarrow True \\ \bigwedge (M^*, \sigma^*) \notin \{(M_1, \sigma_1), (M_2, \sigma_2), \cdots, (M_q, \sigma_q)\}\end{array}\right] \geq \epsilon(\ell)$$

in the above model.

2.5.5 Blind Signature

Blind signature was introduced by Chaum [Cha83]. Roughly speaking, a blind signature scheme allows a signer to interactively issue signatures for a user such that the signer learns nothing about the message being signed (blindness) while the user cannot compute any additional signature without the help of the signer (unforgeability).

Syntax. Formally, a blind signature scheme is an interactive scheme that consists of a tuple of algorithms $(\mathsf{Kg}, \mathsf{Sign}, \mathsf{User}, \mathsf{Vf})$. Suppose that the system security parameter is ℓ. The signer generates a key pair via the key generation algorithm $(pk, sk) \xleftarrow{\$} \mathsf{Kg}(\ell)$. To obtain a signature on a message $m \in \{0,1\}^*$, the user and signer engage in an interactive signing protocol dictated by the algorithms $\mathsf{User}(pk, m)$ and $\mathsf{Sign}(sk)$. After the protocol completes, the User algorithm locally outputs a signature σ_m on m. To verify the validity of a signature σ_m, the verification algorithm Vf takes as input pk, m and σ_m, outputs $True$ if the signature is valid and $False$ otherwise. A blind signature scheme has correctness if $\mathsf{Vf}(pk, m, \sigma_m) = True$ for any $(pk, sk) \xleftarrow{\$} \mathsf{Kg}(\ell)$, any message m and any signature σ_m output by $\mathsf{User}(pk, m)$ after interacting with $\mathsf{Sign}(sk)$.

A blind signature is *deterministic* if for each public key pk and each message m, there exists only one signature σ_m such that $\mathsf{Vf}(pk, m, \sigma) = True$.

Security. The security of blind signature is twofold: *unforgeability* and *blindness*. The details are as follows.

Unforgeability. An efficient (i.e., polynomial-time) adversary against unforgeability aims to generate $q_s + 1$ valid message/signature pairs with distinct messages after being given the public key as input and at most q_s completed interactions with the signing oracle, where q_s is adaptively determined by the adversary during the attack. Note that this notion is different from the standard security notion of a signature scheme as the queried message in an interaction is not sent in clear. We say a blind scheme is *one-more-unforgeable* if any polynomial time adversary that queries the signing oracle with q_s distinct messages can only forge $q_s + 1$ valid message/signature pairs with negligible probability.

Blindness. Another notion, namely *blindness*, requires that the signer cannot tell apart the message it is signing. To be more precise, the blindness condition says that it should be infeasible for a malicious signer to decide which of the two messages has been signed first in two executions with an honest user. Note that even if the malicious signer is allowed to generate the public key maliciously, such a condition should still hold.

2.5.6 Public Key Encryption with Keyword Search

Boneh *et al.* [BCOP04] introduced a primitive, namely Public Key Encryption with Keyword Search (PEKS) that enables a user to search encrypted data in the asymmetric encryption setting. In a PEKS system, using the receiver's public key, the sender attaches some encrypted keywords (referred to as PEKS ciphertexts) with the encrypted data. The receiver then sends the trapdoor of a to-be-searched keyword to the server for data searching. Given the trapdoor and the PEKS ciphertext, the server can test whether the keyword underlying the PEKS ciphertext is equal to the one selected by the receiver. The formal definition is as follows.

Syntax. A non-interactive PEKS is defined by the following algorithms.

$\mathsf{KeyGen}(\ell)$. Taking as input the security parameter ℓ, it outputs the public/private key pair of the receiver as (pk_R, sk_R).

$\mathsf{PEKS}(pk_R, kw)$. Taking as input the public key pk_R and the keyword kw, it outputs the PEKS ciphertext of kw as CT_{kw}.

$\mathsf{Trapdoor}(sk_R, kw')$. Taking as input the secret key sk_R and the keyword kw', it outputs the the trapdoor of kw as $T_{kw'}$.

$\mathsf{Test}(pk_R, CT_{kw}, T_{kw'})$. Taking as input the public key pk_R, the PEKS ciphertext CT_{kw} and the trapdoor $T_{kw'}$, it outputs $True$ if $kw = kw'$, otherwise output $False$.

The consistency condition requires a PEKS scheme to satisfy that no adversary can find two different keywords such that the Test algorithm taking as input the PEKS ciphertext of one keyword and the trapdoor of the other keyword returns $True$.

Security Model. The security notion of a PEKS is called *semantic-security against chosen keyword attack* (SS-CKA) which states that the PEKS ciphertext does not reveal any information about the underlying keyword unless the matching trapdoor is available. More specifically, the SS-CKA security guarantees that no adversary is able to distinguish a keyword from another one given the PEKS ciphertext before he/she obtains the corresponding trapdoor. Formally, the SS-CKA game is defined as follows.

Setup. The challenger $\mathcal{C}$ runs the algorithm $\mathsf{KeyGen}(\ell)$, generates key pairs (pk_R, sk_R) and sends pk_R to the attacker $\mathcal{A}$.

Trapdoor Query-I. The attacker $\mathcal{A}$ can adaptively make the trapdoor query for any keyword.

Challenge. The attacker $\mathcal{A}$ sends the challenger two keywords kw_0, kw_1. The restriction here is that none of kw_0 nor kw_1 has been queried in the Trapdoor Query-I. The challenger $\mathcal{C}$ picks $b \xleftarrow{\$} \{0,1\}$ and generates $CT^* \leftarrow \mathsf{PEKS}(pk_R, kw_b)$ The challenger $\mathcal{C}$ then sends CT^* to the attacker.

Trapdoor Query-II. The attacker $\mathcal{A}$ can continue the query for the trapdoor of any keyword of its choice except of the challenge keywords kw_0, kw_1.

Output. Finally, the attacker $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$ on b and wins the game if $b = b'$.

Definition 2.15 (SS-CKA) *We say that a PEKS scheme is $(T, q, \epsilon(\ell))$-semantic-security against chosen keyword attack (SS-CKA) if no PPT adversary $\mathcal{A}$ making q trapdoor queries can win the game with the advantage*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SS\text{-}CKA}} = \left| \Pr[b' = b] - \frac{1}{2} \right| \geq \epsilon(\ell)$$

in the above model.

2.5.7 Randomness Extractor

The notion of the average-case strong extractor is firstly described in [DORS08]. We start with the introduction of average-case min-entropy.

Average-Case Min-Entropy. The *min-entropy* of a random variable X is $\mathrm{H}_\infty(X) = -\log(\max_x \Pr[X = x])$. Dodis *et al.* [DORS08] formalized the notion of average min-entropy that captures the unpredictability of a random variable X given the value of a random variable Y, formally defined as $\widetilde{\mathrm{H}}_\infty(X|Y) = -\log(\mathrm{E}_{y \leftarrow Y}[2^{-\mathrm{H}_\infty(X|Y=y)}])$. They also showed the following result on average min-entropy in [DORS08].

Lemma 1 ([DORS08]). *If Y has 2^ℓ possible values, then $\widetilde{H}_\infty(X|Y) \geq \widetilde{H}_\infty(X) - \ell$.*

Definition 2.16 (Average-Case Strong Extractor [DORS08]) *Let $\ell \in \mathbb{N}$ be a security parameter. A function $\mathsf{Ext} : \{0,1\}^{n(\ell)} \times \{0,1\}^{t(\ell)} \leftarrow \{0,1\}^{l(\ell)}$ is said to be an average-case (m, ϵ)-strong extractor if for all pairs of random variables (X, I) such that $X \in \{0,1\}^{n(\ell)}$ and $\widetilde{H}_\infty(X|I) \geq m$, it holds that*

$$\mathsf{SD}((\mathsf{Ext}(X, S), S, I), (U, S, I)) \leq \epsilon,$$

as long as $l(\ell) \leq m - 2\log(1/\epsilon)$, where $S \xleftarrow{\$} \{0,1\}^{t(\ell)}$ is the extraction key and $U \xleftarrow{\$} \{0,1\}^{l(\ell)}$.

2.5.8 Pseudo-Random Function

Here we describe the notion of pseudo-random function (PRF) defined in [GGM86] and its specific class, namely pseudo-random function with pairwise-independent random sources (πPRF), which was proposed by Okamoto in [Oka07].

PRF. Let $\ell \in \mathbb{N}$ be a security parameter. A function family F is associated with $\{\mathsf{Seed}_\ell\}_{\ell\in\mathbb{N}}$, $\{\mathsf{Dom}_\ell\}_{\ell\in\mathbb{N}}$ and $\{\mathsf{Rng}_\ell\}_{\ell\in\mathbb{N}}$. Formally, for any $\sum \xleftarrow{\$} \mathsf{Seed}_\ell$, $\sigma \xleftarrow{\$} \sum$, $\mathcal{D} \xleftarrow{\$} \mathsf{Dom}_\ell$ and $\mathcal{R} \xleftarrow{\$} \mathsf{Rng}_\ell$, $\mathsf{F}_\sigma^{\ell,\sum,\mathcal{D},\mathcal{R}}$ defines a function which maps an element of $\mathcal{D}$ to an element of $\mathcal{R}$. That is, $\mathsf{F}_\sigma^{\ell,\sum,\mathcal{D},\mathcal{R}}(\rho) \in \mathcal{R}$ for any $\rho \in \mathcal{D}$.

Definition 2.17 (PRF) *We say that* F *is a pseudo-random function (PRF) family if*

$$\{\mathsf{F}_\sigma^{\ell,\sum,\mathcal{D},\mathcal{R}}(\rho_i)\} \stackrel{c}{\equiv} \{RF(\rho_i)\}$$

for any $\{\rho_i \in \mathcal{D}\}$ *adaptively chosen by any polynomial time distinguisher, where RF is a truly random function. That is, for any* $\rho \in \mathcal{D}, RF(\rho) \xleftarrow{\$} \mathcal{R}$.

πPRF. Roughly speaking, πPRF refers to a pseudo-random function family that if a specific key σ is pairwise-independent from other keys, then the output of function with key σ is computationally indistinguishable from a random element.

Formally, let $Z_\sum$ be a set of random variables over $\sum$, and $I_\sum$ be a set of indices regarding $\sum$ such that there exits a deterministic polynomial-time algorithm, $f_\sum : I_\sum \to Z_\sum$, which on input the index $i \in I_\sum$, output $\sigma_i \in Z_\sum$. Consider the random variables $\{\sigma_{i_j}\}_{j=0,...,q(\ell)} = \{f_\sum(i_j)\}_{j=0,...,q(\ell)}$ where $i_j \in I_\sum$ and $q(\ell)$ a polynomial function of ℓ. We say that σ_{i_0} is *pairwisely independent* from other variables $\sigma_{i_1}, ..., \sigma_{i_{q(\ell)}}$ if for any pair of $(\sigma_{i_0}, \sigma_{i_j})(j = 1, ..., q(\ell))$, for any $(x, y) \in \sum^2$, we have $\Pr[\sigma_{i_0} \to x \wedge \sigma_{i_j} \to y] = 1/|\sum|^2$.

Definition 2.18 (πPRF) *Define* $\widetilde{\mathsf{F}}(\rho_j) = \mathsf{F}_{\sigma_{i_j}}^{\ell,\sum,\mathcal{D},\mathcal{R}}(\rho_j)$ *for* $i_j \in I_\sum$, $\rho_j \in \mathcal{D}$. *We say that* F *is a* π*PRF family if*

$$\{\widetilde{\mathsf{F}}(\rho_j)\} \stackrel{c}{\equiv} \{\widetilde{\mathsf{RF}}(\rho_j)\}$$

for any $\{i_j \in I_\sum, \rho_j \in \mathcal{D}\}$ $(j = 0, 1, ..., q(\ell))$ *adaptively chosen by any polynomial time distinguisher such that* σ_{i_0} *is pairwisely independent from* $\sigma_{i_j}(j > 0)$, *where* $\widetilde{\mathsf{RF}}$ *is the same as* $\widetilde{\mathsf{F}}$ *except that* $\widetilde{\mathsf{RF}}(\rho_0)$ *is replaced by a truly random value in* $\mathcal{R}$.

2.5.9 Smooth Projective Hash Functions

Smooth projective hash function(SPHF) is originally introduced by Cramer and Shoup [CS02] and extended for constructions of many cryptographic primitives [GL03, HK12, KV11, ABB+13, BBC+13c]. We start with the original definition.

Syntax. An SPHF can be defined based on a domain $\mathcal{X}$ and an $\mathcal{NP}$ language $\mathcal{L}$, where $\mathcal{L}$ contains a subset of the elements of the domain $\mathcal{X}$, i.e., $\mathcal{L} \subset \mathcal{X}$. An SPHF system over a language $\mathcal{L} \subset \mathcal{X}$, onto a set $\mathcal{Y}$, is defined by the following five algorithms ($\mathsf{SPHFSetup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}$):

$\mathsf{SPHFSetup}(1^\ell)$: generates the global parameters param and the description of an $\mathcal{NP}$ language instance $\mathcal{L}$;

$\mathsf{HashKG}(\mathcal{L}, \mathsf{param})$: generates a hashing key hk for $\mathcal{L}$;

$\mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}))$: derives the projection key hp from the hashing key hk;

$\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W)$: outputs the hash value $\mathsf{hv} \in \mathcal{Y}$ for the word W from the hashing key hk;

$\mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), W, w)$: outputs the hash value $\mathsf{hv}' \in \mathcal{Y}$ for the word W from the projection key hp and the witness w for the fact that $W \in \mathcal{L}$.

Property. A smooth projective hash function $\mathcal{SPHF}$ should satisfy the following properties,

Correctness. If a word $W \in \mathcal{L}$ with w the witness, then for all hashing key hk and projection key hp, we have

$$\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W) = \mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), W, w).$$

Smoothness. Let a point W be not in the language, i.e., $W \in \mathcal{X} \backslash \mathcal{L}$. Then the following two distributions are statistically indistinguishable :

$$\mathcal{V}_1 = \{(\mathcal{L}, \mathsf{param}, W, \mathsf{hp}, \mathsf{hv}) | \mathsf{hv} = \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W)\},$$

$$\mathcal{V}_2 = \{(\mathcal{L}, \mathsf{param}, W, \mathsf{hp}, \mathsf{hv}) | \mathsf{hv} \xleftarrow{\$} \mathcal{Y}\},$$

To be more precise, we have $\mathcal{V}_1 \overset{\$}{\equiv} \mathcal{V}_2$.

For the cryptographic purpose, we usually require the $\mathcal{NP}$ language $\mathcal{L}$ to be membership indistinguishable, which is formally defined as follows.

Definition 2.19 (Hard Subset Membership Problem) *For a finite set $\mathcal{X}$ and an NP language $\mathcal{L} \subset \mathcal{X}$, we say the subset membership problem is hard if for any word $W \xleftarrow{\$} \mathcal{L}$, W is computationally indistinguishable from any random element chosen from $\mathcal{X} \backslash \mathcal{L}$.*

For a smooth projective hash function that works on the above-defined language, the following property holds, which was introduced in [GL03].

Definition 2.20 (Pseudo-Random SPHF) *A smooth projective hash function* $\mathcal{SPHF} = (\mathsf{SPHFSetup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{WordG}, \mathsf{Hash}, \mathsf{ProjHash})$ *is pseudo-random, if for any PPT adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathsf{PR}}_{\mathcal{A}}(\ell) = \Pr\left[\begin{array}{c} (\mathsf{param}, \mathcal{L}) \leftarrow \mathsf{SPHFSetup}(1^\ell); \\ \mathsf{hk} \leftarrow \mathsf{HashKG}(\mathcal{L}, \mathsf{param}); \\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}, \mathsf{param})); \\ W \xleftarrow{\$} \mathcal{L}, b \xleftarrow{R} \{0,1\}; \\ \mathsf{hv}_0 \xleftarrow{\$} \mathcal{Y}; \mathsf{hv}_1 = \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W); \\ b' \leftarrow \mathcal{A}(\mathsf{param}, \mathcal{L}, \mathsf{hp}, W, \mathsf{hv}_b) : \\ b' = b. \end{array}\right] - 1/2$$

is negligible in ℓ*.*

Part I

Secure Data Storage

Chapter 3

A Formal Treatment on Merkle Hash Tree-Based Dynamic Cloud Auditing

In this chapter, we give a formal treatment on a well-known data structure named Merkle Hash Tree (MHT), which has been widely used in data integrity and authentication applications. We first revisit the existing MHT construction for dynamic data auditing protocol and show that an improper use of MHT could render the data auditing protocol insecure or impractical. Our main contribution is to introduce a new *rank-based Merkle Hash Tree*, which to the best of our knowledge is the first rigorously defined MHT that can support verifiable dynamic data operations. As another contribution of this work, we employ the proposed rank-based MHT to fix the security flaws in a recently proposed dynamic data auditing protocol named Oruta without sacrificing any desirable features of the protocol. The experiment result also shows that it is efficient to employ the rMHT for dynamic data auditing.

3.1 Introduction

In order to enable efficient remote data auditing, there have been several solutions proposed, among which *Proof of Retrievability* (POR), introduced by Juels and Kaliski in [JJ07], is the basis of many cloud data auditing schemes. Compared to *Provable Data Possession* (PDP), which was proposed by Ateniese *et al.* [ABC+07] and provides the guarantee of data integrity, POR directly captures the requirements on data retrievability. The original POR scheme [JJ07] can only support a limited number of data auditing queries and cannot achieve public verifiability. Using homomorphic authenticators, Shacham and Waters [SW08] proposed a publicly verifiable POR scheme with provable security, which supports an unlimited number of queries. Ateniese *et al.* [AKK09] later demonstrated that homomorphic authenticators can be used as a general building block for constructing communication-efficient proof of storage systems since an auditor can be convinced that a linear combination of data blocks are correctly generated by simply verifying an aggregation of the corresponding authenticators.

A number of practical PDP/POR schemes [JJ07, ABC+07, SW08, YY13, BJO09] employ the sampling (or spot-checking) technique where the data file is divided into

blocks and the auditor only queries a small set of randomly chosen blocks for integrity checking. This is to achieve low overhead of bandwidth and computation during the auditing process and thus has been utilized in many subsequent works. According to the result of [ABC+07], a public auditor can detect data damage with a high probability by only choosing a small subset of all the data blocks in each auditing. Suppose that the number of blocks stored in the server is 1,000,000, then if 1% of those blocks are lost or damaged, the auditor can detect these corrupted blocks with probability greater than 99% by choosing only 460 blocks randomly. However, since the untrusted server may want to cover up the data loss, it could launch a so-called *replace attack* in order to bypass the auditing. To be more precise, it uses the uncorrupted blocks to replace the corrupted ones during the auditing. The above result implies that if the auditing scheme is vulnerable to such attacks, the server can cheat as long as the number of corrupted blocks is smaller than 999540. In order to resist the replace attack, many auditing schemes are constructed with the *sequence-enforcing* property [JJ07, ABC+07, SW08, YY13]. That is, the authenticator of the i-th data block is constructed based on the data and also the block index information i so that the i-th block and its corresponding authenticator cannot be used to replace another (damaged) block requested by the auditor.

3.1.1 Motivations

Although the schemes aforementioned can be well adapted for data auditing, they only consider the case of *static* data storage, i.e., a file outsourced to the server never changes. However, in reality, users would regularly partially update their data stored on the cloud servers [KRS+03, LKMS04, MVS00]. Actually, one can easily find some application scenarios of dynamic data storage in practice [Dro, Goo, Bit].

Despite that the *sequence-enforced authenticator* construction is suitable for static data auditing, it suffers from the following drawbacks with respect to dynamic data storage. (1) *Significant Efficiency Loss.* Take the data insertion operation as an example, when the user inserts a new block into the original file, he/she needs to retrieve all the blocks which are located after the insertion position since their positions will be changed after the insertion operation. The user then needs to recompute the authenticators for these data blocks using the new position information, and uploads the new authenticators to the cloud server for data auditing in the future. We can see that this approach is very inefficient especially when the file is huge and thus impractical. (2) *Security Guarantee Loss.* The untrusted server may be able to launch a so-called *replay attack* to fool the auditor. The untrusted server may not perform the update operations correctly on the user's file, or have lost the most updated version. In order to fool the auditor, the server may just use

the old data instead of the updated one to generate a proof for an auditing query. This attack can be successful due to the fact that dynamic data operations in the server may not be verifiable and thus the proof generated from the old data may still pass the verification. Moreover, one can see from existing static data auditing protocols, e.g., in [JJ07, ABC$^+$07, SW08], that the modification of an authenticator would leak some secret information to the untrusted server and hence the server can also launch a *forgery attack* by forging the authenticator of any data block.

Desirable Properties of Dynamic Data Auditing Protocols. Since the static data auditing protocols can not be applied for dynamic data storage, it is of practical importance to design robust dynamic data auditing protocols that achieve the following desirable properties:

1. **Integrity Assurance (IA).** A dynamic auditing protocol should be able to achieve the data integrity assurance. That is, the user/auditor can be convinced that his/her data is well maintained by the remote server through the execution of the dynamic data auditing protocol.

2. **Position Checking (PC).** To resist against the replace attack, the auditor should be able to check that the returned proof are generated from the queried data blocks that he/she has chosen instead of other blocks. As shown above, the *sequence-enforced authenticator* construction for static data is not suitable for dynamic data auditing protocol.

3. **Verifiable Dynamic Data Operations (VDDO).** As mentioned above, the server may just use the old data blocks and authenticators stored in an archive or backup server instead of the updated ones to generate the data integrity proof for an auditing request, while the updated data is actually lost or damaged. Therefore, how to verify that the server has performed the updating operations requested from users and has used the most updated version in the data auditing process is another challenge for dynamic data auditing.

MHT-Based Solutions. Several practical auditing protocols have been proposed for dynamic data storage. Instead of directly embedding the position information into the authenticator of a data block, many of them [OR07, WWR$^+$11, LCY$^+$14] utilized a well-known data structure called Merkle Hash Tree (MHT) to achieve authentication of both block data and their positions. In [OR07], Oprea and Reiter proposed to use MHT to achieve data integrity for encrypted storage. However, their scheme can only support limited dynamic operations. To be more precise, the dynamic operations only include appending a new data block at the end of a file or

Table 3.1: Comparison of bMHT, sMHT, rMHT

Merkle Hash Tree	IA	PC	VDDO
bMHT	$\surd$	$\times$	$\times$
sMHT	$\surd$	$\surd$ (*only for static data*)	$\times^*$(not fully supported)
rMHT	$\surd$	$\surd$	$\surd$

removing the last data block from a file. Wang *et al.* [WWR+11] firstly extended the publicly verifiable POR scheme of Shacham and Waters [SW08] to support efficient dynamic data operations by employing a sequence-enforced MHT (denoted as sMHT in our work) for the authentication of both the values and the positions of data blocks. They treated the leaf nodes (representing data blocks) as a left-to-right sequence, and claimed that the positions of the data blocks can be ensured by following the way of computing the root of the MHT. Since their introduction, many subsequent dynamic auditing protocols, e.g., in [WCW+13, BH11, YNA+14, NYMX14] have been proposed through directly applying the sMHT from [WWR+11] for dynamic data auditing.

Unfortunately, after a rigorous analysis, we find that the sMHT cannot well support dynamic data auditing. Precisely, although the above sMHT-based schemes [WWR+11, WCW+13, BH11, YNA+14, NYMX14] claimed that they can achieve efficient dynamic data auditing using sMHT, none of these schemes has considered the question of how to use MHT to verify the positions of the data blocks during auditing. Actually, in this work, we find that the definition of sMHT is improper and hence fails to achieve the claimed purpose of authenticating the positions of the data blocks during the auditing process, which means replace attack could happen. Essentially speaking, the auxiliary authentication information (AAI) provided by the sMHT does not contain enough information to support position checking.

3.1.2 Contributions

To address the aforementioned problem, we perform a rigorous and detailed study on MHTs that are suitable for static and/or dynamic data auditing schemes. Our contributions can be summarized as follows:

1). We review the basic MHT (bMHT) and show that it cannot achieve the goal of block position checking, which means in order to prevent replace attack, some special treatment must be done in order to use MHT in data auditing schemes. We then show a simple way to extend a bMHT to a sequence-enforced MHT (sMHT) introduced in [WWR+11] for position checking in static data auditing.

2). Although sMHT works well for static data storage, it is not suitable for dynamic storage. The main contribution of this work is to propose a new MHT, named

rank-based MHT (rMHT) that can achieve the goal of data and position verification in dynamic data auditing. We present the details of the construction and the algorithms for both verification and dynamic data operations with some concrete examples.

3). We review a dynamic data auditing protocol named Oruta recently proposed in [WLL14] and show that the protocol is vulnerable to replace and replay attacks. We then employ the proposed rMHT to fix the problem without sacrificing any good feature of the original Oruta protocol.

A comparison of the three types of MHT (namely, bMHT, sMHT and rMHT) is given in Table 3.1.

3.1.3 Related Work

In additional to the existing work aforementioned, we also note that there are some other protocols proposed based on the MHT for cloud data storage. Mo *et al.* [MXZC14] proposed a new authentication to support the deletion of outsourced data. Instead of proving the existence of users' data on the cloud servers for integrity insurance, their introduced structure, namely Recursively Encrypted Red-black Key tree (RERK), is to confirm the non-existence of deleted data on the cloud servers. The work in [MZC12] did not notice the improper application of MHT in [WWR$^+$11] either. They developed an authenticated data structure called Cloud Merkle B+ tree (CMBT) to improve the performance of sMHT in [WWR$^+$11]. We should note that they also applied *rank-based idea* for their new CBMT definition but in a different way as the definition of each node in the proposed B+ tree is different from ours. Recently, Liu *et al.* [LCY$^+$14] also proposed another dynamic auditing scheme using MHT where each leaf node contains a rank to indicate the length of the data represented by that leaf node. It is worth noting that our new rMHT is different from that defined by Liu *et al.* [LCY$^+$14]. In particular, the rank defined in this chapter has a different meaning. Also, Liu *et al.*'s scheme uses the same AAI format as other auditing schemes and thus cannot effectively perform position checking. We also note that in a recent independent work [LRY$^+$14], a new novel multi-replica Merkle hash tree (MR-MHT) was introduced to mainly addressed the efficiency problem in verifiable updates for cloud storage with multiple replicas. They proposed a multi-replica public auditing (MuR-DPA) scheme which is based on MR-MHT, where all replica blocks for each data block are organized into the same replica sub-tree.

Another line of work to design dynamic auditing protocols is through applying non-MHT techniques. In [ADPMT08], Ateniese *et al.* proposed a dynamic version

Table 3.2: Notations for MHT

Notation	Description
v	a node in the MHT
h_v	the value of node v
r_v	the rank of node v
l_v	the level of node v
s_v	the side information of node v
a_i	the i-th authentic data block in the MHT
Ω_{a_i}	the auxiliary authentication information of a_i
CoR_{a_i}	the computation record of verification for a_i
R	the root of the Merkle Hash Tree
h'_R	the computed root value for verification
h^*_R	the updated root value
$P^*_{a_i}$	the updated path nodes of a_i

of their prior PDP scheme which supports limited dynamic data operation and imposes a bound on the number of queries. Erway *et al.* [EKPT09] constructed a full dynamic version of the PDP solution using a ranked-based skip list. However, public auditing is not supported by default. The first dynamic proof of retrievability construction was proposed by Stefanov *et al.* [SvDOJ11] which is applied in Iries, a cloud-based file system. Relying on Oblivious RAM (ORAM), Cash *et al.* [CKW13] proposed a POR construction to obtain asymptotic efficiency in dynamic storage. In order to reduce the bandwidth overhead required in [SvDOJ11] and [CKW13], Shi *et al.* [SSP13] proposed a light-weight dynamic POR construction that achieves comparable bandwidth overhead and client-side computation with a standard Merkle hash tree.

3.2 Merkle Hash Tree

In this section, we introduce a widely used authentication structure named Merkle Hash Tree and then show how to extend it for position checking. First, we define some notations that will be used in this chapter in Table 3.2.

3.2.1 Merkle Hash Tree

Merkle Hash Tree was proposed as an authentication structure [Mer87] to ensure data integrity. An MHT is a binary tree where each node v stores a hash value h_v of its children nodes. To be more precise, the leaf nodes contain the hash of the original data and each internal node contains the hash value of its two children nodes.

Integrity Assurance. Assume that the receiver/verifier knows the root value of an MHT. When receiving the i-th data block a_i, apart from the data itself, the receiver also requires some additional information in order to check the integrity of

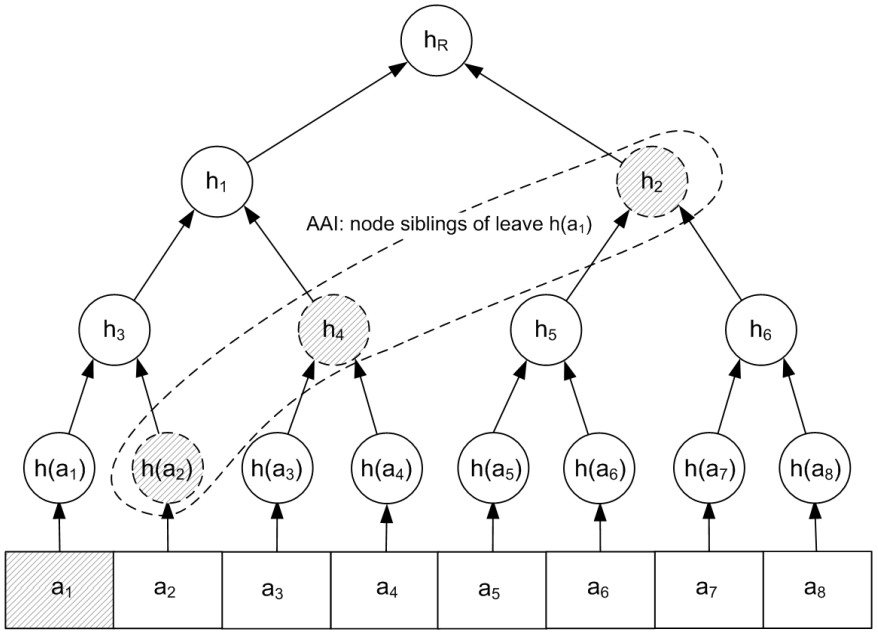


Figure 3.1: MHT for Authentication of Data Elements $(a_1, ..., a_8)$

the returned block a_i. The additional auxiliary authentication information (AAI) Ω_{a_i} for block a_i contains the sibling nodes from the i-th leaf node to the root. The authentication correctness and soundness can be guaranteed by the one way hash function used in the MHT construction.

Fig. 3.1 describes an example of MHT which is constructed for an ordered set of data elements $a_1, ..., a_8$. Any internal node value including the root value is derived from its two children nodes. For example, $h_5 = h(h(a_5)||h(a_6)), h_2 = h(h_5||h_6)$ and $h_R = h(h_1||h_2)$. Suppose that a receiver requests the first block, then in addition to a_1, the prover also provides the receiver with the corresponding auxiliary authentication information (AAI) as

$$\Omega_{a_1} =< h(a_2), h_4, h_2 >$$

To verify that block a_1 is undamaged and unaltered, the receiver first computes $h(a_1)$, $h_3 = h(h(a_1)||h(a_2)), h_1 = h(h_3||h_4), h'_R = h(h_1||h_2)$ and then checks if $h'_R = h_R$ (h_R is maintained locally by the verifier).

It is easy to see that although a basic MHT (bMHT) can be applied to authenticate the value of a data block, it does not provide the function of position checking. In the above example, when the verifier requests to verify the correctness of block a_1, the prover can simply use another block a_3 and its AAI to fool the receiver (i.e., a replace attack can be successful). It indicates that in addition to the root value verification, some special treatment must be performed in order to do position checking, which is important for data auditing schemes, otherwise the cloud server can use uncorrupted data blocks to replace the corrupted ones that are requested

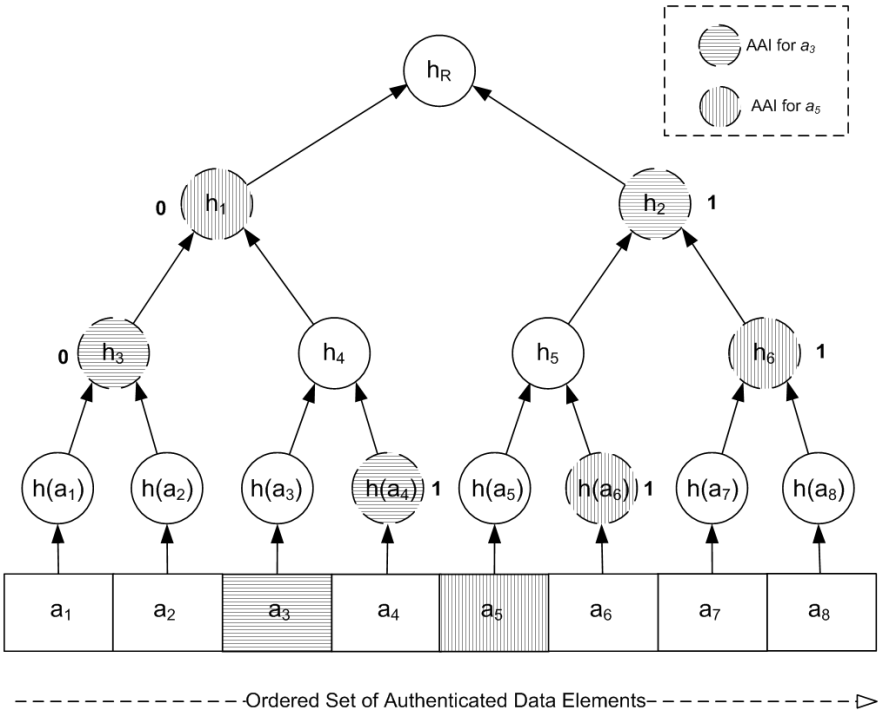


Figure 3.2: sMHT for Sequence-Enforced Authentication of Data Elements

by the auditor.

3.2.2 Sequence-Enforced Merkle Hash Tree (sMHT)

As we have mentioned before, both the value and the position of a data block should be checked in a data auditing protocol. However, as demonstrated above, the bMHT cannot directly achieve the goal of position checking. In this section, we introduce a new type of MHT named sequence-enforced MHT which is originally proposed by Wang *et al.* [WWR+11] without formal descriptions. In fact the only change we made in sMHT compared with bMHT is the verification procedure.

Integrity Assurance with Position Checking. In order to support efficient data integrity checking and position verification, we introduce a new notion named computation record (CoR). The CoR will indicate the relative position (left or right w.r.t. a leaf node) of each AAI node and hence can be used to determine the position of the leaf node when all the leaf nodes are located at the same level. After receiving a leaf node and the corresponding AAI nodes, the verifier now needs to record the computation order when generating the root value, and then use that information to verify the leaf node's position. To be more precise, for each AAI node, its position (first - 0 or second - 1) in the input of the hash function is recorded during the computation of the root value. Fig. 3.2 describes an example. Suppose the verifier wants to verify the leaf node a_3 of the MHT where the root value is h_R. The AAI

Table 3.3: Computation Record of $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$

Leaf Node a_i	**Computation Record** $\boldsymbol{CoR}_{a_i}$
a_1	111
a_2	011
a_3	101
a_4	001
a_5	110
a_6	010
a_7	100
a_8	000

for a_3 is $\Omega_{a_3} = \{h(a_4), h_3, h_2\}$. The value of a_3 can be verified by

$$h(\underline{h(h_3||\underline{h(h(a_3)||h(a_4))})||h_2}) \stackrel{?}{=} h_R.$$

Therefore, the computation record of a_3 is $CoR_{a_3} = 101$ since the computation order is $h(a_4) \rightarrow h_3 \rightarrow h_2$. As another example, for the leaf node a_5, the AAI is $\{h(a_6), h_6, h_1\}$ and the verification can be done as follows

$$h(h_3||\underline{h(\underline{h(h(a_5)||h(a_6))}||h_6)}) \stackrel{?}{=} h_R.$$

Therefore the computation record of a_5 is $CoR_{a_5} = 110$. The computation records of the other leaf nodes can be computed in the same way. Table 3.3 shows the computation records of all the leaf nodes.

We can see that each leaf node has a unique computation record. Hence, using CoR, the verifier can verify the position of each leaf node, under the condition that all the leaf nodes are at the same level.

Inefficient Verification. However, the sMHT cannot support efficient verification as the elements in the AAI do not indicate their position information, i.e., first or second position, in hash computation. Therefore, it results in an inefficient verification as the verifier has to compute the root value for many trial rounds until the correct computation order is used. To be more precise, suppose there are n data blocks, then the computation complexity is $O(n \log_2 n)$. As for the above example, the verifier does not know whether to compute h_1 as $h(h_3||h_4)$ or $h(h_4||h_3)$. And for the worst case, the verifier needs to compute 8 different root values before the integrity checking of a_3 completes.

Problem with Dynamic Storage. Another drawback of sMHT is that it cannot be used for dynamic storage. The reason is that the dynamic data operations, specifically, data insertion and deletion would change the height of the sMHT and make the tree unbalanced. That is, a node with the same index can be in different level after each dynamic operation. Note that in a stateless auditing protocol, the

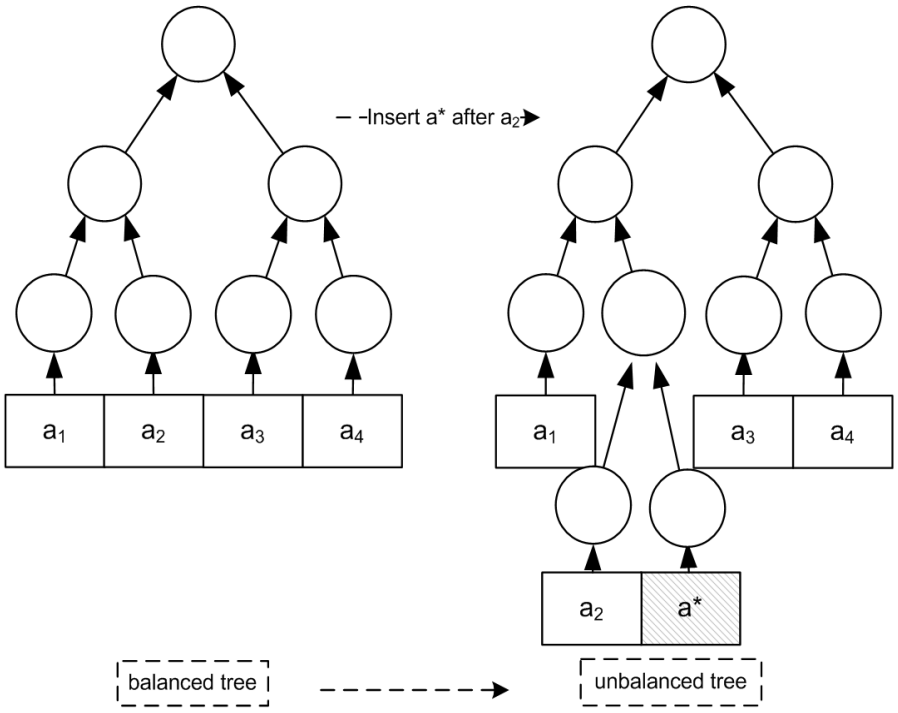


Figure 3.3: sMHT Updating for Data Insertion Operation

verifier has no knowledge about the structure of the corresponding sMHT. As a result, there will be no unified computation record for position verification. Fig. 3.3 shows an example, before the insertion operation, the sMHT is balanced and the computation records for a_1, a_2, a_3, a_4 are $11, 01, 10, 00$ respectively. However, after the insertion of a^*, the leaf nodes of a_2, a^* would be at a lower level compared with other leaf nodes and hence the sMHT is unbalanced. Therefore, the computation records for a_1, a_2, a^*, a_3, a_4 are $11, 101, 001, 10, 00$ respectively. One can note that the computation record of the second block a_2 changes due to the insertion operation. Since the verifier involved in a stateless auditing protocol has no knowledge about this alteration, it is infeasible for it to check the position of a_2 anymore.

3.3 Rank-Based Merkle Hash Tree

In order to make MHT suitable for dynamic data auditing, we develop a new type of Merkle Hash Tree, which we call rank-based Merkle Hash Tree (rMHT).

3.3.1 Construction

In our rMHT, in addition to the hash value h_v, each node v stores the number of leaf nodes at the bottom level that are under v. This value is called the *rank* of v and denoted by r_v. One can see that the rank of a non-leaf node is the sum of the rank of its children nodes. Different from sMHT, the value h_v in an rMHT is the

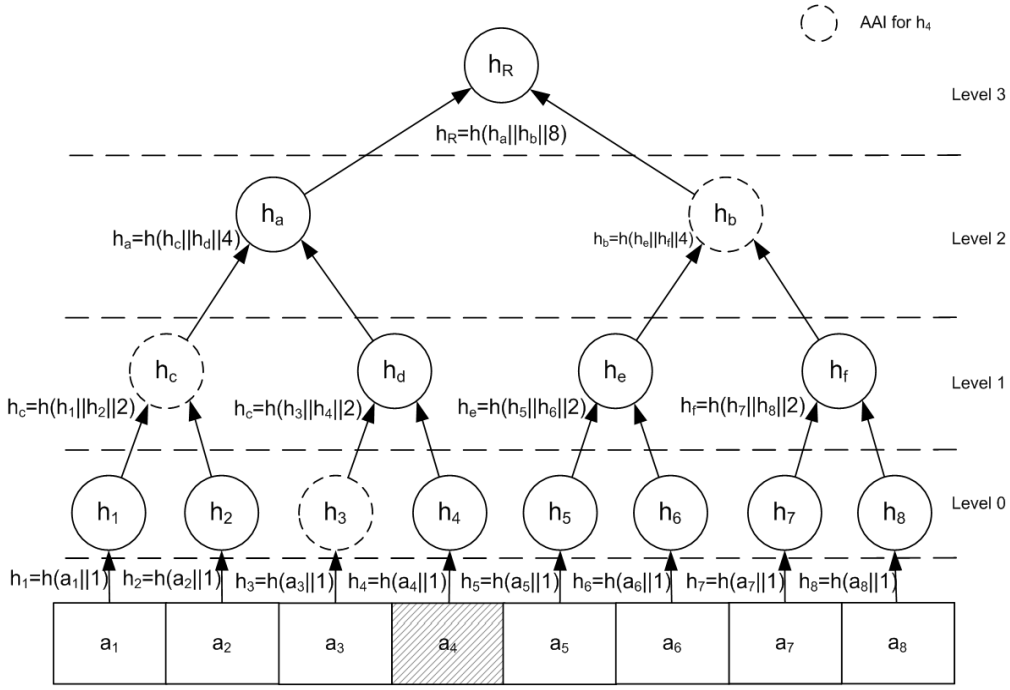


Figure 3.4: rMHT for Sequence-Enforced Authentication of Data Elements

hash of v's two children value and its rank value r_v. Another value stored in a node v is its level (height), denoted by l_v. The last value is called the side information, i.e., left or right (denoted by 0 and 1 respectively) to its parent node and is denoted by s_v. Therefore, the node v in an rMHT is denoted by

$$v = (h_v, r_v, l_v, s_v).$$

More precisely, suppose the child nodes of v are v_l (left child) and v_r (right child). Then the value of node v can be computed as followings,

$$l_v = l_{v_l} + 1,$$

$$r_v = r_{v_l} + r_{v_r},$$

$$h_v = h(h_{v_l}||h_{v_r}||r_v).$$

One should note that the root R does not include the side information and hence s_R is Null.

In the rMHT, the authentcation information (i.e., AAI) for the i-th authentic data element a_i is

$$\Omega_{a_i} = \{(v_1, ..., v_k)|v_j = (h_{v_j}, r_{v_j}, l_{v_j}, s_{v_j}), 1 \leq j \leq k\}.$$

Notice that each node in the AAI is from a different level, here we require that for

Algorithm 1 : $\{0, 1\} = \mathsf{Verify}\ (i, a_i, \Omega_{a_i}, h_R)$

Let $\Omega_{a_i} = \{v_1, ..., v_k\}$ where $v_j = (h_{v_j}, r_{v_j}, l_{v_j}, s_{v_j})$ for $1 \leq j \leq k$;
$CoR = 1; Ra = 2; h'_R = 0$;
if $s_{v_1} = 0$ **then**
  $h'_R = h(h_{v_1}||h(a_i)||Ra)$;
  $CoR = CoR + 1$;
else $h'_R = h(h(a_i)||h_{v_1}||Ra)$;
end if
for $j = 2$; $j \leq k$; $j++$ **do**
  $Ra = Ra + r_{v_j}$;
  if $s_{v_j} = 0$ **then**
    $h'_R = h(h_{v_j}||h'_R||Ra)$;
    $CoR = CoR + r_{v_j}$;
  else $h'_R = h(h'_R||h_{v_j}||Ra)$;
  end if
end for
if $h'_R \neq h_R$ **then**
  return 0;
else if $i \neq CoR$ **then**
  return 0;
else return 1;
end if

any $0 \leq j < t \leq k$, $l_{v_j} < l_{v_t}$, which means the nodes in the AAI are sorted by their levels. This is for the purpose of efficient verification as the computation of the root value is bottom-up. To give a clear picture, we provide an example of an rMHT in Fig. 3.4.

3.3.2 Efficient Verification

Here we show how to employ the rMHT for efficient verification of data integrity and position. Algorithm 1 describes the verification procedure. For all the AAI nodes used during the computation of the root value, we compute the sum of the ranks of those left-side nodes for position checking if the root value passes the verification. Specifically, CoR now records the number of leaf nodes which are on the left side of the verified node (the leaf nodes are treated as a left-to-right sequence in rMHT).

Back to the example in Fig. 3.4, suppose that the verifier wants to verify both the value and the position of an authentic data element say a_4, the verifier receives the corresponding auxiliary authentication information (AAI), that is

$$\Omega_{a_4} = \{v_1, v_2, v_3\} = \{(h_3, 1, 0, 0), (h_c, 2, 1, 0), (h_b, 4, 2, 1)\}$$

from the prover. Then the verifier sets an integer $CoR = 1$ and computes by following our verification algorithm as follows.

Algorithm 2 : $(h_R^*, P_{a_i^*}^*) = \mathsf{Mod}\ (i, a_i^*, \Omega_{a_i})$

Let $\Omega_{a_i} = \{v_1, ..., v_k\}$ where $v_j = (h_{v_j}, r_{v_j}, l_{v_j}, s_{v_j})$ for for $1 \leq j \leq k$; and the update path nodes $P_{a_i^*}^* = \{p_1, ..., p_{k+1}\}$ where $p_j = (h_{p_j}, r_{p_j}, l_{p_j}, s_{p_j})$ for $1 \leq j \leq k+1$;
$h_R^* = 0; h_{p_1} = h(a_i^*); r_{p_1} = 1; l_{p_1} = 0;$
for $j = 1;\ j \leq k;\ j++$ **do**
  $r_{p_{j+1}} = r_{p_j} + r_{v_j};$
  $l_{p_{j+1}} = l_{v_j} + 1;$
  if $s_{v_j} = 0$ **then**
    $h_{p_{j+1}} = h(h_{v_j}||h_{p_j}||r_{p_{j+1}});$
    $s_{p_j} = 1;$
  else if $s_{v_j} = 1$ **then**
    $h_{p_{j+1}} = h(h_{p_j}||h_{v_j}||r_{p_{j+1}});$
    $s_{p_j} = 0;$
  end if
end for
$h_R^* = h_{p_{k+1}};$
return $h_R^*, P_{a_i^*}^*;$

1). Compute rank of d as $r_d = 1 + 1 = 2$ and value of d as $h_d = h(h_3||h(a_4||1)||2)$, set $CoR = 1 + 1 = 2$;

2). Compute rank of a as $r_a = 2 + 2 = 4$ and value of a as $h_a = h(h_c||h_d||4))$, set $CoR = 2 + 2 = 4$;

3). Compute rank of root h'_R as $4 + 4 = 8$ and value of h'_R as $h(h_a||h_b||8)$;

4). Check if $h'_R = h_R$ and verify if $CoR = 4$.

From the above example, one can see that the rMHT can indeed support efficient verification of both value and position of the authentic data. The reason is that the side information indicates the position for hash computation and hence results in an efficient root value generation.

3.3.3 Verifiable Dynamic Data Operations

In this section, we introduce how to employ the rMHT for dynamic data operations, including modification, insertion, and deletion. Below we show the algorithms for updating the rMHT for these three operations.

Modification. A data modification operation refers to the replacement of specified data blocks with new ones. Algorithm 2 describes the procedure of rMHT update for data modification.

The input of the algorithm consists of (i, a_i^*, Ω_{a_i}) where i indicates that the data to be modified is the i-th data block which is to be replaced by a_i^*. Ω_{a_i} is the AAI of the original data a_i. The algorithm Mod then updates the rMHT as follows.

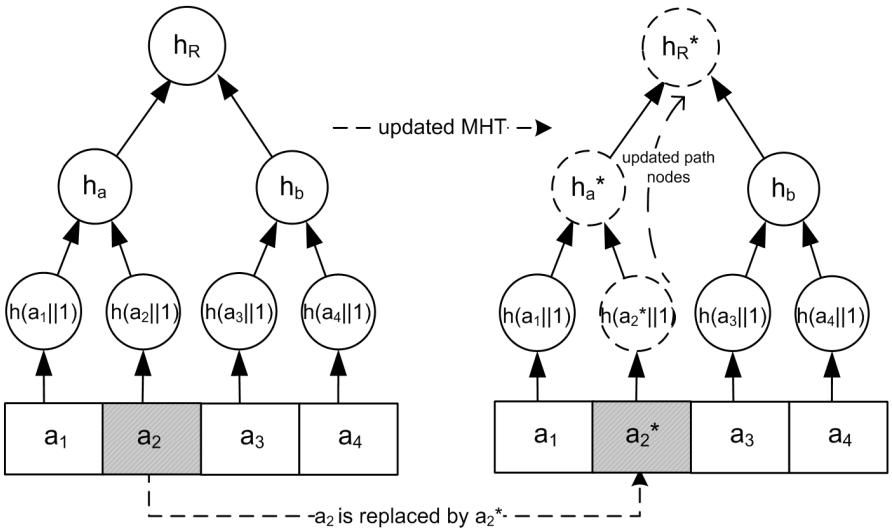


Figure 3.5: rMHT Updating for Data Modification

1). Compute the hash of a_i^* as the new value of the i-th leaf node and set its rank and level values respectively;

2). For each sibling node in the AAI, compute its parent node and set the side information of the same-level path node;

3). Finally, return the new root value h_R^* and output the new path nodes $P_{a_i^*}^*$ of a_i^*.

Fig. 3.5 shows an example of the update operation of the rMHT regarding data modification. Suppose that the data element a_2 is replaced by a new data a_2^*. The nodes need to be updated are those on the path from the leaf node to the root of the rMHT, i.e., $h(a_2^*||1), h_a^*, h_R^*$. It is worth noting that the data modification operation does not change the rank value or side information of each node in the rMHT.

Insertion. A data insertion operation refers to inserting new data blocks into some specified positions of the original data set. Algorithm 3 shows the operation of an rMHT update for data insertion.

The algorithm takes $(i, v_{a_i}, a_i^*, \Omega_{a_i})$ as input which indicates that the new data block a_i^* is to be inserted into the i-th position and Ω_{a_i} is the AAI for the original i-th block a_i. Then the algorithm Insexecutes as follows.

1). Compute the hash of a_i^* as the new value of the i-th leaf node and set its rank and level values respectively;

2). Compute the parent node of a_i^* and a_i;

3). For each sibling node in the AAI, compute its parent node and set the side information of the same-level path node;

Algorithm 3 : $(h_R^*, P_{a_i^*}^*) = \mathsf{Ins}\ (i, v_{a_i}, a_i^*, \Omega_{a_i})$

Let $\Omega_{a_i} = \{v_1, ..., v_k\}$ where $v_j = (h_{v_j}, r_{v_j}, l_{v_j}, s_{v_j})$ for $1 \leq j \leq k$; and $P_{a_i^*}^* = \{p_1, ..., p_{k+2}\}$ where $p_j = (h_{p_j}, r_{p_j}, l_{p_j}, s_{p_j})$ for $1 \leq j \leq k+2$; Let $v_{a_i} = (h_{v_{a_i}}, r_{v_{a_i}}, l_{v_{a_i}}, s_{v_{a_i}})$ be the original i-th leaf node.
$h_R^* = 0; h_{p_1} = h(a_i^*); r_{p_1} = 1; l_{p_1} = l_{v_{a_i}} - -; s_{v_{a_i}} = 1; s_{p_1} = 0;$
$h_{p_2} = h(h_{p_1}||h_{v_{a_i}}||2); r_{p_2} = 2; l_{p_2} = l_{p_1} + 1;$
for $j = 1;\ j \leq k;\ j++$ **do**
  $r_{p_{j+2}} = r_{p_{j+1}} + r_{v_j};$
  $l_{p_{j+2}} = l_{v_j} + 1;$
  if $s_{v_j} = 0$ **then**
    $h_{p_{j+2}} = h(h_{v_j}||h_{p_{j+1}}||r_{p_{j+2}});$
    $s_{p_{j+1}} = 1;$
  else if $s_{v_j} = 1$ **then**
    $h_{p_{j+2}} = h(h_{p_{j+1}}||h_{v_j}||r_{p_{j+2}});$
    $s_{p_{j+1}} = 0;$
  end if
end for
$h_R^* = h_{p_{k+2}};$
return $h_R^*, P_{a_i^*}^*;$

4). Finally, return the new root value h_R^* and output the new path nodes $P_{a_i^*}^*$ of a_i^*.

Fig. 6.4 describes the update operation of the rMHT regarding data insertion. Suppose that a new data block a_2^* is to be inserted before the data block a_2. The nodes need to be updated are those nodes on the path from the inserted leaf node to the root of the rMHT. More precisely, a new internal node is inserted as the parent node of the new inserted leaf node h_2 and the leaf node h_3 as $h(h_2||h_3||2)$. Moreover, since the number of the leaf nodes increases in the new rMHT, the value of h_a^* and root h_R^* are also recomputed as: $h_a^* = h(h(a_1||1)||h(h_2||h_3||2)||3), h_R^* = h(h_a^*||h_b||5)$. One can see that data insertion operation changes the rank value of the nodes on the updated path.

Deletion. A data deletion operation refers to removing a specified data element from the original data set. Algorithm 4 describes the operation of an rMHT update for data deletion. Compared to the algorithms Mod and Ins, the input of algorithm Del just consists of (i, Ω_{a_i}) which indicates that the data to be deleted is the i-th block. The algorithm then updates the corresponding path nodes using the AAI of the original data element a_i as follows.

1). Recompute the level and side value of the first sibling node in Ω_{a_i} and set it to be the first path node;

2). For each other sibling node in the AAI, compute its parent node as the new path node;

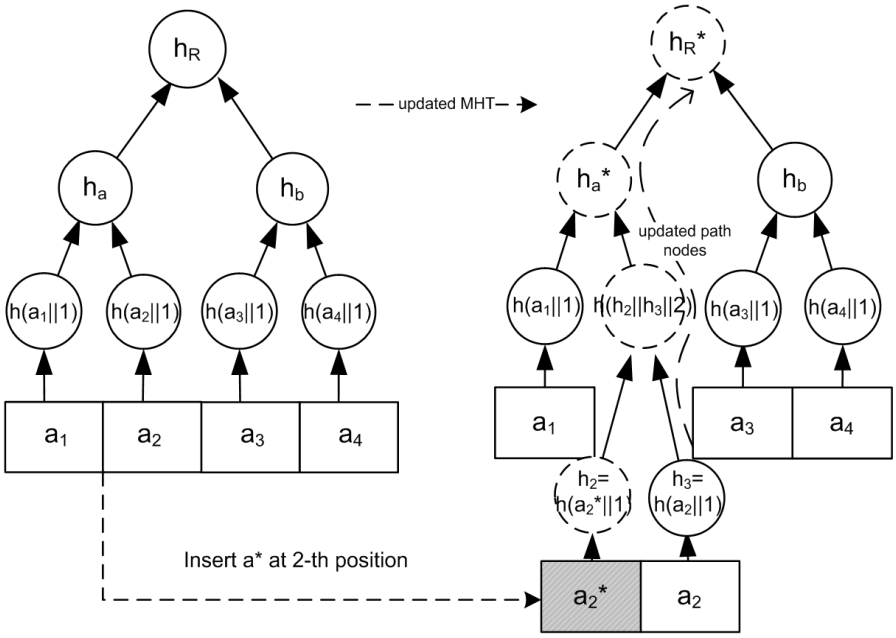


Figure 3.6: rMHT Updating for Data Insertion

3). Finally, return the new root value h_R^* and output the new path nodes $P^*_{a_{i-1}}$ of the $i-1$-th node a_{i-1}.

Fig. 3.7 shows the update operation of the rMHT regarding data deletion. Suppose that the data element a_2 is removed. The original internal node h_a now becomes the leaf node $h_a^* = h(a_1||1)$. Hence the root value is updated to $h_R^* = h(h_a^*||h_b||3)$. One can also see that data insertion operation changes the rank value of the nodes on the updated path.

Verifiable Dynamic Data Operations. In order to ensure that the cloud server has performed the dynamic operations honestly, the data owner can perform a verification on the new root value to check if the cloud server has indeed performed the update operation. The details will be given in Section 3.5.1. However, as mentioned before, when a delegated TPA performs the data auditing, it is possible for the cloud server to use an old version of the data and the corresponding authentication information to generate an auditing proof. In a stateless data auditing scheme, the root of the rMHT should be signed by the user using a secure digital signature scheme in order to prevent the server from modifying the root value, and the TPA does not need to store the root value since it can be recomputed using any leaf node and the corresponding AAI chosen in an auditing request, and then verified using the user's signature. However, in order to prevent the replay attack in which the cloud server can use an old rMHT rather than the updated version, we should require the TPA to keep the latest root value of the rMHT. Another relatively simpler solution to solving the problem is to let the user add a timestamp when signing the root value, and inform the TPA the time of the most recent update.

Algorithm 4 : $(h_R^*, P_{a_{i-1}}^*) = \mathsf{Del}\ (i, \Omega_{a_i})$

Let $\Omega_{a_i} = \{v_1, ..., v_k\}, P_{a_{i-1}}^* = \{p_1, ..., p_k\}$ where $v_j = (h_{v_j}, r_{v_j}, l_{v_j}, s_{v_j}); p_j = (h_{p_j}, r_{p_j}, l_{p_j}, s_{p_j})$ for $1 \leq j \leq k$;
$h_R^* = 0; h_{p_1} = h_{v_1}; r_{p_1} = r_{v_1}; l_{p_1} = l_{v_1} + 1$;
for $j = 1;\ j \leq k;\ j++$ **do**
  $r_{p_{j+1}} = r_{p_j} + r_{v_j}$;
  $l_{p_{j+1}} = l_{P_j} + 1$;
  if $s_{v_j} = 0$ **then**
    $h_{p_{j+1}} = h(h_{v_j} || h_{p_j} || r_{p_{j+1}})$;
    $s_{p_j} = 1$;
  else if $s_{v_j} = 1$ **then**
    $h_{p_{j+1}} = h(h_{p_j} || h_{v_j} || r_{p_{j+1}})$;
    $s_{p_j} = 0$;
  end if
end for
$h_R^* = h_{p_k}$;
return $h_R^*, P_{a_{i-1}}^*$;

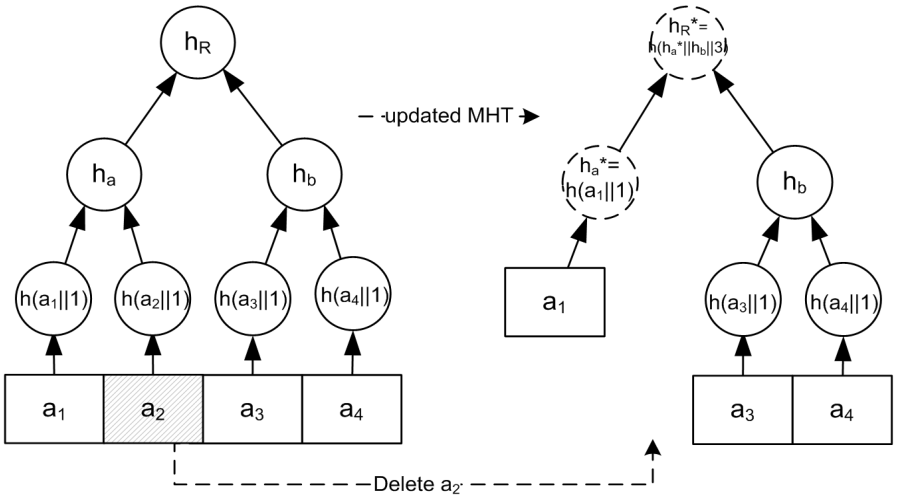


Figure 3.7: rMHT Updating for Data Deletion

3.4 Review of Oruta

In this section, we review the details of a dynamic auditing protocol named Oruta recently proposed in [WLL14]. Oruta was proposed with the purpose of supporting dynamic data operations and ensuring user privacy using the techniques of ring signature. However, below we show that the protocol is vulnerable to the replace attack.

3.4.1 Construction

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_\tau$ be three multiplicative cyclic groups of prime order p, and g_1 and g_2 be the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\tau$ be a bilinear

map, and $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ be a computable isomorphism with $\psi(g_2) = g_1$. Let $H_1 : \{0,1\}^* \to \mathbb{G}_1, H_2 : \{0,1\}^* \to \mathbb{Z}_q$, and $h : \mathbb{G}_1 \to \mathbb{Z}_p$ denote three cryptographic hash functions. The global system parameters are $(e, \psi, p, q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\tau, g_1, g_2, H_1, H_2, h)$.

An outsourced data file M is divided into n blocks, and each block m_j is divided into k elements in $\mathbb{Z}_p$. Then, the data component M can be viewed as an $n \times k$ matrix. Also, let d denote the number of users in the group where the file is shared.

KeyGen. Each user u_i randomly chooses $x_i \in \mathbb{Z}_p$ and computes $w_i = g_2^{x_i}$. So the user u_i's public key is w_i, and the private key is x_i. Besides, the user who firstly create the file should generate a public aggregate key $pak = \{\eta_1, ..., \eta_k\}$, where each η_l $(1 \leq l \leq k)$ is a random elements of $\mathbb{G}_1$.

SignGen. Given all members' public keys $\{w_1, ..., w_d\}$, a block $m_j = \{m_{j,1}, ..., m_{j,k}\}$, the identifier of the block id_j, a public aggregate key $pak = \{\eta_1, ..., \eta_k\}$ and the private key of the signer x_s, the user u_s generates a ring signature for this block as follows.

1). The signer first aggregates block m_j with the public aggregate key pak, and computes $\beta_j = H_1(id_j) \prod_{l=1}^{k} \eta_l^{m_{j,l}} \in \mathbb{G}_1$.

2). Then the signer randomly chooses $a_{j,i} \in \mathbb{Z}_p$ and sets $\sigma_{j,i} = g_1^{a_{j,i}}$ for all $i \neq s$. And for $i = s$, he/she computes $\sigma_{j,i} = (\frac{\beta_j}{\psi(\prod_{i \neq s} w_i^{a_{j,i}})})^{1/x_s}$. Therefore, the ring signature of block m_j is $\sigma_j = \{\sigma_{j,1}, ..., \sigma_{j,d}\}$.

ProofGen. The third-party auditor (TPA) generates the challenge in the following way:

1). The TPA picks c elements in set $[1, n]$, where n is the total number of blocks, to indicate the blocks that will be checked. Let J denote the indices of the chosen blocks.

2). For $j \in J$, the TPA randomly chooses $y_j \in \mathbb{Z}_q$. Then the TPA sends $\{(j, y_j)\}_{j \in J}$ to the cloud server as a challenge message.

After receiving $\{(j, y_j)\}_{j \in J}$, the cloud server generates the proof as follows:

1). For $l \in [1, k]$, the cloud server randomly chooses $r_l \in \mathbb{Z}_q$, and computes $\lambda_l = \eta_l^{r_l} \in \mathbb{G}_1$, and $\mu_l = \sum_{j \in J} y_j m_{j,l} + r_l h(\lambda_l) \in \mathbb{Z}_p$.

2). For $i \in [1, d]$, the cloud server computes $\phi_i = \prod_{j \in J} \sigma_{j,i}^{y_j}$.

Then the cloud server sends the proof $\{\{\lambda_l\}_{l \in [1,k]}, \{\mu_l\}_{l \in [1,k]}, \{\phi_i\}_{i \in [1,d]}, \{id_j\}_{j \in J}\}$ to TPA.

ProofVerify. After receiving the proof, and given the public aggregate key $pak = \{\eta_1, ..., \eta_k\}$ and all the public keys $\{w_1, ..., w_d\}$ of the group members, the TPA verifies the proof by checking:

$$e(\prod_{j\in J} H_1(id_j)^{y_j} \cdot \prod_{l=1}^{k} \eta_l^{\mu_l}, g_2) \stackrel{?}{=} (\prod_{i=1}^{d} e(\phi_i, w_i)) \cdot e(\prod_{i=1}^{k} \lambda_l^{h(\lambda_l)}, g_2)$$

Remarks. Instead of using the index of a block as its identifier (e.g. the index of block m_j is j), this scheme utilises index hash tables. An identifier from this table is described as $id_j = \{v_j, r_j\}$, where v_j is the virtual index of block m_j, and $r_j = H_2(m_j||v_j)$ is generated using a collision-resistance hash function $H_2 : \{0,1\}^* \in \mathbb{Z}_q$. Here q is a prime that is much smaller than p. If $v_i < v_j$, then block m_i is in front of m_j in the file. The initial virtual index of block m_j is set as $v_j = j \cdot \delta$, where δ indicates the number of data block that can be inserted into m_j and m_{j+1}. For example, if m'_j is inserted, then $v'_j = (v_{j-1} + v_j)/2, r'_j = H_2(m'_j||v'_j)$ is inserted into the index hash table; if m_j is deleted, then the corresponding entry is removed from the table.

3.4.2 Replace Attack on Oruta

Below we show that the Oruta scheme is vulnerable to the replace attack. Suppose the challenge query from the auditor is $\{(j, y_j)\}_{j\in J, y_j\in Y}$. We show that the server can redirect the challenge on the corrupted blocks to uncorrupted blocks in order to pass the verification. Without loss of generality, let that the corrupted challenged blocks be $\{m_t\}_{t\in T}$ where $T \subseteq J$. Then the server choose other $|T|$ uncorrupted blocks $\{m_{t^*}\}_{t^*\in T^*}$ where $|T^*| = |T|$ and $T^* \subseteq [1, n]/J$. Let $J^* = T^* \cup J/T, Y^* = Y$. The server generates the auditing proof as follows,

1). For $l \in [1, k]$, the cloud server randomly chooses $r_l \in \mathbb{Z}_q$, and computes $\lambda_l = \eta_l^{r_l} \in \mathbb{G}_1$.

2). For $l \in [1, k]$, the cloud server also computes $\mu_l = \sum_{j\in J^*, y_j\in Y^*} y_j m_{j,l} + r_l h(\lambda_l) \in \mathbb{Z}_p$.

3). For $i \in [1, d]$, the cloud server computes $\phi_i = \prod_{j\in J^*, y_j\in Y^*} \sigma_{j,i}^{y_j}$.

Finally, the cloud server sends the auditing proof $\{\{\lambda\}, \{\mu\}, \{\phi\}, \{id_j\}_{j\in J^*}\}$ to the TPA. The TPA verifies the proof by checking: $e(\prod_{j\in J^*, y_j\in Y^*} H_1(id_j)^{y_j} \cdot \prod_{l=1}^{k} \eta_l^{\mu_l}, g_2) \stackrel{?}{=} (\prod_{i=1}^{d} e(\phi_i, w_i)) \cdot e(\prod_{i=1}^{k} \lambda_l^{h(\lambda_l)}, g_2)$.

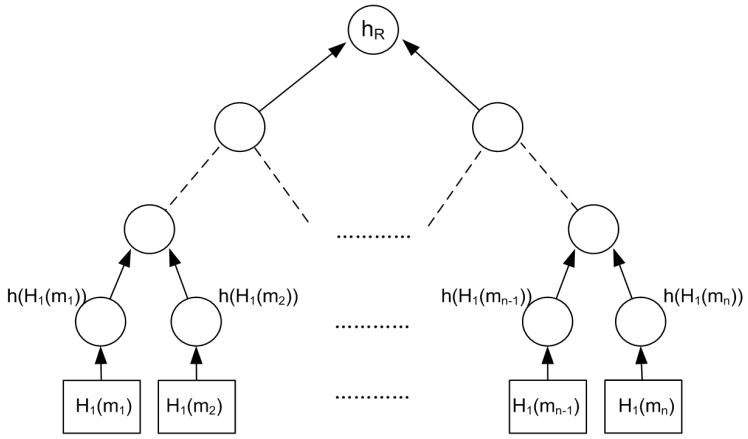


Figure 3.8: rMHT Construction for the Improved Oruta

The correctness is ensured by:

$$
\begin{aligned}
& (\prod_{i=1}^{d} e(\phi_i, w_i)) \cdot e(\prod_{i=1}^{k} \lambda_l^{h(\lambda_l)}, g_2) \\
= & \prod_{j \in J^*, y_j \in Y^*} (\prod_{i=1}^{d} e(\sigma_{j,i}^{y_j}, w_i) \cdot e(\prod_{l=1}^{k} \lambda_l^{h(\lambda_l)}, g_2) \\
= & \prod_{j \in J^*, y_j \in Y^*} \left(\prod_{i=1}^{d} e(\sigma_{j,i}, w_i) \right)^{y_j} \cdot e(\prod_{l=1}^{k} \eta_l^{r_l h(\lambda_l)}, g_2) \\
= & \prod_{j \in J^*, y_j \in Y^*} e(H_1(id_j)^{y_j} \cdot \prod_{l=1}^{k} \eta_l^{y_j m_{j,l} + r_l h(\lambda_l)}, g_2) \\
= & \; e(\prod_{j \in J^*, y_j \in Y^*} H_1(id_j)^{y_j} \cdot \prod_{l=1}^{k} \eta_l^{\mu_l}, g_2)
\end{aligned}
$$

Although Oruta uses the hash table to support dynamic data operations, our attack above essentially shows that such a technique is not suitable since it makes the scheme vulnerable to the replace attack.

3.4.3 Replay Attack

The problem of replay attack against Oruta has already been noticed by the authors of [WLL14]. It is easy to see that even if the server does not perform any update operation, it can still pass the data auditing later by using the old file and authentication information. The authors of [WLL14] didn't provide a solution to solving the problem and left it as their future work.

Table 3.4: Auditing Protocol of the Improved Oruta

TPA		**Server**
Generate query $\{j, y_j\}_{j\in J}$		
	$\xrightarrow{\{j,y_j\}_{j\in J}}$	
		Compute $\{\lambda, \mu, \phi\}$; Generate $\{H_1(m_j), \Omega_{m_j}\}_{j\in J}$;
	$\xleftarrow{\{\lambda,\mu,\phi,\{H_1(m_j),\Omega_{m_j}\}_{j\in J},\sigma_R\}}$	
$\mathsf{Verify}(j, \Omega_{m_j}, H_1(m_j), \sigma_R) \rightarrow V_j$; Run $\mathsf{ProofVerify}$ using $(\{y_j, H_1(m_j)\}_{j\in J}, \lambda, \mu, \phi)$.		

3.5 Improving Oruta Using rMHT

In this section, we show how to use the introduced rank-based Merkle Hash Tree (rMHT) to improve the Oruta protocol to prevent the replace attack and to support verifiable dynamic data operations.

Construction of rMHT. As shown in Fig. 3.8, the sequence-enforced authentic data of the rMHT are the identifier tags, which are the hashes of the data blocks.

Signature Generation. The global system parameters and the key generation algorithm are the same as those in the original Oruta scheme [WLL14]. For a block $m_j = \{m_{j,1}, ..., m_{j,k}\}$, the actual signer, denoted by u_s, computes

$$\beta_j = H_1(m_j) \prod_{l=1}^{k} \eta_l^{m_{j,l}} \in \mathbb{G}_1.$$

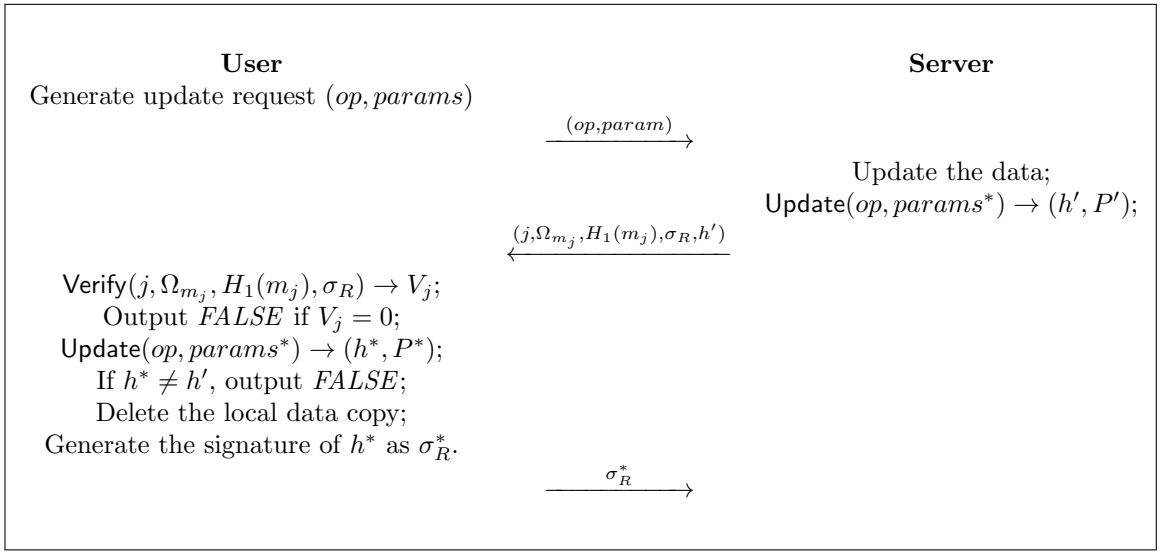
The other part of the ring signature generation for m_j follows the procedure in [WLL14]. Given all the d users' public keys $(pk_1, ..., pk_d) = (w_1, ..., w_d)$, u_s randomly chooses $a_i \in \mathbb{Z}_p$ for all $i \neq s$, where $i \in [1, d]$, and computes $\sigma_i = g_1^{a_i}$. Then he computes $\beta_R = H_1(R) \in \mathbb{G}_1$ for the root R and sets

$$\sigma_s = (\frac{\beta_R}{\psi(\prod_{i\neq s} w_i^{a_{j,i}})})^{1/x_s} \in \mathbb{G}_1.$$

Therefore, the ring signature for the root R is $\sigma_R = (\sigma_1, ..., \sigma_d) \in \mathbb{G}_1^d$. Then the user uploads σ_R, the data blocks, and corresponding ring signatures to the server and then deletes the local copy. It is worth noting that here we also use ring signature to sign the root R in order to preserve the identity privacy property of Oruta.

rMHT-Based Auditing. Table 3.4 describes the auditing protocol of the improved Oruta. After receiving the challenge query $\{j, y_j\}_{j\in J}$, the server computes

Table 3.5: Verifiable Dynamic Data Operations



the auditing proof λ, μ, ϕ in the same way of the original Oruta. The server also provides the public verifier with the identifier tags $\{H_1(m_j)\}_{j\in J}$ and the AAIs $\{\Omega_j\}_{j\in J}$ using the rMHT. In addition, the server also returns σ_R which is the ring signature for the root R of the rMHT. Therefore, the auditing proof for a challenge query $\{j, y_j\}_{j\in J}$ is $\{\lambda, \mu, \phi, \{H_1(m_j), \Omega_j\}_{j\in J}, \sigma_R\}$. Upon receiving the auditing proof from the server, for each $j \in J$, the verifier runs the algorithm Verify (Section 3.3.2) to verify $H_1(m_j)$. It is worth noting that according to its original definition, the inputs to the algorithm Verify should be $(j, H_1(m_j), \Omega_j, h_R)$ instead of $(j, H_1(m_j), \Omega_j, \sigma_R)$. However, such a modification does not affect the verification since the verifier will recompute h_R and uses σ_R to verify that the computed root value is correct. If the verification fails, the verifier rejects by emitting *FALSE*. Otherwise, the verifier runs the algorithm ProofVerify to check,

$$e(\prod_{j\in J} H_1(m_j)^{y_j} \cdot \prod_{l=1}^{k} \eta_l^{\mu_l}, g_2) \stackrel{?}{=} (\prod_{i=1}^{d} e(\phi_i, w_i)) \cdot e(\prod_{\lambda=1}^{k} \lambda_l^{h(\lambda_l)}, g_2)$$

If so, output *TRUE*, otherwise output *FALSE*. The correctness of the above equation can be easily obtained by following **Theorem 5** in [WLL14].

3.5.1 Verifiable Dynamic Data Operations Using rMHT

Here we show that the constructed rMHT in the improved Oruta can efficiently handle dynamic data operations including data modification (modify the j-th block m_j to m'_j), insertion (insert a new block m^* at the j-th position) and deletion

Algorithm 5 : $(h^*, P^*) = \mathsf{Update}\ (op, params^*)$

1: **if** $op = \mathcal{M}$ **then**
2: $(h^*, P^*) = \mathsf{Mod}\ (params^*)$;
3: **else if** $op = \mathcal{I}$ **then**
4: $(h^*, P^*) = \mathsf{Ins}(params^*)$;
5: **else if** $op = \mathcal{D}$ **then**
6: $(h^*, P^*) = \mathsf{Del}\ (params^*)$;
7: **end if**
8: **return** h^*, P^*;

Table 3.6: Description of Notations in Table 3.5

Operation	op	$params$	$params^*$
Modification	$\mathcal{M}$	(j, m'_j, σ'_j)	(j, m'_j, Ω_{m_j})
Insertion	$\mathcal{I}$	(j, m^*, σ^*)	$(j, v_{m_j}, m^*, \Omega_{m_j})$
Deletion	$\mathcal{D}$	(j)	(j, Ω_{m_j})

(remove the j-th block m_j). Moreover, the dynamic data operations on the server side can be verified by the user and hence our work solves the problem pointed out in [WLL14].

Table 3.5 shows the procedure of data updating and the algorithm Update (Algorithm 5) is for rebuilding the rMHT. The inputs of Update include $op \in \{\mathcal{M}, \mathcal{I}, \mathcal{D}\}$ where $\mathcal{M}, \mathcal{I}, \mathcal{D}$ represent the operations of modification, insertion and deletion respectively, and $params^*$ indicates the parameters for the above algorithms. To be more precise, when the server receives the requests $(op, params)$, it constructs the corresponding parameters $(op, params^*)$ for algorithm Update to rebuild the rMHT. The details of parameters are given in Table 3.6. To verify that the server has correctly rebuilt the rMHT, the user also runs the updating algorithm to get the new root value for comparison with the one sent by the server. Therefore, the dynamic data operations in the server are verifiable by the user. In order to make sure that the third party auditor later can also audit the updated data rather than old data (i.e., to prevent replay attack), the strategies given in Section 3.3.3 can be applied.

3.5.2 Batch Auditing Using rMHT

In order to make the auditor be able to handle multiple auditing tasks efficiently, Oruta is extended in [WLL14] to support batch auditing, which can verify the correctness of multiple auditing tasks simultaneously. Although we employ the rMHT for index checking in the improved Oruta protocol, it is easy to see that this modification still supports batch auditing due to the properties of the bilinear map. Therefore, we omit the description here and refer the readers to [WLL14] for more details.

3.5.3 Security of the Improved Oruta

In this chapter, we discuss the security properties of the improved Oruta. Since the improved Oruta mainly aims to address the security issues in the original Oruta by applying rMHT, the security proofs given in [WLL14] can be easily extended for the improved protocol. In particular, the ring signature on the rMHT root R is unforgeable following the analysis in [WLL14], which prevents the server from modifying the root value of the rMHT. Regarding the identity privacy, one can see that the improved Oruta can still preserve identity privacy since it performs ring signature on both data blocks and also the rMHT root. For the privacy of the audited data, since the rMHT construction uses the hashes of the data blocks as the leafs, the public verifier still cannot reveal any information of the audited data due to the one-way hash function.

It is easy to observe that the improved protocol is also secure against the replace attack as it uses rMHT for position checking during the verification of an auditing proof. For the replay attack, our protocol allows the user to verify that the server has correctly performed the dynamic data operations. However, to prevent the replay attack against a third-party auditor, the strategies described in Section 3.3.3 should be applied.

3.6 Performance Evaluation

In this section, we first analysis the computation and communication costs of the improved Oruta, and then evaluate the performance of Oruta in experiments.

3.6.1 Comparison With Oruta

We show the comparisons as follows.

Computation Cost. Due to the employment of the rMHT in the improved Oruta, the extra computation overhead in the new protocol compared with the original one in [WLL14] comes from the rMHT construction and the verification of the AAI information in an auditing task. Roughly speaking, the hash function evaluations are very efficient and hence can be neglected. Hence, the main computation overhead comes from the generation and verification of the ring signature on the rMHT root, which takes several exponentiation/pairing operations.

At the beginning of the protocol execution, the TPA chooses some random values to construct the auditing message and hence only introduces a small cost in computation. Upon receiving the challenge message, the cloud server needs to compute a proof $\{\lambda, \mu, \phi, \{H_1(m_j), \Omega_{m_j}\}_{j\in J}, \sigma_R\}$. The cost of this calculation is about

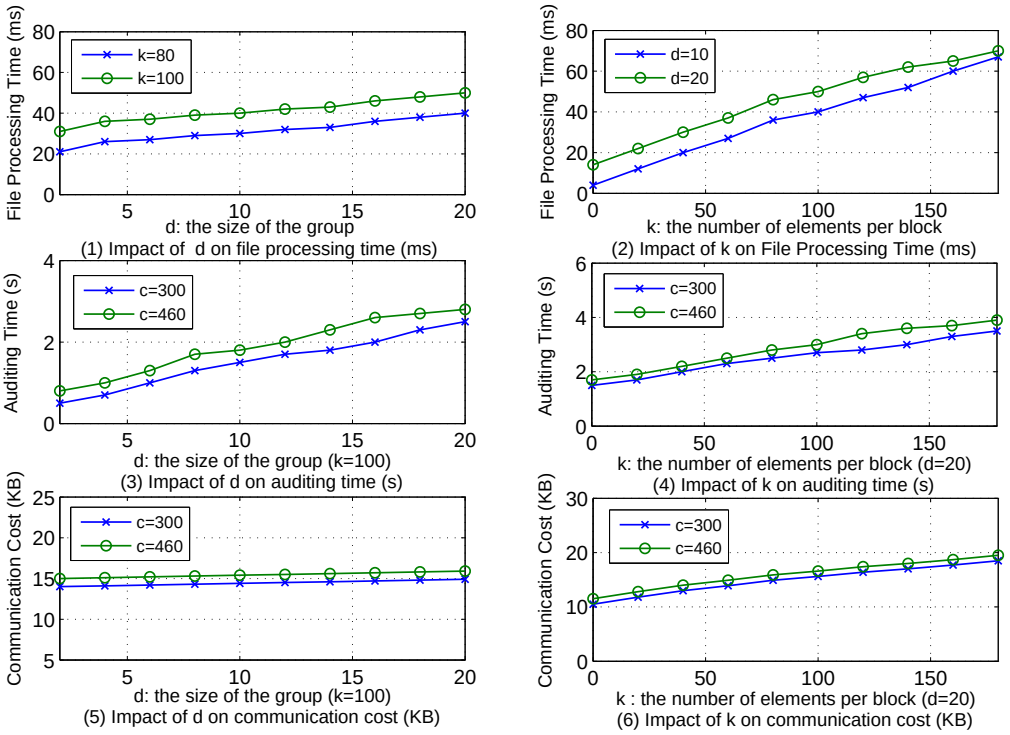


Figure 3.9: Experiment Results of the Performance of the Improved Oruta

$(k + dc)\text{Exp}_{\mathbb{G}_1} + dc\text{Mul}_{\mathbb{G}_1} + dc\text{Mul}_{\mathbb{Z}_P} + k\text{Hash}_{\mathbb{Z}_p}$ where $\text{Exp}_{\mathbb{G}_1}$ denote the computation of one exponentiation in $\mathbb{G}_1$, $\text{Mul}_{\mathbb{G}_1}$ denotes the costs of one multiplication in $\mathbb{G}_1$, $\text{Mul}_{\mathbb{Z}_p}$ and $\text{Hash}_{\mathbb{Z}_p}$ respectively denote the cost of one multiplication and one hashing operation in $\mathbb{Z}_p$. As for the verification, the total computation cost is $(2k + c)\text{Exp}_{\mathbb{G}_1} + (2k + c)\text{Mul}_{\mathbb{G}_1} + d\text{Mul}_{\mathbb{G}_T} + (c\log n)\text{Hash}_{\mathbb{G}_1} + (2d + 3)\text{Pairing}_{\mathbb{G}_1,\mathbb{G}_2}$ where $\text{Pairing}_{\mathbb{G}_1,\mathbb{G}_2}$ denotes one pairing computation in $\mathbb{G}_1$ and $\mathbb{G}_2$.

Communication Cost. Regarding the communication cost, compared with the original Oruta, the server needs to respond the values of the challenged nodes and their AAI, and a ring signature of the rMHT root, which contains d (the number of users sharing the file) elements in the group $\mathbb{G}_1$, in each auditing. According to the experimental results in [WWR+11] and [WLL14], the communication cost of the improvement is acceptable for both the cloud users and the cloud server. In particular, the communication cost of each auditing message is $c(|q|+|n|)$ bits, where $|q|$ is the length of an element of $\mathbb{Z}_q$ and $|n|$ is the length of an index. Each auditing proof consists of $(k+2d+c)$ elements of $\mathbb{G}_1$, k elements of $\mathbb{Z}_p$ and c elements of AAI . Therefore, the communication cost of one auditing proof is $(2k + 2d + c)|p| + c|v|$ bits where $|v|$ is the length of an AAI.

3.6.2 Experimental Results

To evaluate the efficiency of the improved Oruta in experiments, we also implement the protocols utilizing the GNU Multiple Precision Arithmetic (GMP) library and Pairing Based Cryptography (PBC) library. The following experiments are based on coding language C on Linux system (more precise, 2.6.35-22-generic version) with an Intel(R) Core(TM) 2 Duo CPU of 3.33 GHZ and 2.00-GB RAM. For the elliptic curve, we choose an MNT curve with a base filed size of 159 bits and p=160 bits and $|q|$=80 bits. We assume the total number of blocks, i.e., n= 1,000,000 and $|n|$=20 bits. The size of shared data is 2 GB. Suppose 1% of those blocks are lost or damaged, in order to detect these corrupted blocks with probability greater than 99% and 95%, we set the value of c=460 and c=300 respectively. This is according to the result shown in [ABC$^+$07]. We also assume the size of the group $d \in [2, 20]$ in our experiment analysis.

File Processing. Before uploading the file to the cloud server, a user needs to process his/her file by generating the ring signature of each block and computing the corresponding rMHT. The signature generation time is determined by the number of users (d) per group and the number of elements (k) per block while the construction time of rMHT is determined by the block number (n) of the file. As shown in Fig.3.9-(1) and Fig.3.9-(2), the file processing time increases with the size of the group when k is fixed and increases with the number of elements per block when d is fixed. Specifically, the file processing time is 40.12 milliseconds when $d = 20, k = 80$.

Auditing Time. As illustrated in Fig.3.9-(3), the auditing time is increasing with the size of the group and the number of challenged blocks. For example, suppose $d = 10, k = 100$, the auditing time is about 1.5 seconds when $c = 300$ and 1.8 seconds when $c = 460$. When the number of the group member increases to 20, the auditing time cost comes be about 2.5 seconds and 2.8 seconds respectively. When we fix the value of $d = 20$, the auditing time increases with the number of elements per block as shown in Fig.3.9-(4).

Communication. One can see from Fig.3.9-(3) that the communication cost consumed during the auditing protocol is very small compared to the size of the stored file (2 GB). More precisely, When the selected blocks $c = 300, d = 20$, the communication cost is 15 KB and comes to be 16 KB when $c = 460$. The communication cost of the auditing protocols increases with the the number of elements per block. It is worth noting that compared with the original Oruta [WLL14], the auditing task in the improved Oruta naturally consumes more communication cost as the proof consists of the leaf nodes and their AAI additionally.

3.7 Chapter Summary

In this chapter, we gave a formal treatment on Merkle Hash Tree for secure dynamic cloud auditing. We first revisited a well-known authentication structure named Merkle Hash Tree (MHT) and demonstrated how to extend its basic version to a sequence-enforced version that allows position checking. In order to support efficient and verifiable dynamic data operations, we further proposed a variant of MHT, named rank-based MHT (rMHT) that can be used to support verifiable dynamic data auditing. We also reviewed a cloud storage data auditing protocol named Oruta and showed that the protocol is vulnerable to replace and replay attacks. We then employed the proposed rMHT to fix the security problems in Oruta without sacrificing any desirable features of the protocol. It is of independent interest to find other security applications for rMHT.

Chapter 4

Block-Level Message-Locked Encryption for Secure Deduplicaiton

Deduplication is a popular technique widely used to save storage spaces in the cloud. To achieve secure deduplication of encrypted files, Bellare *et al.* formalized a new cryptographic primitive named *Message-Locked Encryption* (MLE) in Eurocrypt 2013. Although an MLE scheme can be extended to obtain secure deduplication for large files, it requires a lot of metadata maintained by the end user and the cloud server. In this chapter, we propose a new approach to achieving more efficient deduplication for (encrypted) large files. Our approach, named *Block-Level Message-Locked Encryption* (BL-MLE), can achieve file-level and block-level deduplication, block key management, and proof of ownership simultaneously using a small set of metadata. We also show that our BL-MLE scheme can be easily extended to support data auditing, which makes it multi-purpose for secure cloud storage.

4.1 Introduction

According to the architecture and the granularity of data processing, deduplication strategies can be mainly classified into the following types. In terms of deduplication granularity, there are two main deduplication strategies. (1) *File-level deduplication*: the data redundancy is exploited on the file level and thus only a single copy of each file is stored on the server. (2) *Block-level deduplication*: each file is divided into blocks, and the server exploits data redundancy at the block level and hence performs a more fine-grained deduplication. It is worth noting that for block-level deduplication, the block size can be either fixed or variable in practice, and each method has its advantages and disadvantages [Ded]. In this work, we focus on the block-level deduplication with fixed block size. From the perspective of deduplication architecture, there are also two strategies. (1) *Target-based deduplication*: users are unaware of any deduplication that might occur to their outsourced files. They just upload the files to the data storage server which then performs deduplication upon receiving the data. (2) *Source-based deduplication:* unlike target-based deduplication, before uploading the data, the user first sends an identifier/tag of the data (e.g., a hash value of the data and thus much shorter) to the server for redundancy checking and thus duplicated data would not be sent over the network.

Large File Deduplication. In this chapter, we focus on large file deduplication.

Normally block-level deduplication can provide more space savings than file-level deduplication does in large file storage. Taking as an example, Alice and Bob want to store the same large file M in a server. Suppose the server performs file-level deduplication, which means only one copy of M will be saved. Later, Bob downloads M, appends several new pages to it, and uploads the modified file (denoted by M') to the server. Since M' is different from M, the server needs to store the whole file M'. However, if block-level deduplication is used, the server only needs to store the appended pages (denoted by ΔM), reducing the space cost from $\mathcal{O}(|M| + |M'|)$ to $\mathcal{O}(|M| + |\Delta M|)$. This approach can bring a significant space saving since $|\Delta M| \ll |M'|$. One drawback of the more fine-grained block-level deduplication is that it requires more processing resources. For some storage systems, e.g., backup systems where there are many duplicated files, it is computationally inefficient to perform block-level deduplication since the server has to exploit redundancy per block and thus requires extensive processing resources especially for large files which contain numerous blocks. Fortunately, file-level deduplication and block-level deduplication are not incompatible with each other. In this chapter, we present a technique that can achieve both of them (i.e., dual-level deduplication).

Another aspect that should be taken into consideration is the bandwidth savings from large file deduplication. It has been reported that the cost of transferring data is almost the same as the space cost of storing the same amount of data for two months in the Amazon S3 server [HPS10]. Since uploading large files would consume extensive bandwidth, source-based deduplication seems to be a better choice for large file outsourcing. Unlike target-based deduplication which requires users to upload their files regardless of the potential data redundancy among those files, source-based deduplication could save the bandwidth significantly by eliminating the retransmission of duplicated data. Therefore, in contrast to target-based deduplication which saves only space, source-based deduplication can, in addition, save network bandwidth, which makes it more attractive in large file deduplication. However, source-based deduplication also has a drawback. A dishonest user who has learnt a piece of information about a file may claim that he/she owns the file. Such a problem has been identified by Halevi *et al.* in [HHPS11]. To overcome such an attack, they proposed a new notion called *Proof of Ownership* (PoW) where the user proves to the server that he/she indeed owns the entire file. It is clear that PoW should be implemented along with source-based deduplication. In the rest of the chapter, we consider PoW as a default component in source-based deduplication.

From the above analysis, we can see that it is desirable to have **Dual-Level Source-Based (DLSB) Deduplication** for large files. Such a mechanism can achieve the best savings on space, computation, and bandwidth. *In a DLSB Deduplication system, the user firstly sends a file identifier to the server for file redun-*

dancy checking. If the file to-be-stored is duplicated in the server, the user should convince the server that he/she indeed owns the file by performing a PoW protocol. Otherwise, the user uploads the identifiers/tag of all the file blocks to the server for block-level deduplication checking. Finally, the user uploads data blocks which are not stored on the server.

Data Privacy. In the discussions above, we haven't considered data privacy issues. In reality, end users may not entirely trust the cloud storage servers. In order to protect data privacy, files may be encrypted first before being uploaded to the server. This brings a new challenge for deduplication since different users will use different keys to perform encryption, which would make two identical files completely different after being encrypted. Although searchable encryption [BCOP04,SWP00,YTHW10] can support equality testing of encrypted data, cloud storage providers still cannot perform any deduplication. Specifically, the work in [BCOP04] only considers the keyword search on the encrypted data for the same user and hence cannot be applied for deduplication of data from different users. A similar drawback exists for the work in [SWP00] which supports symmetric-key searchable encryption and is not suitable for cross-user deduplication. While some other works, e.g., [BBO07, YTHW10], support equality testing of data from different users, they still do not meet the requirement of deduplication. The main reason is that, if a user (say Bob) does not store his encrypted file on the server due to deduplication, e.g., another user Alice has stored the same file in the server, then Bob could not retrieve the original file later since he cannot decrypt Alice's file.

4.1.1 Motivations

To resolve the above problem, Douceur *et al.* [DAB+02] proposed a solution called *Convergent Encryption* (CE). CE is a deterministic symmetric encryption scheme in which the key K is derived from the message M itself by computing $K = H(M)$ and then encrypting the message as $C = E(K, M) = E(H(M), M)$ where H is a cryptographic hash function and E is a block cipher. Using CE, any user holding the same message will produce the same key and ciphertext, enabling deduplication. Although CE and its variants have been widely deployed in many systems [ABC+02, SGLM08, WW08], a formal treatment on their security is missing.

In Eurocrypt'13, Bellare *et al.* [BKR13] formalized a new cryptographic primitive called *Message-Locked Encryption* (MLE) which subsumes Convergent Encryption. Although MLE schemes can perform secure deduplication of encrypted data, they were proposed originally for file-level and target-based deduplication. We could extend an MLE scheme for secure DLSB-deduplication of large files by performing MLE on each data block (i.e., treating a data block as a file) and employing an

existing Proof of Ownership scheme (e.g., the PoW scheme in [HHPS11]). However, as we will show shortly, such an approach is not efficient due to the large amount of metadata produced in order to achieve all the security goals. Different from the files that are stored in a secondary storage, the metadata is usually stored in the primary memory for fast access upon every upload request. However, since the primary memory storage cost is more expensive than that of secondary storage, the additional cost by the metadata storage could be very high even if it is not so large compared to the outsourced files.

Metadata I: Block Identifiers. In DLSB-deduplication, in addition to the file identifier, the server also has to store a large number of block identifiers for redundancy checking. Although a block identifier is much shorter than the corresponding data block, the overall storage costs can be significant due to the large amount of data blocks in large files.

Metadata II: Block Keys. To apply MLE at the block level, each file block should be encrypted using a *block key* which is derived from the data block itself. Therefore, the number of block keys scales linearly with the number of data blocks. The user has to maintain a lot of block keys for decryption. The block key management hence is a challenge for the user especially when the outsourced files are large. A simple solution to solve the problem is that each user encrypts all the block keys with a *master key* and uploads both the encrypted block keys together with the encrypted file to the server. In this way, each user only needs to keep the master key locally which can be used to recover the block keys and hence the encrypted data blocks. However, one drawback of such a block key management mechanism is that the server requires more space to store the (extra) encrypted block keys in addition to the block identifiers. The drawback seems inherent since we cannot apply deduplication on the encrypted block keys when the master keys are different. Such an observation motivated us to tackle the problem from a different angle by encapsulating the block key inside the block identifier, which allows us to apply the above block key management mechanism without introducing extra storage overhead.

Metadata III: PoW Tags. As we have mentioned above, it is a requirement to perform the Proof of Ownership protocol in source-based deduplication, especially for large files. A trivial solution for PoW is to request the prover (i.e., users) to upload some random data blocks specified by the server (i.e., spot-checking). However, as discussed above, since the actual data blocks are stored in the secondary storage, it is more practical to use some short PoW tags to perform the verification. Similar to the encrypted block keys, the PoW tags would also require extra spaces on the data server.

To give a clearer picture, we take the practical MLE scheme **HCE2**, introduced

by Bellare *et al.* [BKR13], as an example to demonstrate the cost of performing DLSB-deduplication. Given the public parameters *params* and a file $\mathbf{M}$, the user first computes the file identifier $T_0 \leftarrow H(P, H(P, \mathbf{M}))$ by applying a cryptographic hash function H, and then separates $\mathbf{M}$ into n blocks, i.e., $\mathbf{M} = \mathbf{M}[1]||...||\mathbf{M}[n]$. For each block $\mathbf{M}[i]$, the user computes the corresponding block key as $k_i \leftarrow H(P, \mathbf{M}[i])$, block identifier as $T_i \leftarrow H(P, k_i)$ and then encrypts each block as $\mathbf{C}[i] \leftarrow \mathcal{SE}(P, k_i, \mathbf{M}[i])$ using a symmetric encryption algorithm $\mathcal{SE}$. In addition, the block keys will also be encrypted and uploaded to the server. Let C_{k_i} be the encrypted form of the block key k_i using the user's master key. Therefore, the server needs to store the metadata of size $\mathcal{O}(|T_0| + \sum_i^n |T_i| + \sum_i^n |C_{k_i}| + |T_{PoW}|)$ for a file $\mathbf{M}$ where T_{PoW} is the tag for Proof of Ownership. This would result in a significant space cost when the file (or block number) is large. Among all the metadata, we can see that the block identifiers and the encrypted block keys form the major storage overhead. This motivated us to design a new scheme that can combine the block identifier T_i and the encrypted block key C_{k_i} into one single element. More details are provided in Section 4.6, where we compare several MLE-based schemes with our proposed BL-MLE scheme for DLSB-deduplication.

Noting that all the schemes following the MLE framework are inherently subject to the brute-force attacks that can recover files falling into a known set, Bellare *et al.* [KBR13] proposed an architecture that provides secure deduplicated storage resisting brute-force attacks, and realized it in a system called DupLESS. It is worth noting that their motivation is different from that of this work. Precisely, their system aims at providing a more secure and easily-to-deploy solution for MLE schemes while we focus on reducing the metadata size for DLSB-deduplication. Nevertheless, we should note that their solution is compatible with our scheme and hence can also be used to enhance the security of our scheme.

4.1.2 Contributions

In this chapter, we formalize the notion of *Block-Level Message-Locked Encryption* (**BL-MLE**) for secure and space-efficient large file deduplication in cloud storage. We remark that our proposed notion is the first block-level message-locked encryption that achieves dual-level deduplication.

1. We propose the formal definition of BL-MLE which captures new functionalities, i.e., dual-level deduplication, block key management and proof of ownership, than the conventional MLE. Security models are also defined separately for each functionality to capture clear security guarantees.

2. We present a concrete BL-MLE scheme that can efficiently realize our design

ideas outlined above. Moreover, we also show that our BL-MLE scheme can be easily modified to support efficient data auditing and obtain an improved PoW protocol with stronger security, which makes our scheme multi-purpose for secure cloud storage.

3. We show that our proposed scheme can indeed achieve significant space savings and PoW bandwidth savings. We also fully prove the security of the constructed scheme under the proposed security models.

From the above analysis, we know that for an extended **HCE2**, the space of the total metadata for DLSB-deduplication is $\mathcal{O}(|T_0| + \sum_i^n |T_i| + \sum_i^n |C_{k_i}| + |T_{PoW}|)$ for a file consisting of n blocks, which however, can be reduced to $\mathcal{O}(|T_0| + \sum_i^n |T_i^*|)$ by using our scheme where $\{T_i^*\}_{i\in[1,n]}$ are the multi-purpose tags. When there are u users that store similar but different files on the server, Suppose the similarity among the u files is δ, the metadata space cost of the extended HCE2 is $\mathcal{O}(u|T_0|+(1+(u-1)(1-\delta)\sum_i^n |T_i| + u\sum_i^n |C_{k_i}| + u|T_{PoW}|)$ while that of our scheme is $\mathcal{O}(u|T_0| + u\sum_i^n |T_i^*|)$. We should note that for both approaches it is infeasible to perform deduplication on the encrypted block keys since we cannot derive a common master key for block key encryption when the files are different.

4.1.3 Related Work

Message-Locked Encryption. According to [BKR13], a standard message-locked encryption scheme consists of teh following five algorithms.

$\mathsf{Setup}(1^\ell)$. Takes 1^ℓ, returns a public parameter $params$;

$\mathsf{KeyGen}(P, M)$. Takes $params$ and a message M, returns a message-derived key K;

$\mathsf{Enc}(P, K, M)$. Takes $params$, key K and message M, returns a ciphertext C;

$\mathsf{Dec}(P, K, C)$. Takes $params$, key K and ciphertext C, returns a message M;

$\mathsf{TagGen}(P, C)$. Takes $params$ and ciphertext C, returns a tag T.

An MLE scheme is a standard symmetric-key encryption scheme which uses a deterministic function to map a message to an encryption key. An MLE scheme also has a tag generation algorithm that derives a tag from a ciphertext. The tag serves as an *identifier* of the message for equality test. Identical data always result in equal tags regardless of the ciphertext which can be randomized. Apart from the formal definition of MLE, they also proposed the formal security definitions, including *privacy* and *tag consistency*, to capture the security requirements on MLE. Due to

the special key generation mechanism, no MLE scheme can achieve the conventional IND-type security. Bellare *et al.* then defined a new privacy model which captures chosen distribution attacks for MLE schemes. For tag consistency, it means the message indicated by a tag must be consistent with that underlying the ciphertext. In Crypto'13, Abadi *et al.* [ABM+13] proposed a stronger notion of MLE. However, the scheme proposed in [ABM+13] is less efficient than the original schemes proposed by Bellare *et al.* [BKR13].

Proof of Ownership. Proof-of-Ownership (PoW) [HHPS11] is an interactive protocol between a prover (file owner) and a verifier (data server). By executing the protocol, the prover convinces the verifier that he/she is an owner of a file stored by the verifier. As mentioned earlier, PoW is necessary for source-based deduplication. Here, we briefly describe the PoW protocol in [HHPS11] which presents three schemes that differ in terms of security and performance. All three require both the user and the server to build the Merkle trees [Mer87] on a buffer, whose content is derived from the pre-processed file. The server only keeps root and challenges the client to present valid sibling paths for a subset of leaves of the Merkle tree. Therefore, the bandwidth costs would be a super-logarithmic number of sibling paths of the Merkle tree.

4.2 Block-Level Message-Locked Encryption

We now describe the definition of *Block-Level Message-Locked Encryption* (BL-MLE) for DLSB-deduplication. It will capture additional functionalities including dual-level deduplication, block keys management, and proof of ownership, compared with a normal file-level MLE.

4.2.1 Definition of BL-MLE

A block-level message-locked encryption scheme is defined as follows:

$\mathsf{Setup}(1^\ell)$. Takes a security parameter ℓ as input and returns the parameters $params$.

$\mathsf{KeyGen}(params, M)$. Takes the public parameters $params$ and a file message $\mathbf{M} = \mathbf{M}[1]||...||\mathbf{M}[n]$ as input, and returns a master key k_{mas} and block keys $\{k_i\}_{1\leq i\leq n}$ generated using the following two sub-algorithms respectively,

- $\mathsf{M\text{-}KeyGen}(params, \mathbf{M})$. Takes $params$ and $\mathbf{M}$ as input, returns the master key k_{mas};
- $\mathsf{B\text{-}KeyGen}(params, \mathbf{M}[i])$. Takes $params$ and $\mathbf{M}[i]$ as input, returns the block key k_i.

$\mathsf{Enc}(params, \mathbf{M}[i], k_i)$. Takes public parameters $params$, a block message $\mathbf{M}[i]$ and the corresponding block key k_i as input, returns the block ciphertext $\mathbf{C}[i]$.

$\mathsf{Dec}(params, \mathbf{C}[i])$. Takes public parameters $params$, a block ciphertext $\mathbf{C}[i]$ and a block key k_i as input, returns a block message $\mathbf{M}[i]$ or $\perp$.

$\mathsf{TagGen}(params, \mathbf{M})$. Takes public parameters $params$ and a file $\mathbf{M}$ as input, returns the file tag T_0 and block tags $\{T_i\}_{1 \leq i \leq n}$ generated using the following two sub-algorithms respectively,

- $\mathsf{F\text{-}TagGen}(params, \mathbf{M})$. Takes $params$ and $\mathbf{M}$ as input, returns the file tag T_0;
- $\mathsf{B\text{-}TagGen}(params, \mathbf{M}, i)$. Takes $params$, $\mathbf{M}$ and the block index i as input, returns the block tag T_i.

$\mathsf{ConTest}(T_i, \mathbf{C}[i])$. Takes a block tag T_i and a block ciphertext $\mathbf{C}[i]$ as input, returns *True* or *False*.

$\mathsf{EqTest}(T, T', T_0, T_0')$. Takes as input two block tags T, T' and the corresponding file tags T_0, T_0', returns *True* or *False*.

$\mathsf{B\text{-}KeyRet}(k_{mas}, T_i, \mathbf{C}[i])$. Takes a master key k_{mas}, a block tag T_i, and a block ciphertext $\mathbf{C}[i]$ as input, returns a block key k_i / $\perp$.

$\mathsf{PoWPrf}(Q, \mathbf{M})$. Takes a challenge Q and a file $\mathbf{M}$ as input, returns a response $\mathcal{P}$.

$\mathsf{PoWVer}(Q, T_0, \{T_i\}_{1 \leq i \leq n}, \mathcal{P})$. Takes a challenge Q, the file tag T_0, the block tags $\{T_i\}_{1 \leq i \leq n}$, and the response $\mathcal{P}$ as input, returns *True* or *False*.

Remark I. Unlike a standard MLE scheme, our BL-MLE scheme performs encryption block by block. Given a file, we split it into *blocks* before encryption. The KeyGen algorithm produces the *master key* and *block keys* using the sub-algorithm M-KeyGen and B-KeyGen respectively. The former is used to encrypt block keys while the latter are derived from the file blocks and used to encrypt and decrypt block messages. For the tag generation algorithm TagGen, we also define two sub-algorithms, F-TagGen and B-TagGen. The *file tag* can be used as the file identifier for file-level deduplication, and the *block tag* severs multi-purposes, including block identifier, encrypted block key, and PoW tag.

To achieve DLSB-deduplication, equality testing of data block can be done with block identifiers using the algorithm EqTest. Therefore, we need to ensure that the block identifier (i.e., block tag) is actually consistent with the uploaded data block to prevent *duplicate faking attack*. The algorithm ConTest is introduced for this purpose. As the block tag also serves as an encrypted block key, we define the

algorithm B-KeyRet for block key retrieval. Given a block tag, the corresponding block ciphertext and the master key, the user can compute the block key and then decrypt the block ciphertext.

Also, we require that the block tags constructed in a BL-MLE scheme can be used as PoW tags. Here we introduce two algorithms PoWPrf and PoWVer for proof of ownership. The algorithm PoWPrf outputs the response to a challenge based on the data file and PoWVer is used to verify whether the response is correct or not.

Correctness. For BL-MLE, except the file space $\mathrm{MsgSp}_{BL\text{-}MLE}(\ell)$, we also define the block space $\mathrm{BlSp}_{BL\text{-}MLE}(\ell)$ for any $\ell \in \mathbb{N}$. The following correctness conditions are required for a BL-MLE. For all $\ell \in \mathbb{N}$, $params \leftarrow \mathsf{Setup}(1^\ell)$ and all $\mathbf{M} \in \mathrm{MsgSp}_{BL-MLE}(\ell)$, we require the following correctness.

1. *Decryption Correctness.* For all block message $\mathbf{M}[i] \in \mathrm{BlSp}_{BL\text{-}MLE}(\ell)$, block key $k_i \leftarrow \mathsf{B\text{-}KeyGen}(\mathbf{M}[i])$[a] and block ciphertext $\mathbf{C}[i] \leftarrow \mathsf{Enc}(k_i, \mathbf{M}[i])$, we have that, $\mathsf{Dec}(k_i, \mathbf{C}[i]) = \mathbf{M}[i]$;

2. *Tag Correctness.* For any two block message $\mathbf{M}[i], \mathbf{M}'[t] \in \mathrm{BlSp}_{BL\text{-}MLE}(\ell)$ such that $\mathbf{M}[i] = \mathbf{M}'[t]$, block key $k_i \leftarrow \mathsf{B\text{-}KeyGen}(\mathbf{M}[i])$, block ciphertext $\mathbf{C}[i] \leftarrow \mathsf{Enc}(k_i, \mathbf{M}[i])$, block tag $T_i \leftarrow \mathsf{B\text{-}TagGen}(\mathbf{M}, i)$ and $T'_t \leftarrow \mathsf{B\text{-}TagGen}(\mathbf{M}', t)$, we have, $\Pr[\mathsf{ConTest}(T_i, \mathbf{C}[i]) = True] = 1$ and $\Pr[\mathsf{EqTest}(T_i, T'_t) = True] = 1$;

3. *B-Key-Retrieval Correctness.* For any block message $\mathbf{M}[i] \in \mathrm{BlSp}_{BL\text{-}MLE}(\ell)$, master key $k_{mas} \leftarrow \mathsf{M\text{-}KeyGen}(\mathbf{M})$, block key $k_i \leftarrow \mathsf{B\text{-}KeyGen}(\mathbf{M}[i])$, block ciphertext $\mathbf{C}[i] \leftarrow \mathsf{Enc}(k_i, \mathbf{M}[i])$ and block tag $T_i \leftarrow \mathsf{B\text{-}TagGen}(\mathbf{M}, i)$, we have that, $\mathsf{B\text{-}KeyRet}\ (k_{mas}, T_i, \mathbf{C}[i]) = k_i$;

4. *PoW Correctness.* For all tags $\mathbf{T} \leftarrow \mathsf{TagGen}(\mathbf{M})$, any challenge Q, $\mathcal{P} \leftarrow \mathsf{PoWPrf}(\mathbf{M}, Q)$, we have that, $\Pr[\mathsf{PoWVer}(\mathbf{T}, \mathcal{P}) = True] = 1$.

4.2.2 Security Definitions for BL-MLE

Guessing Probability. Given a random variable X with min-entropy $\mathbf{H}_\infty(X) = -\log(\mathrm{Max}_x \Pr[X = x])$, the guessing probability of X is $\mathbf{GP}(X) = \mathrm{Max}_x \Pr[X = x] = 2^{-\mathbf{H}_\infty(X)}$. Given a random variable Y, the conditional guessing probability $\mathbf{GP}(X|Y)$ of a random variable X with conditional min-entropy $\mathbf{H}_\infty(X|Y)$ is $\mathbf{GP}(X|Y) = \sum_y \Pr[Y = y] \cdot \mathrm{Max}_x \Pr[X = x|Y = y] = 2^{-\mathbf{H}_\infty(X|Y)}$.

Unpredictable Block-Source. A block-source is a polynomial time algorithm $\mathcal{M}$ that on input 1^ℓ returns $(\mathbf{M}, Z)$ where $\mathbf{M}$ is a message vector over $\{0,1\}^*$ and $Z \in \{0,1\}^*$ denotes some auxiliary information. Let $n(\ell)$ denote the vector length, i.e.,

[a]For simplicity, we will omit *params* in the input of the algorithms in the rest of the chapter.

the number of blocks. For all $i \in [1, n(\ell)]$, $\mathbf{M}[i]$ represents the i^{th} block of the message $\mathbf{M}$. We say $\mathcal{M}$ is an *unpredictable block-source* if $\mathbf{GP}_{\mathcal{M}} = \mathrm{Max}_i\{\mathbf{GP}(\mathbf{M}[i]|Z)\}$ is negligible.

In this section, we formalize the security definitions for BL-MLE schemes. In a BL-MLE scheme, a large file is split into blocks before being encrypted, hence apart from the requirement that the message is unpredictable, each block should also be unpredictable when considering the privacy of block encryption. Therefore, we only consider *unpredictable block-source* in this chapter (i.e., $\mathbf{GP}_{\mathcal{M}}$ is negligible).

Privacy. Similar to the MLE scheme, our BL-MLE scheme cannot achieve the conventional semantic security due to the special key generation mechanism. For MLE, Bellare *et al.* [BKR13] proposed PRV$-CDA which is a strong privacy notion where the encryption of an unpredictable message must be indistinguishable from a random string of the same length [RBBK01].

In this chapter, we follow the idea in [BKR13] to define the privacy model. Here, we modify the notion PRV$-CDA slightly for BL-MLE (denoted by PRV$-CDA-B) as it produces file tag and block tags separately from the ciphertext. We say a BL-MLE scheme is secure under chosen distribution attacks if no polynomial-time adversary $\mathcal{A}$ has a non-negligible advantage in the following PRV$-CDA-B game:

Setup. The adversary $\mathcal{A}$ sends the challenger the description of a unpredictable block-source $\mathcal{M}$. The challenger then generates and sends $\mathcal{A}$ the system parameter $params$.

Challenge. The challenger picks randomly $b \leftarrow \{0, 1\}$. If $b = 0$, then runs the source $\mathcal{M}$ as, $(\mathbf{M}_0, Z) \leftarrow \mathcal{M}(\ell)$. Otherwise, if $b = 1$, chooses $\mathbf{M}_1$ uniformly at random from $\{0, 1\}^{|\mathbf{M}_0|}$. Set $\mathbf{M} = \mathbf{M}_b$. Suppose $n(\ell)$ is the block numbers. For each $i = 1, ..., n(\ell)$, the challenger computes $k_i \leftarrow$B-KeyGen$(\mathbf{M}[i])$ and then computes the ciphertext as, $\mathbf{C}[i] \leftarrow \mathsf{Enc}(k_i, \mathbf{M}[i])$. The challenger also computes the file tag and block tags as follows, $T_0 \leftarrow$F-TagGen$(\mathbf{M}), T_i \leftarrow$B-TagGen$(\mathbf{M}[i])$. Set $\mathbf{T} = \{T_0, T_1, ..., T_{n(\ell)}\}$. Finally, the challenger gives auxiliary information Z, tags $\mathbf{T}$, and the ciphertext $\mathbf{C}$ to the adversary.

Output. After receiving $(\mathbf{C}, \mathbf{T}, Z)$, the adversary outputs his guess b' on b and wins the game if $b' = b$.

We refer to such an adversary $\mathcal{A}$ as a PRV$-CDA-B adversary and define adversary $\mathcal{A}$'s advantage as,

$$\mathsf{Adv}_{\mathcal{A}}^{\text{PRV\$-CDA-B}}(\ell) = |\Pr[b = b'] - \frac{1}{2}|.$$

Definition 4.1 *We say that a BL-MLE scheme is* PRV$-CDA-B *secure if for any unpredictable block source* $\mathcal{M}$ *and any polynomial-time* PRV$-CDA-B *adversary* $\mathcal{A}$*, the advantage in the chosen distribution attack game,* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{PRV\$\text{-}CDA\text{-}B}}(\ell)$*, is negligible.*

Tag Consistency. To prevent *duplicate faking attack*, we need to check the consistency of the block tag and corresponding ciphertext. The TC notion proposed in [BKR13] can only ensure that an honest user can detect corruption. STC (strong tag consistency), on the other hand, can ensure that the adversary cannot fool the server to erase honestly generated ciphertexts and hence data owner can recover the original message. However, the notions proposed in [BKR13] are only for *deterministic* tags.

For BL-MLE, we define a similar security notion as STC by following the general definition, STC2 in [ABM+13] since we also consider randomized tags. Therefore, instead of comparing tag value directly, our definition uses ConTest and EqTest algorithms to check the tag consistency. We say a BL-MLE scheme is secure under duplicate faking attack (DFA) if no polynomial-time adversary $\mathcal{A}$ has a non-negligible advantage in the following DFA game:

Setup. The challenger generates and sends $\mathcal{A}$ all the system parameters *params*.

Output. Eventually, $\mathcal{A}$ outputs $< \mathbf{M}^*, i, c^*, T^* >$. If $\mathsf{ConTest}(T^*, c^*) \rightarrow$ False, output 0; Otherwise, if $\mathbf{M}^*[i] \neq \mathsf{Dec}(\text{B-KeyGen}(\mathbf{M}^*[i]), c^*)$,$\mathsf{EqTest}(\text{B-TagGen}(\mathbf{M}^*[i]), T^*) \rightarrow True$, output 1.

We refer to such an adversary $\mathcal{A}$ as a DFA adversary and define adversary $\mathcal{A}$'s advantage as $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{DFA}}(\ell)$ in the above game as the probability the game outputs 1.

Definition 4.2 *We say a BL-MLE scheme is* DFA*-secure if for any polynomial-time* DFA *adversary* $\mathcal{A}$*, the advantage* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{DFA}}(\ell)$ *is negligible.*

PoW Security. As for the security of proof-of-ownership, similar to the security definition in [18], we consider the probability that an attacker who knows *partial information* about the file can convince the server that he/she owns the entire file. Based on the idea of the "bounded retrieval model" [CLW06, Dzi06], we assume that the attacker only knows partial information (a bounded number of blocks) of the file. We say a BL-MLE scheme is secure against an uncheatable chosen distribution attack if no polynomially bounded adversary $\mathcal{A}$ has a non-negligible advantage against the challenger in the following UNC-CDA game:

Setup. The challenger generates and sends $\mathcal{A}$ the system parameters *params*.

Challenge. The adversary sends challenger the BL-MLE-valid source $\mathcal{M}$. And the challenger runs $\mathcal{M}$ as $(M, Z) \leftarrow \mathcal{M}(\ell)$ and sends the proof query $Q = (i, v_i)$ with the auxiliary information Z to the adversary.

Output. Finally, the adversary outputs the proof $\mathcal{P}^*$, which passes the verification, i.e., $\mathsf{PoWVer}(\mathsf{TagGen}\ (M), \mathcal{P}^*, Q) \rightarrow True$. Let the expected honest response be $\mathcal{P}$, i.e.,$\mathsf{PoWPrf}(M, Q) \rightarrow \mathcal{P}$. If $\mathcal{P}^* \neq \mathcal{P}$, the challenger outputs 1, otherwise output 0.

We refer to such an adversary $\mathcal{A}$ as an UNC-CDA adversary and define adversary $\mathcal{A}$'s advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{UNC\text{-}CDA}}(\ell)$ as the probability that the game outputs 1.

Definition 4.3 *We say that a BL-MLE scheme is* UNC-CDA *secure if for any unpredictable block-source $\mathcal{M}$ and any polynomial time* UNC-CDA *adversary $\mathcal{A}$, the advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{UNC\text{-}CDA}}(\ell)$ is negligible.*

4.3 The proposed BL-MLE Scheme

In this section, we adapt the BL-MLE framework and introduce the proposed BL-MLE scheme in detail.

4.3.1 Construction

$\mathsf{Setup}(1^\ell)$. On input 1^ℓ, the algorithm generates a prime $params$, the descriptions of two groups $\mathbb{G}_1, \mathbb{G}_\tau$ of order $params$, a generator g of $\mathbb{G}_1$ and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_\tau$. Choose an integer $s \in \mathbb{N}$ and three hash function $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p$, $H_2 : \{\mathbb{Z}_p\}^s \rightarrow \mathbb{G}_1$, $H_3 : \mathbb{G}_1 \rightarrow \{\mathbb{Z}_p\}^s$. Pick s elements randomly $u_1, u_2, ..., u_s \overset{R}{\leftarrow} \mathbb{G}_1$. The system parameters are $params =< p, g, \mathbb{G}_1, \mathbb{G}_\tau, e, H_1, H_2, H_3, s, u_1, u_2, ..., u_s >$.

$\mathsf{KeyGen}(\mathbf{M})$. Given a data file $\mathbf{M} = \mathbf{M}[1]||...||\mathbf{M}[n]$ where for all $1 \leq i \leq n$, $\mathbf{M}[i] \in \{\mathbb{Z}_p\}^s$, compute the master key k_{mas} and each block key k_i as follows,

$\mathsf{M\text{-}KeyGen}(\mathbf{M})$. take M, output $k_{mas} = H_1(\mathbf{M})$;

$\mathsf{B\text{-}KeyGen}(\mathbf{M}[i])$. take $\mathbf{M}[i]$, output $k_i = H_2(\mathbf{M}[i])$.

$\mathsf{Enc}(k_i, \mathbf{M}[i])$. Given a block message $\mathbf{M}[i]$, and the corresponding block key k_i, output the block ciphertext as

$$\mathbf{C}[i] = H_3(k_i) \oplus \mathbf{M}[i].$$

$\mathsf{Dec}(k_i, \mathbf{C}[i])$. Given a block ciphertext $\mathbf{C}[i]$, and the corresponding block key k_i, compute

$$\mathbf{M}[i] = H_3(k_i) \oplus \mathbf{C}[i].$$

If $k_i = H_2(\mathbf{M}[i])$, output $\mathbf{M}[i]$; otherwise output $\perp$.

$\mathsf{TagGen}(\mathbf{M})$. Given the file $\mathbf{M} = \mathbf{M}[1]||...||\mathbf{M}[n]$, output the file tag T_0 and each block tag T_i as follows,

$\mathsf{M\text{-}TagGen}(\mathbf{M})$. take $\mathbf{M}$, generate the master key k_{mas}, output $T_0 = g^{k_{mas}}$;

$\mathsf{B\text{-}TagGen}(\mathbf{M}, i)$. take $\mathbf{M}$ and the block index i, generate the master key k_{mas}, the corresponding block key k_i and block ciphertext $\mathbf{C}[i]$, split $\mathbf{C}[i]$ into s sectors: $\{\mathbf{C}[i][j]\}_{1\leq j\leq s}$, and output,

$$T_i = (k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{k_{mas}}.$$

In our scheme, some auxiliary data $aux_i = e(k_i, T_0)$ is also generated and attached to the block tag T_i during block tag generation. Please refer to Section 4.3.2 for some discussions on T_i and aux_i.

$\mathsf{ConTest}(T_i, \mathbf{C}[i])$. Given a block ciphertext $\mathbf{C}[i]$ and the block tag T_i with auxiliary data aux_i, split $\mathbf{C}[i]$ into s sectors: $\{\mathbf{C}[i][j]\}_{1\leq j\leq s}$. Let T_0 be the corresponding file tag. Check whether,

$$e(T_i, g) \stackrel{?}{=} aux_i \cdot e(\prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]}, T_0).$$

If so, output 1; otherwise, output 0.

$\mathsf{EqTest}(T_i, T_i', T_0, T_0')$. Given two block tags T_i, T_i' and the corresponding file tags T_0, T_0', check whether $e(T_i, T_0') \stackrel{?}{=} e(T_i', T_0)$. If so, output 1; otherwise, output 0.

$\mathsf{B\text{-}KeyRet}(k_{mas}, T_i, \mathbf{C}[i])$. Given a block ciphertext $\mathbf{C}[i]$ and the block tag T_i, split $\mathbf{C}[i]$ into s sectors: $\{\mathbf{C}[i][j]\}_{1\leq j\leq s}$, and compute the corresponding block key,

$$k_i = T_i^{k_{mas}^{-1}} \cdot (\prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{-1}.$$

If $\mathsf{Dec}(k_i, \mathbf{C}[i])=\perp$, output $\perp$; otherwise, output k_i.

$\mathsf{PoWPrf}(\mathbf{M}, Q)$. For a challenge query $Q = \{(i, \upsilon_i)\}$, compute the block tag T_i where i is the auditingition of a queried block. Finally, output the proof P_T

as $P_T = \prod_{(i,v_i)\in Q} T_i^{v_i}$.

$\mathsf{PoWVer}(P_T, \{T_i\}_{1\le i\le n}, Q)$. Given a proof P_T for a challenge query $Q = \{(i, v_i)\}$, compute the verification information $V_T = \prod_{(i,v_i)\in Q} T_i^{v_i}$ and check $P_T \stackrel{?}{=} V_T$. If so, output 1; otherwise, output 0.

4.3.2 Design Considerations

Guarded Decryption. In our scheme, the decryption algorithm additionally checks the validity of the decrypted message. By recomputing the block key using the decrypted message, it can tell whether the output message is the correct one or not. If it fails, then $\perp$ is returned. This additional property enables the user to be sure that the encrypted data downloaded from the cloud server is the one he/she intends to obtain (TC-secure).

Block Tag Generation. Since the tag constructed in a BL-MLE scheme should enable equality testing of block data and block key management, we embed the master key, block key and the ciphertext in the block tag. The master key serves as the encryption key to encrypt the block key. Since both the encryption algorithm and the master key generation algorithm are deterministic, different owners of the same file would produce the same block tags and hence the server can also perform deduplication on the block tags, which reduces the space cost.

Block Sectors. Another consideration is the block tag size. It is desirable that the length of the block tag should be less than that of the corresponding block ciphertext. In order to shorten the size of the block tag, the block ciphertext is split into s sectors. Each sector is one element of $\mathbb{Z}_p$ and hence the size of a block tag is just $1/s$ of the corresponding block ciphertext. It is worth noting that in our scheme we assume $\mathbf{M}[i] \in \{\mathbb{Z}_p\}^s$ and hence $\mathbf{C}[i][j] \in \mathbb{Z}_p$ as $|\mathbf{M}[i]| = |\mathbf{C}[i]|$. However, we should be careful about the length of each block message which is represented as bit-strings in practice. In order to make sure that each sector is the element of $\mathbb{Z}_p$, we set $|\mathbf{M}[i]| = s \cdot (\log_2 p - 1)$ instead of $s \cdot \log_2 p$.

Consistency Testing. In a BL-MLE system, we require that the block tag construction should achieve strong tag consistency. For our algorithm $\mathsf{ConTest}$, besides the block ciphertext $\mathbf{C}[i]$ and the block tag T_i, we need some additional information, i.e., $aux_i = e(k_i, T_0)$, for the consistency checking. However, aux_i does not need to be stored on the server. It is only required when the user uploads the file block and the block tag. Once the server has checked that $\mathbf{C}[i]$ and T_i are consistent, aux_i can be discarded. Moreover, we should note that the auxiliary information would not leak the block key to the server. Specifically, if there is an adversary $\mathcal{A}$ that can

derive k_i from $aux_i = e(k_i, T_0)$, we can construct an algorithm $\mathcal{B}$ to solve the CDH problem by using the adversary $\mathcal{A}$ as a subroutine as follows. Given a CDH problem instance (g, g^a, g^b), $\mathcal{B}$ sets $aux_i = e(k_i, T_0) = e(g^a, g^b) = e(g^{ab}, g)$ by implicitly setting $k_i = g^{ab}, T_0 = g$. $\mathcal{B}$ then runs $\mathcal{A}$ with input $e(k_i, T_0)$. If $\mathcal{A}$ can derive k_i (i.e., k_i appears in a hash query), then $\mathcal{B}$ can successfully solve the CDH problem. The details are referred to the security proof in Section 4.4.

Equality Testing. Note that in our scheme two identical block messages may belong to different files and hence have two distinct block tags. In order to support block data redundancy checking using these distinct block tags, we use paring to do the equality testing. This approach has been used in [ABM+13, YTHW10].

4.3.3 Correctness Analysis

We can observe that the BL-MLE scheme satisfies the requirement of *decryption correctness* as we use symmetric encryption. It is also obvious that the construction also achieves *PoW correctness*. For the other algorithms, we verify their correctness as follows.

Tag Correctness. Consider the block ciphertext $\mathbf{C}[i]$ of a block message $\mathbf{M}[i]$ and the corresponding block tag

$$T_i = (k_i \cdot \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{k_{mas}}$$

where k_{mas} is the corresponding master key and file tag $T_0 = g^{k_{mas}}$. For $\mathsf{ConTest}$ algorithm we have

$$\begin{aligned} e(T_i, g) &= e((k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{k_{mas}}, g) \\ &= e(k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]}, g^{k_{mas}}) \\ &= e(k_i, T_0) \cdot e(\prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]}, T_0). \end{aligned}$$

For EqTest algorithm, consider another block tag $T'_t = (k'_t \prod_{j=1}^{s} u_j^{\mathbf{C}'[t][j]})^{k'_{mas}}$ where $\mathbf{C}'[t]$ is the ciphertext of block message $\mathbf{M}'[t]$ and k'_t is the corresponding block key, k'_{mas} is the corresponding master key, and $T'_0 = g^{k'_{mas}}$ is the corresponding file tag.

We have,

$$\begin{aligned} e(T_i, T_0') &= e((k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{k_{mas}}, g^{k'_{mas}}) \\ &= e(k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]}, g)^{k_{mas} \cdot k'_{mas}}, \end{aligned}$$

$$\begin{aligned} e(T_t', T_0) &= e((k_t' \prod_{j=1}^{s} u_j^{\mathbf{C}'[t][j]})^{k'_{mas}}, g^{k_{mas}}) \\ &= e(k_t' \prod_{j=1}^{s} u_j^{\mathbf{C}'[t][j]}, g)^{k_{mas} \cdot k'_{mas}}. \end{aligned}$$

Suppose that $\mathbf{M}[i] = \mathbf{M}'[t]$, since the encryption and key generation algorithm are deterministic, $k_t' = k_i, \mathbf{C}'[t][j] = \mathbf{C}[i][j]$ for $(1 \leq j \leq s)$ (notice that $k_{mas}, k_{mas'}$ would be different when $\mathbf{M} \neq \mathbf{M}'$). Therefore, we have $e(T_i, T_0') = e(T_t', T_0)$.

B-Key-Retrieving Correctness. For a given block ciphertext $\mathbf{C}[i]$ and block tag

$$T_i = (k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{k_{mas}}$$

where k_{mas} is the corresponding master key, we have

$$T_i^{k_{mas}^{-1}} = (k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{k_{mas} \cdot k_{mas}^{-1}} = k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]}$$

which means $k_i = T_i^{k_{mas}^{-1}} (\prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{-1}$.

4.4 Security Analysis

In this section, following the security models defined previously, we show that the proposed scheme achieves the design goals in terms of security guarantees, more precisely, data privacy, tag consistency and PoW security.

4.4.1 Privacy

We prove the PRV$-CDA-B security of our scheme when modeling H_1, H_2, H_3 as random oracles.

Theorem 4.1 *Let H_1, H_2, H_3 be random oracles. Then if there exists a* PRV$-CDA-B *adversary $\mathcal{A}$ with advantage $\epsilon(\ell)$ against our scheme, there is an algorithm $\mathcal{B}$ that solves the CDH problem with probability $\mathsf{Adv}^{\mathcal{B}}_{CDH}(\ell)$ such that for all $\ell \in \mathbb{N}$,*

$$\mathsf{Adv}^{\mathsf{CDH}}_{\mathcal{B}}(\ell) \geq \frac{2\epsilon(\ell)}{n(\ell)q_{2,3}(\ell)} - \frac{q_1(\ell)}{2^{\mu(\ell)\cdot n(\ell)}} - \frac{q_{2,3}(\ell)\cdot n(\ell)}{2^{\mu(\ell)}},$$

where $\mu(\ell)$ is the min-entropy of block-source $\mathcal{M}$, $n(\ell)$ is the block message number and $q_1(\ell), q_{2,3}(\ell)$ are the number of queries to $H_1, H_{2,3}$ respectively by the adversary.

Proof: Algorithm $\mathcal{B}$ is given as input a random instance of CDH problem, $g_0, g_0^a, g_0^b \in \mathbb{G}_1$. Its goal is to output $g_0^{ab} \in \mathbb{G}_1$. Algorithm $\mathcal{B}$ simulates the challenger and interacts with the adversary $\mathcal{A}$ as follows.

Setup. $\mathcal{A}$ sends its chosen block-source $\mathcal{M}$ to $\mathcal{B}$. $\mathcal{B}$ then sets the system parameters $params =< p, g, \mathbb{G}_1, \mathbb{G}_\tau, e, H_1, H_2, H_3,\ s, u_1, u_2, ..., u_s >$ as follows: let $g = g_0^a$, and for all $k \in \{1, s\}$, chooses randomly $r_k \xleftarrow{R} \mathbb{Z}_p$ and computes $u_k = g_0^{ar_k}$. Finally, $\mathcal{A}$ is given $< p, g, \mathbb{G}_1, \mathbb{G}_\tau, e, s, u_1, u_2, ..., u_s >$ while H_1, H_2, H_3 are random oracles controlled by $\mathcal{B}$ as described below.

H_1-queries. $\mathcal{B}$ maintains a list of tuples $< M_i, h_{M_i} >$, which is called the H_1-list and initially empty $< *, * >$. When $\mathcal{A}$ queries the oracle H_1 at a point $M_j \in \{0,1\}^*$, if M_j already appears on the H_1-list in a tuple $< M_j, h_{M_j} >$ then $\mathcal{B}$ responds with $H_1(M_j) = h_{M_j}$, otherwise picks randomly $h_{M_j} \xleftarrow{R} \mathbb{Z}_p$, adds $< M_j, h_{M_j} >$ to the H_1-list and responds to $\mathcal{A}$ with $H_1(M_j) = h_{M_j}$.

$H_{2,3}$-queries. $\mathcal{B}$ maintains a list of tuples $< m_i, k_i, h_{k_i} >$, which is called the $H_{2,3}$-list and initially empty $< *, *, * >$.

1. When $\mathcal{A}$ queries the oracle H_2 at a point $m_j \in \{\mathbb{Z}_p\}^s$, if m_j already appears on the $H_{2,3}$-list in a tuple $< m_j, k_j, h_{k_j} >$ then $\mathcal{B}$ responds with $H_2(m_j) = k_j$, otherwise picks randomly $k_j \xleftarrow{R} \mathbb{G}_1, h_{k_j} \xleftarrow{R} \{\mathbb{Z}_p\}^s$, adds $< m_j, k_j, h_{k_j} >$ to the $H_{2,3}$-list and responds to $\mathcal{A}$ with $H_2(m_j) = k_j$.
2. When $\mathcal{A}$ queries the oracle H_3 at a point $k_j \in \mathbb{G}_1$, if k_j already appears on the $H_{2,3}$-list in a tuple $< m_j, k_j, h_{k_j} >$ then $\mathcal{B}$ responds with $H_3(k_j) = h_{k_j}$, otherwise picks randomly $m_j \xleftarrow{R} \{\mathbb{Z}_p\}^s, h_{k_j} \xleftarrow{R} \{\mathbb{Z}_p\}^s$, adds $< m_j, k_j, h_{k_j} >$ to the $H_{2,3}$-list and responds to $\mathcal{A}$ with $H_3(k_j) = h_{k_j}$.

Challenge. Finally, $\mathcal{B}$ runs $\mathcal{M}$ and gets $\mathbf{M}_0$ and Z. Then $\mathcal{B}$ chooses $\mathbf{M}_1$ uniformly at random from $\{0,1\}^{|\mathbf{M}_0|}$. Finally, $\mathcal{B}$ picks randomly $b \xleftarrow{R} \{0,1\}$, sets $\mathbf{M} = \mathbf{M}_b$ and computes as follows. If there exist a tuple $< M_i, h_{M_i} >$ on the H_1-list such that $M_i = \mathbf{M}$, $\mathcal{B}$ fails and aborts. Otherwise, $\mathcal{B}$ regards the master key as $H_1(\mathbf{M}) = a^{-1}$ and computes the file tag as $T_0 = g^{a^{-1}} = g_0$. For each $1 \leq j \leq$

$n(\ell)$, if there exist a tuple $< m_i, k_i, h_{k_i} >$ on the $H_{2,3}$-list such that $m_i = \mathbf{M}[j]$, $\mathcal{B}$ fails and aborts. Otherwise, for each $\mathbf{M}[j]$, $\mathcal{B}$ picks $\alpha_j \leftarrow \mathbb{Z}_p, \mathbf{C}[j] \xleftarrow{R} \{\mathbb{Z}_p\}^s$ randomly, splits $\mathbf{C}[j]$ into s sectors $\mathbf{C}[j][1], \mathbf{C}[j][2], ..., \mathbf{C}[j][s] \in \mathbb{Z}_p$ and computes the block tag as,

$$\mathbf{T}[j] = g_0^{\alpha_j b} \prod_{k=1}^{s} g_0^{r_k \mathbf{C}[j][k]} = (g_0^{\alpha_j ab} \prod_{k=1}^{s} u_k^{\mathbf{C}[j][k]})^{a^{-1}}.$$

Here, the block key $\mathbf{K}[j] = H_2(\mathbf{M}[j]) = g_0^{\alpha_j ab}$, $aux_j = e(\mathbf{K}[j], T_0) = e(g_0^{\alpha_j b}, g_0^a)$. Hence, $\mathbf{T}[j]$ is consistent with the block ciphertext $\mathbf{C}[j]$. Finally, $\mathcal{B}$ sends $(\mathbf{C}, \mathbf{T}, T_0, Z)$ to $\mathcal{A}$.

Output. Eventually algorithm $\mathcal{A}$ outputs his guess b' on b. At this point, $\mathcal{B}$ picks a random tuple $< m_i, k_i, h_{k_i} >$ from $H_{2,3}$-list, chooses $j \xleftarrow{R} [1, n(\ell)]$ and output $k_i^{\alpha_j^{-1}}$ as the solution to the given instance of CDH.

Note that the game is identical to PRV$-CDA-B game from the view of the adversary unless the challenge message $\mathbf{M}$ or $\mathbf{M}[j]$ ($j \in [1, n(\ell)]$) has been queried by the adversary.

Let $\mathcal{H}_1$ be the event that any of the H_1-queries by adversary equals $\mathbf{M}$ and $\mathcal{H}_2$ be the event that any of the H_2-queries equals $\mathbf{M}[j]$. Due to the fact that $\mathbf{M}_1$ is chosen uniformly at random from $\{0,1\}^{|\mathbf{M}_0|}$ and $\mathbf{C}, \mathbf{T}, T^F$ are computed independently of $\mathbf{M}$, we can apply the min-entropy of the block-source $\mathcal{M}$ to bound the probability $\Pr[\mathcal{H}_1]$ and $\Pr[\mathcal{H}_2]$ as follows,

$$\Pr[\mathcal{H}_1] \leq \frac{q_1(\ell)}{2^{\mu(\ell) \cdot n(\ell)}}, \quad \Pr[\mathcal{H}_2] \leq \frac{q_{2,3}(\ell) \cdot n(\ell)}{2^{\mu(\ell)}}.$$

In the simulation above, we refer to $\mathbf{K}$ as the block keys of $\mathbf{M}$. Let $\mathcal{H}_3$ be the event that $\mathcal{A}$ issues a query for $H_3(\mathbf{K}[j])$ at some point for any $j \in [1, n(\ell)]$. In the real attack, if $\mathcal{A}$ never issues such a query then the ciphertext $\mathbf{C}$ is independent of $\mathcal{A}$'s view (since each $H_3(K[j])$ is independent of $\mathcal{A}$'s view). In this case, the probability that $\mathcal{A}$ wins in the game is $\Pr[b = b' | \neg\mathcal{H}_3] = 1/2$. As $\mathcal{A}$ has advantage $\epsilon(\ell)$ against our scheme, we have $|Pr[b = b'] - 1/2| \geq \epsilon(\ell)$. Since

$$\Pr[b = b'] \leq \Pr[b = b' | \neg\mathcal{H}_3] \Pr[\neg\mathcal{H}_3] + \Pr[\mathcal{H}_3] = \frac{1}{2} + \frac{1}{2} \Pr[\mathcal{H}_3],$$

$$\Pr[b = b'] \geq \Pr[b = b' | \neg\mathcal{H}_3] \Pr[\neg\mathcal{H}_3] = \frac{1}{2} - \frac{1}{2} \Pr[\mathcal{H}_3],$$

we have $\epsilon(\ell) \leq |\Pr[b = b'] - 1/2| \leq \frac{1}{2} \Pr[\mathcal{H}_3]$, i.e., $\Pr[\mathcal{H}_3] \geq 2\epsilon(\ell)$.

Suppose that $\mathcal{A}$ issues a query for $H_3(\mathbf{K}[j^*])$ ($j^* \in [1, n(\ell)]$) at some point. Now we analyze the probability that the output by $\mathcal{B}$ is the correct solution to

the CDH problem. As $\mathcal{B}$ picks a tuple $< m_i, k_i, h_{k_i} >$ from $H_{2,3}$-list randomly, the probability that the picked tuple contains $\mathbf{K}[j^*]$ (i.e., $k_i = \mathbf{K}[j^*]$) is $1/q_{2,3}(\ell)$. In this case, since $\mathcal{B}$ chooses randomly $j \xleftarrow{R} [1, n(\ell)]$ and output $k_i^{\alpha_j^{-1}}$ as the solution to the CDH problem, we say that the solution is correct only when $j = j^*$, of which the probability is $1/n(\ell)$.

Therefore, $\mathcal{B}$ solves the CDH problem with the probability below,

$$\mathsf{Adv}_{\mathcal{B}}^{\mathsf{CDH}}(\ell) \geq \frac{2\epsilon(\ell)}{n(\ell)q_{2,3}(\ell)} - \frac{q_1(\ell)}{2^{\mu(\ell)\cdot n(\ell)}} - \frac{q_{2,3}(\ell)\cdot n(\ell)}{2^{\mu(\ell)}}.$$

This completes the proof of the theorem and therefore, we have the following conclusion.

Theorem 4.2 *Let H_1, H_2, H_3 be the random oracles, then if the Computational Diffie-Hellman (CDH) problem is hard, our scheme is* PRV$-CDA-B*-secure.*

4.4.2 Tag Consistency

For the tag consistency of our scheme, we have the following result.

Theorem 4.3 *Let H_2, H_3 be the random oracles. Let $\mathcal{A}$ be a* DFA *adversary that has advantage $\epsilon(\ell)$ against our scheme. Then there exist an algorithm $\mathcal{B}$ that solves the Discrete Log Problem (DLP) with advantage at least $\epsilon(\ell)/2$.*

Proof: Algorithm $\mathcal{B}$ is given as a random instance $< g, g^a >$ of the DLP. Its goal is to output a. Algorithm $\mathcal{B}$ finds the solution to this DLP by interacting with $\mathcal{A}$ as follows:

Setup. $\mathcal{B}$ sets the system parameters $params =< p, g, \mathbb{G}_1, \mathbb{G}_\tau, e, H_1, H_2, H_3, s, u_1, u_2, ..., u_s >$ as follows: for all $k \in \{1, s\}$, chooses randomly $a_k \xleftarrow{R} \mathbb{Z}_p$ and computes $u_k = g^{a_k}$. Finally, $\mathcal{A}$ is given $< p, g, \mathbb{G}_1, \mathbb{G}_\tau, e, H_1, s, u_1, u_2, ..., u_s >$ while H_2, H_3 are random oracles controlled by $\mathcal{B}$ as described below.

$H_{2,3}$-queries. $\mathcal{B}$ maintains a $H_{2,3}$-list $< m_i, k_i, r_i, h_{k_i}, coin_i >$ for H_2, H_3-queries.

1. When $\mathcal{A}$ queries the oracle H_2 at a point $m_j \in \{\mathbb{Z}_p\}^s$, algorithm $\mathcal{B}$ responds as follows. If the query m_j already appears on the $H_{2,3}$-list in a tuple $< m_j, k_j, r_j, h_{k_j}, coin_j >$ then algorithm $\mathcal{B}$ responds with $H(m_j) = k_j \in \mathbb{G}_1$. Otherwise, $\mathcal{B}$ generates a random coin $coin_j \xleftarrow{R} \{0, 1\}$, and picks a random $r_j \in \mathbb{Z}_p$, computes $k_j = g^{r_j + (1-coin_j)a} \in \mathbb{G}_1$. If the query k_j already appears on the $H_{2,3}$-list in a tuple $< *, k_j, *, h_{k_j}, * >$, then $\mathcal{B}$ fills the tuple as $< m_j, k_j, r_j, h_{k_j}, coin_j >$. Otherwise $\mathcal{B}$ picks $h_{k_j} \xleftarrow{R} \{\mathbb{Z}_p\}^s$, adds the tuple $< m_j, k_j, r_j, h_{k_j}, coin_j >$ to the $H_{2,3}$-list. Finally, $\mathcal{B}$ responds to $\mathcal{A}$ by setting $H(m_j) = k_j$.

2. When $\mathcal{A}$ queries the oracle H_3 at a point $k_j \in \mathbb{G}_1$, algorithm $\mathcal{B}$ responds as follows. If the query k_j already appears on the $H_{2,3}$-list in a tuple $< m_j, k_j, r_j, h_{k_j}, coin_j >$ then algorithm $\mathcal{B}$ responds with $H(k_j) = h_{k_j}$. Otherwise, $\mathcal{B}$ picks $h_{k_j} \xleftarrow{R} \{\mathbb{Z}_p\}^s$, adds the tuple $< *, k_j, *, h_{k_j}, * >$ to the $H_{2,3}$-list and responds to $\mathcal{A}$ by setting $H(k_j) = h_{k_j}$.

Output. Eventually algorithm $\mathcal{A}$ outputs $< \mathbf{M}^*, i, c', T' >$. If there is no tuple on the $H_{2,3}$-list containing $\mathbf{M}^*[i]$, then $\mathcal{B}$ queries itself for H_2 to ensure that such a tuple exists. If there is no tuple $< m', k', r', h_{k'}, coin' >$ on the $H_{2,3}$-list that satisfies $m' \oplus h_{k'} = c'$, $\mathcal{B}$ reports failure and terminates. Next, algorithm $\mathcal{B}$ finds the tuples $< m^*, k^*, r^*, h_{k^*}, coin^* >$ and $< m', k', r', h_{k'}, coin' >$ on the $H_{2,3}$-list. If $coin^* = coin'$ then $\mathcal{B}$ reports failure and terminates. Otherwise, let T^* be the block tag of $\mathbf{M}^*[i]$. Since $\mathsf{EqTest}(T^*, T') = 1$, we have, $k^* \cdot \prod_{j=1}^{s} u_j^{c_j^*} = k' \cdot \prod_{j=1}^{s} u_j^{c_j'}$. Without loss of generality, Suppose that $coin^* = 0, coin' = 1$, then $k^* = g^{r^*}, k' = g^{r'+a}$, hence, $g^{r^*} \cdot \prod_{j=1}^{s} u_j^{c_j^*} = g^{r'+a} \cdot \prod_{j=1}^{s} u_j^{c_j'}$. Therefore, we have, $g^a = g^{r^*} \cdot \prod_{j=1}^{s} u_j^{c_j^*} / (g^{r'} \prod_{j=1}^{s} u_j^{c_j'}) = g^{(r^* - r' + \sum_{j=1}^{s} a_j (c_j^* - c_j'))}$. Then $\mathcal{B}$ outputs the required a as $a = r^* - r' + \sum_{j=1}^{s} a_j (c_j^* - c_j')$.

As $\mathcal{A}$ has advantage $\epsilon(\ell)$ against our scheme, it must have queried m' to H_2 otherwise k' is independent of $\mathcal{A}$'s view and hence it cannot obtain the advantage. Since $\Pr[coin^* \neq coin'] = 1/2$, $\mathcal{B}$ solves the DLP with the probability $\epsilon(\ell)/2$. This completes the proof of the theorem.

Theorem 4.4 *Let H_2, H_3 be the random oracles, then if the Discrete Log Problem (DLP) is hard, our scheme is* DFA*-secure.*

4.4.3 PoW Security

For the security of our basic PoW protocol, following the model of UNC-CDA, we give a detailed probability analysis for the following theorem.

Theorem 4.5 *Let H_2 be a random oracle, then the adversary's advantage $\epsilon(\ell)$ in the* UNC-CDA *game against our scheme is,*

$$\epsilon(\ell) \leq (\frac{t(\ell)}{n(\ell)})^{q(\ell)} + \frac{1}{2^{\mu(\ell)}} \cdot (1 - (\frac{t(\ell) - q(\ell) + 1}{n(\ell) - q(\ell) + 1})^{q(\ell)})$$

where $n(\ell)$ is the total block number of the challenge-file, $t(\ell)$ is the number of blocks known to the adversary given the auxiliary information, $q(\ell)$ is the number of queried blocks and $\mu(\ell)$ is the min-entropy of block-source $\mathcal{M}$.

Proof: Suppose that the challenge-file $\mathbf{M}_f$ consists of $n(\ell)$ blocks, i.e., $\mathbf{M}_f = \{m_1, ..., m_{n(\ell)}\}$ and $\mathbf{M}_q = \{m_{i_1}, ..., m_{i_{q(\ell)}}\}_{i_1,...,i_{q(\ell)} \in [1,n(\ell)]}$ are the blocks queried during

the challenge stage, where $q(\ell)$ is the number. We also denote the $t(\ell)$ blocks known to the adversary as $\mathbf{M}_k = \{m_{j_1}, ..., m_{j_{t(\ell)}}\}_{j_1,...,j_{t(\ell)} \in [1,n(\ell)]}$.

Let Bad be the event that all the blocks queried are known to the adversary, i.e., $\mathbf{M}_q \subseteq \mathbf{M}_k$. As the query is chosen randomly, we have,

$$\Pr[Bad] = \frac{C_{t(\ell)}^{q(\ell)}}{C_{n(\ell)}^{q(\ell)}} = \prod_{i=0}^{q(\ell)-1} \frac{t(\ell)-i}{n(\ell)-i} \leq (\frac{t(\ell)}{n(\ell)})^{q(\ell)}.$$

Therefore, the probability that there exists at least a queried block which is unknown to the adversary, i.e., $\mathbf{M}_q \nsubseteq \mathbf{M}_k$, is,

$$\begin{aligned} \Pr[\mathbf{M}_q \nsubseteq \mathbf{M}_k] &= \Pr[\neg Bad] = 1 - \Pr[Bad] \\ &= 1 - \prod_{i=0}^{q(\ell)-1} \frac{t(\ell)-i}{n(\ell)-i} \\ &\leq 1 - (\frac{t(\ell)-q(\ell)+1}{n(\ell)-q(\ell)+1})^{q(\ell)}. \end{aligned}$$

Let $\mathcal{A}_{wins}$ be the event that the adversary wins in the UNC-CDA game, then we have,

$$\begin{aligned} \Pr[\mathcal{A}_{wins}] = \Pr[\mathcal{A}_{wins}|Bad] \Pr[Bad] \\ + \Pr[\mathcal{A}_{wins}|\neg Bad] \Pr[\neg Bad]. \end{aligned}$$

When the event Bad occurs, the adversary wins in the game with probability 1 since it knows all the blocks queried by the challenger. When the event Bad does not occur, i.e., $\mathbf{M}_q \nsubseteq \mathbf{M}_k$, we define $\mathbf{I}_c = \{i_1, ..., i_{q(\ell)}\} \cap \{j_1, ..., j_{t(\ell)}\}$ and refer to T_c as the aggregation of the block tags of those queried blocks known to the adversary, i.e., $T_c = \prod_{i \in \mathbf{I}_c} T_i$. Suppose that the verification information is V_T. We refer to $\Delta T = V_T / T_c$ as the aggregation of the block tags of those queried blocks unknown to the server. However, for those queried blocks unknown to the adversary, the corresponding block tags would be independent from the view of the adversary if the adversary does not query them to the random oracle H_2 and hence ΔT is independent from the view of the adversary. As the whole challenge-file is chosen from the *block-source*, we then apply the min-entropy to bound the probability that adversary wins when event Bad does not occur. Specially, we have, $\Pr[\mathcal{A}_{wins}|\neg Bad] \leq \frac{1}{2^{\mu(\ell)}}$. We can therefore bound the advantage $\Pr[\mathcal{A}_{wins}]$ of the adversary in the UNC-CDA game as follows,

$$\Pr[\mathcal{A}_{wins}] \leq (\frac{t(\ell)}{n(\ell)})^{q(\ell)} + \frac{1}{2^{\mu(\ell)}} \cdot (1 - (\frac{t(\ell)-q(\ell)+1}{n(\ell)-q(\ell)+1})^{q(\ell)}).$$

Theorem 4.6 *Let H_2 be a random oracle, then our scheme is* UNC-CDA*-secure.*

4.5 Extension for Data Auditing

In this section, we describe how to adjust our scheme for data auditing in the cloud.

4.5.1 Extension of Our Scheme

In this section, we extend our BL-MLE scheme to allow secure and efficient auditing protocol. In an efficient auditing protocol, it is required that the server does not need to access the entire file in order to convince the user that the data is intact. This is important, especially for large files. In our BL-MLE scheme, the server stores the file tag and block tags for the purpose of deduplication, block key management, and Proof of Ownership. An interesting question is: can we also use these tags for auditing? In this section, we affirm the answer by presenting an auditing system which can be considered as a variant of the auditing system in [WWR+11].

Setup. The setup is the same as our original BL-MLE scheme. Let $params = (p, g, \mathbb{G}_1, \mathbb{G}_\tau, e, H_1, H_2, H_3, s, u_1, u_2, ..., u_s)$ be the system parameters.

Authenticator. In an auditing system [ABC+07, AKK09, SW08], an authenticator is generated by the data owner for each data block, and uploaded to the server. During the auditing protocol, the verifier (either the data owner or a third-party auditor) employs a spot-checking mechanism to ask the server to present some (aggregated) data blocks and the corresponding (aggregated) authenticators for verification. In our BL-MLE scheme, a user generates both file tag and block tags for a file. Inspired by the auditing system in [SW08], we found that our master key k_{mas} in fact can serve as a user secret key and the corresponding file tag $T_0 = g^{k_{mas}}$ can serve as the public key. Therefore, the block tags can be used as the authenticators for individual data blocks. In other words, the file tag and the block tags in our BL-MLE scheme can play an additional role in auditing.

Merkle-Hash-Tree. We use the rank-based MHT, which is proposed in Chapter 3 in our auditing system. The user creates an rMHT where the i-th leaf node is $e(k_i, T_0)$ (or aux_i in our BL-MLE scheme). The root of the rMHT is kept by the verifier (either the user or a third-party auditor), and all the leave nodes are sent to the data server. Notice that aux_i has to be sent to the data server anyway for tag consistency checking. However, the server now should keep these leaf nodes in order to perform the auditing protocol.

Auditing. There are two parties involved in the auditing protocol, the verifier (the user or a third-party) and the prover (data sever). Assume that the data file contains

n blocks. The verifier first sends a challenge query $Q = \{(i, \upsilon_i)\}$ ($i \in [1, n], \upsilon_i \in \mathbb{Z}_p$) to the prover. Upon receiving the challenge, the server computes $T = \prod_{(i,\upsilon_i)\in Q} T_i^{\upsilon_i}$ and $\mu_j = \sum_{(i,\upsilon_i)\in Q} \upsilon_i \mathbf{C}[i][j]$ for all $1 \leq j \leq s$. In additional to these information, the server also provides the verifier $\{e(k_i, T_0), \Omega_i\}_{i\in Q}$ where Ω_i contains the siblings on the path from the leave $e(k_i, T_0)$ to the root R of the rMHT. Therefore, the response sent to the verifier contains $\mathcal{P} = \{T, \{\mu_j\}_{1\leq j\leq s}, \{e(k_i, T_0), \Omega_i\}_{i\in Q}\}$. After receiving the response, the verifier computes the root R' and verifies the root signature. If the checking fails, the verifier outputs *False*. Otherwise, the verifier checks, $e(T, g) \stackrel{?}{=} \prod_{(i,\upsilon_i)\in Q} e(k_i, T_0)^{\upsilon_i} \cdot e(\prod_{j=1}^{s} u_j^{\mu_j}, T_0)$. If so, output *True*; otherwise output *False*.

4.5.2 Security Analysis

Correctness. For an honest response to a query $Q = \{(i, \upsilon_i)\}$ containing $\mu_j = \sum_{(i,\upsilon_i)\in Q} \upsilon_i \mathbf{C}[i][j]$ and $T = \prod_{(i,\upsilon_i)\in Q} T_i^{\upsilon_i}$, we have

$$\begin{aligned} T &= \prod_{(i,\upsilon_i)\in Q} T_i^{\upsilon_i} = \prod_{(i,\upsilon_i)\in Q} (k_i \prod_{j=1}^{s} u_j^{\mathbf{C}[i][j]})^{k_{mas}\cdot\upsilon_i} \\ &= (\prod_{(i,\upsilon_i)\in Q} k_i^{\upsilon_i} \cdot \prod_{j=1}^{s} u_j^{\mu_j})^{k_{mas}}. \end{aligned}$$

Therefore,

$$e(T, g) = e((\prod_{(i,\upsilon_i)\in Q} k_i^{\upsilon_i} \cdot \prod_{j=1}^{s} u_j^{\mu_j})^{k_{mas}}, g) = \prod_{(i,\upsilon_i)\in Q} e(k_i, T_0)^{\upsilon_i} \cdot e(\prod_{j=1}^{s} u_j^{\mu_j}, T_0),$$

which shows the correctness of the protocol.

Security. We define the security for an auditing protocol using the Data Possession Game in [ABC+07] which captures the requirement that except with negligible probability, an adversary cannot successfully construct a valid proof without using the correct (i.e., unmodified) file blocks (and authenticators) corresponding to a given challenge. We have the following result for our auditing system.

Theorem 4.7 *If there exists an adversary $\mathcal{A}$ that has advantage ϵ in the data possession game, then there exists an algorithm $\mathcal{B}$ that solves the CDH problem also with advantage ϵ.*

4.5.3 Improved PoW Protocol with Stronger Security

Following the observation from [XZ14], we improve our PoW protocol for stronger security based on the auditing protocol construction. Note that in order to allow auditing, the server should maintain the leaves of the rMHT. Equipped with these

additional data, we show that our proof of ownership protocol can be improved to obtain stronger security by requiring the prover (user) to returning data blocks instead of aggregated block tags. Notice that here we can allow the adversary to have all the tags of a file.

In a proof of ownership protocol, upon receiving a challenge query $Q = \{(i, \upsilon_i)\}$ ($i \in [1, n], \upsilon_i \in \mathbb{Z}_p$), we require the user/prover to return $\mu_j = \sum_{(i,\upsilon_i)\in Q} \upsilon_i \mathbf{C}[i][j]$, $1 \leq j \leq s$. After receiving the proof, the server computes $\prod_{(i,\upsilon_i)\in Q} e(k_i, T_0)^{\upsilon_i}$ and $T = \prod_{(i,\upsilon_i)\in Q} T_i^{\upsilon_i}$, and then checks $e(T, g) \stackrel{?}{=} \prod_{(i,\upsilon_i)\in Q} e(k_i, T_0)^{\upsilon_i} \cdot e(\prod_{j=1}^{s} u_j^{\mu_j}, T_0)$.

Now in the PoW security game, similar to the Data possession Game for auditing, we require that except with negligible probability, the adversary cannot pass the verification without knowing all the data blocks corresponding to a given challenge. Formally, we have,

Theorem 4.8 *If the computational Diffie-Hellman problem is hard, then except with negligible probability, no adversary can construct a valid proof in the revised PoW game defined above.*

4.6 Performance Analysis

We combine each of the three MLE schemes proposed in [BKR13], namely **CE**, **HCE2**, and **RCE**, with the **PoW** protocol in [HHPS11]. We refer to **XXX+PoW** (**XXX** $\in$ { **CE**, **HCE2**, **RCE** }) as the *extended MLE scheme* for DLSB-deduplication. It is worth noting that in the extended MLE schemes, we assume that the master key for block key encryption is also derived from the file in a determined way, e.g., using a hash function.

Metadata Size. Below we provide a concrete comparison on the Dedup-Metadata size for different schemes. More precisely, let the block size in the extended MLE schemes be b-bits, then the block number is $n_1 = \lceil t/b \rceil$ for a duplicated file of size t-bits. Suppose the hash function applied in the scheme is SHA-256 of which the output has 256 bits. We then know from the constructions of **CE**, **HCE2** and **RCE** that both the block tag and the block key are of 256 bits. Therefore, the total dedup-metadata size of the extended MLE schemes is $512 \cdot (n_1 + 1)$ bits. One should note that here we assume both the file tag and the PoW tag are also of 256 bits. As for our BL-MLE scheme, to achieve 128-bit security, we choose the (Elliptic Curve) group $\mathbb{G}_1$ with prime order *params* such that $|p|$=257 (note that each sector in our scheme has $|p|$-1=256 bits). Therefore, the block size in our scheme is $256 \cdot s$ bits (s is the number of sectors per block) and the block number is $n_2 = \lceil t/(256 \cdot s) \rceil$. Since each block tag is an element of the elliptic curve group $\mathbb{G}_1$, we have that the total dedup-metadata size of our BL-MLE scheme is $257 \cdot (n_2 + 1)$ bits. To give a

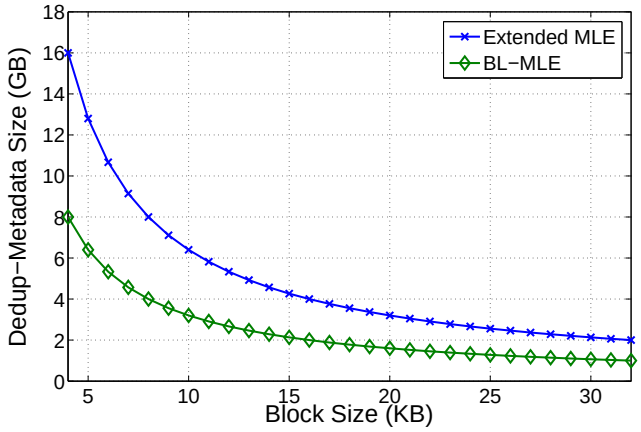


Figure 4.1: Impact of Block Size on Dedup-Metadata Size (f=1)

clear picture, we suppose that the cloud server stores f distinct files and each file is of 1 TB on average. To give a fair comparison, in both the extended MLE scheme and our BL-MLE scheme, we set the block to be of the same size (i.e., $b = 256 \cdot s$). In particular, when the block size is 4 KB, 8 KB, 16 KB, s is set to be 128, 256 and 512 respectively.

In the first case, we assume duplicated (i.e., same) files are stored in the cloud storage. In this case, the space cost of both the extended MLE scheme and our BL-MLE scheme is independent from the number of users who share the same file, under the assumption that the master key is directly derived from the file. However, our approach still performs better than the extended MLE scheme in terms of the total metadata size. The reason is that the tag in our construction can serve multi-purposes (i.e., block identifier, the encrypted block key, PoW tag), whereas the extended MLE scheme requires different tags for different purposes. As shown in Fig. 4.1, the dedup-metadata size of the extended MLE schemes decreases when the block size grows. Specifically, the dedup-metadata size of the extended MLE schemes is about 16 GB when the block size is 4 KB whereas the dedup-metadata of our scheme is approx 8 GB under the same setting. Generally speaking, the size of dedup-metadata in our scheme is much smaller than that of the extended MLE scheme when the block size is the same. Moreover, as illustrated in Fig. 4.2, the space saving from our BL-MLE scheme would be more significant when the file number increases. In particular, when f=20, the dedup-metadata size of the extended MLE schemes is 320 GB while that of our BL-MLE scheme is 160 GB when the block size is 4 KB.

For the second case, we consider similar but different files uploaded by different users. In this case, for both the extended MLE scheme and the proposed scheme, if two users upload two similar files, the server has to store all the encrypted block keys for both files since the corresponding master keys are different. Nevertheless,

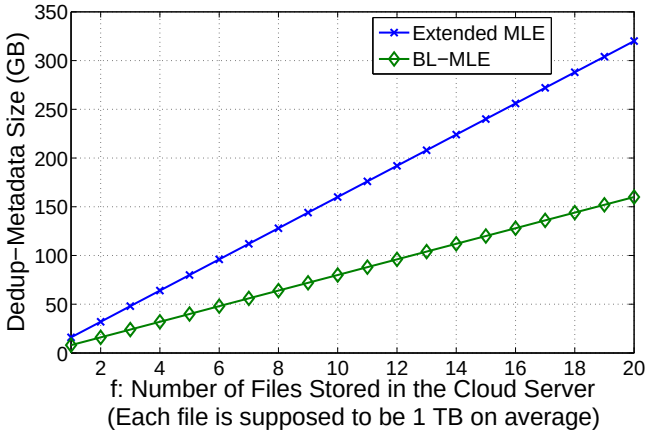


Figure 4.2: Impact of f on Dedup-Metadata Size (Block size: 4 KB)

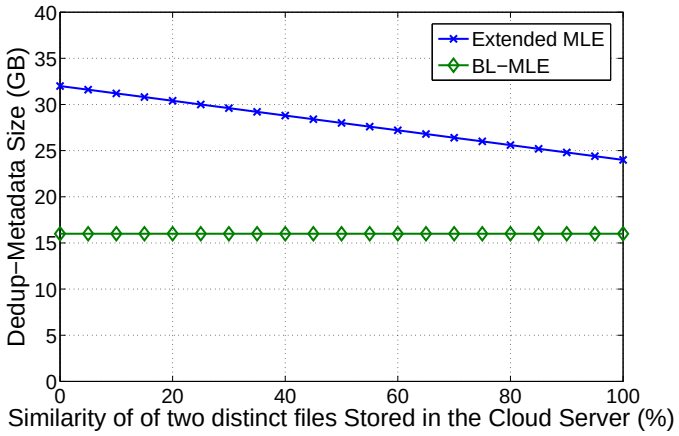


Figure 4.3: Impact of Similarity on Dedup-Metadata Size (Block size: 4 KB)

our construction can still reduce the metadata size, since for the extended MLE scheme, it still requires the block identifiers (one for all the duplicated data blocks) in addition to the encrypted block keys. The comparison between the two approaches for this case is illustrated in Fig. 4.3. The dedup-metadata size of extended MLE schemes decreases with the growth of the similarity while that of our BL-MLE stays the same regardless of the similarity. The reason is that in our BL-MLE scheme, the encrypted block key can also be used as the block identifier. Particularly, when the similarity is 0%, which means that these two files are completely different, the case is the same as shown in Fig. 4.2 (when f=2). When these two files have the similarity of 50%, the dedup-metadata size of extended MLE is 28 GB. Note that the difference will be more significant when the number of similar files is large.

Communication. For the bandwidth consumption of PoW, our BL-MLE scheme features a constant bandwidth since we use aggregated block tags as the response. However, the security of the basic PoW protocol in our BL-MLE scheme is weaker than that in [HHPS11] as we assume that the attacker does not know all the block

Table 4.1: Computation Time of Tag Generation and Block Key Retrieval

Algorithms	**Number of Sectors per Block**			
	64	128	256	512
TagGen	0.412 s	0.823 s	1.644 s	3.288 s
B-KeyRet	0.422 s	0.829 s	1.648 s	3.292 s

tags. For the extended BL-MLE scheme with auditing, the communication cost of the PoW is $\mathcal{O}(s)$ but now we can achieve a much stronger security. However, the cost is still smaller than that of the PoW protocol in [HHPS11].

Computation. To evaluate the computation cost of our BL-MLE scheme, we implemented the scheme using the Pairing Based Cryptography (PBC) library[b] (version 0.5.14). Our experiments are conducted using C on a Linux machine (2.6.35-22-generic version) with an Intel(R) Core(TM) 2 Duo CPU of 3.33 GHZ and 2.00-GB RAM. To achieve 128-bit level security, the scheme is implemented using bilinear groups of prime order *params* where $|p| = 256$. Since the encryption and decryption algorithms of our BL-MLE scheme are symmetric, we only analyze the computation cost related to tag generation and block key retrieval, which needs to be done by an end user and involves expensive modular exponentiation operations. As illustrated in Table 4.1, the computation time of both tag generation and block key retrieval increases with the number of sectors per block. This is due to the fact that for each sector, we need to compute one exponentiation operation. Moreover, the computation cost of block key retrieval is slightly higher than that of tag generation due to the additional group inversion operation. Specifically, when $s = 128$ (block size is 4 KB), the computation time is 0.823 seconds for tag generation and 0.829 seconds for block key retrieval.

We should note that the MLE schemes are in general more efficient in computation than our BL-MLE scheme since we use public-key techniques to construct *randomized* tags in our BL-MLE scheme whereas the tag construction in MLE schemes is deterministic. Particularly, when using SHA256 for instantiating the **CE**, the tag generation for a block of 4 KB consumes around 0.021s. We should remark that the efficiency loss seems indispensable in order to achieve significant space savings using randomized tag construction. A similar result has been observed in [ABM+13] which describes a fully randomized MLE scheme and leaves the construction of efficient randomized MLE schemes as an open problem.

[b] http://crypto.stanford.edu/pbc

4.7 Chapter Summary

In this chapter, we formalized a new primitive called Block-Level Message-Locked Encryption for DLSB-deduplication of large files to achieve space-efficient storage in cloud. We also presented a concrete BL-MLE scheme that can efficiently realize our design ideas. Moreover, we also showed that our BL-MLE scheme can be easily modified to achieve efficient data auditing, which makes our scheme multi-purpose for secure cloud storage.

Part II

Secure Data Retrieval

Chapter 5

Dual-Server Public-Key Encryption with Keyword Search

Searchable encryption is of increasing interest for protecting the data privacy in secure searchable cloud storage. In this chapter, we investigate the security of a well-known cryptographic primitive, namely Public Key Encryption with Keyword Search (PEKS) which is very useful in many applications of cloud storage. Unfortunately, it has been shown that the traditional PEKS framework suffers from an inherent vulnerability under the inside Keyword Guessing Attack (KGA) launched by the malicious server. To address this security vulnerability, we propose a new PEKS framework named Dual-Server Public Key Encryption with Keyword Search (DS-PEKS). As another main contribution, we define a new variant of the Smooth Projective Hash Functions (SPHFs) referred to as linear and homomorphic SPHF (LH-SPHF). We then show a generic construction of secure DS-PEKS from LH-SPHF. To illustrate the feasibility of our new framework, we provide an efficient instantiation of the general framework from a DDH-based LH-SPHF and show that it can achieve the strong security against inside KGA.

5.1 Introduction

In reality, end users may not entirely trust the CSP and may prefer to encrypt their data before uploading them to the cloud server in order to protect the data privacy. This usually makes the data utilization more difficult than the traditional storage where data is kept in the absence of encryption. One of the typical solutions is the searchable encryption which allows the user to retrieve the encrypted documents that contain the user-specified keywords, where given the keyword trapdoor, the server can find the data required by the user without decryption.

Searchable encryption can be realized in either symmetric or asymmetric encryption setting. In [SWP00], Song *et al.* proposed keyword search on ciphertext, known as Searchable Symmetric Encryption (SSE) and afterwards several SSE schemes [AKSX04, CGKO06] were designed for improvements. Although SSE schemes enjoy high efficiency, they suffer from complicated secret key distribution. Precisely, users have to securely share secret keys which are used for data encryption. Otherwise, they are not able to share the encrypted data outsourced to the cloud. To resolve this problem, Boneh *et al.* [BCOP04] introduced a more flexible primitive,

namely Public Key Encryption with Keyword Search (PEKS) that enables a user to search encrypted data in the asymmetric encryption setting. In a PEKS system, using the receiver's public key, the sender attaches some encrypted keywords (referred to as PEKS ciphertexts) with the encrypted data. The receiver then sends the trapdoor of a to-be-searched keyword to the server for data searching. Given the trapdoor and the PEKS ciphertext, the server can test whether the keyword underlying the PEKS ciphertext is equal to the one selected by the receiver. If so, the server sends the matching encrypted data to the receiver. One can see that this has many potential applications such as the private email system where the receiver may want the gateway to route his/her emails based on the specified keyword.

5.1.1 Motivations

Unfortunately, despite that being free from secret key distribution, PEKS schemes suffer from an inherent insecurity regarding the trapdoor keyword privacy, namely *inside Keyword Guessing Attack* (KGA). The reason leading to such a security vulnerability is that anyone who knows receiver's public key can generate the PEKS ciphertext of arbitrary keyword himself. Specifically, given a trapdoor, the adversarial server can choose a guessing keyword from the keyword space and then use the keyword to generate a PEKS ciphertext. The server then can test whether the guessing keyword is the one underlying the trapdoor. This *guessing-then-testing* procedure can be repeated until the correct keyword is found. Such a guessing attack has also been considered in many password-based systems. However, the attack can be launched more efficiently against PEKS schemes since the keyword space is roughly the same as a normal dictionary (e.g., all the meaningful English words), which has a much smaller size than a password dictionary (e.g., all the words containing 6 alphanumeric characters). It is worth noting that in SSE schemes, only secret key holders can generate the keyword ciphertext and hence the adversarial server is not able to launch the inside KGA. As the keyword always indicates the privacy of the user data, it is therefore of practical importance to overcome this security threat for secure searchable encrypted data outsourcing.

5.1.2 Contributions and Techniques

The contributions of this chapter are four-fold.

First, we formalize a new PEKS framework named *Dual-Server Public Key Encryption with Keyword Search* (DS-PEKS) to address the security vulnerability of PEKS. Our key idea to achieve the security against inside KGA is to split the testing functionality of the PEKS system into two steps which are handled by two independent servers: *front server* and *back server*. We then show that under such a setting

neither the front server nor the back server can launch the keyword guessing attack. It is worth noting that we require that the two servers do not collude since otherwise it goes back to the one-server setting in which inside KGA cannot be prevented.

Second, we introduce a new variant of *Smooth Projective Hash Function* (SPHF) referred to as *linear and homomorphic SPHF* for a generic construction of DS-PEKS. Roughly speaking, an SPHF can be defined based on a domain $\mathcal{X}$ and an $\mathcal{NP}$ language $\mathcal{L}$, where $\mathcal{L} \subset \mathcal{X}$. In our chapter, the SPHF used is based on the hard-on-the-average $\mathcal{NP}$-language and hence is also *pseudo-random* [GL03]. Our newly defined variant, named Lin-Hom SPHF, additionally requires the underlying language $\mathcal{L}$ and the SPHF to be *linear* and *homomorphic*. To be more precise, we require that (1) any word $W \in \mathcal{L}$ can be mapped to another word $W^* \in \mathcal{L}$ using a witness Δw. Correspondingly, the hash value of W^* can be computed using that of W with Δw; (2) any two words $W_1, W_2 \in \mathcal{L}$ can be transferred to a new word $W' \in \mathcal{L}$ under a defined operation. Correspondingly, the hash value of W' can be derived from that of W_1 and W_2.

Third, we show a generic construction of DS-PEKS using the proposed Lin-Hom SPHF. In a generic DS-PEKS construction, to generate the PEKS ciphertext of a keyword, the sender picks a word randomly from $\mathcal{L}$ and computes its two hash values using the witness and the public key (projection key) of the front and back servers respectively. The keyword is then concealed with these two hash values in the PEKS ciphertext. The receiver generates the trapdoor in the same way. In the pre-processing stage, the front server first removes the pseudo-random hash values in the trapdoor and the PEKS ciphertext for the back server using its private key. Due to the *linear* and *homomorphic* properties of Lin-Hom SPHF, the front server can re-randomise the *internal testing-state* to preserve the keyword privacy from the back server who can only determine whether the two keywords underlying the *internal testing-state* are the same or not. We can see that in this way, the security of DS-PEKS against inside keyword guessing attack can be obtained.

Fourth, to illustrate the feasibility of our new framework, an efficient instantiation of our SPHF based on the Diffie-Hellman language is presented in this chapter. We show that the Diffie-Hellman language is linear and homomorphic and the derived SPHF is a Lin-Hom SPHF based on the Diffie-Hellman assumption. We then present a practical and secure DS-PEKS scheme without pairings, which is more efficient compared with other pairing-based PEKS schemes [BCOP04, WBDS04, ABC+05, Kha06].

5.1.3 Related Work

In this subsection, we describe a classification of PEKS schemes based on their security.

Traditional PEKS. Following Boneh *et al.*'s seminal work [BCOP04], Abdalla *et al.* [ABC+05] formalized anonymous IBE (AIBE) and presented a generic construction of searchable encryption from AIBE. They also showed how to transfer a hierarchical IBE (HIBE) scheme into a public key encryption with temporary keyword search (PETKS) where the trapdoor is only valid in a specific time interval. Waters [WBDS04] showed that the PEKS schemes based on the bilinear map could be applied to build encrypted and searchable auditing logs. In order to construct a PEKS secure in the standard model, Khader [Kha06] proposed a scheme based on the k-resilient IBE and also gave a construction supporting multiple-keyword search. The first PEKS scheme without pairings was introduced by Di Crescenzo and Saraswat [CS07]. The construction is derived from Cock's IBE scheme [Coc01] which is not very practical.

Secure Channel Free PEKS. The original PEKS scheme [BCOP04] requires a secure channel to transmit the trapdoors. To overcome this limitation, Baek *et al.* [BSS08] proposed a new PEKS scheme without requiring a secure channel, which is referred to as a secure channel-free PEKS (SCF-PEKS). The idea is to add the server's public/private key pair into a PEKS system. The keyword ciphertext and trapdoor are generated using the server's public key and hence only the server (designated tester) is able to perform the search. Rhee *et al.* [RPSL09] later enhanced Baek *et al.*'s security model [BSS08] for SCF-PEKS where the attacker is allowed to obtain the relationship between the non-challenge ciphertexts and the trapdoor. They also presented an SCF-PEKS scheme secure under the enhanced security model in the random oracle model.

Against Outside KGA. Byun *et al.* [BRPL06] introduced the off-line keyword guessing attack against PEKS as keywords are chosen from a much smaller space than passwords and users usually use well-known keywords for searching documents. They also pointed out that the scheme proposed in Boneh *et al.* [BCOP04] was susceptible to keyword guessing attack. Inspired by the work of Byun *et al.* [BRPL06], Yau *et al.* [YHG08] demonstrated that outside adversaries that capture the trapdoors sent in a public channel can reveal the encrypted keywords through off-line keyword guessing attacks and they also showed off-line keyword guessing attacks against the (SCF-)PEKS schemes in [BSS08,BSS06]. The first PEKS scheme secure against outside keyword guessing attacks was proposed by Rhee *et al.* [RSK09]. In [RPSL10], the notion of trapdoor indistinguishability was proposed and the authors showed

that trapdoor indistinguishability is a sufficient condition for preventing outside keyword-guessing attacks.

Against Inside KGA. Nevertheless, all the schemes mentioned above are found to be vulnerable to keyword guessing attacks from a malicious server (i.e., inside KGA). Jeong *et al.* [JKHL09] showed a negative result that the consistency/correctness of PEKS implies insecurity to inside KGA in PEKS. Their result indicates that constructing secure and consistent PEKS schemes against inside KGA is impossible under the original framework. A potential solution is to propose a new framework of PEKS. In [XJWW13], Peng *et al.* proposed the notion of Public-key Encryption with Fuzzy Keyword Search (PEFKS) where each keyword corresponds to an exact trapdoor and a fuzzy trapdoor. The server is only provided with the fuzzy trapdoor and thus can no longer learn the exact keyword since two or more keywords share the same fuzzy keyword trapdoor. However, their scheme suffers from several limitations regarding the security and efficiency. On one hand, although the server cannot exactly guess the keyword, it is still able to know which small set the underlying keyword belongs to and thus the keyword privacy is not well preserved from the server. On the other hand, their scheme is impractical as the receiver has to locally find the matching ciphertext by using the exact trapdoor to filter out the non-matching ones from the set returned by the server.

5.2 Dual-Server PEKS

In this section, we formally define the Dual-Server Public Key Encryption with Keyword Search (DS-PEKS) and its security model.

5.2.1 Overview

In our proposed framework, namely DS-PEKS, we disallow the stand-alone testing to obtain the security against inside keyword guessing attacks. Roughly speaking, DS-PEKS consists of (KeyGen, DS-PEKS, DS-Trapdoor, FrontTest, BackTest). To be more precise, the KeyGen algorithm generates the public/private key pairs of the front and back servers instead of that of the receiver. Moreover, the trapdoor generation algorithm DS-Trapdoor defined here is public while in the traditional PEKS definition [BCOP04, BSS08], the algorithm Trapdoor takes as input the receiver's private key. Such a difference is due to the different structures used by the two systems. In the traditional PEKS, since there is only one server, if the trapdoor generation algorithm is public, then the server can launch an off-line guessing attack against a keyword ciphertext to recover the encrypted keyword. As a result, it is impossible to achieve the semantic security as defined in [BCOP04, BSS08]. However, as we will

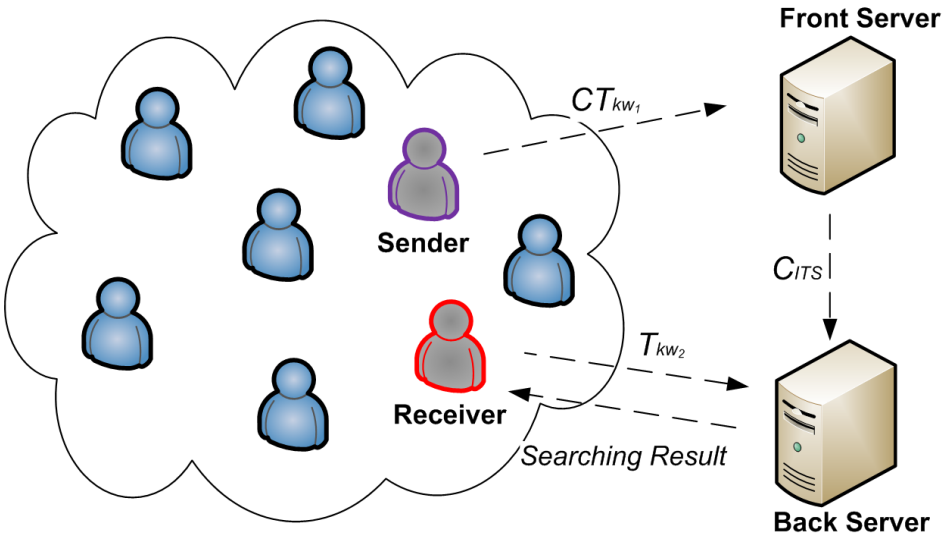


Figure 5.1: System Model of Dual-Server PEKS

show later, under the DS-PEKS framework, we can still achieve semantic security when the trapdoor generation algorithm is public. Another difference between the traditional PEKS and our proposed DS-PEKS is that the test algorithm is divided into two algorithms, FrontTest and BackTest run by two independent servers. This is essential for achieving security against the inside keyword guessing attacks.

The system model is as shown in Fig. 5.1. Upon receiving a query from the receiver, the front server pre-processes the trapdoor and all the PEKS ciphertexts using its private key, and then sends some *internal testing-states* to the back server with the corresponding trapdoor and PEKS ciphertexts hidden. The back server can then decide which documents are queried by the receiver using its private key and the received internal testing-states from the front server.

5.2.2 Formal Definition

Syntax. A DS-PEKS scheme is defined by the following algorithms.

Setup(1^ℓ). Takes as input the security parameter ℓ, generates the system parameters $params$;

KeyGen($params$). Takes as input the systems parameters $params$, outputs the public/secret key pairs (pk_{FS}, sk_{FS}), and (pk_{BS}, sk_{BS}) for the front server, and the back server respectively;

DS-PEKS($params, pk_{FS}, pk_{BS}, kw_1$). Takes as input $params$, the front server's public key pk_{FS}, the back server's public key pk_{BS} and the keyword kw_1, outputs the PEKS ciphertext CT_{kw_1} of kw_1;

DS-Trapdoor$(params, pk_{FS}, pk_{BS}, kw_2)$. Takes as input $params$, the front server's public key pk_{FS}, the back server's public key pk_{BS} and the keyword kw_2, outputs the trapdoor T_{kw_2};

FrontTest$(params, sk_{FS}, CT_{kw_1}, T_{kw_2})$. Takes as input $params$, the front server's secret key sk_{FS}, the PEKS ciphertext CT_{kw_1} and the trapdoor T_{kw_2}, outputs the internal testing-state C_{ITS};

BackTest$(params, sk_{BS}, C_{ITS})$. Takes as input $params$, the back server's secret key sk_{BS} and the internal testing-state C_{ITS}, outputs the testing result 0 or 1;

Correctness. It is required that for any keyword kw_1, kw_2, and

$$CT_{kw_1} \leftarrow \text{DS-PEKS}(pk_{FS}, pk_{BS}, kw_1)^{\text{a}},$$

$$T_{kw_2} \leftarrow \text{DS-Trapdoor}(pk_{FS}, pk_{BS}, kw_2),$$

we have that $\text{BackTest}(sk_{BS}, C_{ITS}) = 1$ if $kw_1 = kw_2$, otherwise 0.

5.2.3 Security Models

In this subsection, we formalize the security models for DS-PEKS. We define the following security models for a DS-PEKS scheme against the adversarial front and back servers, respectively. We should note that the following security models also imply the security guarantees against the outside adversaries which have less capability compared to the servers.

Adversarial Front Server. In this part, we define the security against an adversarial front server. Precisely, we introduce two games, namely semantic-security against chosen keyword attack and indistinguishability against keyword guessing attack to capture the security of PEKS ciphertext and trapdoor, respectively.

Semantic-Security against Chosen Keyword Attack (SS-CKA). In the following, we define the semantic-security against chosen keyword attack which guarantees that no adversary is able to distinguish a keyword from another one given the corresponding PEKS ciphertext. That is, the PEKS ciphertext does not reveal any information about the underlying keyword to any adversary.

Setup. The challenger $\mathcal{C}$ runs the $\text{KeyGen}(\ell)$ algorithm to generate key pairs (pk_{FS}, sk_{FS}) and (pk_{BS}, sk_{BS}). It gives $(pk_{FS}, sk_{FS}, pk_{BS})$ to the attacker $\mathcal{A}$;

Test query-I. The attacker $\mathcal{A}$ can adaptively make the test query for any keyword and any PEKS ciphertext of its choice. The challenger $\mathcal{C}$ returns 1 or 0 as the test result to the attacker;

[a]For simplicity, we will omit $params$ in the input of the algorithms in the rest of the chapter.

Challenge. The attacker $\mathcal{A}$ sends the challenger $\mathcal{C}$ two keywords kw_0, kw_1. The challenger picks $b \xleftarrow{\$} \{0,1\}$ and generates

$$CT^*_{kw} \leftarrow \mathsf{DS\text{-}PEKS}(pk_{FS}, pk_{BS}, kw_b).$$

The challenger $\mathcal{C}$ then sends CT^*_{kw} to the attacker $\mathcal{A}$;

Test query-II. The attacker $\mathcal{A}$ can continue the test query for any keyword and any PEKS ciphertext of its choice except the challenge keywords kw_0, kw_1. The challenger $\mathcal{C}$ returns 1 or 0 as the test result to the attacker $\mathcal{A}$;

Output. Finally, the attacker $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$ on b and wins the game if $b = b'$.

We refer to such an adversarial front server $\mathcal{A}$ in the above game as an SS-CKA adversary and define its advantage as

$$\mathsf{Adv}^{\mathsf{SS\text{-}CKA}}_{\mathcal{FS},\mathcal{A}}(\ell) = \Pr[b = b'] - 1/2.$$

Indistinguishability against Keyword Guessing Attack (IND-KGA). This model captures that the trapdoor reveals no information about the underlying keyword to the adversarial front server. We define the security model as follows.

Setup. The challenger $\mathcal{C}$ runs the $\mathsf{KeyGen}(\ell)$ algorithm to generate key pairs (pk_{FS}, sk_{FS}) and (pk_{BS}, sk_{BS}). It gives $(pk_{FS}, sk_{FS}, pk_{BS})$ to the attacker $\mathcal{A}$;

Test query-I. The attacker $\mathcal{A}$ can adaptively make the test query for any keyword and any PEKS ciphertext of its choice. The challenger $\mathcal{C}$ returns 1 or 0 as the test result to the attacker;

Challenge. The attacker sends the challenger $\mathcal{C}$ two keywords kw_0, kw_1. The challenger $\mathcal{C}$ picks $b \xleftarrow{\$} \{0,1\}$ and generates

$$T^*_{kw} \leftarrow \mathsf{DS\text{-}Trapdoor}(pk_{FS}, pk_{BS}, kw_b).$$

The challenger $\mathcal{C}$ then sends T^*_{kw} to the attacker $\mathcal{A}$;

Test query-II. The attacker $\mathcal{A}$ can continue issue the test query for any keyword and any PEKS ciphertext of its choice except the challenge keywords kw_0, kw_1. The challenger $\mathcal{C}$ returns 1 or 0 as the test result to the attacker $\mathcal{A}$;

Output. Finally, the attacker $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$ on b and wins the game if $b = b'$.

We refer to such an adversarial front server $\mathcal{A}$ in the above game as an IND-KGA adversary and define its advantage as

$$\mathsf{Adv}^{\mathsf{IND\text{-}KGA}}_{\mathcal{FS},\mathcal{A}}(\ell) = \Pr[b = b'] - 1/2.$$

Adversarial Back Server. The security models of SS-CKA and IND-KGA in terms of an adversarial back server are similar to those against an adversarial front server.

Semantic-Security against Chosen Keyword Attack. Here the game against an adversarial back server is the same as the one against an adversarial front server except that the adversary is given the private key of the back server instead of that of the front server. We omit the details here for simplicity.

We refer to the adversarial back server $\mathcal{A}$ in the SS-CKA game as an SS-CKA adversary and define its advantage as $\mathsf{Adv}^{\mathsf{SS\text{-}CKA}}_{\mathcal{BS},\mathcal{A}}(\ell) = \Pr[b = b'] - 1/2$.

Indistinguishability against Keyword Guessing Attack. Similarly, this security model aims to capture that the trapdoor does not reveal any information to the back server and hence is the same as that against the front server except that the adversary owns the private key of the back server instead of that of the front server. Therefore, we also omit the details here.

We refer to the adversarial back server $\mathcal{A}$ in the IND-KGA game as an IND-KGA adversary and define its advantage as $\mathsf{Adv}^{\mathsf{IND\text{-}KGA}}_{\mathcal{BS},\mathcal{A}}(\ell) = \Pr[b = b'] - 1/2$.

Indistinguishability against Keyword Guessing Attack-II (IND-KGA-II). Apart from the above two security models, we should also guarantee that the internal testing-state does not reveal any information about the keyword to the back server. We hence define another type of keyword guessing attack to capture such a requirement. The security, namely Indistinguishability against Keyword Guessing Attack-II guarantees that the back server cannot learn any information about the keywords from the internal testing-state. The security model is defined as follows.

Setup. The challenger $\mathcal{C}$ runs the $\mathsf{KeyGen}(\ell)$ algorithm to generates key pairs (pk_{FS}, sk_{FS}) and (pk_{BS}, sk_{BS}). It gives $(pk_{FS}, pk_{BS}, sk_{BS})$ to the attacker $\mathcal{A}$.

Challenge. The attacker $\mathcal{A}$ sends the challenger $\mathcal{C}$ three different keywords kw_0, kw_1, kw_2. The challenger $\mathcal{C}$ picks $\{b_1, b_2\} \subset \{0, 1, 2\}$ randomly and computes

$$CT^*_{kw} \leftarrow \mathsf{DS\text{-}PEKS}(pk_{FS}, pk_{BS}, kw_{b_1}),$$
$$T^*_{kw} \leftarrow \mathsf{DS\text{-}Trapdoor}(pk_{FS}, pk_{BS}, kw_{b_2}),$$
$$C^*_{ITS} \leftarrow \mathsf{FrontTest}(sk_{FS}, CT^*_{kw}, T^*_{kw}).$$

The challenger $\mathcal{C}$ then sends C^*_{ITS} to the attacker $\mathcal{A}$.

Output. Finally, the attacker $\mathcal{A}$ outputs its guess on $\{b_1, b_2\}$ as $\{b_1', b_2'\} \subset \{0, 1, 2\}$ and wins the game if $\{b_1', b_2'\} = \{b_1, b_2\}$.

We refer to such an adversary $\mathcal{A}$ in the above two games as a IND-KGA-II adversary and define its advantage as,

$$\mathsf{Adv}^{\mathsf{IND\text{-}KGA\text{-}II}}_{\mathcal{BS},\mathcal{A}}(\ell) = \Pr[\{b_1', b_2'\} = \{b_1, b_2\}] - 1/3.$$

We should remark that in the above game, b_1 and b_2 can be equivalent. In this case, the adversary (i.e., back server) will know that the same keyword has been used in the generation of the PEKS ciphertext and the trapdoor, and the adversary's goal is to guess which keyword among the three has been used.

Based on the security models defined above, we give the following security definition for a DS-PEKS scheme.

Definition 5.1 *We say that a DS-PEKS is secure if for any polynomial time attacker $\mathcal{A}_i$ $(i = 1, \ldots, 5)$, we have that $\mathsf{Adv}^{\mathsf{SS\text{-}CKA}}_{\mathcal{BS},\mathcal{A}_1}(\ell)$,$\mathsf{Adv}^{\mathsf{SS-CKA}}_{\mathcal{BS},\mathcal{A}_2}(\ell)$, $\mathsf{Adv}^{\mathsf{IND\text{-}KGA}}_{\mathcal{FS},\mathcal{A}_3}(\ell)$, $\mathsf{Adv}^{\mathsf{IND\text{-}KGA}}_{\mathcal{BS},\mathcal{A}_4}(\ell)$ and $\mathsf{Adv}^{\mathsf{IND\text{-}KGA\text{-}II}}_{\mathcal{BS},\mathcal{A}_5}(\ell)$ are all negligible functions of the security parameter ℓ.*

5.3 Linear and Homomorphic SPHF

In this chapter, we consider a new variant of smooth projective hash function. We consider two new properties: linear and homomorphic, which are defined below. It is worth noting that Abdalla *et al.* [ACP09] introduced conjunction and disjunction of languages for smooth projective hashing that was later used in the construction of blind signature [BBC+13b, BPV], oblivious signature-based envelope [BPV], and authenticated key exchange protocols for algebraic languages [BBC+13a]. As shown in the following, our definition for the new SPHF here is different from their work since we consider the operations on the words belonging to the same language, whereas theirs considers operations among different languages.

Let $\mathcal{SPHF}$=(SPHFSetup, HashKG, ProjKG, Hash, ProjHash) be a smooth projective hash function over the language $\mathcal{L} \subset \mathcal{X}$ onto the set $\mathcal{Y}$ and $\mathcal{W}$ be the witness space of $\mathcal{L}$. We first describe the operations on the sets $< \mathcal{L}, \mathcal{Y}, \mathcal{W} >$ as follows.

a). $\circledcirc : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$. For any $W_1 \in \mathcal{L}, W_2 \in \mathcal{L}$, $W_1 \circledcirc W_2 \in \mathcal{L}$;

b). $\circledast : \mathcal{Y} \times \mathcal{Y} \to \mathcal{Y}$. For any $y_1 \in \mathcal{Y}, y_2 \in \mathcal{Y}$, $y_1 \circledast y_2 \in \mathcal{Y}$;

c). $\odot, \oplus : \mathcal{W} \times \mathcal{W} \to \mathcal{W}$. For any $w_1 \in \mathcal{W}, w_2 \in \mathcal{W}$, $w_1 \odot w_2 \in \mathcal{W}$ and $w_1 \oplus w_2 \in \mathcal{W}$;

d). $\otimes : \mathcal{W} \times \mathcal{L} \to \mathcal{L}$. For any $w \in \mathcal{W}, W \in \mathcal{L}$, $w \otimes W \in \mathcal{L}$;

e). $\bullet : \mathcal{W} \times \mathcal{Y} \to \mathcal{Y}$. For any $w \in \mathcal{W}, y \in \mathcal{Y}$, $w \bullet y \in \mathcal{Y}$.

Moreover, for any element $y \in \mathcal{Y}$, we define $y \circledast y^{-1} = 1_{\mathcal{Y}}$ which is the identity element of $\mathcal{Y}$.

Our new SPHF requires the underlying language to be also *linear and homomorphic language*, which is defined below.

Definition 5.2 (Linear and Homomorphic Language) *A language $\mathcal{L}$ is linear and homomorphic if it satisfies the following properties.*

1). For any word $W \in \mathcal{L}$ with witness w and $\Delta w \in \mathcal{W}$, there exists a word $W^* \in \mathcal{L}$ such that $\Delta w \otimes W = W^*$ with the witness $w^* = \Delta w \odot w$.

2). For any two words $W_1, W_2 \in \mathcal{L}$ with the witness $w_1, w_2 \in \mathcal{W}$ respectively, there exists a word $W^* \in \mathcal{L}$ such that $W_1 \circledcirc W_2 = W^*$ with the witness $w^* = w_1 \oplus w_2$.

We then give the definition of *Lin-Hom SPHF* as follows.

Definition 5.3 (Lin-Hom SPHF (LH-SPHF)) *We say $\mathcal{SPHF}$ is a Lin-Hom SPHF (LH-SPHF) if the underlying language $\mathcal{L}$ is a linear and homomorphic language and $\mathcal{SPHF}$ satisfies the following properties.*

1). For any word $W \in \mathcal{L}$ with the witness $w \in \mathcal{W}$ and $\Delta w \in \mathcal{W}$, we have

$$\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), \Delta w \otimes W) = \Delta w \bullet \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W).$$

In other words, suppose $\Delta w \otimes W = W^*$, we have,

$$\mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), W^*, w^*) = \Delta w \bullet \mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), W, w),$$

where $w^* = \Delta w \odot w$.

2). For any two words $W_1, W_2 \in \mathcal{L}$ with the witness $w_1, w_2 \in \mathcal{W}$, we have

$$\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W_1 \circledcirc W_2) =$$

$$\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W_1) \circledast \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W_2).$$

In other words, suppose $W_1 \circledcirc W_2 = W^*$, we have,

$$\mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), W^*, w^*) =$$

$$\mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), W_1, w_1) \circledast \mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), W_2, w_2)$$

where $w^* = w_1 \oplus w_2$.

In this chapter, we also assume that the LH-SPHF has the following property: for any $y \in \mathcal{Y}$, $W \in \mathcal{L}$ and the witness $w \in \mathcal{W}$ of W, there exists a projection key hp such that $\mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}, \mathsf{param}), W, w) = y$.

5.4 Generic Construction of DS-PEKS

In this section, we show how to generically construct a Dual-Server Public Key Encryption with keyword search based on *Lin-Hom Smooth Projective Hash Functions*.

5.4.1 Generic Construction

Suppose $\mathcal{SPHF} = (\mathsf{SPHFSetup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash})$ is an LH-SPHF over the language $\mathcal{L}$ onto the set $\mathcal{Y}$. Let $\mathcal{W}$ be the witness space of the language $\mathcal{L}$ and $\mathcal{KW}$ be the keyword space. Our generic construction $\mathcal{DS}$-$\mathcal{PEKS}$ works as follows.

$\mathsf{Setup}(1^\ell)$. Take as input the security parameter ℓ, run $\mathsf{SPHFSetup}$ algorithm and generate the global parameters param, the description of the language $\mathcal{L}$ and a collision-resistant hash function $\Gamma : \mathcal{KW} \to \mathcal{Y}$. Set the system parameter $P =< \mathsf{param}, \mathcal{L}, \Gamma >$.

$\mathsf{KeyGen}(P)$. Take as input P, run the algorithms $< \mathsf{HashKG}, \mathsf{ProjHash} >$ to generate the public/private key pairs (pk_{FS}, sk_{FS}), (pk_{BS}, sk_{BS}) for the front server and the back server respectively.

$$pk_{FS} \leftarrow \mathsf{HashKG}(P), sk_{FS} = \mathsf{ProjKG}(pk_{FS}),$$

$$pk_{BS} \leftarrow \mathsf{HashKG}(P), sk_{BS} = \mathsf{ProjKG}(pk_{BS}).$$

$\mathsf{DS\text{-}PEKS}(pk_{FS}, pk_{BS}, kw_1)$. Take as input $params$, pk_{FS}, pk_{BS} and the keyword kw_1, pick a word $W_1 \in \mathcal{L}$ randomly with the witness w_1 and generate the PEKS ciphertext CT_{kw_1} of kw_1 as following.

$$x_1 = \mathsf{ProjHash}(pk_{FS}, W_1, w_1),$$

$$y_1 = \mathsf{ProjHash}(pk_{BS}, W_1, w_1),$$

$$C_1 = x_1 \circledast y_1 \circledast \Gamma(kw_1).$$

Set $CT_{kw_1} =< W_1, C_1 >$ and return CT_{kw_1} as the keyword ciphertext.

$\mathsf{DS\text{-}Trapdoor}(pk_{FS}, pk_{BS}, kw_2)$. Take as input $params$, pk_{FS}, pk_{BS} and the keyword kw_2, pick a word $W_2 \in \mathcal{L}$ randomly with the witness w_2 and generate the

trapdoor T_{kw_2} of kw_2 as follows.

$$x_2 = \mathsf{ProjHash}(pk_{FS}, W_2, w_2),$$

$$y_2 = \mathsf{ProjHash}(pk_{BS}, W_2, w_2),$$

$$C_2 = x_2 \circledast y_2 \circledast \Gamma(kw_2)^{-1}.$$

Set $T_{kw_2} =< W_2, C_2 >$ and return T_{kw_2} as the trapdoor.

$\mathsf{FrontTest}(sk_{FS}, CT_{kw}, T_{kw})$. Takes as input *params*, the front server's secret key sk_{FS}, the PEKS ciphertext $CT_{kw_1} =< W_1, C_1 >$ and the trapdoor $T_{kw_2} =< W_2, C_2 >$, pick $\Delta w \in \mathcal{W}$ randomly, generate the internal testing-state C_{ITS} as follows.

$$W = W_1 \circledcirc W_2,$$

$$x = \mathsf{Hash}(sk_{FS}, W),$$

$$C = C_1 \circledast C_2 \circledast x^{-1},$$

$$W^* = \Delta w \otimes W, C^* = \Delta w \bullet C.$$

Set $C_{ITS} =< W^*, C^* >$ and return C_{ITS} as the internal testing-state.

$\mathsf{BackTest}(sk_{BS}, C_{ITS})$. Takes as input *params*, the back server's secret key sk_{BS} and the internal testing-state $C_{ITS} =< W^*, C^* >$, test as follows.

$$\mathsf{Hash}(sk_{BS}, W^*) \stackrel{?}{=} C^*$$

If yes output 1, else output 0.

Correctness Analysis. One can see that the correctness of this construction is guaranteed by the important properties of the LH-SPHF. To be more precise, we give the analysis as follows.

For the algorithm $\mathsf{FrontTest}$, we have

$$\begin{aligned} x &= \mathsf{Hash}(sk_{FS}, W) \\ &= \mathsf{Hash}(sk_{FS}, W_1 \odot W_2) \\ &= \mathsf{Hash}(sk_{FS}, W_1) \circledast \mathsf{Hash}(sk_{FS}, W_2) \\ &= \mathsf{ProjHash}(pk_{FS}, W_1, w_1) \circledast \mathsf{ProjHash}(pk_{FS}, W_2, w_2) \\ &= x_1 \circledast x_2. \end{aligned}$$

Therefore,

$$\begin{aligned} C &= C_1 \circledast C_2 \circledast x^{-1} \\ &= x_1 \circledast y_1 \circledast \Gamma(kw_1) \circledast x_2 \circledast y_2 \circledast \Gamma(kw_2)^{-1} \circledast (x_1 \circledast x_2)^{-1} \\ &= y_1 \circledast y_2 \circledast \Gamma(kw_1) \circledast \Gamma(kw_2)^{-1}. \end{aligned}$$

For the algorithm $\mathsf{BackTest}$, we have

$$\begin{aligned} & \mathsf{Hash}(sk_{BS}, W^*) \\ = \ & \mathsf{Hash}(sk_{BS}, \Delta w \otimes W) \\ = \ & \Delta w \bullet \mathsf{Hash}(sk_{BS}, W) \\ = \ & \Delta w \bullet \mathsf{Hash}(sk_{BS}, W_1 \circledcirc W_2). \\ = \ & \Delta w \bullet (\mathsf{Hash}(sk_{BS}, W_1) \circledast \mathsf{Hash}(sk_{BS}, W_2)) \\ = \ & \Delta w \bullet (\mathsf{ProjHash}(pk_{BS}, W_1, w_1) \circledast \mathsf{ProjHash}(pk_{BS}, W_2, w_2)) \\ = \ & \Delta w \bullet (y_1 \circledast y_2). \end{aligned}$$

It is easy to see that if $kw_1 = kw_2$, then $\mathsf{Hash}(sk_{BS}, W^*) = \Delta w \bullet C = C^*$. Otherwise, $\mathsf{Hash}(sk_{BS}, W^*) \neq C^*$ due to the collision-resistant property of the hash function Γ.

5.4.2 Security Analysis

In this subsection, we analyse the security of the above generic construction.

Theorem 5.1 *The generic construction $\mathcal{DS}$-$\mathcal{PEKS}$ is semantically secure under chosen keyword attacks.*

The above theorem can be obtained from the following two lemmas.

Lemma 5.2 *For any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{SS\text{-}CKA}}_{\mathcal{FS},\mathcal{A}}(\ell)$ is a negligible function.*

Proof: We define a sequence of games as follows.

$\mathsf{Game}0$. This is the original SS-CKA game against the adversarial front server.

Setup. The challenger runs the Setup, KeyGen to generate system parameter $params$, key pairs (pk_{FS}, sk_{FS}) and (pk_{BS}, sk_{BS}). It then gives adversary $\mathcal{A}$ the key pair $(P, pk_{FS}, sk_{FS}, pk_{BS})$.

$\mathsf{Test\ Query\text{-}I}$. The adversary makes a query on $< kw, CT >$. Suppose $CT = (W, C)$, the challenger computes the following.

$$T \leftarrow \mathsf{DS\text{-}Trapdoor}(pk_{FS}, pk_{BS}, kw),$$

$$C_{ITS} \leftarrow \mathsf{FrontTest}(sk_{FS}, C, T).$$

The challenger then runs the algorithm $\mathsf{BackTest}(sk_{BS}, C_{ITS})$ and returns the output to the adversary.

Challenge. $\mathcal{A}$ chooses two keywords kw_0, kw_1 and sends kw_0, kw_1 to the challenger. The challenger first picks $b \xleftarrow{\$} \{0,1\}$, and then picks a word $W_1 \in \mathcal{L}$ randomly with the witness w_1 and generates the PEKS ciphertext CT^*_{kw} of kw_b as follows.

$$x_1 = \mathsf{ProjHash}(pk_{FS}, W_1, w_1), y_1 = \mathsf{ProjHash}(pk_{BS}, W_1, w_1),$$

$$C_1 = x_1 \circledast y_1 \circledast \Gamma(kw_b).$$

The challenger sets $CT^*_{kw} =< W_1, C_1 >$ as the keyword ciphertext and sends CT^*_{kw} to $\mathcal{A}$.

Test Query-II. The procedure is the same as that in Test Query-I.

Output. Finally, $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$ on b and wins the game if $b = b'$.

We define the advantage of $\mathcal{A}$ in Game0 as $\mathsf{Adv}^{\mathsf{Game0}}_{\mathcal{FS},\mathcal{A}}(\ell)$ and have that

$$\mathsf{Adv}^{\mathsf{Game0}}_{\mathcal{FS},\mathcal{A}}(\ell) = \mathsf{Adv}^{\mathsf{SS-CKA}}_{\mathcal{FS},\mathcal{A}}(\ell)$$

as Game0 strictly follows the SS-CKA model.

Game1. Let Game1 be the same game as Game0, except that the challenger chooses $y_1 \xleftarrow{\$} \mathcal{Y}$ instead of computing y_1 as $\mathsf{ProjHash}(pk_{BS}, W_1, w_1)$. Due to the correctness and pseudo-randomness of $\mathcal{SPHF}$, that is, the distribution $\{(W_1, pk_{BS}, y_1)|y_1 = \mathsf{ProjHash}(pk_{BS}, W_1, w_1)\}$ is computationally indistinguishable from the distribution $\{(W_1, pk_{BS}, y_1)|y_1 \xleftarrow{\$} \mathcal{Y}\}$, we have that

$$|\mathsf{Adv}^{\mathsf{Game1}}_{\mathcal{FS},\mathcal{A}}(\ell) - \mathsf{Adv}^{\mathsf{Game0}}_{\mathcal{FS},\mathcal{A}}(\ell)| \leq \mathsf{Adv}^{\mathsf{PR}}_{\mathcal{A}}(\ell).$$

Game2. Let Game2 be the same game as Game1, except that the challenger chooses $C_1 \xleftarrow{\$} \mathcal{Y}$ instead of computing $C_1 = x_1 \circledast y_1 \circledast \Gamma(kw_b)$. We can see that

$$\mathsf{Adv}^{\mathsf{Game2}}_{\mathcal{FS},\mathcal{A}}(\ell) = \mathsf{Adv}^{\mathsf{Game1}}_{\mathcal{FS},\mathcal{A}}(\ell).$$

It is easy to see that the adversary in Game2 can only win with probability $1/2$ as C_1 is independent of b. Therefore, we have that $\mathsf{Adv}^{\mathsf{Game2}}_{\mathcal{FS},\mathcal{A}}(\ell) = 0$.

Therefore, from Game0, Game1 and Game2, we have that

$$|\mathsf{Adv}^{\mathsf{Game2}}_{\mathcal{FS},\mathcal{A}}(\ell) - \mathsf{Adv}^{\mathsf{SS-CKA}}_{\mathcal{FS},\mathcal{A}}(\ell)| \leq \mathsf{Adv}^{\mathsf{PR}}_{\mathcal{A}}(\ell).$$

As $\mathsf{Adv}^{\mathsf{Game2}}_{\mathcal{FS},\mathcal{A}}(\ell) = 0$ and $\mathsf{Adv}^{\mathsf{PR}}_{\mathcal{A}}(\ell)$ is negligible, we have that $\mathsf{Adv}^{\mathsf{SS\text{-}CKA}}_{\mathcal{FS},\mathcal{A}}(\ell)$ is also negligible, which completes the proof.

Lemma 5.3 *For any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{SS\text{-}CKA}}_{\mathcal{BS},\mathcal{A}}(\ell)$ is a negligible function.*

The proof of **Lemma** 5.3 can be easily obtained by following the proof of **Lemma** 5.2, and hence is omitted.

Theorem 5.4 *The generic construction $\mathcal{DS}$-$\mathcal{PEKS}$ is secure against keyword guessing attack.*

The above theorem can be obtained from the following lemmas.

Lemma 5.5 *For any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{IND\text{-}KGA}}_{\mathcal{FS},\mathcal{A}}(\ell)$ is a negligible function.*

Lemma 5.6 *For any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{IND\text{-}KGA}}_{\mathcal{BS},\mathcal{A}}(\ell)$ is a negligible function.*

The proofs of **Lemma** 5.5 and **Lemma** 5.6 are similar to those of **Lemma** 5.2 and **Lemma** 5.3 as the generation of a trapdoor is the same as that of a PEKS ciphertext, and the security model of IND-KGA is also similar to that of SS-CKA. Therefore, we omit the proof details here. For the security against the keyword guessing attack-II, we have the following lemma.

Lemma 5.7 *For any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{IND\text{-}KGA\text{-}II}}_{\mathcal{BS},\mathcal{A}}(\ell)$ is a negligible function.*

Proof: In the IND-KGA-II game, if $b_1 = b_2$, then it is easy to see that the adversary has no advantage since the two keywords are canceled out in the internal testing-state C_{ITS}, which means C_{ITS} is independent of the keywords. In the following, we focus on the case that $b_1 \neq b_2$.

Here, we use the game-hopping technique again to prove this lemma. We define a sequence of attack games as follows.

Game0. Let the original IND-CKA game be Game0.

Setup. The challenger runs the Setup, KeyGen to generate system parameter $params$, key pairs (pk_{FS}, sk_{FS}) and (pk_{BS}, sk_{BS}). It gives adversary $\mathcal{A}$ the key pairs $(P, pk_{FS}, pk_{BS}, sk_{BS})$.

Challenge. $\mathcal{A}$ chooses challenge keywords kw_0, kw_1, kw_2 adaptively and sends them to the challenger. The challenger firstly picks $\{b_1, b_2\} \subset \{0, 1, 2\}$ randomly.

The challenger picks two words $W_1, W_2 \in \mathcal{L}$ randomly with the witness w_1, w_2 respectively and generates the internal testing-state C_{ITS} as follows.

$$x_1 = \mathsf{ProjHash}(pk_{FS}, W_1, w_1), y_1 = \mathsf{ProjHash}(pk_{BS}, W_1, w_1),$$

$$x_2 = \mathsf{ProjHash}(pk_{FS}, W_2, w_2), y_2 = \mathsf{ProjHash}(pk_{BS}, W_2, w_2),$$

$$W = W_1 \odot W_2, x = \mathsf{Hash}(sk_{FS}, W),$$

$$C = x_1 \circledast x_2 \circledast y_1 \circledast y_2 \circledast x^{-1} \circledast \Gamma(kw_{b_1}) \circledast \Gamma(kw_{b_2})^{-1},$$

$$W^* = \Delta w \otimes W, C^* = \Delta w \bullet C.$$

Set $C^*_{ITS} =< W^*, C^* >$ and return C^*_{ITS} to $\mathcal{A}$.

Output. Finally, $\mathcal{A}$ outputs its guess on $\{b_1, b_2\}$ as $\{b'_1, b'_2\} \subset \{0, 1, 2\}$ and wins the game if $\{b'_1, b'_2\} = \{b_1, b_2\}$.

We define the advantage of $\mathcal{A}$ in **Game0** as $\mathsf{Adv}^{\mathsf{Game0}}_{\mathcal{BS},\mathcal{A}}(\ell)$ and have that

$$\mathsf{Adv}^{\mathsf{Game0}}_{\mathcal{BS},\mathcal{A}}(\ell) = \mathsf{Adv}^{\mathsf{IND\text{-}KGA\text{-}II}}_{\mathcal{BS},\mathcal{A}}(\ell)$$

as **Game0** strictly follows the IND-KGA-II model.

Game1. Let **Game1** be the same game as **Game0**, except that the challenger chooses $y \xleftarrow{\$} \mathcal{Y}$ and computes C^* as follows.

$$C^* = (x_1 \circledast x_2 \circledast y_1 \circledast y_2 \circledast x^{-1}) \circledast y.$$

In other words, the challenger replaces the part $\Delta w \cdot (\Gamma(kw_{b_1}) \circledast \Gamma(kw_{b_2})^{-1})$ with a random chosen element $y \in \mathcal{Y}$ during the generation of C^*. We now prove that the replacement in this way can make at most a negligible difference, that is,

Claim. *For any PPT adversary $\mathcal{A}$,*

$$|\mathsf{Adv}^{\mathsf{Game1}}_{\mathcal{BS},\mathcal{A}}(\ell) - \mathsf{Adv}^{\mathsf{Game0}}_{\mathcal{BS},\mathcal{A}}(\ell)| \leq \mathsf{Adv}^{\mathsf{PR}}_{\mathcal{A}}(\ell).$$

Proof: Since the language $\mathcal{L}$ is a linear and homomorphic language, we have that the witness of W^* is $w^* = \Delta w \otimes w$ where w is the witness of W. Then based on our definition of LH-SPHF there exists a projection key hp' that

$$\mathsf{ProjHash}(\mathsf{hp}', W, w) = \Gamma(kw_{b_1}) \circledast \Gamma(kw_{b_2})^{-1}.$$

As $\mathcal{SPHF}$ is a Lin-Hom SPHF, we have that

$$\begin{aligned} \mathsf{ProjHash}(\mathsf{hp}', W^*, w^*) &= \Delta w \bullet \mathsf{ProjHash}(\mathsf{hp}', W, w) \\ &= \Delta w \bullet (\Gamma(kw_{b_1}) \circledast \Gamma(kw_{b_2})^{-1}). \end{aligned}$$

Moreover, the distribution $\{(W^*, \mathsf{hp}', y) | y = \mathsf{ProjHash}(\mathsf{hp}', W^*, w^*)\}$ is computationally indistinguishable from the distribution $\{(W^*, \mathsf{hp}', y) | y \xleftarrow{\$} \mathcal{Y}\}$ due to the correct-

ness and pseudo-randomness of $\mathcal{SPHF}$. Therefore, we have that

$$|\mathsf{Adv}^{\mathsf{Game1}}_{\mathcal{BS},\mathcal{A}}(\ell) - \mathsf{Adv}^{\mathsf{Game0}}_{\mathcal{BS},\mathcal{A}}(\ell)| \leq \mathsf{Adv}^{\mathsf{PR}}_{\mathcal{A}}(\ell).$$

Game2. Let Game2 be the same game as Game1, except that the challenger chooses $C^* \xleftarrow{\$} \mathcal{Y}$. We can see that

$$\mathsf{Adv}^{\mathsf{Game2}}_{\mathcal{BS},\mathcal{A}}(\ell) = \mathsf{Adv}^{\mathsf{Game1}}_{\mathcal{BS},\mathcal{A}}(\ell).$$

It is easy to see that the adversary can only win in the Game2 with probability $1/3$ as C^* is independent of b_1, b_2. Therefore, we have that $\mathsf{Adv}^{\mathsf{Game2}}_{\mathcal{BS},\mathcal{A}}(\ell) = 0$.

Therefore, from Game0, Game1 and Game2, we have that

$$|\mathsf{Adv}^{\mathsf{Game2}}_{\mathcal{BS},\mathcal{A}}(\ell) - \mathsf{Adv}^{\mathsf{IND\text{-}KGA\text{-}II}}_{\mathcal{BS},\mathcal{A}}(\ell)| \leq \mathsf{Adv}^{\mathsf{PR}}_{\mathcal{A}}(\ell).$$

As $\mathsf{Adv}^{\mathsf{Game2}}_{\mathcal{BS},\mathcal{A}}(\ell) = 0$ and $\mathsf{Adv}^{\mathsf{PR}}_{\mathcal{A}}(\ell)$ is negligible, we have that $\mathsf{Adv}^{\mathsf{IND\text{-}KGA\text{-}II}}_{\mathcal{BS},\mathcal{A}}(\ell)$ is also a negligible function, which proves the lemma.

5.5 The Proposed DS-PEKS Scheme

In this section, we first introduce a concrete LH-SPHF and then show how to construct a DS-PEKS based on it.

5.5.1 LH-SPHF Based on The Diffie-Hellman Language

In the following, we present the language we use in the instantiation of LH-SPHF. Specifically, we introduce the Diffie Hellman language $\mathcal{L}_{DH}$ and show how to construct a LH-SPHF based on it.

Diffie-Hellman Language. Let $\mathbb{G}$ be a group of primer order *params* and $g_1, g_2 \in \mathbb{G}$ the generators of $\mathbb{G}$. The Diffie-Hellman Language is defined as follows.

$$\mathcal{L}_{DH} = \{(u_1, u_2) | \exists r \in \mathbb{Z}_p, s.t., u_1 = g_1^r, u_2 = g_2^r\}$$

One can see that the witness space of $\mathcal{L}_{DH}$ is $\mathcal{W} = \mathbb{Z}_p$ and $\mathcal{L}_{DH} \subset \mathbb{G}^2$. We have the following theorems.

Theorem 5.8 *The Diffie-Hellman language $\mathcal{L}_{DH}$ is a linear and homomorphic language.*

Proof: We show that $\mathcal{L}_{DH}$ satisfies the properties of a linear and homomorphic language.

1). For a word $W = (g_1^r, g_2^r)$ with the witness $w = r \in \mathbb{Z}_p$ and $\Delta r \in \mathbb{Z}_p$, we have,

$$W^* = \Delta r \otimes W = (g_1^{r\Delta r}, g_2^{r\Delta r}) \in \mathcal{L}_{DH},$$

which has the witness $w^* = r\Delta r$.

2). For any two word $W_1 = (g_1^{r_1}, g_2^{r_1})$ (witness $w_1 = r_1$), $W_2 = (g_1^{r_2}, g_2^{r_2})$ (witness $w_2 = r_2$), we have,

$$W^* = W_1 \circledcirc W_2 = (g_1^{r_1+r_2}, g_2^{r_1+r_2}) \in \mathcal{L}_{DH},$$

which has the witness $w^* = w_1 \oplus w_2 = r_1 + r_2$.

LH-SPHF on $\mathcal{L}_{DH}$. Here we show how to construct an LH-SPHF (denoted by $\mathcal{SPHF}_{DH}$) over the language $\mathcal{L}_{DH} \subset \mathcal{X} = \mathbb{G}^2$ onto the group $\mathcal{Y} = \mathbb{G}$. The concrete construction is as follows.

$\mathsf{SPHFSetup}(1^\ell)$: $\mathsf{param} = (\mathbb{G}, p, g_1, g_2)$;

$\mathsf{HashKG}(\mathcal{L}_{DH}, \mathsf{param})$: $\mathsf{hk} = (\alpha_1, \alpha_2) \xleftarrow{\$} \mathbb{Z}_p^2$;

$\mathsf{ProjKG}(\mathsf{hk}, (\mathcal{L}_{DH}, \mathsf{param}))$: $\mathsf{hp} = g_1^{\alpha_1} g_2^{\alpha_2} \in \mathbb{Z}_p$;

$\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}_{DH}, \mathsf{param}), W = (g_1^r, g_2^r))$: $\mathsf{hv} = g_1^{r\alpha_1} g_2^{r\alpha_2} \in \mathbb{Z}_p$;

$\mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}_{DH}, \mathsf{param}), W = (g_1^r, g_2^r), w = r)$: $\mathsf{hv}' = \mathsf{hp}^r \in \mathbb{Z}_p$.

Theorem 5.9 *$\mathcal{SPHF}_{DH}$ is a smooth projective hash function.*

Proof: We show that $\mathcal{SPHF}_{DH}$ is projective, smooth and pseduo-random.

1). *Correctness.* With the above notations, we have

$$\mathsf{Hash}(\mathsf{hk}, (\mathcal{L}_{DH}, \mathsf{param}), W) = g_1^{r\alpha_1} g_2^{r\alpha_2} = \mathsf{hp}^r = \mathsf{ProjHash}(\mathsf{hp}, (\mathcal{L}_{DH}, \mathsf{param}), W, w).$$

2). *Smoothness.* Suppose $g_2 = g_1^\theta$. Note that $\mathsf{hp} = g_1^{\alpha_1} g_2^{\alpha_2}$ which constraints (α_1, α_2) to satisfy

$$\log_{g_1} \mathsf{hp} = \alpha_1 + \theta\alpha_2. \tag{5.1}$$

Let $W' = (g_1^{r_1}, g_2^{r_2}) \in \mathcal{X} \backslash \mathcal{L}_{DH}$ where $r_1 \neq r_2$, then the hash value hv_1 of W' is

$$\mathsf{hv}_1 = \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}_{DH}, \mathsf{param}), W') = g_1^{r_1\alpha_1} g_2^{r_2\alpha_2},$$

which also constraints (α_1, α_2) to satisfy

$$\log_{g_1} \mathsf{hv}_1 = r_1\alpha_1 + r_2\theta\alpha_2. \tag{5.2}$$

For the above two equations, we have

$$(\alpha_1, \alpha_2) \cdot \mathbf{A} = (\log_{g_1} \mathsf{hp}, \log_{g_1} \mathsf{hv}_1),$$

where $\mathbf{A}$ is a matrix defined as

$$\mathbf{A} = \begin{bmatrix} 1 & r_1 \\ \theta & r_2\theta \end{bmatrix}.$$

Since the determinant of $\mathbf{A}$ is $\theta \cdot (r_2 - r_1)$ that is nonzero ($r_1 \neq r_2$), we have that the equation (1) is independent of the equation (2). Therefore, we have that hv_1 is statistically indistinguishable from any element randomly chosen from $\mathbb{G}$.

3). *Pseudo-randomness.* In the following, we prove that $\mathcal{SPHF}_{DH}$ is pseudo-random. More precisely, we show that if there is an adversary that wins in the PR-game with non-negligible probability, then we can use it to solve the decision Diffie-Hellman (DDH) problem. Formally, suppose the adversary $\mathcal{A}$ has the advantage $\mathsf{Adv}^{\mathsf{PR}}_{\mathcal{A}}$, then we have an algorithm $\mathcal{B}$ that solves the DDH problem with advantage $\mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{B}} = 2 \cdot \mathsf{Adv}^{\mathsf{PR}}_{\mathcal{A}}$. Let the DDH instance be (g, g^a, g^b, Z), $\mathcal{B}$ then works as follows to decide whether Z equals g^{ab} (outputs 1) or not (outputs 0).

Setup. $\mathcal{B}$ sets $g_1 = g, g_2 = g^a$, picks $\mathsf{hk} = (\alpha_1, \alpha_2) \xleftarrow{\$} \mathbb{Z}_p$ and sets $\mathsf{param} = (g_1, g_2, \mathbb{G}, p)$computes $\mathsf{hp} = g_1^{\alpha_1} g_2^{\alpha_2}$. $\mathcal{B}$ sends $(\mathsf{param}, \mathsf{hp})$ to $\mathcal{A}$.

Challenge. The challenger computes $\mathsf{hv} = g_1^{b\alpha_1} Z^{\alpha_2}$ and sends (g_1^b, Z, hv) to $\mathcal{A}$

Output. Finally, $\mathcal{A}$ outputs its guess b' which $\mathcal{B}$ outputs as its solution to the given DDH problem.

Let $W' = (g_1^b, Z)$. If $Z = g^{ab}$, then $W' \in \mathcal{L}_{DH}$ as $Z = g^{ab} = g_2^b$ which is the case that $b = 1$ in the PR-game. If $Z \neq g^{ab}$, then $W' \notin \mathcal{L}_{DH}$ and hence $\mathsf{hv} = g_1^{b\alpha_1} Z^{\alpha_2} = \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}_{DH}, \mathsf{param}), W')$ is statistically indistinguishable from a random element $v \xleftarrow{\$} \mathcal{Y}$ which is the case $b = 0$. Therefore, we have that

$$\begin{aligned} \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{B}} &= \Pr[b' = 1 | Z = g^{ab}] - \Pr[b' = 1 | Z \neq g^{ab}] \\ &= \Pr[b' = 1 | Z = g^{ab}] - (1 - \Pr[b' = 0 | Z \neq g^{ab}]) \\ &= \mathsf{Adv}^{\mathsf{PR}}_{\mathcal{A}} + 1/2 - (1 - (\mathsf{Adv}^{\mathsf{PR}}_{\mathcal{A}} + 1/2)) \\ &= 2 \cdot \mathsf{Adv}^{\mathsf{PR}}_{\mathcal{A}}. \end{aligned}$$

Theorem 5.10 *$\mathcal{SPHF}_{DH}$ is a Lin-Hom SPHF.*

Proof: As shown above, $\mathcal{L}_{DH}$ is a linear and homomorphic language. Now we show that $\mathcal{SPHF}_{DH}$ satisfies the following properties.

1). For a word $W = (g_1^r, g_2^r)$ with the witness $w = r \in \mathbb{Z}_p$ and $\Delta r \in \mathbb{Z}_p$, we have $\Delta r \otimes W = (g_1^{r\Delta r}, g_2^{r\Delta r})$. Therefore, we have

$$\begin{aligned} & \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}_{DH}, \mathsf{param}), \Delta r \otimes W) \\ = \; & g_1^{r\Delta r\alpha_1} g_2^{r\Delta r\alpha_2} \\ = \; & (g_1^{r_1\alpha_1} g_2^{r_2\alpha_2})^{\Delta r} \\ = \; & \Delta r \bullet \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}_{DH}, \mathsf{param}), W). \end{aligned}$$

2). For any two word $W_1 = (g_1^{r_1}, g_2^{r_1})$, $W_2 = (g_1^{r_2}, g_2^{r_2})$, we have $W_1 \circledcirc W_2 = (g_1^{r_1+r_2}, g_2^{r_1+r_2}) \in \mathcal{L}_{DH}$. Therefore, we have

$$\begin{aligned} & \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}_{DH}, \mathsf{param}), W_1 \circledcirc W_2) \\ = \; & g_1^{(r_1+r_2)\cdot\alpha_1} \cdot g_2^{(r_1+r_2)\cdot\alpha_2} \\ = \; & g_1^{r_1\alpha_1} g_2^{r_1\alpha_2} \cdot g_1^{r_2\alpha_1} g_2^{r_2\alpha_2} \\ = \; & \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W_1) \circledast \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W_2). \end{aligned}$$

This proves the theorem.

5.5.2 A Concrete DS-PEKS Scheme Based on $\mathcal{SPHF}_{DH}$

Construction. The concrete scheme based on $\mathcal{SPHF}_{DH}$ introduced above is as follows.

Setup. Let $\mathbb{G}$ be a group with prime order *params* and g_1, g_2 be two generators of $\mathbb{G}$. $H : \{0,1\}^* \to \mathbb{G}$ is a collision-resistant hash function. The system parameter is $(p, g_1, g_2, \mathbb{G}, H)$.

KeyGen. Pick $\alpha_1, \alpha_2, \beta_1, \beta_2$ from $\mathbb{Z}_p$ randomly and generate the public/secret key pair (pk_{FS}, sk_{FS}), (pk_{BS}, sk_{BS}) for the front server and the back server respectively as follows,

$$pk_{FS} = h_1 = g_1^{\alpha_1} g_2^{\alpha_2}, sk_{FS} = (\alpha_1, \alpha_2),$$

$$pk_{BS} = h_2 = g_1^{\beta_1} g_2^{\beta_2}, sk_{BS} = (\beta_1, \beta_2).$$

PEKS. For a keyword kw_1, pick $r_1 \xleftarrow{\$} \mathbb{Z}_p$, generate the PEKS ciphertext of kw_1 as follows,

$$CT_{kw} = (g_1^{r_1},\ g_2^{r_1},\ h_1^{r_1} h_2^{r_1} H(kw_1))$$

Trapdoor. For a keyword kw_2, pick $r_2 \xleftarrow{\$} \mathbb{Z}_p$, generate the trapdoor of kw_2 as follows,

$$T_{kw} = (g_1^{r_2},\ g_2^{r_2},\ h_1^{r_2} h_2^{r_2} H(kw_2)^{-1})$$

FrontTest. Pick $\gamma \xleftarrow{\$} \mathbb{Z}_p$ and compute the internal testing-state C_{ITS} as follows,

$$CT_{kw} \cdot T_{kw'} = (C_1, C_2, C_3) = (g_1^{r_1+r_2},\ g_2^{r_1+r_2},\ h_1^{r_1+r_2} h_2^{r_1+r_2} (H(kw_1)H(kw_2)^{-1}))$$

$$C_{ITS} = (C_1^*, C_2^*, C_3^*) = (C_1^{\gamma}, C_2^{\gamma}, (C_3/(C_1^{\alpha_1} C_2^{\alpha_2}))^{\gamma})$$

BackTest. For an internal testing-state $C_{ITS} = (C_1^*, C_2^*, C_3^*)$, do the testing as follows,

$$C_1^{*\beta_1} C_2^{*\beta_2} \stackrel{?}{=} C_3^*.$$

If the equation holds, outputs 1, otherwise outputs 0.

Correctness. It is easy to obtain the correctness as $C_1^{\alpha_1} C_2^{\alpha_2} = g_1^{(r_1+r_2)\alpha_1} g_2^{(r_1+r_2)\alpha_2} = h_1^{r_1+r_2}$ and we have that,

$$C_3^* = (C_3/(C_1^{\alpha_1} C_2^{\alpha_2}))^{\gamma} = h_2^{(r_1+r_2)\gamma} (H(kw)H(kw')^{-1})^{\gamma}.$$

Therefore, if $kw_1 = kw_2$, then

$$C_1^{*\beta_1} C_2^{*\beta_2} = C_3^*,$$

otherwise, the equation does not hold due to the collision resistance property of H.

Security. The following corollary can be obtained directly from **Theorem** 7.1 and **Theorem** 7.2.

Corollary 5.11 *The concrete construction is a secure DS-PEKS scheme.*

5.6 Performance Evaluation

In this section, we first give a comparison between existing schemes and our scheme in terms of computation, size and security. We then evaluate its performance in experiments.

Computation Costs. All the existing schemes [BCOP04, RPSL10, XJWW13] require the pairing computation during the generation of PEKS ciphertext and testing and hence are less efficient than our scheme, which does not need any pairing computation. In our scheme, the computation cost of PEKS generation, trapdoor generation and testing are $4\text{Exp}_{\mathbb{G}_1}+1\text{Hash}_{\mathbb{G}_1}+2\text{Mul}_{\mathbb{G}_1}$, $4\text{Exp}_{\mathbb{G}_1}+1\text{Hash}_{\mathbb{G}_1}+2\text{Mul}_{\mathbb{G}_1}$,

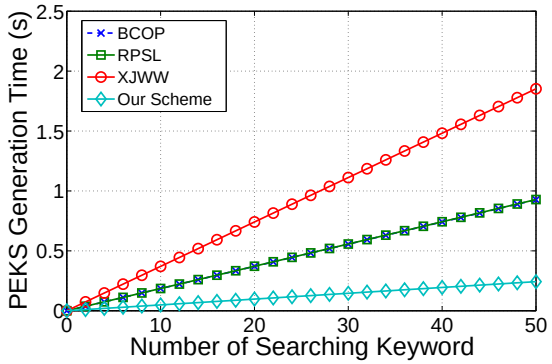


Figure 5.2: Computation Cost of PEKS Generation in Different Schemes

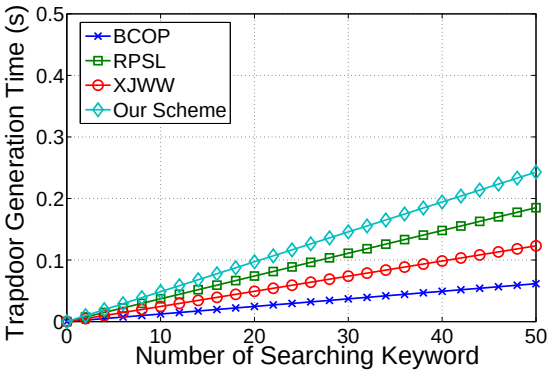


Figure 5.3: Computation Cost of Trapdoor Generation in Different Schemes

and $7\text{Exp}_{\mathbb{G}_1}+3\text{Mul}_{\mathbb{G}_1}$ respectively, where $\text{Exp}_{\mathbb{G}_1}$ denotes the computation of one exponentiation in $\mathbb{G}_1$, $\text{Mul}_{\mathbb{G}_1}$ denotes the costs of one multiplication in $\mathbb{G}_1$, $\text{Mul}_{\mathbb{G}_1}$ and $\text{Hash}_{\mathbb{G}_1}$ respectively denote the cost of one multiplication and one hashing operation in $\mathbb{G}_1$.

Communication Costs. It is worth noting that although our scheme outperforms the existing schemes in terms of computational cost, it requires more communication costs. The size of PEKS ciphertext and trapdoor in our scheme is slightly larger than that of the existing schemes. In particular, both the PEKS and trapdoor of our scheme consist of three group elements ($3|\mathbb{G}_1|$), while the PEKS size (bits) of [BCOP04], [RPSL10], [XJWW13] is $1|\mathbb{G}_1|+\log p$, $1|\mathbb{G}_1|+\lambda$ and $3|M|+2|\mathbb{G}_1|$ respectively, and the trapdoor size is $1|\mathbb{G}_1|$, $2|\mathbb{G}_1|$ and $2|\mathbb{G}_1|$ respectively. Moreover, our scheme requires additional communication costs between the two servers, since the front server needs to transfer the internal testing-state to the back server for the final testing. The size of the internal testing-state is of size of $3|\mathbb{G}_1|$. We should also note that in [XJWW13], additional communication also occurs between the server and the user.

Experiment Results. To evaluate the efficiency of schemes in experiments, we also implement the scheme utilizing the GNU Multiple Precision Arithmetic (GMP)

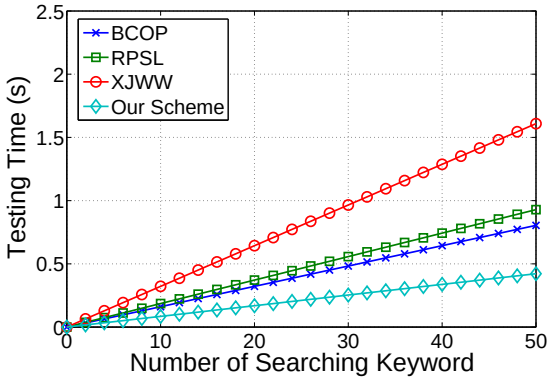


Figure 5.4: Computation Cost of Testing in Different Schemes

library and Pairing Based Cryptography (PBC) library. The following experiments are based on coding language C on Linux system (more precise, 2.6.35-22-generic version) with an Intel(R) Core(TM) 2 Duo CPU of 3.33 GHZ and 2.00-GB RAM. For the elliptic curve, we choose an MNT curve with a base filed size of 159 bits and p=160 bits and $|q|$=80 bits.

As shown in Fig. 6.4, our scheme is the most efficient in terms of PEKS computation. It is because that our scheme does not include pairing computation. Particularly, the scheme [XJWW13] requires the most computation cost due to 2 pairing computation per PEKS generation. As for the trapdoor generation indicated in Figure 6.5, as all the existing schemes do not involve pairing computation, the computation cost is much lower than that of PEKS generation. It is worth noting that the trapdoor generation in our scheme is slightly higher than those of existing schemes due to the additional exponentiation computations. When the searching keyword number is 50, the total computation cost of our scheme is about 0.25 seconds. As illustrated in Fig. 6.6, the scheme [XJWW13] cost the most time due to an additional pairing computation in the exact testing stage. One should note that this additional pairing computation is done on the user side instead of the server. Therefore, it could be the computation burden for users who may use a light device for searching data. In our scheme, although we also require another stage for the testing, our computation cost is actually lower than that of any existing scheme as we do not require any pairing computation and all the searching work is handled by the server.

5.7 Chapter Summary

In this chapter, we proposed a new framework, named Dual- Server Public Key Encryption with Keyword Search (DSPEKS), that can prevent the inside keyword guessing attack which is an inherent vulnerability of the traditional PEKS frame-

work. We also introduced a new Smooth Projective Hash Function (SPHF) and used it to construct a generic DSPEKS scheme. An efficient instantiation of the new SPHF based on the Diffie-Hellman problem is also presented in the chapter, which gives an efficient DS-PEKS scheme without pairings.

Chapter 6

Server-Aided Public Key Encryption with Keyword Search

This chapter also studies the inside (off-line) KGA problem in the conventional PEKS system. Unlike the work introduced in Chapter 5 that aims at completely preventing the inside KGA, the work in this chapter provides a more practical and applicable treatment that works transparently with any existing PEKS system. We formalize a new PEKS system named Server-Aided Public Key Encryption with Keyword Search (SA-PEKS) for the security against the off-line KGA. We then introduce a universal transformation from any PEKS scheme to a secure SA-PEKS scheme using the deterministic blind signature. To illustrate its feasibility, we present the first instantiation of SA-PEKS scheme. Moreover, we describe how to securely implement the client-KS protocol with a rate-limiting mechanism against on-line KGA and evaluate the performance of our solution in experiments. The results show that our proposed scheme enjoys the high efficiency with resistance against off-line and on-line KGAs.

6.1 Introduction

In this work, we aim at designing a more practical treatment to address the security issue of the PEKS system, which has been studied in Chapter 5. Moreover, we are interested in building a system that works transparently with any existing PEKS system. That is, the system will be backward-compatible and make no modification on the implementation details of the underlying PEKS system.

6.1.1 Contributions

The contributions of this chapter are four-fold.

First, we formalize a new PEKS system named *Server-Aided Public Key Encryption with Keyword Search* (SA-PEKS) to address the security vulnerability against (inside) off-line KGA in existing PEKS systems. Unlike prior solutions that primarily alter the traditional PEKS framework and suffer from efficiency issues, we investigate a more practical, easily-to-deploy system to prevent the off-line KGA. Our proposed solution can work transparently with any existing PEKS system and hence is much more applicable in practice.

Secondly, we show a generic construction of SA-PEKS scheme with formal security analysis. Precisely, we propose a universal transformation from any PEKS scheme to an SA-PEKS scheme by utilizing a deterministic blind signature. The advantage of our solution is the prospect of multi-tiered security. In particular, we can achieve the best-case security guarantee when the adversary does not have access to the keyword server (KS). Also, even if the attacker has access to the KS, the KGA is limited to be on-line and hence much less effective, especially when the KS imposes some rate-limiting measures. For the worst case where the private key of KS is compromised to the attacker, our system still preserves the security of the underlying PEKS. Moreover, from the security of the underlying signature scheme, the inputs of the client (user) are also hidden from the KS.

Thirdly, to illustrate the feasibility of the proposed generic transformation, an instantiation of the SA-PEKS scheme is presented in this chapter. Specifically, we instantiate the scheme from the FDH-RSA blind signature and the PEKS scheme proposed by Boneh *et al.* in [BCOP04]. We then show that the construction is secure following the defined models as long as the FDH-RSA has one-more-unforgeability and blindness.

Lastly, we present the implementation of our proposed solution and analyze its performance in experiments. Particularly, we show how to securely implement the client-KS protocol with a rate-limiting mechanism against on-line KGA. Following the designed protocol, we then analyze the performance of the FDH-RSA in terms of latency, packet drop rate, and on-line KGA resistance. Moreover, we evaluate the efficiency of our instantiated scheme. Our results show that the proposed solution can significantly reduce the brute force attack rate without degrading the performance.

6.1.2 Related Work

In [XJWW13], Peng *et al.* proposed the notion of Public-key Encryption with Fuzzy Keyword Search (PEFKS) where each keyword corresponds to an exact trapdoor and a fuzzy trapdoor. The server is only provided with the fuzzy trapdoor and thus can no longer learn the exact keyword since two or more keywords share the same fuzzy keyword trapdoor. However, their scheme suffers from several limitations regarding the security and efficiency. On one hand, the malicious server is still able to identify a small set the underlying keyword belongs to and thus the keyword privacy is not well preserved from the server. On the other hand, their scheme is impractical as the receiver has to locally find the matching ciphertext by using the exact trapdoor to filter out the non-matching ones from the set returned by the server. Another work by Chen *et al.* [CMY+15] proposed a new framework

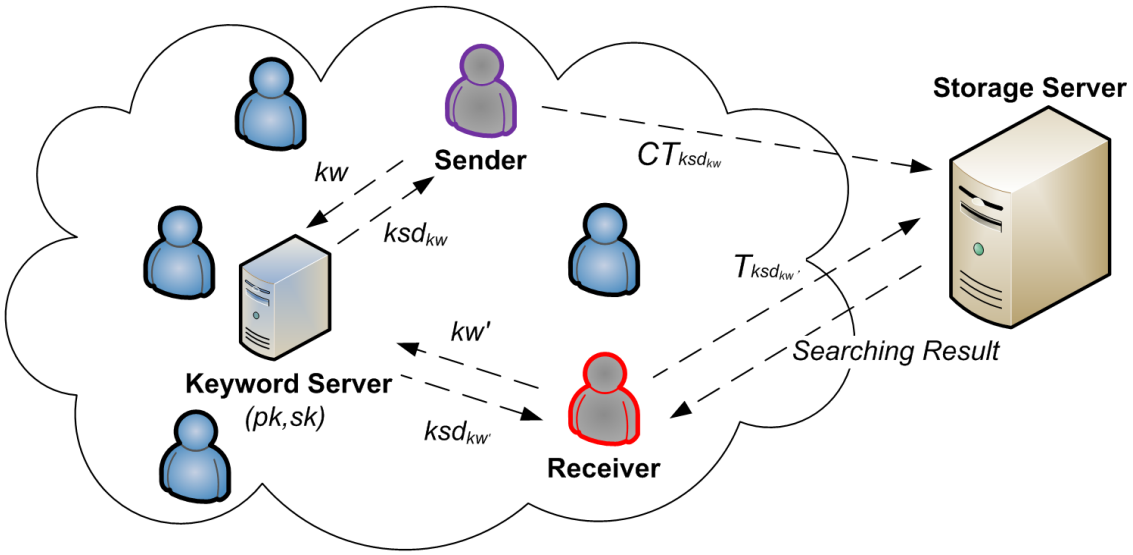


Figure 6.1: System Model of Server-Aided PEKS

of PEKS, namely Dual-Server Public Key Encryption with Keyword Search (DS-PEKS) to achieve the security against inside KGA. Their central idea is to disallow the stand-alone testing of PEKS by splitting the testing functionality of the PEKS system into two parts which are handled by two independent servers. Therefore, the security against the off-line KGA can be obtained as long as the two servers do not collude. Nevertheless, the two-server PEKS may still suffer from the inefficiency as the keyword searching is now separately processed by two servers. Another efficiency issue is that the communication between the two servers can cost a lot of bandwidths especially when the data size is large.

6.2 Server-Aided PEKS

In this section, we formally define the Server-Aided PEKS (SA-PEKS) for the security against the off-line KGA.

6.2.1 Overview

SA-PEKS is motivated by the observation that the off-line KGA can be dealt with by employing a semi-trusted third party, namely *Keyword Server* (KS) which is separated from the *Storage Server* (SS), as shown in Figure.6.1. Roughly speaking, in an SA-PEKS system, the KS owns the public/secret key pair (pk, sk). Users authenticate themselves to the KS and are provisioned with per-user credentials. Different from the PEKS framework where the PEKS ciphertext and the trapdoor are derived from the original keyword directly, the user needs to interact with the KS in an authenticated way to obtain the pre-processed keyword, namely *KS-derived*

keyword, before the generation of the PEKS ciphertext and the trapdoor.

More specifically, given an original keyword kw, the sender has to access the KS through authentication and run an interactive protocol with the KS. At the end of the protocol execution, the sender gets the corresponding KS-derived keyword of kw as ksd_{kw}. The sender then generates the PEKS ciphertext by regarding the KS-derived keyword ksd_{kw} as the final keyword. Similarly, taking as input a specified keyword kw', the receiver runs the interactive protocol with the KS to obtain the KS-derived keyword $ksd_{kw'}$ and then generates the corresponding trapdoor. It is required that the derivation algorithm from original keyword to KS-derived keyword should be deterministic, otherwise the SA-PEKS cannot work correctly. That is, if $kw = kw'$, then we have that $ksd_{kw} = ksd_{kw'}$. We can see that in this way, the generation of PEKS ciphertexts and trapdoors turns to be in an on-line manner (through protocol) and hence the security against the off-line KGA can be obtained. Moreover, the KS can function as a single point of control for implementing rate-limiting measures to reduce the on-line KGA rate. More details are deferred to Section 6.1 where we describe a client-KS protocol with online KGA rate-limiting control.

6.2.2 Formal Definition

An SA-PEKS scheme is defined by the six-tuple ($\mathsf{SA\text{-}KeyGen}_{\mathcal{KS}}$, $\mathsf{SA\text{-}KeyGen}_{\mathcal{R}}$, SA-KSD, SA-PEKS, SA-Trapdoor, SA-Test) as follows.

$\mathsf{SA\text{-}KeyGen}_{\mathcal{KS}}(\ell)$. Taking as input the security parameter ℓ, it outputs the public/private key pair of the KS as (pk_{ks}, sk_{ks}).

$\mathsf{SA\text{-}KeyGen}_{\mathcal{R}}(\ell)$. Taking as input parameter ℓ, it outputs the public/private key pair of the receiver as (pk_R, sk_R).

$\mathsf{SA\text{-}KSD}(pk_{ks}, sk_{ks}, kw)$. Taking as input the key pair of the KS and the keyword kw, it returns the KS-derived keyword ksd_{kw}.

$\mathsf{SA\text{-}PEKS}(pk_R, ksd_{kw})$. Taking as input the public key pk_R of the receiver and the KS-derived keyword ksd_{kw}, it outputs the PEKS ciphertext of kw as $CT_{ksd_{kw}}$.

$\mathsf{SA\text{-}Trapdoor}(sk_R, ksd_{kw'})$. Taking as input the secret key sk_R of the receiver and the KS-derived keyword $ksd_{kw'}$, it outputs the the trapdoor as $T_{ksd_{kw'}}$.

$\mathsf{SA\text{-}Test}(pk_R, CT_{ksd_{kw}}, T_{ksd_{kw'}})$. Taking as input the public key pk_R, the PEKS ciphertext $CT_{ksd_{kw}}$ and the trapdoor $T_{ksd_{kw'}}$, it outputs $True$ if $kw = kw'$; otherwise outputs $False$.

Correctness. It is required that for any two keywords kw, kw', we have that $\mathsf{SA\text{-}Test}(pk_R, CT_{ksd_{kw}}, T_{ksd_{kw'}}) = True$ if $kw = kw'$, where $ksd_{kw} \leftarrow \mathsf{SA\text{-}KSD}(pk_{ks}, sk_{ks}, kw)$, $ksd_{kw'} \leftarrow \mathsf{SA\text{-}KSD}(pk_{ks}, sk_{ks}, kw')$, $CT_{ksd_{kw}} \leftarrow \mathsf{SA\text{-}PEKS}(pk_R, ksd_{kw})$ and $T_{ksd_{kw'}} \leftarrow \mathsf{SA\text{-}Trapdoor}(sk_R, ksd_{kw'})$.

Remark 1. The algorithm SA-KSD is an interactive protocol between the user (sender or receiver) and the KS. Both the KS and the user take as input the public information pk_{ks}. The private input of the KS is sk_{ks} while that for the user is the original keyword. The KS and the user engage in the KS-derived keyword issuing protocol and stop in polynomial time. When the protocol completes, the private output of the user contains the KS-derived keyword.

6.2.3 Security Models

In this subsection, we define the security models for the SA-PEKS in terms of the adversarial SS, the honest but curious KS and adversarial users respectively. One should note that here we suppose the KS to be honest but curious which means that the KS would always execute the protocol honestly and return the valid KS-derived keywords for the user. However, the KS may be curious about the original keyword of the user during the execution of the interactive protocol. Moreover, the KS is not allowed to collude with the SS in the SA-PEKS, otherwise the security against the off-line KGA cannot be obtained.

Adversarial Storage Server (SS). An adversarial SS may try to learn the underlying keyword of a PEKS ciphertext. Here we propose a new notion, namely *Semantic-Security against Chosen Keyword Guessing Attack* (SS-CKGA) for the SA-PEKS. Similar to the notion of SS-CKA in PEKS, SS-CKGA guarantees that the PEKS ciphertext in the SA-PEKS does not reveal any information about the underlying keyword. The difference between the SS-CKGA and SS-CKA is that the adversary against the SA-PEKS is allowed to obtain the matching trapdoor of the challenge PEKS ciphertext. That is, the keyword privacy can still be preserved even if the adversary is given the corresponding trapdoor. Note that this is not allowed in the SS-CKA model of the PEKS as the adversary can launch an off-line attack on the matching trapdoor and hence be able to distinguish a keyword from another given the PEKS ciphertext and the trapdoor. Formally, we define the game of SS-CKGA as follows.

Setup. The challenger generates key pairs (pk_R, sk_R), (pk_{ks}, sk_{ks}) and sends (pk_R, pk_{ks}) to the attacker.

Query-I. The attacker can adaptively query the challenger for the trapdoor and PEKS ciphertext of any keyword.

Challenge. The attacker sends the challenger two keywords kw_0, kw_1. The restriction here is that none of w_0 nor w_1 has been queried in the Query-I. The challenger picks $b \xleftarrow{\$} \{0,1\}$ and generates

$$ksd_{kw_b} \leftarrow \mathsf{SA\text{-}KSD}(pk_{ks}, sk_{ks}, kw_b),\ CT^* \leftarrow \mathsf{SA\text{-}PEKS}(pk_R, ksd_{kw_b}),$$
$$T^* \leftarrow \mathsf{SA\text{-}Trapdoor}(sk_R, ksd_{kw_b}).$$

The challenger then sends (CT^*, T^*) to the attacker.

Query-II. The attacker can continue the query for the trapdoor and PEKS ciphertext of any keyword of its choice except the challenge keywords kw_0, kw_1.

Output. Finally, the attacker outputs its guess $b' \in \{0,1\}$ on b and wins the game if $b = b'$.

We refer to such an adversary $\mathcal{A}$ in the above game as an SS-CKGA adversary and define its advantage as

$$\mathsf{Adv}^{\mathsf{SS\text{-}CKGA}}_{\mathcal{SS},\mathcal{A}}(\ell) = \Pr[b = b'] - 1/2.$$

Honest but Curious Keyword Server (KS). Noting that the KS may be curious about the original keyword from the user (sender or receiver) and there may exist adversaries that can be outside attackers who are able to either gain access to the KS or eavesdrop on the communication channel between the KS and the user, we require that the user should take the original keyword as a private input during the execution of the interactive protocol to obtain the KS-derived keyword. Back to the Fig.1, it is required that at the end of the protocol execution, the sender learns exactly the KS-derived keyword ksd_{kw} while the KS should learn nothing at all about kw. That is, the protocol cannot reveal any information about the private input of the user to the KS or other outside attackers. Formally, we define the game of *Indistinguishability against Chosen Keyword Attack* (IND-CKA) as follows.

Setup. The challenger runs the algorithm $\mathsf{KeyGen}(\ell)$ and sends the attacker the public/private key pair (pk_{ks}, sk_{ks}). The attacker then sends the challenger two keywords w_0, w_1.

Challenge. The challenger picks $b \xleftarrow{\$} \{0,1\}$, runs the KS-derived keyword issuing protocol with the attacker by taking as input the keyword kw_b.

Output. After the protocol execution completes, the attacker outputs its guess $b' \in \{0,1\}$ on b and wins the game if $b = b'$.

We refer to such an adversary $\mathcal{A}$ in the above game as an IND-CKA adversary and define its advantage as

$$\mathsf{Adv}^{\mathsf{IND\text{-}CKA}}_{\mathcal{KS},\mathcal{A}}(\ell) = \Pr[b = b'] - 1/2.$$

Adversarial Users. It is a requirement that only the KS can generate the correct KS-derived keywords, otherwise the security of SA-PEKS falls to that of original PEKS, i.e., being insecure against off-line KGA. Also, we should prevent an adversarial user from generating the KS-derived keyword based on the previous KS-derived keywords obtained from the KS. Therefore, to best capture such a security requirement, we define the game of *One-More-Unforgeability under Chosen Keyword Attack* (OMU-CKA) as follows.

Setup. The challenger runs algorithm $\mathsf{KeyGen}(\ell)$ to obtain key pair (pk_{ks}, sk_{ks}). The attacker is given pk_{ks}.

KSD-Query. The attacker can adaptively query the challenger for the KS-derived keyword for at most q_k distinct original keywords of his choice $kw_1, kw_2, ..., kw_{q_k}$ through the protocol.

Output. Finally, the attacker outputs $q_k + 1$ pairs $\{w_i, ksd_{kw_i}\}_{i\in[1,q_k+1]}$ and wins the game if (1) $kw_i \neq kw_j$, for any $i, j \in [1, q_k + 1]$ where $i \neq j$, and (2) ksd_{kw_i} is a valid KS-derived keyword of kw_i for any $i \in [1, q_k + 1]$.

We refer to such an adversary $\mathcal{A}$ in the above game as an OMU-CKA adversary and define its advantage $\mathsf{Adv}^{\mathsf{OMU\text{-}CKA}}_{\mathcal{U},\mathcal{A}}(\ell)$ to be the probability that $\mathcal{A}$ wins in the above game.

Based on the security models defined above, we give the following security definition for an SA-PEKS scheme.

Definition 6.1 (Secure SA-PEKS) *We say that an SA-PEKS is secure if for any polynomial time attacker $\mathcal{A}_i$ ($i = 1, 2, 3$), we have that $\mathsf{Adv}^{\mathsf{SS\text{-}CKGA}}_{\mathcal{SS},\mathcal{A}_1}(\ell)$, $\mathsf{Adv}^{\mathsf{IND\text{-}CKA}}_{\mathcal{KS},\mathcal{A}_2}(\ell)$ and $\mathsf{Adv}^{\mathsf{OMU\text{-}CKA}}_{\mathcal{U},\mathcal{A}_3}(\ell)$ are all negligible functions of the security parameter ℓ.*

6.3 PEKS-to-SA-PEKS Transformation

In this section, we propose a universal transformation from PEKS to SA-PEKS.

6.3.1 A Universal Transformation

In this subsection, we show a universal transformation from PEKS to SA-PEKS. Given a deterministic blind signature scheme $\mathcal{BS} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{User}, \mathsf{Vf})$ and a PEKS

scheme $\mathcal{PEKS} = (\mathsf{KeyGen}, \mathsf{PEKS}, \mathsf{Trapdoor}, \mathsf{Test})$, the resulting SA-PEKS scheme is as follows. Let $\widehat{H} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be a cryptographic collision-resistant hash function.

$\mathsf{SA\text{-}KeyGen}_{\mathcal{KS}}(\ell)$. Take as input the security parameter ℓ and output the public/private key pair of the KS by running the algorithm Kg of $\mathcal{BS}$ as $(pk_{ks}, sk_{ks}) \xleftarrow{\$} \mathsf{Kg}(\ell)$.

$\mathsf{SA\text{-}KeyGen}_{\mathcal{R}}(\ell)$. Take as input the parameter ℓ and output the key pair of the receiver by running the algorithm KeyGen of $\mathcal{PEKS}$ as $(pk_R, sk_R) \xleftarrow{\$} \mathsf{KeyGen}(\ell)$.

$\mathsf{SA\text{-}KSD}(pk_{ks}, sk_{ks}, kw)$. Take as input the key pair of the KS and the keyword kw, run the algorithm $\mathsf{User}(pk_{ks}, kw)$ and the algorithm $\mathsf{Sign}(sk_{ks})$ of $\mathcal{BS}$ in an interactive signing protocol to obtain the valid signature σ_{kw} of kw. Compute and output the KS-derived keyword as,

$$ksd_{kw} = \widehat{H}(kw, \sigma_{kw}).$$

$\mathsf{SA\text{-}PEKS}(pk_R, ksd_{kw})$. Take as input the public key pk_R of the receiver and the KS-derived keyword ksd_{kw}, run the algorithm PEKS of $\mathcal{PEKS}$ as,

$$CT_{ksd_{kw}} \leftarrow \mathsf{PEKS}(pk_R, ksd_{kw}).$$

Output $CT_{ksd_{kw}}$ as the PEKS ciphertext of kw.

$\mathsf{SA\text{-}Trapdoor}(sk_R, ksd_{kw'})$. Take as input the secret key sk_R of the receiver and the KS-derived keyword $ksd_{kw'}$, run the algorithm $\mathsf{Trapdoor}$ of $\mathcal{PEKS}$ as,

$$T_{ksd_{kw'}} \leftarrow \mathsf{Trapdoor}(sk_R, ksd_{kw'}).$$

Output $T_{ksd_{kw'}}$ as the trapdoor of kw'.

$\mathsf{SA\text{-}Test}(pk_R, CT_{ksd_{kw}}, T_{ksd_{kw'}})$. Take as input the public key pk_R, the PEKS ciphertext $CT_{ksd_{kw}}$ and the trapdoor $T_{ksd_{kw'}}$, run the algorithm Test of $\mathcal{PEKS}$ as,

$$TF \leftarrow \mathsf{PEKS}(pk_R, CT_{ksd_{kw}}, T_{ksd_{kw'}}).$$

Output TF as the testing result.

Correctness Analysis. One can see that the correctness condition of this construction holds due to the collision-resistant hash function $\widehat{H}$, the deterministic scheme $\mathcal{BS}$ and correctness of $\mathcal{PEKS}$. To be more precise, for any keywords kw, kw', we have that $\sigma_{kw} = \sigma'_{kw}$ and hence $ksd_w = ksd_{kw'}$ if $kw = kw'$. Therefore, due to

the correctness of $\mathcal{PEKS}$, we have that $True \leftarrow \mathsf{PEKS}(pk_R, CT_{ksd_{kw}}, T_{ksd_{kw'}})$. If $kw \neq kw'$, we can see that $ksd_{kw} \neq ksd_{kw'}$ and hence the testing result would be $False$.

Remark 2. It is worth noting that unlike the work in [CMY+15] where the additional server participates in testing stage, the semi-trusted server introduced in SA-PEKS is only in charge of keyword pre-processing and all the algorithms in the underlying PEKS remain unchanged. Therefore, it is conceivable that our solution can be more practical compared with the previous solutions. One can note that the above universal transformation is also applicable for other non-interactive PEKS systems such as SCF-PEKS [BSS08].

6.3.2 Security Analysis

In this subsection, we analyze the security of $\mathcal{SA}$-$\mathcal{PEKS}$ based on the security models defined above.

SS-CKGA Security. Formally, the SS-CKGA security of the above SA-PEKS scheme $\mathcal{SA}$-$\mathcal{PEKS}$ is guaranteed by the following theorem.

Theorem 6.1 *Let hash function $\widehat{H}$ be a random oracle. Suppose that there exists a polynomial-time adversary $\mathcal{A}$ that can break the SS-CKGA security of the above scheme $\mathcal{SA}$-$\mathcal{PEKS}$ with advantage $\mathsf{Adv}^{\mathsf{SS\text{-}CKGA}}_{\mathcal{SS},\mathcal{A}}$, then there exists a polynomial-time adversary $\mathcal{B}$ that can break the one-more unforgeability of the underlying signature scheme $\mathcal{BS}$ with advantage at least*

$$\mathsf{Adv}^{\mathsf{OMU}}_{\mathcal{BS},\mathcal{B}} \geq 1/q_{\widehat{H}} \cdot \mathsf{Adv}^{\mathsf{SS\text{-}CKGA}}_{\mathcal{SS},\mathcal{A}}$$

where $q_{\widehat{H}}$ is the number of queries to $\widehat{H}$.

Proof: We prove the theorem by constructing an algorithm $\mathcal{B}$ who simulates the challenger in the SS-CKGA model to play the game with $\mathcal{A}$. The goal of $\mathcal{B}$ is to break the *one-more-unforgeability* security of the scheme $\mathcal{BS}$. Suppose that $\mathcal{B}$ is given the public key pk_{ks} of $\mathcal{BS}$. Then $\mathcal{B}$ interacts with $\mathcal{A}$ as follows.

In the Setup stage, $\mathcal{B}$ runs the algorithm KeyGen to generate the key pair (pk_R, sk_R) and gives pk_R, pk_{ks} to the adversary $\mathcal{A}$.

In the $\mathsf{Query\text{-}I}$ stage, when $\mathcal{A}$ queries a keyword kw for the PEKS ciphertext (or the trapdoor), $\mathcal{B}$ first interacts with the signing oracle to obtain the signature of kw and accesses to the random oracle $\widehat{H}$ for the KS-derived keyword ksd_{kw}. $\mathcal{B}$ then generates the PEKS ciphertext $CT_{ksd_{kw}}$ (or the trapdoor $T_{ksd_{kw}}$) using (pk_R, sk_R) and returns the result to $\mathcal{A}$.

In the $\mathsf{Challenge}$ stage, upon receiving two challenge keywords kw_0, kw_1 from $\mathcal{A}$, instead of querying kw_b to the signing oracle for the signature σ_{kw_b}, $\mathcal{B}$ just picks

randomly r and generates the challenge PEKS ciphertext and trapdoor (CT^*, T^*) of kw_b by regarding r as the signature of kw_b. $\mathcal{A}$ then sends (CT^*, T^*) to $\mathcal{A}$.

In the Query-II stage, $\mathcal{B}$ simulates as that in Query-I stage.

In the Ouput stage, after $\mathcal{A}$ outputs its guess b' on b, $\mathcal{B}$ picks an input-element from the random oracle $\widehat{H}$ randomly and outputs it as its forgery signature.

Let Q be the event that $\mathcal{A}$ queried (kw_b, σ_{kw_b}) to the random oracle $\widehat{H}$. Then the advantage of $\mathcal{A}$ wins in the above game is

$$\mathsf{Adv}^{\mathsf{SS\text{-}CKGA}}_{\mathcal{SS},\mathcal{A}} = \Pr[b' = b|Q] + \Pr[b' = b|\overline{Q}] - 1/2.$$

One can note that in the above simulation, if event Q does not happen, that is, $\mathcal{A}$ did not ever query $\widehat{H}$ with (kw_b, σ_{kw_b}), then the above game is identical to the original SS-CKGA game from the view of $\mathcal{A}$ due to the property of random oracle $\widehat{H}$. However, the probability of $\mathcal{A}$ wins in this game under this case is at most $1/2$ since (CT^*, T^*) is independent of b, i.e. $\Pr[b' = b|\overline{Q}] = 1/2$. Therefore, we have that $\Pr[b' = b|Q] = \mathsf{Adv}^{\mathsf{SS\text{-}CKGA}}_{\mathcal{SS},\mathcal{A}}$, which means that the event Q happens with probability $\mathsf{Adv}^{\mathsf{SS\text{-}CKGA}}_{\mathcal{SS},\mathcal{A}}$. Therefore, $\mathcal{B}$ can successfully forgery a signature as (kw_b, σ_{kw_b}) with advantage $\mathsf{Adv}^{\mathsf{OMU}}_{\mathcal{BS},\mathcal{B}} \geq 1/q_{\widehat{H}} \cdot \mathsf{Adv}^{\mathsf{SS\text{-}CKGA}}_{\mathcal{SS},\mathcal{A}}$, if the number of queries to $\widehat{H}$ is $q_{\widehat{H}}$.

IND-CKA Security. As for the IND-CKA security which guarantees that except the user no other entity can learn any information about the private input (keyword) of the user, we have the following theorem.

Theorem 6.2 *If there exits a polynomial-time adversary $\mathcal{A}$ that can break the IND-CKA security of the above scheme $\mathcal{SA}$-$\mathcal{PEKS}$ with advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CKA}}_{\mathcal{KS},\mathcal{A}}$, then there exists a polynomial-time adversary $\mathcal{B}$ that can break the blindness security of the underlying signature scheme $\mathcal{BS}$ with advantage at least*

$$\mathsf{Adv}^{\mathsf{Blindness}}_{\mathcal{BS},\mathcal{B}} \geq \mathsf{Adv}^{\mathsf{IND\text{-}CKA}}_{\mathcal{KS},\mathcal{A}}.$$

Proof: We prove the theorem above by constructing an algorithm $\mathcal{B}$ who runs the adversary $\mathcal{A}$ as a subroutine to break the security of *blindness* of the underlying scheme $\mathcal{BS}$ as follows. Suppose the challenger in the *blindness* security attack game against the scheme $\mathcal{BS}$ is $\mathcal{C}$.

In the Setup stage, $\mathcal{B}$ receives the public/private key pair (pk, sk) from $\mathcal{C}$, sends the key pair to $\mathcal{A}$ and then receives the challenge keywords (kw_0, kw_1) from $\mathcal{A}$. $\mathcal{B}$ then forwards (kw_0, kw_1) as the challenge message to $\mathcal{C}$.

In the Challenge stage, $\mathcal{B}$ simulates as the adversarial signer from the view of $\mathcal{C}$ and as the challenger of the IND-CKA game against the scheme $\mathcal{SA}$-$\mathcal{PEKS}$ from the view of $\mathcal{A}$. Once $\mathcal{C}$ starts the first execution of the signing protocol, $\mathcal{B}$ starts the KS-derived keyword issuing protocol with $\mathcal{A}$ and forwards the message between

$\mathcal{A}$ and $\mathcal{C}$. During the second execution of the protocol, $\mathcal{B}$ interacts with $\mathcal{C}$ honestly using the public/private key pair (pk, sk).

In the Ouput stage, after $\mathcal{A}$ outputs its guess b', $\mathcal{B}$ output b' as its guess to $\mathcal{C}$.

It is easy to see that the above simulation is indistinguishability from the IND-CKA game from the view of $\mathcal{A}$. Therefore, we have that $\mathsf{Adv}^{\mathsf{Blindness}}_{\mathcal{BS},\mathcal{B}} \geq \mathsf{Adv}^{\mathsf{IND\text{-}CKA}}_{\mathcal{KS},\mathcal{A}}$, which completes the proof.

OMU-CKA Security. Here we discuss the OMU-CKA security of our universal transformation. This notion guarantees that a user cannot forge a new KS-derived keyword without the help of the KS even if it has seen many KS-derived keywords before. Formally, we have the following theorem.

Theorem 6.3 *If there exits a polynomial-time adversary $\mathcal{A}$ that can break the OMU-CKA security of the above scheme $\mathcal{SA}$-$\mathcal{PEKS}$ with advantage $\mathsf{Adv}^{\mathsf{OMU\text{-}CKA}}_{\mathcal{U},\mathcal{A}}$, then there exists a polynomial-time adversary $\mathcal{B}$ that can break the one-more-unforgeability security of the underlying signature scheme $\mathcal{BS}$ with advantage at least*

$$\mathsf{Adv}^{\mathsf{OMU}}_{\mathcal{BS},\mathcal{B}} \geq \mathsf{Adv}^{\mathsf{OMU\text{-}CKA}}_{\mathcal{U},\mathcal{A}}.$$

Proof: We give the proof of the theorem above by constructing an algorithm $\mathcal{B}$ who invokes the adversary $\mathcal{A}$ to break the *one-more-unforgeability* security of the scheme $\mathcal{BS}$ as follows.

In the Setup stage, $\mathcal{B}$ receives the public key pk_{ks} from the signing oracle and sends pk_{ks} to $\mathcal{A}$.

In the $\mathsf{KSD\text{-}Query}$ stage, upon receiving a queried keyword kw, $\mathcal{B}$ queries kw to the signing oracle to obtain the signature σ_{kw}, returns the hash value of the returned signature as $\widehat{H}(kw, \sigma_{kw})$ to $\mathcal{A}$ as the KS-derived keyword of kw.

In the Ouput stage, if $\mathcal{A}$ outputs $q_k + 1$ valid pairs $\{(kw_i, ksd_{kw_i})\}_{1 \leq i \leq q_k+1}$ where q_k is the KS-derived keyword query number. Then for each i, $\mathcal{B}$ looks up kw_i in the hash query record for the corresponding signature σ_i and outputs $\{(kw_i, \sigma_i)\}_{1 \leq i \leq q_k+1}$ as the $q_k + 1$ valid message/signature pairs as its forgery signatures. Otherwise, $\mathcal{B}$ aborts.

One can see that the simulation above by $\mathcal{B}$ is indistinguishable from the original OMU-CKA game from the view of the adversary $\mathcal{A}$, therefore, $\mathcal{A}$ can successfully output $q_k + 1$ valid keyword/KS-derived keywords pairs with the advantage $\mathsf{Adv}^{\mathsf{OMU\text{-}CKA}}_{\mathcal{U},\mathcal{A}}(\ell)$. That is, $\mathcal{B}$ can break the *one-more-unforgeability* security of $\mathcal{BS}$ with advantage at least $\mathsf{Adv}^{\mathsf{OMU}}_{\mathcal{BS},\mathcal{B}} \geq \mathsf{Adv}^{\mathsf{OMU\text{-}CKA}}_{\mathcal{U},\mathcal{A}}$.

Based on the theorems above, we have the following observation.

Theorem 6.4 *The universal transformation above results in a secure SA-PEKS scheme if the underlying blind signature is secure in terms of one-more-unforgeability and blindness.*

Further Discussions On the Multi-tiered Security. Ideally, we hope that the adversary does not have authorized access to the KS. This means that the adversary can launch neither off-line nor on-line KGA. However, in reality, the adversary (including the adversarial SS) may have remote access to the KS. In this case, the resulting SA-PEKS still remains SS-CKA secure as long as the underlying PEKS is SS-CKA secure. However, we should note that even in this case, the adversarial SS cannot efficiently launch the KGA since it needs to query the signature of each guessing keyword in an on-line manner though the protocol and hence the brute-force attack will be rendered less effective. As will shown in Section 6.1, with an effective rate-limiting mechanism, the on-line KGA can be slowed down significantly.

Note that in the previous formal security models, we assume that the KS does not collude with the SS since otherwise the construction cannot achieve the desirable security guarantees since the SS can launch the off-line KGA once it obtains the private key of the KS. Nevertheless, we can see that even if the private key of the KS is leaked to the SS through any possible means, the security of the SA-PEKS is still at the same level as that of the underlying PEKS. It is because that the PEKS ciphertext (trapdoor) generation procedure in the SA-PEKS is the same as that of the underlying PEKS scheme. Therefore, we have,

Theorem 6.5 *The universal transformation above results in an SS-CKA secure SA-PEKS scheme if the underlying PEKS is SS-CKA secure.*

6.4 An Instantiation of SA-PEKS

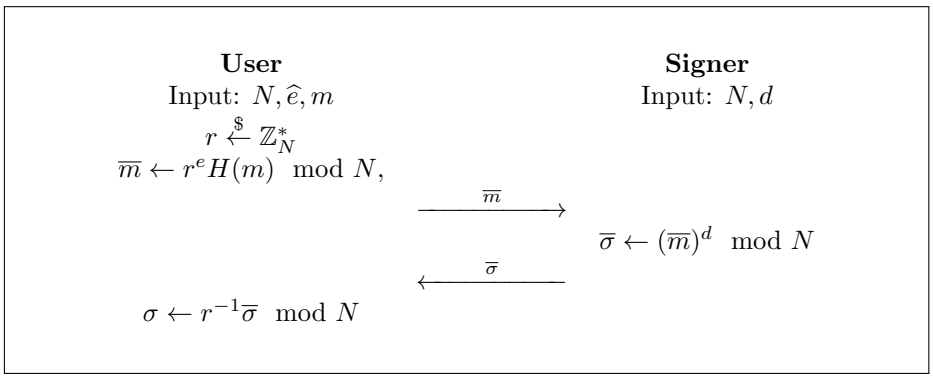
In this section, we show how to implement the proposed SA-PEKS scheme by presenting an instantiation.

6.4.1 Underlying Schemes

Following the above universal transformation, here we show an instantiation of the proposed SA-PEKS scheme based on the FDH-RSA blind signature [BNPS03] and the PEKS scheme (denoted by BCOP-PEKS) proposed by Boneh *et al.* in [BCOP04]. We start with the introduction of the two building blocks.

FDH-RSA. The RSA blind signature is described in Fig.6.1. The signer has public key $N, \widehat{e}$ and private key N, d where $\widehat{e}d \equiv 1 \mod \phi(N)$, modulus N is the product of two distinct primes of roughly equal length. The user uses a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_N^*$ to hash the message m to an element of $\mathbb{Z}_N$ and then blinds result with a random group element $r \stackrel{\$}{\leftarrow} \mathbb{Z}_N^*$. The resulting blinded hash, denoted $\overline{m}$ is then sent to the signer. The signer signs $\overline{m}$ with its private key d by computing $\overline{\sigma} \leftarrow (\overline{m})^d \mod N$ and sends back $\overline{\sigma}$. The user then derive the signature by removing

Table 6.1: Blind signing protocol for FDH-RSA



the blinding element through $\sigma \leftarrow r^{-1}\overline{\sigma} \mod N$. The correctness can be obtained due to the fact that $\overline{\sigma} = (\overline{m})^d \mod N = (r^{\widehat{e}}H(m))^d \mod N = rH(m)^d \mod N$ and hence $\sigma = r^{-1}\overline{\sigma} \mod N$.

We can see that the *blindness* security of the above FDH-RSA is guaranteed by the one-time element chosen randomly to blind the signed message. As for the unforgeability security, based on the result from [BNPS03], we have the conclusion that the FDH-RSA blind signature scheme is polynomially-secure against one-more forgery if the *RSA known-target inversion* problem is hard. More details can be found in [BNPS03].

BCOP-PEKS. Here, we show the PEKS scheme proposed in [BCOP04]. This scheme is based on a variant of the *Computation Diffie-Hellman Problem*. Let $\mathbb{G}_1, \mathbb{G}_\tau$ be two groups with prime order p and the bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_\tau$. Then the non-interactive searchable encryption scheme works as follows.

KeyGen. The input security parameter determines the size p of the groups $\mathbb{G}_1, \mathbb{G}_\tau$. The algorithm then picks a random $\alpha \xleftarrow{\$} \mathbb{Z}_p^*$, a generate g of $\mathbb{G}_1$ and choose two hash functions $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \mathbb{G}_\tau \leftarrow \{0,1\}^{\log p}$. Then it outputs $pk_R = (g, h = g^\alpha, H_1, H_2), sk_R = \alpha$.

PEKS. For a keyword kw, pick a random $r \xleftarrow{\$} \mathbb{Z}_p^*$ and compute $t = e(H_1(kw), h^r)$. Output the PEKS ciphertext as $CT_{kw} = (g^r, H_2(t))$.

Trapdoor. For a keyword kw', output the trapdoor as $T_{kw'} = H_1(kw')^\alpha \in \mathbb{G}_1$.

Test. Take as input $CT_{kw} = (A, B)$ and $T_{kw'}$, if $H_2(e(T_{kw'}, A)) = B$, output $True$, $False$ otherwise.

The correctness of the PEKS scheme above can be easily obtained. In terms of security, the scheme is secure against the SS-CKA [BCOP04] but insecure against

the off-line KGA. Actually, the off-line KGA against the BCOP-PEKS scheme can be launched in a much simpler way. Given a trapdoor $T_{kw^*} = H_1(kw^*)^\alpha$, the attacker can easily test whether its guessing keyword kw is the underlying keyword of T_{kw^*} by checking $e(T_{kw^*}, g) \stackrel{?}{=} e(H_1(kw), h)$.

6.4.2 Resulting SA-PEKS

Here we show the resulting SA-PEKS derived from the FDH-RSA and the BCOP-PEKS schemes. The details are described in Table. 6.2.

Scheme Descriptions. Note that the KS-derived keyword issuing protocol in our scheme requires that $N < \widehat{e}$ should be verified by the user. This is to avoid that the KS may generate the keys dishonestly in order to learn some information about the keyword. This condition ensures that $\gcd(\phi(N), \widehat{e}) = 1$ even if N is maliciously generated and thus ensures that the map $f_{\widehat{e}} : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$, defined by $f_{\widehat{e}}(x) = x^{\widehat{e}} \mod N$ for all $x \in \mathbb{Z}_N^*$, is a permutation on $\mathbb{Z}_N^*$. Since $f_{\widehat{e}}$ is a permutation and the user can verify the validity of the signature, even a malicious KS cannot force the output of signature to be a fixed value.

To obtain a KS-derived keyword, the user takes as input the private keyword kw and accesses the KS through authentication, and then activate the KS-derived keyword issuing protocol, i.e., the interactive RSA blind signature scheme. Note that the user should verify that $\widehat{e} > N$ after being given the public key to avoid the maliciously generated key. The user then blinds the hash value $H(w)$ with a randomly chosen element from $\mathbb{Z}_N^*$ and sends the blinded message to the KS for signing. The user removes the randomness upon receiving the signature from the KS and thus obtains the signature of kw. The user then applies the hash function $\widehat{H}$ to compute the KS-derived keyword ksd_{kw}. The other parts are the same as the BCOP-PEKS scheme except that the PEKS ciphertext and the trapdoor are derived from the KS-derived keyword instead of the original keyword.

Correctness. It is easy to see that for any two keywords kw, kw', if $kw = kw'$, then $ksd_{kw} = \widehat{H}(w, \sigma_{kw}) = \widehat{H}(kw, H(kw)^d) = \widehat{H}(kw', H(kw')^d) = ksd_{kw'}$. Therefore, as for the corresponding PEKS ciphertext $CT_{ksd_{kw}} = (A, B)$ and the trapdoor $T_{ksd_{kw'}}$, we have that

$$H_2(e(T_{ksd_{kw'}}, A)) = H_2(e(H_1(ksd_{kw'}), h^r)) = B.$$

Otherwise, $H_2(e(T_{ksd_{kw'}}, A)) \neq B$ as $ksd_{kw} \neq ksd_{kw'}$.

Security Analysis. The security of the resulting SA-PEKS scheme can be easily obtained based on **Theorem** 6.4 as the FDH-RSA is *one-more-unforgeable* and of *blindness*. Formally, we have the following collusion.

Corollary 6.6 *The concrete SA-PEKS scheme presented above is secure.*

Table 6.2: An SA-PEKS scheme from FDH-RSA and BCOP-PEKS scheme

Let $\mathbb{G}_1, \mathbb{G}_\tau$ be two groups with prime order p and the bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_\tau$. The public RSA-exponent $\widehat{e}$ is fixed as part of the scheme. Let $\widehat{H} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be a cryptographic collision-resistant hash function.

SA-KeyGen$_{\mathcal{KS}}(\ell)$. The algorithm runs $\mathsf{Kg}(\ell, \widehat{e})$ to get N, d such that $\widehat{e}d \equiv 1 \mod \phi(N)$ and $N < \widehat{e}$. It chooses hash function $H : \{0,1\}^* \to \mathbb{Z}_N^*$, and then outputs $pk_{ks} = (N, \widehat{e}, H, \widehat{H}), sk_{ks} = (N, d)$.

SA-KeyGen$_{\mathcal{R}}(\ell)$. The algorithm takes as input the security parameter, determines the size p of the groups $\mathbb{G}_1, \mathbb{G}_\tau$ and picks a random $\alpha \xleftarrow{\$} \mathbb{Z}_p^*$, a generate g of $\mathbb{G}_1$, computes $h = g^\alpha$ and chooses hash functions $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \mathbb{G}_\tau \leftarrow \{0,1\}^{\log p}$. it then outputs $pk_R = (g, h, H_1, H_2), sk_R = \alpha$.

SA-KSD. For a keyword kw, the algorithm runs the KS-derived keyword issuing protocol to obtain the valid signature σ_{kw} of kw as follows.

User $(N, \widehat{e}, H, kw)$		**KS** (N, d)
If $\widehat{e} \leq N$ then abort		
$r \xleftarrow{\$} \mathbb{Z}_N^*$		
$\overline{kw} \leftarrow r^{\widehat{e}} \cdot H(kw) \mod N$		
	$\xrightarrow{\overline{kw}}$	
		$\overline{\sigma} \leftarrow (\overline{kw})^d \mod N$
	$\xleftarrow{\overline{\sigma}}$	
$\sigma_{kw} \leftarrow r^{-1}\overline{\sigma} \mod N$		
If $\sigma_{kw}^{\widehat{e}} \mod N \neq H(kw)$ then abort		
Else return σ_{kw}		

The algorithm then computes $ksd_{kw} = \widehat{H}(kw, \sigma_{kw}) = \widehat{H}(kw, H(kw)^d)$, and outputs ksd_{kw} as the KS-derived keyword of kw.

SA-PEKS. For a KS-derived keyword ksd_{kw}, the algorithm picks a random $r \xleftarrow{\$} \mathbb{Z}_p^*$, computes

$$t = e(H_1(ksd_{kw}), h^r), CT_{ksd_{kw}} = (g^r, H_2(t)).$$

The algorithm outputs $CT_{ksd_{kw}}$ as the PEKS ciphertext of ksd_{kw}.

SA-Trapdoor. For a KS-derived keyword $ksd_{kw'}$, the algorithm computes

$$T_{ksd_{kw'}} = H_1(ksd_{kw'})^\alpha.$$

The algorithm then outputs $T_{ksd_{kw'}}$ as the trapdoor of $ksd_{kw'}$.

SA-Test. The algorithm takes as input $CT_{ksd_{kw}} = (A, B)$ and $T_{ksd_{kw'}}$, and checks

$$H_2(e(T_{ksd_{kw'}}, A)) \stackrel{?}{=} B.$$

If yes, output $True$, else output $False$.

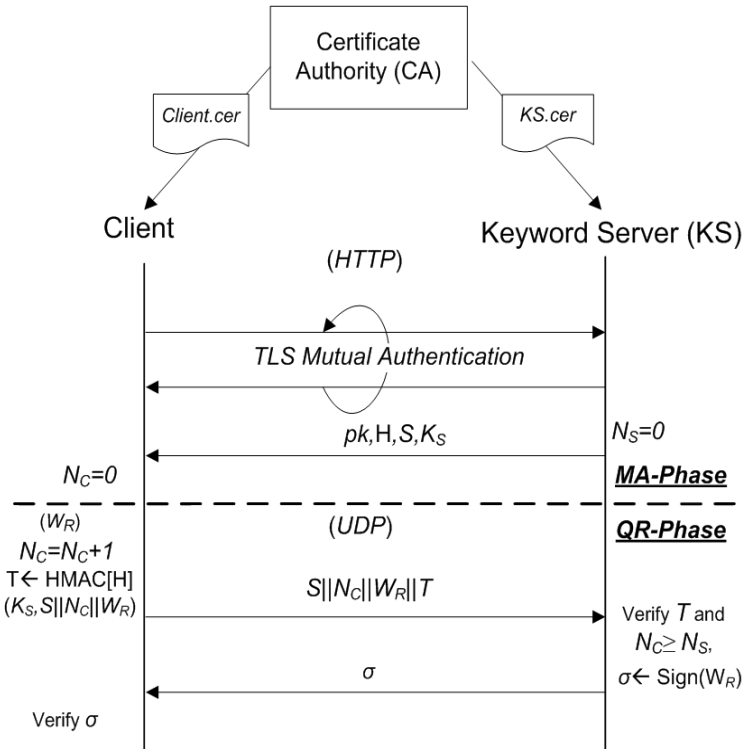


Figure 6.2: The Client-KS Protocol in SA-PEKS

6.5 Implementation and Performance

In this section, we will first describe the client-KS protocol and analyze the performance of our solution in experiments.

6.5.1 The Client-KS Protocol

Motivated by the work in [KBR13], we show a protocol for client-KS interaction and the rate-limiting strategies which limit client queries to slow down on-line keyword guessing attack. Our design goal is to give a low-latency protocol to avoid performance degrading.

Protocol Implementation. The proposed protocol relies on a CA providing the KS and each client with a unique verifiable TLS certificates. As shown in Fig. 6.2, the execution of protocol consists of the Mutual Authentication (MA) phase and the Query-Response (QR) phase, of which the first one is over HTTP while the later one is over UDP.

Phase I: Mutual Authentication. The MA procedure starts with a TLS handshake with mutual authentication, initiated by a client. The KS responds immediately with the verification key pk of the underlying blind signature scheme, a hash function H (by default SHA-256), a random session identifier $S \in \{0,1\}^{128}$, and a random session key $K_S \in \{0,1\}^k$. The KS then initializes

and records a sequence number with this session as $N_S = 0$. The client stores pk, S, K_S and also initializes a sequence number $N_C = 0$. In our protocol, similar to [KBR13], we also set each session to last for a fixed time period.

Phase II: Query-Response. In the QR phase, the client first generates a blinded value of a keyword as W_R, increases the recorded sequence number $N_C \leftarrow N_C + 1$, and then computes a MAC tag using the KS's session key K_S, as $T \leftarrow \mathsf{HMAC}[\mathsf{H}](K_S, S||N_C||W_R)$. The client then sends $S||N_C||W_R||T$ to the KS in a UDP packet. Upon receiving the query information, the KS first checks that $N_C \geq N_S$ and verifies the correctness of the MAC T. If the verification fails, then the KS drops the packet without any further response. Otherwise, it would sign the blinded keyword and returns the signature σ to the client.

Per-Client Rate-Limiting Mechanism Against On-line KGA. Although the off-line KGA cannot be launched against our scheme anymore, an attacker (possible the malicious server) who is able to access to the KS can perform the on-line KGA to break the keyword privacy of the trapdoor. As a countermeasure, we explore the so-called *exponential delay mechanism* to achieve a balancing between defence against the on-line KGA and the latency of a KS request. For the first query, the KS performs the response with an initial small delay t_I, and the delay time is doubled after each query from the client. The doubling then stops at an upper limit t_U. The KS maintains synchronized epochs and an active client list. It checks the status of active clients after each epoch. The delay would be reset to the initial value if the client makes no queries during an entire epoch. It would also drop any query from the active client who is in the list and awaiting responses.

Protocol Security. Attackers can attempt to eavesdrop and even tamper with the communications between clients and the KS. In the protocol, due to the mutual authentication TLS handshake in the session initialization, no adversary can start a session pretending to be a valid client. Moreover, without the session key K_S, no adversary can create a fresh query packet without a successful MAC forgery. Packets can neither be replayed across sessions due to the randomly picked session identifier nor be replayed within a session due to the increasing sequence number.

Experiment Results. For the client-KS protocol, we implement FDH-RSA (RSA1024) using SHA256 in the standard way. Similar to [KBR13], the PKI setup uses RSA2048 certificates and ECDHE-RSA-AES128-SHA ciphersuite is fixed for the handshake in our protocol. In our implementation, the client machine is located in a university LAN and equipped with Linux system (more precise, 2.6.35-22-generic version) with an Intel(R) Core(TM) 2 Duo CPU of 3.33 GHZ and 2.00-GB RAM.

Latency of Protocol. Table.6.3 shows the latency of different phases in the form

Table 6.3: Latency of Protocol Under Different Load

Operation	Latency (ms)
Ping (1 Round-Trip Time)	96 ±02
MA Phase	312 ± 48
QR Phase (1000 q/s)	103 ± 32
QR Phase (2000 q/s)	134 ±43
QR Phase (3000 q/s)	157 ±40
QR Phase(4000 q/s)	193 ±46
QR Phase (5000 q/s)	252 ±51
QR Phase (6000 q/s)	327 ±49
QR Phase (7000 q/s)	376 ±44

of median time plus/minus one standard deviation over 500 trials. We can note that when the KS load is low (e.g., 1000 queries/second (q/s)), the latency is quite small and actually almost the smallest possible time (1 Round-Trip Time (RTT) of Ping operation). The time increases with the growth of the query rate. Specifically, the latency is around 157±40 milliseconds when the query rate is 3000 q/s and becomes 376±44 milliseconds when the query rate increases to 7000 q/s. It is worth noting that we only take the successful operations into account in our experiment. That is, all the replies that timed out three times were excluded from the median calculation.

Packet Drop Rates. We also evaluate the packet drop rates of our protocol for different query rate through a similar experiment as in [KBR13]. The client sends $100i$ ($1 \leq i \leq 64$) UDP request packets per second (q/s) until a total of 10000 packets are sent. We then record the number of replied over 500 trials using min/max/mean/standard deviation, as shown in Fig. 6.3. One can note that the packet loss is negligible at rates up to around 2500 q/s. However, when the query rate is 7000 q/s, the packet loss rate can be as high as 60%.

Performance Against On-line KGA. To evaluate how our rate-limiting mechanism works in real settings, we estimate the effect of on-line KGA against our protocol in experiments. In our protocol, the proposed rate-limiting mechanism, i.e., exponential delay mechanism, gives a balancing between on-line KGA speed and KS request latency, as the delay increases exponentially with the growth of queries from a client.

For the exponential delay mechanism in our experiment, we set the initial small delay t_I to 10 milliseconds and the epoch duration t_E to one week. We then evaluate the performance of protocol, i.e., the maximum query rates (in queries per second) by setting the upper limit t_U to different values. To evaluate the effectiveness of the introduced mechanism, we also run the protocol without rate-limiting. The maximum query rates that an attacker who compromised a client can achieve are given

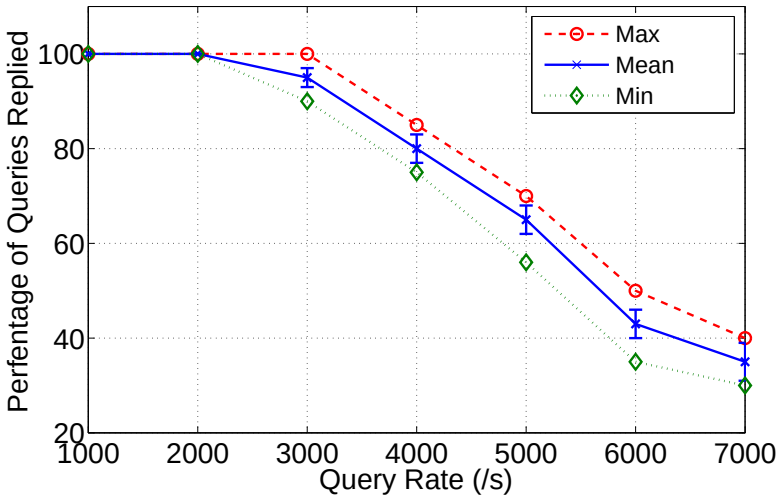


Figure 6.3: Packet Loss of the Client-KS Protocol

Table 6.4: KGA Rate for Different Rate Limiting Approaches

Setting	**Attack Rate** (queries/second)
No Rate Limiting	2700
$t_U = 400$ ms	8.23
$t_U = 600$ ms	4.21
$t_U = 800$ ms	2.54
$t_U = 1000$ ms	1.21

in Table.6.4. One can note from the result that our exponential delay mechanism can significantly slow down the on-line KGA. Specifically, the attack rate is around 2700 q/s if we put no rate-limiting on the KS. By forcing the exponential delay mechanism, the attack rate can be significantly reduced to less than 10 q/s. The attack rate decreases with the growth of the upper limit t_U and is only 1.21 q/s if we set t_U to 1000 ms.

6.5.2 The Instantiated Scheme

In this section, we evaluate the performance of our concrete SA-PEKS scheme. We first compare the computation cost between the PEKS schemes in [BCOP04], [XJWW13] with our instantiated scheme described in Section 6.4.

Comparison of Schemes. As shown in Table 6.5, compared to the BCOP scheme [BCOP04] (the underlying PEKS scheme of our SA-PEKS construction), our scheme requires 4 additional RSA exponentiations during the generation of PEKS ciphertext and trapdoor. In the testing phase, our scheme has the same computation cost as the BCOP scheme does. While the scheme [XJWW13] can also achieve a certain level of security against off-line KGA, its computation cost is much higher due to the additional pairing computation. Specifically, in our scheme, the computation cost of

Table 6.5: Comparisons between Existing Works and Our Scheme

Schemes	**Computation**		
	PEKS Generation	*Trapdoor Generation*	*Testing*
[BCOP04]	$2\text{Exp}_{\mathbb{G}_1}+2\text{Hash}_{\mathbb{G}_1}$ $+ 1\text{Pairing}_{\mathbb{G}_1,\mathbb{G}_\tau}$	$1\text{Hash}_{\mathbb{G}_1}+1\text{Exp}_{\mathbb{G}_1}$	$1\text{Hash}_{\mathbb{G}_1}+ 1\text{Pairing}_{\mathbb{G}_1,\mathbb{G}_\tau}$
[XJWW13]	$4\text{Exp}_{\mathbb{G}_1}+4\text{Hash}_{\mathbb{G}_1}$ $+2\ \text{Pairing}_{\mathbb{G}_1,\mathbb{G}_\tau}$	$2\ \text{Hash}_{\mathbb{G}_1}+2\text{Exp}_{\mathbb{G}_1}$	$2\text{Hash}_{\mathbb{G}_1}+2\text{Pairing}_{\mathbb{G}_1,\mathbb{G}_\tau}$
Our Scheme	$2\text{Exp}_{\mathbb{G}_1}+2\text{Exp}_{\mathbb{Z}_N^*}$ $+2\text{Hash}_{\mathbb{G}_1}+ 1\text{Pairing}_{\mathbb{G}_1,\mathbb{G}_\tau}$	$1\text{Hash}_{\mathbb{G}_1}+1\text{Exp}_{\mathbb{G}_1}$ $+2\text{Exp}_{\mathbb{Z}_N^*}$	$1\text{Hash}_{\mathbb{G}_1}+ 1\text{Pairing}_{\mathbb{G}_1,\mathbb{G}_\tau}$

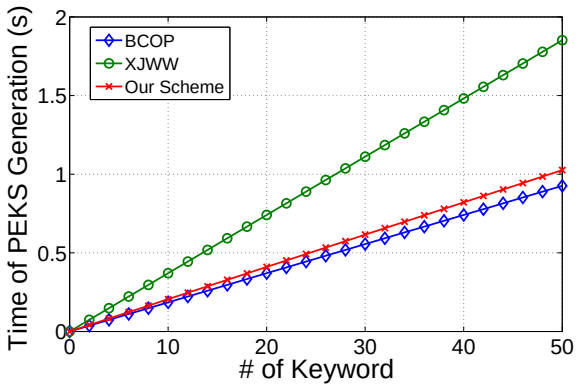


Figure 6.4: Computation Cost of PEKS Generation (excluding latency)

PEKS generation, trapdoor generation and testing are $2\text{Exp}_{\mathbb{G}_1}+4\text{Exp}_{\mathbb{Z}_N^*}+2\text{Hash}_{\mathbb{G}_1}+ 1\text{Pairing}_{\mathbb{G}_1,\mathbb{G}_\tau}$, $1\text{Hash}_{\mathbb{G}_1}+1\text{Exp}_{\mathbb{G}_1}+4\text{Exp}_{\mathbb{Z}_N^*}$ and $1\text{Hash}_{\mathbb{G}_1}+ 1\text{Pairing}_{\mathbb{G}_1,\mathbb{G}_\tau}$ respectively, where $\text{Exp}_{\mathbb{G}_1}, \text{Exp}_{\mathbb{Z}_N^*}$ denote the computation of one exponentiation in $\mathbb{G}_1$ and $\mathbb{Z}_N^*$ respectively, $\text{Hash}_{\mathbb{G}_1}$ denotes the cost of one hashing operation in $\mathbb{G}_1$.

In terms of communication cost, one can note that our scheme only introduces very small communication overhead over the basic scheme [BCOP04], while the work in [XJWW13] and [CMY+15] cost huge bandwidth to transfer either the fuzzy matching data [XJWW13] or internal testing state [CMY+15].

Experiment Results. To evaluate the efficiency of our scheme in experiments, we implement the scheme utilizing the GNU Multiple Precision Arithmetic (GMP) library and Pairing Based Cryptography (PBC) library. The following experiments are based on coding language C on a Linux system (more precise, 2.6.35-22-generic version) with an Intel(R) Core(TM) 2 Duo CPU of 3.33 GHZ and 2.00-GB RAM. For the elliptic curve, we choose an MNT curve with a base field of size 159 bits, $|p|$=160 and $|q|$=80.

We mainly analyze the computation cost of PEKS generation, trapdoor generation and testing in the schemes of [BCOP04, XJWW13] and our scheme. As shown in Fig.6.4 and Fig.6.5, the computation cost of our scheme is only slightly higher than that of the BCOP scheme in terms of PEKS generation and trapdoor generation. It is because that the computation involved in the underlying FDH-RSA scheme is quite small. It is worth noting that the result would also hold when

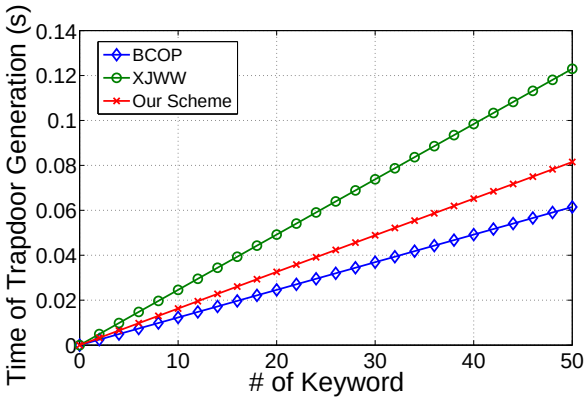


Figure 6.5: Computation Cost of Trapdoor Generation (excluding latency)

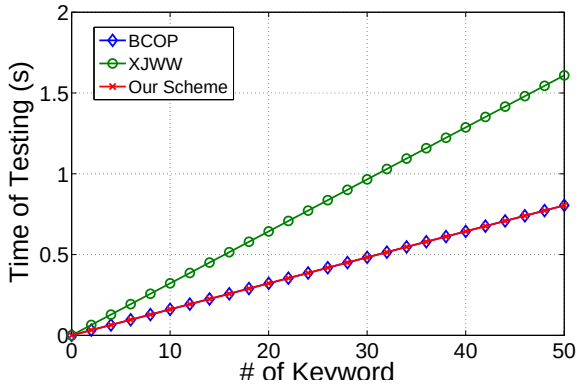


Figure 6.6: Computation Cost of Testing

we adopt our solution using other PEKS system to achieve security against off-line KGA as our client-KS protocol works transparently with the underlying PEKS system. Since our solution does not introduce any additional operation in the testing phase, the corresponding computation cost remains the same with the underlying PEKS system, as illustrated in Fig. 6.6. As for the scheme in [XJWW13] which achieves a certain level of security against off-line KGA, the computation cost is more than that of the PEKS scheme in [BCOP04] and our scheme in terms of all the operations. Particularly, it takes about 2 seconds to generate a PEKS ciphertext for the scheme in [XJWW13] when the keyword number is 50, while that of the scheme in [BCOP04] and our scheme is around 0.9 second and 1 second, respectively. For the trapdoor generation, the computation is slightly higher than that of our scheme as the exponentiation in $\mathbb{G}_1$ is usually more expensive than the exponentiation in $\mathbb{Z}_N^*$. To be more precise, the time of trapdoor generation for 50 keywords in [XJWW13] is about 0.12 seconds while that of our scheme is 0.08 seconds. Regarding the testing operation, the computation cost in [XJWW13] is almost twice that of our scheme. Specifically, the computation cost of testing is around 1.6 second for the scheme in [XJWW13] and 0.8 seconds for our scheme. This is because the testing in [XJWW13] requires an additional pairing computation.

6.6 Chapter Summary

In this work, we provided a practical and applicable treatment on (inside) off-line KGA by formalizing a new PEKS system, namely Server-Aided Public Key Encryption with Keyword Search (SA-PEKS). We introduced a universal transformation from any PEKS scheme to a secure SA-PEKS scheme, also with the first instantiation of SA-PEKS scheme and showed how to securely implement the client-KS protocol with a rate-limiting mechanism against on-line KGA. The experimental results showed that our proposed scheme achieves much better efficiency while providing resistance against both off-line and on-line KGAs.

Chapter 7

Strongly Leakage-Resilient Authenticated Key Exchange

Authenticated Key Exchange (AKE) protocols have been widely deployed in many real-world applications for securing communication channels. In this chapter, we make the following contributions. First, we revisit the security modelling of leakage-resilient AKE protocols, and show that the existing models either impose some unnatural restrictions or do not sufficiently capture leakage attacks in reality. We then introduce a new strong yet meaningful security model, named challenge-dependent leakage-resilient eCK (CLR-eCK) model, to capture challenge-dependent leakage attacks on both long-term secret key and ephemeral secret key (i.e., randomness). Second, we propose a general framework for constructing one-round CLR-eCK-secure AKE protocols based on smooth projective hash functions (SPHFs). This framework ensures the session key is private and authentic even if the adversary learns a large fraction of both long-term secret key and ephemeral secret key, and hence provides stronger security guarantee than existing AKE protocols which become insecure if the adversary can perform leakage attacks during the execution of a session. Finally, we also present a practical instantiation of the general framework based on the Decisional Diffie-Hellman assumption without the random oracle. Our result shows that the instantiation is efficient in terms of the communication and computation overhead and captures more general leakage attacks.

7.1 Introduction

Leakage-resilient cryptography, particularly leakage-resilient cryptographic primitives such as encryption, signature, and pseudo-random function, has been extensively studied in recent years. However, there are only very few works that have been done on the modelling and construction of leakage-resilient authenticated key exchange (AKE) protocols. This is somewhat surprising since AKE protocols are among the most widely used cryptographic primitives. In particular, they form a central component in many network standards, such as IPSec, SSL/TLS, SSH. In practice, the communication channel over a public network can be easily attacked by a malicious attacker and hence is insecure by default for message transmission. An AKE protocol enables a secure channel to be established among a set of communicating parties by first allowing them to agree on a cryptographically strong secret key,

and then applying efficient symmetric key tools to ensure the data confidentiality and authenticity.

Many practical AKE protocols such as the ISO protocol (a.k.a. SIG-DH) [ISO, CK01] and the Internet Key Exchange protocol (a.k.a. SIGMA) [Kra03] have been proposed and deployed in the aforementioned network standards. In such an AKE protocol, each party holds a *long-term public key* and the corresponding *long-term secret key*, which are static in the establishment of different session keys for multiple communication sessions. In order to establish a unique session key for an individual session, each party also generates their own *ephemeral secret key* and exchanges the corresponding *ephemeral public key*. Both parties can derive a common session key based on their own secret keys and the public keys of the peer entity. We should note that in practice, an AKE protocol proven secure in the traditional model could be completely insecure in the presence of leakage attacks. For example, an attacker can launch a memory attack [HSH+08, AGV09] to learn partial information about the static long-term secret key, and also obtain partial information about the ephemeral secret key (i.e., randomness) of an AKE session (e.g., via poorly implemented PRNGs [Mar, SF, Zet]).

7.1.1 Motivations

The general theme in formulating leakage resilience of cryptographic primitives is that in addition to the normal black-box interaction with an honest party, the adversary can also learn some partial information of a user secret via an abstract leakage function f. More precisely, the adversary is provided with access to a leakage oracle: the adversary can query the oracle with a polynomial-time computable function f, and then receive $f(sk)$, where sk is the user secret key. This approach was applied to model leakage resilience of many cryptographic schemes, such as pseudorandom generators [YSPY10], signature schemes [BSW13] and encryption schemes [NS09, CDRW10]. One of the major problems of leakage-resilient cryptography is to define a meaningful leakage function family $\mathcal{F}$ for a cryptographic primitive such that the leakage functions in $\mathcal{F}$ can cover as many leakage attacks as possible while at the same time it is still feasible to construct a scheme that can be proven secure. That is, in order to allow the software-level solution to solve the leakage problem in one go, the leakage function set $\mathcal{F}$ should be *as large as possible* and *adaptively* chosen by the adversary under minimal restrictions.

Limitations in Existing Leakage-Resilient AKE Models. The above modelling approach has been applied to define leakage-resilient AKE protocols in [ADW09, DHLW10, MO, ASB14]. This was done by allowing the adversary to access the leakage oracle in addition to other oracles defined in a traditional AKE security model.

However, we find that the existing leakage-resilient AKE models fail to fully capture general leakage attacks due to the following reasons.

1). *Unnatural Restrictions.* The *de facto* security definition of AKE requires that the real challenge session key should be indistinguishable from a randomly chosen key even when the adversary has obtained some information (e.g., by passively eavesdropping the ephemeral public keys, or injecting an ephemeral public key in an active attack) of the challenge session. However, such a definition will bring a problem when it comes to the leakage setting. During the execution of the challenge session, the adversary can access to the leakage oracle by encoding the available information about the challenge session into the leakage function and obtain partial information about the real session key. The previous security definitions for leakage-resilient AKE, e.g., [ADW09, DHLW10, MO, YMSW13], bypassed the definitional difficulty outlined above by only considering *challenge-independent leakage*. Namely, the adversary *cannot make a leakage query which involves a leakage function f that is related to the challenge session.* Specifically, in those models, the adversary is disallowed to make any leakage query during the challenge session. This approach indeed bypasses the technical problem, but it also puts some unnatural restrictions on the adversary by assuming leakage would not happen during the challenge AKE session. Such a definitional difficulty was also recognized in the prior work on leakage-resilient encryption schemes. For example, Naor and Segev wrote in [NS09] that "it will be very interesting to find an appropriate framework that allows a certain form of challenge-dependent leakage." We should note that there are some recent works on challenge-dependent leakage-resilient encryption schemes [HL11, YZYL14], which addressed the problem by weakening the security notions.

2). *Insufficient Leakage Capturing.* Although the notions proposed in [ADW09, DHLW10, MO, YMSW13, ASB14] have already captured some leakage attacks, they only focused on partial leakage of the long-term secret key. We should note that *the partial leakage is independent from the (long-term/ephemeral) secret key reveal queries* in CK/eCK models. In reality, an attacker may completely reveal one (long-term/ephemeral) secret key and learn partial information about the other (ephemeral/long-term) secret key. Such an adversarial capability has never been considered in the previous models. In practice, as mentioned before, potential weakness of the randomness can be caused due to different reasons such as the poor implementation of pseudo-random number generators (PRNGs) [Mar, SF, Zet]. Moreover, real leakage attacks (e.g., timing or power consumption analysis) can also be closely related to the ran-

domness. The problem has been recognized in prior work on leakage-resilient encryption and signature schemes. For example, Halevi and Lin mentioned in [HL11] that "Another interesting question is to handle leakage from the encryption randomness, not just the secret key", which was later answered by the works in [BCH12, YZYL14]. In terms of the signature schemes, the notion of fully leakage-resilient signatures was also proposed by Katz and Vaikuntanathan [KV09]. In a fully leakage-resilient setting, the adversary is allowed to obtain leakage of the state information, including the secret keys and internal random coins. However, to date there is no formal treatment on the randomness leakage in AKE protocols. This is surprising as randomness plays a crucial role in AKE protocols and determines the value of a session key.

On After-the-Fact Leakage. It is worth noting that inspired by the work in [HL11], Alawatugoda *et al.* [ASB14] modelled after-the-fact leakage for AKE protocols. Their proposed model, named bounded after-the-fact leakage eCK model (BAFL-eCK), captures the leakage of long-term secret keys during the challenge session. However, the BAFL-eCK model has implicitly assumed that the long-term secret has split-state since otherwise their definition is unachievable in the eCK-model. Moreover, the central idea of their AKE construction is to utilize a split-state encryption scheme with a special property (i.e., pair generation indistinguishability), which is a strong assumption. We also note that the split-state approach seems not natural for dealing with ephemeral secret leakage. The work in [ABS14] also introduced a continuous after-the-fact leakage eCK model which is a weaker variant of the one in [ASB14] and hence also suffers from the aforementioned limitations.

Goal of This Work. In this work, we are interested in designing a more general and powerful leakage-resilient AKE model without the aforementioned limitations. Particularly, we ask two questions: *how to generally define a challenge-dependent leakage-resilient AKE security model capturing both long-term and ephemeral secret leakage*, and *how to construct an efficient AKE protocol proven secure under the proposed security model*. The motivation of this work is to solve these two outstanding problems which are of both practical and theoretical importance.

7.1.2 Contributions and Techniques

In this work, we address the aforementioned open problems by designing a strong yet meaningful AKE security model, namely challenge-dependent leakage-resilient eCK (CLR-eCK) model, to capture the challenge-dependent leakage attacks on both the long-term secret key and the ephemeral secret key; we then present a general framework for the construction of CLR-eCK-secure one-round AKE protocol as well as an

efficient instantiation based on the DDH assumption. Below we give an overview of our results.

Overview of Our Model. Our model is the first *split-state-free* model that captures challenge-dependent leakage on both the long-term secret key and the ephemeral secret key (or randomness), which could occur in practice due to side-channel attacks and weak randomness implementations. In our proposed model, we consider the partial *Relative-Leakage* [AGV09]. We should note that the partial leakage here is independent from the secret key reveal queries in CK/eCK models. In our CLR-eCK model, the adversary can make both leakage and key reveal queries for the long-term and ephemeral secret keys. To be more precise, our model allows one (long-term/ephemeral) secret key to be completely revealed and the other (ephemeral/long-term) secret key to be partially leaked. Such an adversarial capability has never been considered in the previous models.

Our CLR-eCK security model addresses the limitations of the previous leakage-resilient models by allowing both long-term and ephemeral key leakage queries before, during and after the test (i.e., challenge) session. Nevertheless, we should prevent an adversary $\mathcal{M}$ from submitting a leakage function which encodes the session key derivation function of the test session since otherwise the adversary can trivially distinguish the real session key from a random key. To address this technical problem, instead of asking adversary $\mathcal{M}$ to specify the leakage functions before the system setup (i.e., non-adaptive leakage), we require $\mathcal{M}$ to commit a set of leakage functions before it obtains (via key reveal queries) all the inputs, except the to-be-leaked one, of the session key derivation function for the test session. Once $\mathcal{M}$ obtains all the other inputs, it can only use the leakage functions specified in the committed set to learn the partial information of the last unknown secret. To be more precise, in the CLR-eCK model, after $\mathcal{M}$ reveals the ephemeral secret key of the test session, it can only use any function $f_1 \in \mathcal{F}_1$ as the long-term secret key leakage function where $\mathcal{F}_1$ is the set of leakage functions committed by $\mathcal{M}$ before it reveals the ephemeral secret key. A similar treatment is done for the ephemeral secret key leakage function f_2. Under such a restriction, neither f_1 nor f_2 can be embedded with the session key derivation function of the test session and $\mathcal{M}$ cannot launch a trivial attack against the AKE protocol. Therefore, the adversary can still make leakage queries during and after the test session, and if the long-term/ephemeral key is not revealed, then the adversary even doesn't need to commit the ephemeral/long-term key leakage functions $\mathcal{F}_1$ or $\mathcal{F}_2$. We can see that our approach still allows the adversary to adaptively choose leakage functions and meanwhile can capture challenge-dependent leakage under the minimum restriction.

Generic AKE Construction. To illustrate the practicality of the model, we present a general framework for the construction of AKE protocol secure in our

newly proposed challenge-dependent leakage-resilient eCK model. The framework can be regarded as a variant of the AKE protocols proposed by Okamoto *et al.* [Oka07, MO]. Roughly speaking, we apply both pseudo-random functions (PRFs) and strong randomness extractors in the computation of ephemeral public key and session key to obtain the security in the presence of key leakage. Specifically, we employ an (extended) smooth projective hash function (SPHF) which is defined based on a domain $\mathcal{X}$ and an $\mathcal{NP}$ language $\mathcal{L} \subset \mathcal{X}$. For any word $W \in \mathcal{L}$, the hash value of W can be computed using either a secret hashing key or a public projection key with the knowledge of the witness for W. The key property of SPHF is that the projection key uniquely determines the hash value of any word in the language $\mathcal{L}$ (*projective*) but gives almost no information about the hash value of any point in $\mathcal{X} \setminus \mathcal{L}$ (*smooth*). During the session execution, both parties generate their ephemeral secret key and apply a strong extractor to extract a fresh seed for a PRF in order to derive a word in $\mathcal{L}$. They then exchange their words with the corresponding witness kept secret locally. Additionally, they also run an ephemeral Diffie-Hellman protocol using the exponent which is also output by the PRF. At the end of session, they derive the session key by computing the hash value of both words along with the Diffie-Hellman shared key. The correctness of the framework can be easily obtained due to the property of SPHF and Diffie-Hellman protocol while the security is guaranteed by the strong extractors, pseudo-random functions, along with the underlying (2-)smooth SPHF built on an $\mathcal{NP}$ language where the subgroup decision problem is hard.

An Efficient Instantiation. We show that the building blocks in our framework can be instantiated efficiently based on the DDH assumption. Precisely, we first introduce the Diffie-Hellman language $\mathcal{L}_{\mathsf{DH}} = \{(u_1, u_2) | \exists r \in \mathbb{Z}_p, s.t., u_1 = g_1^r, u_2 = g_2^r\}$ where $\mathbb{G}$ is a group of primer order p and $g_1, g_2 \in \mathbb{G}$ are generators. We then show that the subset membership problem over $\mathcal{X} = \mathbb{G}^2$ and $\mathcal{L}_{\mathsf{DH}}$ is hard and use it to construct a 2-smooth SPHF, denoted by $\mathcal{SPHF}_{\mathsf{DH}}$. A concrete protocol based on $\mathcal{SPHF}_{\mathsf{DH}}$ is then presented and proved to be CLR-eCK-secure. We should note that the communication costs in eSIG-DH [ADW09] and Enc-DH [DHLW10] is higher than our protocol due to the reason that they require their underlying primitive, i.e., signature or encryption scheme, to be leakage-resilient. For example, according to the result (**Theorem 5.2**) of [DHLW10], to obtain $(1 - \varepsilon)$-leakage resilience, the ciphertexts CT transferred in the Enc-DH protocol has the size of $O(1/\varepsilon)|\mathbb{G}|$. Due to the same reason, the computation overhead of those protocols is also higher than that of our protocol.

7.1.3 Related Work

Traditional AKE Security Notions. The Bellare-Rogaway (BR) model [BR93a] gives the first formal security notion for AKE based on an indistinguishability game, where an adversary is required to differentiate between the real session key from a randomly chosen session key. Its variants are nowadays the *de facto* standard for AKE security analysis. In particular, the Canetti-Krawczyk (CK) model [CK01], which can be considered as the extension and combination of the BR model and the Bellare-Canetti-Krawczyk (BCK) model [BCK98], has been used to prove the security of many widely used AKE protocols such as SIG-DH and SIGMA. Noting that the CK model does not capture several attacks such as the Key Compromise Impersonation (KCI) attacks, LaMacchia *et al.* [LLM07] introduced an extension of the CK model, named eCK model, to consider stronger adversaries (in some aspects) who is allowed to access either the long-term secret key or the ephemeral secret key in the target session chosen by the adversary. We refer the readers to Choo *et al.* [CBH05] for a detailed summary of the differences among the aforementioned AKE models, and to Cremers *et al.* [Cre11] for a full analysis of these models.

Modelling Leakage Resilience. The method of protecting against leakage attacks by treating them in an abstract way was first proposed by Micali and Reyzin [MR04] based on the assumption that *only computation leaks information*. Inspired by the cold boot attack presented by Halderman *et al.* [HSH+08], Akavia et al. [AGV09] formalized a general framework, namely, *Relative Leakage Model*, which implicitly assumes that, a leakage attack can reveal a fraction of the secret key, no matter what the secret key size is. The *Bounded-Retrieval Model* (BRM) [ADW09] is a generalization of the relative leakage model. In BRM, the leakage-parameter forms an independent parameter of the system. The secret key-size is then chosen flexibly depending on the leakage parameter. Another relatively stronger leakage model is the *Auxiliary Input Model* [DKL09] where the leakage is not necessarily bounded in length, but it is assumed to be computationally hard to recover the secret-key from the leakage.

Leakage-Resilient AKE. Alwen, Dodis and Wichs [ADW09] presented an efficient leakage-resilient AKE protocol in the random oracle model. They considered a leakage-resilient security model (BRM-CK) by extending the CK model to the BRM leakage setting. They then showed that a leakage-resilient AKE protocol can be constructed from an entropically-unforgeable digital signature scheme secure under chose-message attacks. Such a leakage-resilient signature-based AKE protocol, namely eSIG-DH, however, is at least 3-round and does not capture ephemeral secret key leakage. Also, the security model considered in [ADW09] does not capture challenge-dependent leakage since the adversary is not allowed to make leakage

queries during the execution of the challenge session. In [DHLW10], Dodis *et al.* proposed new constructions of AKE protocols that are leakage-resilient in the CK security model (LR-CK). Their first construction follows the result of [ADW09], i.e., authenticating Diffie-Hellman (DH) key exchange using a leakage-resilient signature scheme. The second construction, i.e., Enc-DH, is based on a leakage-resilient CCA-secure PKE scheme: both parties authenticate each other by requiring the peer entity to correctly decrypt the DH ephemeral public key encrypted under the long-term public key. Similar to Alwen at al. [ADW09], the security model given by Dodis *et al.* [DHLW10] is not challenge-dependent, and both constructions have at least 3-round and didn't consider randomness leakage. Another leakage-resilient model for AKE protocols is introduced by Moriyama and Okamoto [MO]. Their notion, named λ-leakage resilient eCK (LR-eCK) security, is an extension of the eCK security model with the notion of λ-leakage resilience introduced in [AGV09]. They also presented a 2-round AKE protocol that is λ-leakage resilient eCK secure without random oracles. One limitation of their model is that they just considered the long-term secret key leakage (when the ephemeral secret key is revealed) but not the ephemeral secret key leakage (when the long-term secret key is revealed). Also, their model is not challenge-dependent. Yang *et al.* [YMSW13] initiated the study on leakage resilient AKE in the auxiliary input model. They showed that in the random oracle model, an AKE protocol secure under auxiliary input attacks can be built based on a digital signature scheme that is random message unforgeable under random message and auxiliary input attacks (RU-RMAA). However, their model is based on the CK model and only captures the challenge-independent leakage of the lone-term secret.

7.2 A New Strong Security Model for AKE

We are now ready to introduce our proposed challenge-dependent leakage-resilient eCK (CLR-eCK) security model.

7.2.1 AKE Protocol

An AKE protocol is run among parties $(\mathcal{A}, \mathcal{B}, \mathcal{C}, ...)$ which are modelled as probabilistic polynomial-time Turing Machines. Each party has a *long-term secret key* (lsk) together with a certificate that binds the corresponding *long-term public key* (lpk) to the party. Here we denote $\widehat{A}$ $(\widehat{B})$ as the long-term public key of party $\mathcal{A}$ $(\mathcal{B})$ with the certificate issued by a trusted certificate authority $\mathcal{CA}$.

Any two parties, say $\mathcal{A}$ and $\mathcal{B}$, can be activated to run an instance of the AKE protocol, which is referred to as a *session*, and obtain a shared session key. In this

chapter, we only focus on one-round (i.e., two-pass) AKE protocols. Specifically, during the execution of a session, party $\mathcal{A}$ generates an *ephemeral public/secret key* pair $(epk_{\mathcal{A}}, esk_{\mathcal{A}})$ and sends $(\widehat{B}, \widehat{A}, epk_{\mathcal{A}})$ to the peer $\mathcal{B}$, and vice versa. At the end of the session execution, each party derives the shared session key by taking as input his/her own long-term secret key and ephemeral secret key, along with the long-term public key and ephemeral public key received from the other party.

A session of party $\mathcal{A}$ with peer $\mathcal{B}$ is identified by the session identifier $(\widehat{A}, \widehat{B}, epk_{\mathcal{A}}, epk_{\mathcal{B}})$, and the session $(\widehat{B}, \widehat{A}, epk_{\mathcal{B}}, epk_{\mathcal{A}})$ of party $\mathcal{B}$ is referred to as the *matching session* of $(\widehat{A}, \widehat{B}, epk_{\mathcal{A}}, epk_{\mathcal{B}})$. If the party outputs a session key at the end of the session, we call the session is completed successfully.

7.2.2 eCK Security Model

The extended Canetti-Krawczyk (eCK) model was proposed by LaMacchia, Lauter and Mityagin [LLM07] based on the CK model which was formulated by Canetti and Krawczyk [CK01] for the AKE protocols.

Roughly speaking, in the eCK definition, the adversary $\mathcal{M}$ is modelled as a probabilistic polynomial time Turing machine that controls all communications between the honest parties. Note that $\mathcal{M}$ cannot interfere with communication between a single party and the $\mathcal{CA}$ but is able to register fictitious parties. The adversary plays a central role in the model and is responsible for activating all other parties. That is, $\mathcal{M}$ schedules all activations of parties and message delivery. Initially and upon the completion of each activation, $\mathcal{M}$ decides which party to activate next. The adversary $\mathcal{M}$ also decides which incoming message or external request the activated party is to receive.

To be more precise, in the eCK model, adversary $\mathcal{M}$ is given the (certified) public keys of a set of honest users, and is allowed to issue the following oracle queries.

Send$(\mathcal{A}, \mathcal{B}, message)$. Send *message* to party $\mathcal{A}$ on behalf of party $\mathcal{B}$, and obtain $\mathcal{A}$'s response for this message.

EstablishParty(pid). This query allows the adversary to register a long-term public key on behalf of party pid, which is said to be *dishonest*.

LongTermKeyReveal(pid). This query allows the adversary to learn the long-term secret key of honest party pid.

SessionKeyReveal(sid). This query allows the adversary to obtain the session key of the completed session sid.

$\mathsf{EphemeralKeyReveal}(\mathsf{sid})$. This query allows the adversary to obtain the ephemeral secret key of session sid.

Eventually, in the challenge phase, adversary $\mathcal{M}$ selects a completed session sid^* as the *test session* and makes a query $\mathsf{Test}(\mathsf{sid}^*)$ as follows.

$\mathsf{Test}(\mathsf{sid}^*)$. To answer this query, the challenger pick $b \xleftarrow{\$} \{0,1\}$. If $b = 1$, the challenger returns $SK^* \leftarrow \mathsf{SessionKeyReveal}(\mathsf{sid}^*)$. Otherwise, the challenger sends $\mathcal{M}$ a random key $R^* \xleftarrow{\$} \{0,1\}^{|SK^*|}$.

Note that the Test query can be issued only once but at any time during the game, and the game terminates as soon as $\mathcal{M}$ outputs its guess b' on b. Here, we require the *test session* to be a *fresh session* which is defined as follows.

Definition 7.1 (Fresh Session in eCK Model) *Let* sid *be the completed session owned by an honest party* $\mathcal{A}$ *with peer* $\mathcal{B}$*, who is also honest. If there exists the matching session to session* sid*, we denote the matching session as* $\overline{\mathsf{sid}}$*. Session* sid *is said to be fresh if none of the following conditions hold:*

1). $\mathcal{M}$ *issues a* $\mathsf{SessionKeyReveal}(\mathsf{sid})$ *query or a* $\mathsf{SessionKeyReveal}(\overline{\mathsf{sid}})$ *query (If* $\overline{\mathsf{sid}}$ *exists).*

2). $\overline{\mathsf{sid}}$ *exists and* $\mathcal{M}$ *issues either*

$\mathsf{LongTermKeyReveal}(\mathcal{A}) \wedge \mathsf{EphemeralKeyReveal}(\mathsf{sid})$*, or*

$\mathsf{LongTermKeyReveal}(\mathcal{B}) \wedge \mathsf{EphemeralKeyReveal}(\overline{\mathsf{sid}})$.

3). $\overline{\mathsf{sid}}$ *does not exist and* $\mathcal{M}$ *issues either*

$\mathsf{LongTermKeyReveal}(\mathcal{A}) \wedge \mathsf{EphemeralKeyReveal}(\mathsf{sid})$*, or*

$\mathsf{LongTermKeyReveal}(\mathcal{B})$.

We remark that the freshness of the test session can be identified only after the game is completed as $\mathcal{M}$ can continue the other queries after the Test query. That is, $\mathcal{M}$ wins the game if he correctly guesses the challenge for the test session which remains fresh until the end of the game. Formally, we have the following notion for eCK security.

Definition 7.2 (eCK Security) *Let the test session* sid^* *be fresh where adversary* $\mathcal{M}$ *issues* $\mathsf{Test}(sid^*)$ *query. We define the advantage of* $\mathcal{M}$ *in the eCK game by*

$$\mathsf{Adv}^{\mathsf{eCK}}_{\mathcal{M}}(\ell) = \Pr[b' = b] - 1/2,$$

where k *is the security parameter of the AKE protocol. We say the AKE protocol is eCK-secure if the matching session computes the same session key and for any probabilistic polynomial-time adversary* $\mathcal{M}$*,* $\mathsf{Adv}^{\mathsf{eCK}}_{\mathcal{M}}(\ell)$ *is negligible.*

7.2.3 Challenge-Dependent Leakage-Resilient eCK Model

We introduce a new eCK-based security notion to capture various *side-channel attacks* against AKE protocols. Our notion, named *Challenge-Dependent Leakage-Resilient eCK* (CLR-eCK) model is the first split-state-free security model that captures both long-term and ephemeral key leakage *and* allows the adversary to issue leakage queries even *after the activation of the test session*. Formally, adversary $\mathcal{M}$ is allowed to issue the following queries.

Send$(\mathcal{A}, \mathcal{B}, message)$. Send $message$ to party $\mathcal{A}$ on behalf of party $\mathcal{B}$, and obtain $\mathcal{A}$'s response for this message.

EstablishParty(pid). Register a long-term public key on behalf of party pid, which is said to be *dishonest*.

LongTermKeyReveal(pid). Query the long-term secret key of honest party pid.

SessionKeyReveal(sid). Query the session key of the completed session sid.

EphemeralKeyReveal(sid). Query the ephemeral secret key of session sid.

LongTermKeyLeakage(f_1, pid). This query allows $\mathcal{M}$ to learn $f_1(lsk)$ where f_1 denotes the leakage function and lsk denotes the long-term secret key of party pid.

EphemeralKeyLeakage(f_2, sid). This query allows $\mathcal{M}$ to learn $f_2(esk)$ where f_2 denotes the leakage function and esk denotes the ephemeral secret key used by an honest user in the session sid.

Test(sid*). To answer this query, the challenger pick $b \xleftarrow{\$} \{0,1\}$. If $b = 1$, the challenger returns $SK^* \leftarrow$ SessionKeyReveal(sid*). Otherwise, the challenger sends the adversary a random key $R^* \xleftarrow{\$} \{0,1\}^{|SK^*|}$.

Note that the Test query can be issued only once but at any time during the game, and the game terminates as soon as $\mathcal{M}$ outputs its guess b' on b.

Restrictions on the Leakage Function. In our CLR-eCK security model, we consider several restrictions on the leakage function to prevent the adversary $\mathcal{M}$ from trivially breaking the AKE protocol.

The first restriction is that the output size of the leakage function f_1 and f_2 must be less than $|lsk|$ and $|esk|$, respectively. Specifically, following some previous work on leakage resilient cryptography [NS09], we require the output size of a leakage function f is at most λ bits, which means the entropy loss of sk is at most λ bits upon observing $f(sk)$. Formally, we define the bounded leakage function family $\mathcal{F}_{\mathsf{bbd\text{-}l}}$ for

the long-term secret key and $\mathcal{F}_{\mathsf{bbd\text{-}II}}$ for the ephemeral secret key as follows. $\mathcal{F}_{\mathsf{bbd\text{-}I}}(\ell)$ is defined as the class of all polynomial-time computable functions: $f : \{0,1\}^{|lsk|} \rightarrow \{0,1\}^{\leq \lambda_1(\ell)}$, where $\lambda_1(\ell) < |lsk|$. $\mathcal{F}_{\mathsf{bbd\text{-}II}}(\ell)$ is defined as the class of all polynomial-time computable functions: $f : \{0,1\}^{|esk|} \rightarrow \{0,1\}^{\leq \lambda_2(\ell)}$, where $\lambda_2(\ell) < |esk|$. We then require that the leakage function submitted by the adversary should satisfy that $f_1 \in \mathcal{F}_{\mathsf{bbd\text{-}I}}$ and $f_2 \in \mathcal{F}_{\mathsf{bbd\text{-}II}}$.

Another restriction that must be enforced is related to the challenge-dependent leakage security of AKE protocols. Consider a test session sid^* which is owned by party $\mathcal{A}$ with peer $\mathcal{B}$. Note that for a 2-pass AKE protocol, the session key of sid^* is determined by $(\widehat{A}, \widehat{B}, lsk_{\mathcal{A}}, esk^*_{\mathcal{A}}, lpk_{\mathcal{B}}, epk^*_{\mathcal{B}})$ which contains only two secret keys (i.e., $lsk_{\mathcal{A}}, esk^*_{\mathcal{A}}$). Since $\mathcal{M}$ is allowed to reveal $esk^*_{\mathcal{A}}$ ($lsk_{\mathcal{A}}$) in the eCK model, $\mathcal{M}$ can launch a trivial attack by encoding the session key derivation function into the leakage function of $lsk_{\mathcal{A}}$ ($esk^*_{\mathcal{A}}$) and hence wins the security game. Therefore, adversary $\mathcal{M}$ should not be allowed to adaptively issue leakage query after it obtains all the other (secret) information for session key computation, otherwise the security of AKE protocol is unachievable. More precisely, we describe the restrictions on $\mathsf{LongTermKeyLeakage}(f_1, \mathcal{A})$ and $\mathsf{EphemeralKeyLeakage}(f_2, \mathsf{sid}^*)$ as follows.

1). $\mathcal{M}$ is allowed to ask for arbitrary leakage function $f_1 \in \mathcal{F}_{\mathsf{bbd\text{-}I}}$ before it obtains the ephemeral secret key $esk^*_{\mathcal{A}}$, i.e., by issuing $\mathsf{EphemeralKeyReveal}(\mathsf{sid}^*)$ query; however, after obtaining $esk^*_{\mathcal{A}}$, $\mathcal{M}$ can only use the leakage functions $f_1 \in \mathcal{F}_1 \subset \mathcal{F}_{\mathsf{bbd\text{-}I}}$ where $\mathcal{F}_1$ is a set of leakage functions chosen and submitted by $\mathcal{M}$ before it issues $\mathsf{EphemeralKeyReveal}(\mathsf{sid}^*)$.

2). $\mathcal{M}$ is allowed to ask for arbitrary leakage function $f_2 \in \mathcal{F}_{\mathsf{bbd\text{-}II}}$ before it obtains the long-term secret key $lsk_{\mathcal{A}}$, i.e., by issuing $\mathsf{LongTermKeyReveal}(\mathcal{A})$ query; however, after obtaining $lsk_{\mathcal{A}}$, $\mathcal{M}$ can only use the leakage functions $f_2 \in \mathcal{F}_2 \subset \mathcal{F}_{\mathsf{bbd\text{-}II}}$ where $\mathcal{F}_2$ is a set of leakage functions chosen and submitted by $\mathcal{M}$ before it issues $\mathsf{LongTermKeyReveal}(\mathcal{A})$.

We should note that if $\overline{\mathsf{sid}^*}$ exists, the above restriction must also be enforced for the leakage query $\mathsf{LongTermKeyLeakage}(f_1, \mathcal{B})$ and $\mathsf{EphemeralKeyLeakage}(f_2, \overline{\mathsf{sid}^*})$, since the session key of sid^* is also determined by $(\widehat{A}, \widehat{B}, lpk_{\mathcal{A}}, epk^*_{\mathcal{A}}, lsk_{\mathcal{B}}, esk^*_{\mathcal{B}})$.

Adaptive Leakage. One can see that our proposed model enables adversary $\mathcal{M}$ to choose $\mathcal{F}_1, \mathcal{F}_2$ adaptively and $\mathcal{M}$ can submit $\mathcal{F}_1, \mathcal{F}_2$ even after the challenge phase as long as the restriction holds. That is, $\mathcal{M}$ can specify function set $\mathcal{F}_1, \mathcal{F}_2$ after seeing $epk^*_{\mathcal{A}}$ and $epk^*_{\mathcal{B}}$. Also, if there is no long-term (ephemeral, respectively) key reveal query, then $\mathcal{F}_1$ ($\mathcal{F}_2$, respectively) is the same as $\mathcal{F}_{\mathsf{bbd\text{-}I}}$ ($\mathcal{F}_{\mathsf{bbd\text{-}II}}$, respectively). Implicitly, $\mathcal{M}$ is allowed to obtain $f_1(lsk_{\mathcal{A}}), f'_1(lsk_{\mathcal{B}}), f_2(esk^*_{\mathcal{A}}), f'_2(esk^*_{\mathcal{B}})$ where $f_1, f'_1 \in \mathcal{F}_{\mathsf{bbd\text{-}I}}, f_2, f'_2 \in \mathcal{F}_{\mathsf{bbd\text{-}II}}$ can be dependent on $(lpk_{\mathcal{A}}, lpk_{\mathcal{B}}, epk^*_{\mathcal{A}}, epk^*_{\mathcal{B}})$, or to obtain

$f_1(lsk_{\mathcal{A}}), f_2(esk^*_{\mathcal{B}})$ where $f_1 \in \mathcal{F}_1, f_2 \in \mathcal{F}_2$ can be dependent on $(lpk_{\mathcal{A}}, lpk_{\mathcal{B}}, lsk_{\mathcal{B}}, epk^*_{\mathcal{A}}, epk^*_{\mathcal{B}})$ and $(lpk_{\mathcal{A}}, lpk_{\mathcal{B}}, epk^*_{\mathcal{A}}, esk^*_{\mathcal{A}}, epk^*_{\mathcal{B}})$, respectively. Since the leakage can happen during or after the challenge session and can be related to the challenge session, our proposed security model captures the challenge-dependent leakage security for AKE protocols.

We define the notion of a *fresh session* in the CLR-eCK model as follows.

Definition 7.3 ((λ_1, λ_2)-Leakage Fresh Session in the CLR-eCK Model) *Let* sid *be a completed session owned by an honest party $\mathcal{A}$ with peer $\mathcal{B}$, who is also honest. Let* $\overline{\text{sid}}$ *denote the matching session of* sid*, if it exists. Session* sid *is said to be fresh in the* CLR-eCK *model if the following conditions hold:*

1). sid *is a fresh session in the sense of eCK model.*

2). $\mathcal{M}$ only issues the queries LongTermKeyLeakage$(f_1, \mathcal{A})$, LongTermKeyLeakage$(f'_1, \mathcal{B})$, EphemeralKeyLeakage(f_2, sid), EphemeralKeyLeakage$(f'_2, \overline{\text{sid}})$ *(if* $\overline{\text{sid}}$ *exists), such that f_1, f'_1, f_2, f'_2 satisfy the restriction given above.*

3). The total output length of all the LongTermKeyLeakage *queries to $\mathcal{A}$ ($\mathcal{B}$, respectively) is at most λ_1.*

4). The total output length of all the EphemeralKeyLeakage *query to* sid *(*$\overline{\text{sid}}$*, respectively, if it exists) is at most λ_2.*

We now describe the notion of CLR-eCK security.

Definition 7.4 (CLR-eCK Security) *Let the test session* sid^* *be (λ_1, λ_2)-leakage fresh where adversary $\mathcal{M}$ issues* Test(sid^*) *query. We define the advantage of $\mathcal{M}$ in the* CLR-eCK *game by* $\text{Adv}^{\text{CLR-eCK}}_{\mathcal{M}}(\ell) = \Pr[b' = b] - 1/2$*, where k is the security parameter of the AKE protocol. We say the AKE protocol is (λ_1, λ_2)-challenge-dependent leakage-resilient eCK-secure ((λ_1, λ_2)-*CLR-eCK*-secure) if the matching session computes the same session key and for any probabilistic polynomial-time adversary $\mathcal{M}$,* $\text{Adv}^{\text{CLR-eCK}}_{\mathcal{M}}(\ell)$ *is negligible.*

Remark. Here we give a further discussion on the relationship between the reveal oracle, e.g., LongTermKeyReveal and the leakage oracle, e.g., LongTermKeyLeakage. We can see that it is meaningless for $\mathcal{M}$ to issue the leakage query on the long-term secret key (ephemeral secret key) if it has already obtained the whole key through querying the reveal oracle. Indeed, adversary $\mathcal{M}$ can compute by itself the leakage function $f_1(lsk_{\mathcal{A}})$ if $lsk_{\mathcal{A}}$ is known to him.

Therefore, we can observe that the meaningful queries that adversary $\mathcal{M}$ will ask in CLR-eCK model are as follows. Suppose session sid^* is the test session owned by $\mathcal{A}$ with the peer $\mathcal{B}$. If $\overline{\text{sid}^*}$ exists, $\mathcal{M}$ will only make queries that form a subset of any one of the following cases:

1). $\{\mathsf{LongTermKeyReveal}(\mathcal{A}), \mathsf{LongTermKeyReveal}(\mathcal{B}), \mathsf{EphemeralKeyLeakage}(\mathsf{sid}^*), \mathsf{EphemeralKeyLeakage}(\overline{\mathsf{sid}^*})\}$,[a]

2). $\{\mathsf{EphemeralKeyReveal}(\mathsf{sid}^*), \mathsf{EphemeralKeyReveal}(\overline{\mathsf{sid}^*}), \mathsf{LongTermKeyLeakage}(\mathcal{A}), \mathsf{LongTermKeyLeakage}(\mathcal{B})\}$,

3). $\{\mathsf{LongTermKeyReveal}(\mathcal{A}), \mathsf{EphemeralKeyReveal}(\overline{\mathsf{sid}^*}), \mathsf{EphemeralKeyLeakage}(\mathsf{sid}^*), \mathsf{LongTermKeyLeakage}(\mathcal{B})\}$,

4). $\{\mathsf{EphemeralKeyReveal}(\mathsf{sid}^*), \mathsf{LongTermKeyReveal}(\mathcal{B}), \mathsf{LongTermKeyLeakage}(\mathcal{A}), \mathsf{EphemeralKeyLeakage}(\overline{\mathsf{sid}^*})\}$.

If $\overline{\mathsf{sid}^*}$ does not exist, we have the following cases:

5). $\{\mathsf{LongTermKeyReveal}(\mathcal{A}), \mathsf{EphemeralKeyLeakage}(\mathsf{sid}^*), \mathsf{LongTermKeyLeakage}(\mathcal{B})\}$,

6). $\{\mathsf{EphemeralKeyReveal}(\mathsf{sid}^*), \mathsf{LongTermKeyLeakage}(\mathcal{A}), \mathsf{LongTermKeyLeakage}(\mathcal{B})\}$.

7.3 One-Round CLR-eCK-Secure AKE

In this section, we present a generic construction of one-round CLR-eCK-secure AKE protocol.

7.3.1 Extended Smooth Projective Hash Function

In order to make the SPHF notion well applied for our construction, similar to [CS02], we also need an extension of the SPHF in this chapter. Precisely, we introduce the WordG algorithm and slightly modify the $\mathsf{Hash}, \mathsf{ProjHash}$ algorithms for SPHF as follows.

$\mathsf{WordG}(w)$[b] : generates a word $W \in \mathcal{L}$ with w the witness ;

$\mathsf{Hash}(\mathsf{hk}, W, aux)$: outputs the hash value $\mathsf{hv} \in \mathcal{Y}$ on the word W from the hashing key hk and the auxiliary input aux;

$\mathsf{ProjHash}(\mathsf{hp}, W, w, aux)$: outputs the hash value $\mathsf{hv}' \in \mathcal{Y}$, on the word W from the projection key hp, the witness w for the fact that $W \in \mathcal{L}$ and the auxiliary input aux.

[a] For simplicity, we will omit the leakage function in the input of the leakage query in the rest of the chapter.

[b] For simplicity, we will omit $(\mathcal{L}, \mathsf{param})$ in the input of the SPHF algorithms in the rest of the chapter.

Property. A smooth projective hash function $\mathcal{SPHF}$[c] should satisfy the following properties,

Correctness. Let $W = \mathsf{WordG}(w)$, then for all hashing key hk and projection key hp , we have

$$\mathsf{Hash}(\mathsf{hk}, W, aux) = \mathsf{ProjHash}(\mathsf{hp}, W, w, aux)$$

Smoothness. For any $W \in \mathcal{X}\backslash\mathcal{L}$. Then the following two distributions are perfectly indistinguishable:

$$\mathcal{V}_1 = \{(\mathcal{L}, \mathsf{param}, W, \mathsf{hp}, aux, \mathsf{hv}) | \mathsf{hv} = \mathsf{Hash}(\mathsf{hk}, W, aux)\},$$

$$\mathcal{V}_2 = \{(\mathcal{L}, \mathsf{param}, W, \mathsf{hp}, aux, \mathsf{hv}) | \mathsf{hv} \xleftarrow{\$} \mathcal{Y}\}.$$

Definition 7.5 (2-smooth SPHF) *For any $W_1, W_2 \in \mathcal{X}\backslash\mathcal{L}$, let aux_1, aux_2 be the auxiliary inputs such that $(W_1, aux_1) \neq (W_2, aux_2)$, we say an SPHF is 2-smooth if the following two distributions are perfectly indistinguishable :*

$$\mathcal{V}_1 = \{(\mathcal{L}, \mathsf{param}, W_1, W_2, \mathsf{hp}, aux_1, aux_2, \mathsf{hv}_1, \mathsf{hv}_2) | \mathsf{hv}_2 = \mathsf{Hash}(\mathsf{hk}, W_2, aux_2)\},$$

$$\mathcal{V}_2 = \{(\mathcal{L}, \mathsf{param}, W_1, W_2, \mathsf{hp}, aux_1, aux_2, \mathsf{hv}_1, \mathsf{hv}_2) | \mathsf{hv}_2 \xleftarrow{\$} \mathcal{Y}\}.$$

where $\mathsf{hv}_1 = \mathsf{Hash}(\mathsf{hk}, (\mathcal{L}, \mathsf{param}), W_1, aux_1)$.

7.3.2 General Framework

Table 7.1 describes a generic construction of the CLR-eCK secure AKE protocol. Suppose that ℓ is the system security parameter. Let $\mathbb{G}$ be a group with prime order p and g is a random generator of $\mathbb{G}$. Let $\mathcal{SPHF}$ denote a 2-smooth SPHF over $\mathcal{L} \subset \mathcal{X}$ and onto the set $\mathcal{Y}$ such that the subset membership problem between $\mathcal{L}$ and $\mathcal{X}$ is hard. Denote the hashing key space by $\mathcal{HK}$, the projection key space by $\mathcal{HP}$, the auxiliary input space by $\mathcal{AUX}$ and the witness space by $\mathcal{W}$. Pick two collision-resistant hash functions $H_1 : \{0,1\}^* \to \mathcal{AUX}, H_2 : \mathbb{G} \to \mathcal{Y}$.

Let $\lambda_1 = \lambda_1(\ell)$ be the bound on the amount of long-term secret key leakage and $\lambda_2 = \lambda_2(\ell)$ be that of the ephemeral secret key leakage. Let $\mathsf{Ext}_1, \mathsf{Ext}_2, \mathsf{Ext}_3$ be strong extractors as follows. $\mathsf{Ext}_1 : \mathcal{HK} \times \{0,1\}^{t_1(\ell)} \to \{0,1\}^{l_1(\ell)}$ is an average-case $(|\mathcal{HK}| - \lambda_1, \epsilon_1)$-strong extractor. $\mathsf{Ext}_2 : \{0,1\}^{u(\ell)} \times \{0,1\}^{t_2(\ell)} \to \{0,1\}^{l_2(\ell)}$ is an average-case $(k - \lambda_2, \epsilon_2)$-strong extractor. $\mathsf{Ext}_3 : \mathcal{Y} \times \{0,1\}^{t_3(\ell)} \to \{0,1\}^{l_3(\ell)}$ is an

[c] In the rest of this chapter, all the SPHFs are referred to as the extended SPHF and defined by algorithms (SPHFSetup, HashKG, ProjKG, WordG, Hash, ProjHash).

Table 7.1: Framework for CLR-eCK secure AKE

$\mathcal{A}$		$\mathcal{B}$
$\mathsf{hk} \xleftarrow{\$} \mathsf{HashKG}$,		$\mathsf{hk}' \xleftarrow{\$} \mathsf{HashKG}$,
$\mathsf{hp} \xleftarrow{\$} \mathsf{ProjKG}(\mathsf{hk})$,		$\mathsf{hp}' \xleftarrow{\$} \mathsf{ProjKG}(\mathsf{hk}')$,
$r_{\mathcal{A}_1} \xleftarrow{\$} \{0,1\}^{t_1(\ell)}, r_{\mathcal{A}_2} \xleftarrow{\$} \{0,1\}^{t_2(\ell)}$,		$r_{\mathcal{B}_1} \xleftarrow{\$} \{0,1\}^{t_1(\ell)}, r_{\mathcal{B}_2} \xleftarrow{\$} \{0,1\}^{t_2(\ell)}$,
$lsk_{\mathcal{A}} = \mathsf{hk}, lpk_{\mathcal{A}} = (\mathsf{hp}, r_{\mathcal{A}_1}, r_{\mathcal{A}_2})$.		$lsk_{\mathcal{B}} = \mathsf{hk}', lpk_{\mathcal{B}} = (\mathsf{hp}', r_{\mathcal{B}_1}, r_{\mathcal{B}_2})$.
$esk_{\mathcal{A}} \xleftarrow{\$} \{0,1\}^{u(\ell)}, t_{\mathcal{A}} \xleftarrow{\$} \{0,1\}^{t_3(\ell)}$,		$esk_{\mathcal{B}} \xleftarrow{\$} \{0,1\}^{u(\ell)}, t_{\mathcal{B}} \xleftarrow{\$} \{0,1\}^{t_3(\ell)}$,
$\widehat{lsk}_{\mathcal{A}} = \mathsf{Ext}_1(lsk_{\mathcal{A}}, r_{\mathcal{A}_1})$,		$\widehat{lsk}_{\mathcal{B}} = \mathsf{Ext}_1(lsk_{\mathcal{B}}, r_{\mathcal{B}_1})$,
$\widehat{esk}_{\mathcal{A}} = \mathsf{Ext}_2(esk_{\mathcal{A}}, r_{\mathcal{A}_2})$,		$\widehat{esk}_{\mathcal{B}} = \mathsf{Ext}_2(esk_{\mathcal{B}}, r_{\mathcal{B}_2})$,
$(w_{\mathcal{A}}, x) = \widehat{F}_{\widehat{lsk}_{\mathcal{A}}}(esk_{\mathcal{A}}) + \overline{F}_{\widehat{esk}_{\mathcal{A}}}(r_{\mathcal{A}_1})$,		$(w_{\mathcal{B}}, y) = \widehat{F}_{\widehat{lsk}_{\mathcal{B}}}(esk_{\mathcal{B}}) + \overline{F}_{\widehat{esk}_{\mathcal{B}}}(r_{\mathcal{B}_1})$,
$W_{\mathcal{A}} = \mathsf{WordG}(w_{\mathcal{A}}), X = g^x$,		$W_{\mathcal{B}} = \mathsf{WordG}(w_{\mathcal{B}}), Y = g^y$,
Erase all state except $(esk_{\mathcal{A}}, W_{\mathcal{A}}, X, t_{\mathcal{A}})$.		Erase all state except $(esk_{\mathcal{B}}, W_{\mathcal{B}}, Y, t_{\mathcal{B}})$.
	$\xrightarrow{(\widehat{B}, \widehat{A}, W_{\mathcal{A}}, X, t_{\mathcal{A}})}$	
	$\xleftarrow{(\widehat{A}, \widehat{B}, W_{\mathcal{B}}, Y, t_{\mathcal{B}})}$	
Set $\mathsf{sid} = (\widehat{A}, \widehat{B}, W_{\mathcal{A}}, X, t_{\mathcal{A}}, W_{\mathcal{B}}, Y, t_{\mathcal{B}})$		Set $\mathsf{sid} = (\widehat{A}, \widehat{B}, W_{\mathcal{A}}, X, t_{\mathcal{A}}, W_{\mathcal{B}}, Y, t_{\mathcal{B}})$
$aux = H_1(\mathsf{sid}), K_{\mathcal{A}_1} = Y^x$,		$aux = H_1(\mathsf{sid}), K_{\mathcal{A}_1} = X^y$,
$K_{\mathcal{A}_2} = \mathsf{ProjHash}(lpk_{\mathcal{B}}, W_{\mathcal{A}}, w_{\mathcal{A}}, aux)$,		$K_{\mathcal{B}_2} = \mathsf{Hash}(lsk_{\mathcal{B}}, W_{\mathcal{A}}, aux)$,
$K_{\mathcal{A}_3} = \mathsf{Hash}(lsk_{\mathcal{A}}, W_{\mathcal{B}}, aux)$,		$K_{\mathcal{B}_3} = \mathsf{ProjHash}(lpk_{\mathcal{A}}, W_{\mathcal{B}}, w_{\mathcal{B}}, aux)$,
$s_{\mathcal{A}} = \mathsf{Ext}_3(H_2(K_{\mathcal{A}_1}) \oplus K_{\mathcal{A}_2} \oplus K_{\mathcal{A}_3}, t_{\mathcal{A}} \oplus t_{\mathcal{B}})$,		$s_{\mathcal{B}} = \mathsf{Ext}_3(H_2(K_{\mathcal{B}_1}) \oplus K_{\mathcal{B}_2} \oplus K_{\mathcal{B}_3}, t_{\mathcal{A}} \oplus t_{\mathcal{B}})$,
$SK_{\mathcal{A}} = \widetilde{F}_{s_{\mathcal{A}}}(\mathsf{sid})$.		$SK_{\mathcal{B}} = \widetilde{F}_{s_{\mathcal{B}}}(\mathsf{sid})$.

average-case $(|\mathcal{Y}| - \lambda_1, \epsilon_3)$-strong extractor. Here $\epsilon_1 = \epsilon_1(\ell), \epsilon_2 = \epsilon_2(\ell), \epsilon_3 = \epsilon_3(\ell)$ are negligible.

Let $\widehat{\mathsf{F}}$ and $\overline{\mathsf{F}}$ be PRF families and $\widetilde{\mathsf{F}}$ be a πPRF family as follows.

$$\widehat{\mathsf{F}}^{k,\sum_{\widehat{\mathsf{F}}},\mathcal{D}_{\widehat{\mathsf{F}}},\mathcal{R}_{\widehat{\mathsf{F}}}} : \sum_{\widehat{\mathsf{F}}} = \{0,1\}^{l_1(\ell)}, \mathcal{D}_{\widehat{\mathsf{F}}} = \{0,1\}^{u(\ell)}, \mathcal{R}_{\widehat{\mathsf{F}}} = \mathcal{W} \times \mathbb{Z}_p,$$

$$\overline{\mathsf{F}}^{k,\sum_{\overline{\mathsf{F}}},\mathcal{D}_{\overline{\mathsf{F}}},\mathcal{R}_{\overline{\mathsf{F}}}} : \sum_{\overline{\mathsf{F}}} = \{0,1\}^{l_2(\ell)}, \mathcal{D}_{\overline{\mathsf{F}}} = \{0,1\}^{t_1(\ell)}, \mathcal{R}_{\overline{\mathsf{F}}} = \mathcal{W} \times \mathbb{Z}_p,$$

$$\widetilde{\mathsf{F}}^{k,\sum_{\widetilde{\mathsf{F}}},\mathcal{D}_{\widetilde{\mathsf{F}}},\mathcal{R}_{\widetilde{\mathsf{F}}}} : \sum_{\widetilde{\mathsf{F}}} = \{0,1\}^{l_3(\ell)}, \mathcal{D}_{\widetilde{\mathsf{F}}} = (\Lambda_k)^2 \times \mathcal{L}^2 \times \mathbb{G}^2 \times \{0,1\}^{2t_3(\ell)}, \mathcal{R}_{\widetilde{\mathsf{F}}} = \{0,1\}^{l_4(\ell)}.^{\mathrm{d}}$$

Let $\widehat{F} \leftarrow \widehat{\mathsf{F}}^{k,\sum_{\widehat{\mathsf{F}}},\mathcal{D}_{\widehat{\mathsf{F}}},\mathcal{R}_{\widehat{\mathsf{F}}}}$, $\overline{F} \leftarrow \overline{\mathsf{F}}^{k,\sum_{\overline{\mathsf{F}}},\mathcal{D}_{\overline{\mathsf{F}}},\mathcal{R}_{\overline{\mathsf{F}}}}$ and $\widetilde{F} \leftarrow \widetilde{\mathsf{F}}^{k,\sum_{\widetilde{\mathsf{F}}},\mathcal{D}_{\widetilde{\mathsf{F}}},\mathcal{R}_{\widetilde{\mathsf{F}}}}$.

The system parameter is $(\mathsf{param}, \mathbb{G}, p, g, H_1, H_2, \mathsf{Ext}_1, \mathsf{Ext}_2, \mathsf{Ext}_3, \widehat{F}, \overline{F}, \widetilde{F})$ where $\mathsf{param} \leftarrow \mathsf{SPHFSetup}(1^\ell)$.

Key Generation. At the long-term key generation stage, $\mathcal{A}$ runs the algorithm HashKG to obtain a hashing key hk and then the algorithm ProjKG to obtain the projection key hp, picks $r_{\mathcal{A}_1} \xleftarrow{\$} \{0,1\}^{t_1(\ell)}, r_{\mathcal{A}_2} \xleftarrow{\$} \{0,1\}^{t_2(\ell)}$, then sets its long-term key pair as $lsk_{\mathcal{A}} = \mathsf{hk}, lpk_{\mathcal{A}} = (\mathsf{hp}, r_{\mathcal{A}_1}, r_{\mathcal{A}_2})$. Similarly, $\mathcal{B}$ generates its long-term key pair as $lsk_{\mathcal{B}} = \mathsf{hk}', lpk_{\mathcal{B}} = (\mathsf{hp}', r_{\mathcal{B}_1}, r_{\mathcal{B}_2})$.

Session Execution $(\mathcal{A} \rightleftharpoons \mathcal{B})$**.** The key exchange protocol between $\mathcal{A}$ and $\mathcal{B}$ executes as follows.

$(\mathcal{A} \rightharpoonup \mathcal{B})$. $\mathcal{A}$ performs the following steps.

[d] In this chapter, we denote the space of a certified long-term public key (such as $\widehat{\mathcal{A}}$) by Λ_k.

1. Selects the ephemeral secret key $esk_{\mathcal{A}} \xleftarrow{\$} \{0,1\}^{u(\ell)}$ and picks $t_{\mathcal{A}} \xleftarrow{\$} \{0,1\}^{t_3(\ell)}$.
2. Sets $\widehat{lsk}_{\mathcal{A}} = \mathsf{Ext}_1(lsk_{\mathcal{A}}, r_{\mathcal{A}_1}), \widehat{esk}_{\mathcal{A}} = \mathsf{Ext}_2(esk_{\mathcal{A}}, r_{\mathcal{A}_2})$.
3. Computes $(w_{\mathcal{A}}, x) = \widehat{F}_{\widehat{lsk}_{\mathcal{A}}}(esk_{\mathcal{A}}) + \overline{F}_{\widehat{esk}_{\mathcal{A}}}(r_{\mathcal{A}_1})$.
4. Runs the algorithm $\mathsf{WordG}(w_{\mathcal{A}})$ to obtain a word $W_{\mathcal{A}}$ and computes $X = g^x$.
5. Erase all state except $(esk_{\mathcal{A}}, W_{\mathcal{A}}, X, t_{\mathcal{A}})$, sets $(W_{\mathcal{A}}, X, t_{\mathcal{A}})$ as the ephemeral public key and sends $(\widehat{B}, \widehat{A}, W_{\mathcal{A}}, X, t_{\mathcal{A}})$ to $\mathcal{B}$.

($\mathcal{B} \rightharpoonup \mathcal{A}$). Similarly, $\mathcal{B}$ executes the following steps.

1. Selects the ephemeral secret key $esk_{\mathcal{B}} \xleftarrow{\$} \{0,1\}^{u(\ell)}$ and picks $t_{\mathcal{B}} \xleftarrow{\$} \{0,1\}^{t_3(\ell)}$.
2. Sets $\widehat{lsk}_{\mathcal{B}} = \mathsf{Ext}_1(lsk_{\mathcal{B}}, r_{\mathcal{B}_1}), \widehat{esk}_{\mathcal{B}} = \mathsf{Ext}_2(esk_{\mathcal{B}}, r_{\mathcal{B}_2})$.
3. Computes $(w_{\mathcal{B}}, y) = \widehat{F}_{\widehat{lsk}_{\mathcal{B}}}(esk_{\mathcal{B}}) + \overline{F}_{\widehat{esk}_{\mathcal{B}}}(r_{\mathcal{B}_1})$.
4. Runs the algorithm $\mathsf{WordG}(w_{\mathcal{B}})$ to obtain a word $W_{\mathcal{B}}$ and computes $Y = g^y$.
5. Erase all state except $(esk_{\mathcal{B}}, W_{\mathcal{B}}, Y, t_{\mathcal{B}})$, sets $(W_{\mathcal{B}}, Y, t_{\mathcal{B}})$ as the ephemeral public key and sends $(\widehat{A}, \widehat{B}, W_{\mathcal{B}}, Y, t_{\mathcal{B}})$ to $\mathcal{A}$.

Session Key Output. When $\mathcal{A}$ receives $(\widehat{A}, \widehat{B}, W_{\mathcal{B}}, Y, t_{\mathcal{B}})$, $\mathcal{A}$ sets $\mathsf{sid} = (\widehat{A}, \widehat{B}, W_{\mathcal{A}}, X, t_{\mathcal{A}}, W_{\mathcal{B}}, Y, t_{\mathcal{B}})$ and computes the session key as follows.

1. Reconstructs $(w_{\mathcal{A}}, x)$ from $(lsk_{\mathcal{A}}, lpk_{\mathcal{A}}, esk_{\mathcal{A}})$, and computes $aux = H_1(\mathsf{sid})$.
2. Computes $K_{\mathcal{A}_1} = Y^x, K_{\mathcal{A}_2} = \mathsf{ProjHash}(lpk_{\mathcal{B}}, W_{\mathcal{A}}, w_{\mathcal{A}}, aux), K_{\mathcal{A}_3} = \mathsf{Hash}(lsk_{\mathcal{A}}, W_{\mathcal{B}}, aux)$.
3. Sets $s_{\mathcal{A}} = Ext_3(H_2(K_{\mathcal{A}_1}) \oplus K_{\mathcal{A}_2} \oplus K_{\mathcal{A}_3}, t_{\mathcal{A}} \oplus t_{\mathcal{B}})$.
4. Computes $SK_{\mathcal{A}} = \widetilde{F}_{s_{\mathcal{A}}}(\mathsf{sid})$.

Similarly, party $\mathcal{B}$ sets $\mathsf{sid} = (\widehat{A}, \widehat{B}, W_{\mathcal{A}}, X, t_{\mathcal{A}}, W_{\mathcal{B}}, Y, t_{\mathcal{B}})$ and then computes the session key as follows.

1. Reconstructs $(w_{\mathcal{B}}, y)$ from $(lsk_{\mathcal{B}}, lpk_{\mathcal{B}}, esk_{\mathcal{B}})$ and computes $aux = H_1(\mathsf{sid})$.
2. Computes $K_{\mathcal{B}_1} = X^y, K_{\mathcal{B}_2} = \mathsf{Hash}(lsk_{\mathcal{B}}, W_{\mathcal{A}}, aux), K_{\mathcal{B}_3} = \mathsf{ProjHash}\ (lpk_{\mathcal{A}}, W_{\mathcal{B}}, w_{\mathcal{B}}, aux)$.
3. Sets $s_{\mathcal{B}} = Ext_3(H_2(K_{\mathcal{B}_1}) \oplus K_{\mathcal{B}_2} \oplus K_{\mathcal{B}_3}, t_{\mathcal{A}} \oplus t_{\mathcal{B}})$.

4. Computes $SK_{\mathcal{B}} = \widetilde{F}_{s_{\mathcal{B}}}(\mathsf{sid})$.

Correctness Analysis. One can note that $K_{\mathcal{A}_1} = K_{\mathcal{B}_1}$ as $K_{\mathcal{A}_1} = Y^x = X^y = K_{\mathcal{B}_1} = g^{xy}$. Due to the property of SPHF, we have

$$K_{\mathcal{A}_2} = \mathsf{ProjHash}(lpk_{\mathcal{B}}, W_{\mathcal{A}}, w_{\mathcal{A}}, aux) = \mathsf{Hash}(lsk_{\mathcal{B}}, W_{\mathcal{A}}, aux) = K_{\mathcal{B}_2},$$

$$K_{\mathcal{A}_3} = \mathsf{Hash}(lsk_{\mathcal{A}}, W_{\mathcal{B}}, aux) = \mathsf{ProjHash}(lpk_{\mathcal{A}}, W_{\mathcal{B}}, w_{\mathcal{B}}, aux) = K_{\mathcal{B}_3}.$$

Therefore, we can obtain that $s_{\mathcal{A}} = Ext_3(H_2(K_{\mathcal{A}_1}) \oplus K_{\mathcal{A}_2} \oplus K_{\mathcal{A}_3}, t_{\mathcal{A}} \oplus t_{\mathcal{B}}) = s_{\mathcal{B}} = Ext_3(H_2(K_{\mathcal{B}_1}) \oplus K_{\mathcal{B}_2} \oplus K_{\mathcal{B}_3}, t_{\mathcal{A}} \oplus t_{\mathcal{B}})$, which guarantees that $SK_{\mathcal{A}} = SK_{\mathcal{B}}$.

7.3.3 Security Analysis

Theorem 7.1 *The AKE protocol following the general framework is* (λ_1, λ_2)-CLR-eCK-*secure if the underlying smooth projective hash function is 2-*smooth*, the DDH assumption holds in* $\mathbb{G}$*,* H_1, H_2 *are collision-resistant hash functions,* $\widehat{\mathsf{F}}$ *and* $\overline{\mathsf{F}}$ *are PRF families and* $\widetilde{\mathsf{F}}$ *is a* πPRF *family. Here* $\lambda_1 \leq \min\{|\mathcal{HK}| - 2\log(1/\epsilon_1) - l_1(\ell), |\mathcal{Y}| - 2\log(1/\epsilon_3) - l_3(\ell)\}, \lambda_2 \leq u(\ell) - 2\log(1/\epsilon_2) - l_2(\ell)$.

Proof: Let session $\mathsf{sid}^* = (\widehat{A}, \widehat{B}, W^*_{\mathcal{A}}, X^*, t^*_{\mathcal{A}}, W^*_{\mathcal{B}}, Y^*, t^*_{\mathcal{B}})$ be the target session chosen by adversary $\mathcal{M}$. $\mathcal{A}$ is the owner of the session sid^* and $\mathcal{B}$ is the peer. We then analyze the security of the AKE protocol in the following two disjoint cases.

Case I. *There exists a matching session,* $\overline{\mathsf{sid}^*}$*, of the target session* sid^*.

we analyse the security based on the type of the reveal query and leakage query that the adversary issues to the target session, the matching session and the corresponding parties.

a). $\mathsf{LongTermKeyReveal}(\mathcal{A}), \mathsf{LongTermKeyReveal}(\mathcal{B}), \mathsf{EphemeralKeyLeakage}(\mathsf{sid}^*)$, $\mathsf{EphemeralKeyLeakage}(\overline{\mathsf{sid}^*})$. In this sub-case, suppose that the adversary obtains at most λ_2-bits of the ephemeral secret key of target session sid^*, we have that

$$\widehat{esk}^*_{\mathcal{A}} = \mathsf{Ext}_2(esk^*_{\mathcal{A}}, r_{\mathcal{A}_2}) \stackrel{s}{\equiv}_{\epsilon_2} \widehat{esk}'_{\mathcal{A}} \stackrel{\$}{\leftarrow} \{0,1\}^{l_2(\ell)}, \tag{7.1}$$

Therefore, $(w^*_{\mathcal{A}}, x^*) = \widehat{F}_{\widehat{lsk}_{\mathcal{A}}}(esk^*_{\mathcal{A}}) + \overline{F}_{\widehat{esk}^*_{\mathcal{A}}}(r_{\mathcal{A}_1}) \stackrel{c}{\equiv} (w'_{\mathcal{A}}, x') \stackrel{\$}{\leftarrow} \mathcal{W} \times \mathbb{Z}_p$. Similarly, suppose that the adversary obtains at most λ_2-bits of the ephemeral secret key of matching session $\overline{\mathsf{sid}^*}$, we have that

$$\widehat{esk}^*_{\mathcal{B}} = \mathsf{Ext}_2(esk^*_{\mathcal{B}}, r_{\mathcal{B}_2}) \stackrel{s}{\equiv}_{\epsilon_2} \widehat{esk}'_{\mathcal{B}} \stackrel{\$}{\leftarrow} \{0,1\}^{l_2(\ell)}, \tag{7.2}$$

and thus $(w^*_{\mathcal{B}}, y^*) = \widehat{F}_{\widehat{lsk}_{\mathcal{B}}}(esk^*_{\mathcal{B}}) + \overline{F}_{\widehat{esk}^*_{\mathcal{B}}}(r_{\mathcal{B}_1}) \stackrel{c}{\equiv} (w'_{\mathcal{B}}, y') \stackrel{\$}{\leftarrow} \mathcal{W} \times \mathbb{Z}_p$.

b). $\mathsf{EphemeralKeyReveal}(\mathsf{sid}^*), \mathsf{EphemeralKeyReveal}(\overline{\mathsf{sid}^*}), \mathsf{LongTermKeyLeakage}(\mathcal{A}),$ $\mathsf{LongTermKeyLeakage}(\mathcal{B})$. In this sub-case, suppose that the adversary obtains at most λ_1-bits of the long-term secret key of party $\mathcal{A}$, we have that

$$\widehat{lsk}^*_{\mathcal{A}} = \mathsf{Ext}_1(lsk_{\mathcal{A}}, r_{\mathcal{A}_1}) \overset{s}{\equiv}_{\epsilon_1} \widehat{lsk}'_{\mathcal{A}} \overset{\$}{\leftarrow} \{0,1\}^{l_1(\ell)}, \tag{7.3}$$

hence $(w^*_{\mathcal{A}}, x^*) = \widehat{F}_{\widehat{lsk}^*_{\mathcal{A}}}(esk^*_{\mathcal{A}}) + \overline{F}_{\widehat{esk}^*_{\mathcal{A}}}(r_{\mathcal{A}}) \overset{c}{\equiv} (w'_{\mathcal{A}}, x') \overset{\$}{\leftarrow} \mathcal{W} \times \mathbb{Z}_p$. Similarly, suppose that the adversary obtains at most λ_1-bits of the long-term secret key of party $\mathcal{B}$, we have that

$$\widehat{lsk}^*_{\mathcal{B}} = \mathsf{Ext}_1(lsk_{\mathcal{B}}, r_{\mathcal{B}_1}) \overset{s}{\equiv}_{\epsilon_1} \widehat{lsk}'_{\mathcal{B}} \overset{\$}{\leftarrow} \{0,1\}^{l_1(\ell)}, \tag{7.4}$$

and therefore $(w^*_{\mathcal{B}}, y^*) = \widehat{F}_{\widehat{lsk}^*_{\mathcal{B}}}(esk^*_{\mathcal{B}}) + \overline{F}_{\widehat{esk}^*_{\mathcal{B}}}(r_{\mathcal{B}_1}) \overset{c}{\equiv} (w'_{\mathcal{B}}, y') \overset{\$}{\leftarrow} \mathcal{W} \times \mathbb{Z}_p$.

c). $\mathsf{LongTermKeyReveal}(\mathcal{A}), \mathsf{EphemeralKeyReveal}(\overline{\mathsf{sid}^*}), \mathsf{EphemeralKeyLeakage}(\mathsf{sid}^*),$ $\mathsf{LongTermKeyLeakage}(\mathcal{B})$. In this sub-case, suppose that the adversary obtains at most λ_2-bits of the ephemeral secret key of target session sid^*, at most λ_1-bits of the long-term secret key of party $\mathcal{B}$, then based on the Equation (7.1),(7.4), we have that $(w^*_{\mathcal{A}}, x^*) = \widehat{F}_{\widehat{lsk}^*_{\mathcal{A}}}(esk^*_{\mathcal{A}}) + \overline{F}_{\widehat{esk}^*_{\mathcal{A}}}(r_{\mathcal{A}_1}) \overset{c}{\equiv} (w'_{\mathcal{A}}, x') \overset{\$}{\leftarrow}$ $\mathcal{W} \times \mathbb{Z}_p$ and $(w^*_{\mathcal{B}}, y^*) = \widehat{F}_{\widehat{lsk}^*_{\mathcal{B}}}(esk^*_{\mathcal{B}}) + \overline{F}_{\widehat{esk}^*_{\mathcal{B}}}(r_{\mathcal{B}_1}) \overset{c}{\equiv} (w'_{\mathcal{B}}, y') \overset{\$}{\leftarrow} \mathcal{W} \times \mathbb{Z}_p$.

d). $\mathsf{EphemeralKeyReveal}(\mathsf{sid}^*), \mathsf{LongTermKeyReveal}(\mathcal{B}), \mathsf{LongTermKeyLeakage}(\mathcal{A}),$ $\mathsf{EphemeralKeyLeakage}(\overline{\mathsf{sid}^*})$. In this sub-case, suppose that the adversary obtains at most λ_1-bits of the long-term secret key of party $\mathcal{A}$, at most λ_2-bits of the ephemeral secret key of matching session $\overline{\mathsf{sid}^*}$, then based on Equation (7.2),(7.3), we have that $(w^*_{\mathcal{A}}, x^*) = \widehat{F}_{\widehat{lsk}^*_{\mathcal{A}}}(esk^*_{\mathcal{A}}) + \overline{F}_{\widehat{esk}^*_{\mathcal{A}}}(r_{\mathcal{A}_1}) \overset{c}{\equiv} (w'_{\mathcal{A}}, x') \overset{\$}{\leftarrow}$ $\mathcal{W} \times \mathbb{Z}_p$ and $(w^*_{\mathcal{B}}, y^*) = \widehat{F}_{\widehat{lsk}^*_{\mathcal{B}}}(esk^*_{\mathcal{B}}) + \overline{F}_{\widehat{esk}^*_{\mathcal{B}}}(r_{\mathcal{B}_1}) \overset{c}{\equiv} (w'_{\mathcal{B}}, y') \overset{\$}{\leftarrow} \mathcal{W} \times \mathbb{Z}_p$.

Therefore, regardless of the type of the reveal query and leakage query, (x^*, y^*) are uniformly random elements in $\mathbb{Z}_p^2$ from the view of adversary $\mathcal{M}$. Therefore, $K^*_{\mathcal{A}_1} = K^*_{\mathcal{B}_1} = g^{x^*y^*}$ is computationally indistinguishable from a random element in $\mathbb{G}$ according to the DDH assumption and hence $H_2(K^*_{\mathcal{A}_1})$ is a uniform random string from the view of $\mathcal{M}$ who is given $X^* = g^{x^*}, Y^* = g^{y^*}$. We then have that the seed $s^*_{\mathcal{A}}$ for the πPRF function is uniformly distributed and unknown to the adversary and thus the derived session key $SK^*_{\mathcal{A}}$ is computationally indistinguishable from a random string. It is worth noting that in this case we only require $\tilde{F}$ to be a normal PRF.

Case II. *There exists no matching session of the test session* sid^*.

In this case, the adversary cannot issue $\mathsf{LongTermKeyReveal}$ query to reveal the long-term secret key of $\mathcal{B}$ but may issues the leakage query $\mathsf{LongTermKeyLeakage}$ to

learn some bit-information of $lsk_{\mathcal{B}}$. We prove the security of the AKE protocol as follows.

In the simulation, we modify the security game via the following steps to obtain two new games.

Game 1: Replace $K^*_{\mathcal{A}_2} = \mathsf{ProjHash}(lpk_{\mathcal{B}}, W^*_{\mathcal{A}}, w^*_{\mathcal{A}}, aux^*)$ by $K^*_{\mathcal{A}_2} = \mathsf{Hash}(lsk_{\mathcal{B}}, W^*_{\mathcal{A}}, aux^*)$.

Game 2: Choose $W^*_{\mathcal{A}} \in \mathcal{X} \setminus \mathcal{L}$ instead of deriving it from $\mathcal{L}$ through the algorithm WordG.

We can see that Game 1 is identical to the original game from the view of adversary $\mathcal{M}$ due to the fact that $\mathsf{ProjHash}(lpk_{\mathcal{B}}, W^*_{\mathcal{A}}, w^*_{\mathcal{A}}) = \mathsf{Hash}(lsk_{\mathcal{B}}, W^*_{\mathcal{A}})$, and Game 2 is indistinguishable from Game 1 (and hence also the original game) due to the difficulty of the subset membership problem which ensures that the distribution of $\mathcal{X} \setminus \mathcal{L}$ is indistinguishable from $\mathcal{L}$.

Note that adversary $\mathcal{M}$ may actives a session sid, which is not matching to session sid^*, with $\mathcal{B}$. Precisely, $\mathcal{M}$ can choose $W \in \mathcal{X} \setminus \mathcal{L}$ (e.g., by replaying $W^*_{\mathcal{A}}$), send W to $\mathcal{B}$ and issues $\mathsf{SessionKeyReveal}(\mathsf{sid})$ query to learn the shared key. According to the property of 2-smooth of the underlying smooth projective hash function, we have that $K^*_{\mathcal{A}_2}$ is pairwisely independent from any other such key (denoted by $\widetilde{K}$) and all public information (i.e., $\mathsf{param}, \mathcal{L}, lpk_{\mathcal{B}}, W^*_{\mathcal{A}}, aux^*$) and hence

$$\widetilde{\mathrm{H}}_\infty(K^*_{\mathcal{A}_2} | \widetilde{K}, \mathsf{param}, \mathcal{L}, lpk_{\mathcal{B}}, W^*_{\mathcal{A}}, aux^*) = |\mathcal{Y}|.$$

Suppose that the leakage of $lsk_{\mathcal{B}}$ is at most λ_1-bits (denoted by $\widetilde{lsk_{\mathcal{B}}}$), and therefore (see *Lemma 1*)

$$\begin{aligned} \widetilde{\mathrm{H}}_\infty(K^*_{\mathcal{A}_2} | \widetilde{K}, \mathsf{param}, \mathcal{L}, lpk_{\mathcal{B}}, W^*_{\mathcal{A}}, aux^*, \widetilde{lsk_{\mathcal{B}}}) &\geq \widetilde{\mathrm{H}}_\infty(K^*_{\mathcal{A}_2} | \widetilde{K}, \mathsf{param}, \mathcal{L}, lpk_{\mathcal{B}}, W^*_{\mathcal{A}}, aux^*) - \lambda_1 \\ &= |\mathcal{Y}| - \lambda_1. \end{aligned}$$

Therefore, by using the strong extractor Ext_3, it holds that

$$s^*_{\mathcal{A}} = \mathsf{Ext}_3(H_2(K_{\mathcal{A}_1})^* \oplus K^*_{\mathcal{A}_2} \oplus K^*_{\mathcal{A}_3}, t^*_{\mathcal{A}} \oplus t^*_{\mathcal{B}}) \stackrel{s}{\equiv}_{\epsilon_3} s'_{\mathcal{A}} \stackrel{\$}{\leftarrow} \{0,1\}^{l_3(\ell)}.$$

One can see that $\mathcal{A}$ obtains a variable $s^*_{\mathcal{A}}$ which is pairwisely independent from any other such variables and thus the derived session key $SK^*_{\mathcal{A}}$ is computationally indistinguishable from a truly random element from $\mathcal{M}$'s view due to the application of πPRF, which completes the proof.

Simulation for Non-test Session. Note that for the two cases above, we have to simulate the non-test session correctly with the adversary. Specifically, when adversary $\mathcal{M}$ activates a non-test session with $\mathcal{A}$ or $\mathcal{B}$, the session execution simulated should be identical to the session run by $\mathcal{A}$ or $\mathcal{B}$ from the view of $\mathcal{M}$. One can

note that this can be easily guaranteed when the query $\mathsf{LongTermKeyReveal}(\mathcal{A})$ or $\mathsf{LongTermKeyReveal}(\mathcal{B})$ is issued in the game. Since we know the long-term secret key of $\mathcal{A}$ or $\mathcal{B}$, we can just select an ephemeral secret key and compute the ephemeral public key correctly by using the long-term secret key and long-term public key. Nevertheless, if the query $\mathsf{LongTermKeyReveal}(\mathcal{A})$ or $\mathsf{LongTermKeyReveal}(\mathcal{B})$ is not issued, that is, without the long-term secret key of $\mathcal{A}$ or $\mathcal{B}$, the simulation of the non-test session owned by $\mathcal{A}$ or $\mathcal{B}$ can no longer be simulated as shown above. In this case, we simulate the session as follows. Suppose that we are to simulate the session owned by $\mathcal{A}$ without knowing $lsk_{\mathcal{A}}$, we pick $(r_1, r_2) \xleftarrow{\$} \mathcal{W} \times \mathbb{Z}_p$ and then compute $W_{\mathcal{A}} = \mathsf{WordG}(r_1), X = g^{r_2}$. We say that the session simulated in this way can be identical to the real session from $\mathcal{M}$'s view due to the pseudo-randomness of the PRF. To be more precise, even when $\mathcal{M}$ obtains at most λ_1-bits of $lsk_{\mathcal{A}}$ through $\mathsf{LongTermKeyLeakage}(\mathcal{A})$, the variable $\widehat{lsk}_{\mathcal{A}}$, which comes from $\mathsf{Ext}_1(lsk_{\mathcal{A}}, r_{\mathcal{A}})$ and inputs to the pseudo-random function $\widehat{F}$, still remains unknown to adversary $\mathcal{M}$. Therefore, the value of $\widehat{F}_{\widehat{lsk}_{\mathcal{A}}}(esk_{\mathcal{A}})$ is computationally indistinguishable from a random element.

7.4 An Instantiation from DDH Assumption

In this section, we first introduce an SPHF based on the DDH assumption and then show how to construct a CLR-eCK-secure AKE protocol based on this function.

7.4.1 DDH-based SPHF

In the following, we present the language we use in the instantiation of our generic CLR-eCK-secure AKE protocol. Specifically, we introduce the Diffie-Hellman language $\mathcal{L}_{\mathsf{DH}}$ and show how to construct a 2-smooth SPHF on $\mathcal{L}_{\mathsf{DH}}$.

Diffie-Hellman Language. Let $\mathbb{G}$ be a group of primer order p and $g_1, g_2 \in \mathbb{G}$. The Diffie-Hellman Language is as follows.

$$\mathcal{L}_{\mathsf{DH}} = \{(u_1, u_2) | \exists r \in \mathbb{Z}_p, s.t., u_1 = g_1^r, u_2 = g_2^r\}$$

One can see that the witness space of $\mathcal{L}_{\mathsf{DH}}$ is $\mathcal{W} = \mathbb{Z}_p$ and $\mathcal{L}_{\mathsf{DH}} \subset \mathcal{X} = \mathbb{G}^2$. We have the following theorems.

Theorem 7.2 *The subset membership problem over $\mathcal{X} = \mathbb{G}^2$ and $\mathcal{L}_{\mathsf{DH}}$ is hard.*

Proof: One can easily obtain the theorem above from the DDH assumption and hence we omit the proof here. Actually, if an adversary can distinguish a word randomly picked from $\mathcal{L}_{\mathsf{DH}}$ from a random element chosen from $\mathcal{X} \backslash \mathcal{L}_{\mathsf{DH}}$, we can build a distinguisher for the DDH problem by using the adversary as a subroutine.

SPHF on $\mathcal{L}_{\mathsf{DH}}$. Here we show how to construct a 2-smooth SPHF (denoted by $\mathcal{SPHF}_{\mathsf{DH}}$) over the language $\mathcal{L}_{\mathsf{DH}} \subset \mathcal{X} = \mathbb{G}^2$ onto the group $\mathcal{Y} = \mathbb{G}$. Let $H_1 : \{0,1\}^* \to \mathbb{Z}_p$ denote a collision-resistant hash function. The concrete construction is as follows.

$\mathsf{SPHFSetup}(1^\ell)$: $\mathsf{param} = (\mathbb{G}, p, g_1, g_2)$;

HashKG: $\mathsf{hk} = (\alpha_1, \alpha_2, \beta_1, \beta_2) \xleftarrow{\$} \mathbb{Z}_p^4$;

$\mathsf{ProjKG}(\mathsf{hk})$: $\mathsf{hp} = (\mathsf{hp}_1, \mathsf{hp}_2) = (g_1^{\alpha_1} g_2^{\alpha_2}, g_1^{\beta_1} g_2^{\beta_2}) \in \mathbb{G}_p^2$;

$\mathsf{WordG}(\mathsf{hk}, w = r)$: $W = (g_1^r, g_2^r)$;

$\mathsf{Hash}(\mathsf{hk}, W = (u_1, u_2) = (g_1^r, g_2^r), aux = d = H_1(W, aux'))$: $\mathsf{hv} = u_1^{\alpha_1 + d\beta_1} u_2^{\alpha_2 + d\beta_2}$;

$\mathsf{ProjHash}(\mathsf{hp}, W = (u_1, u_2) = (g_1^r, g_2^r), w = r, aux = d = H_1(W, aux'))$: $\mathsf{hv}' = \mathsf{hp}_1^r \mathsf{hp}_2^{dr}$.

Note that $\mathcal{Y} = \mathbb{G}, \mathcal{HK} = \mathbb{Z}_p^4, \mathcal{HP} = \mathbb{G}_p^2, \mathcal{AUX} = \mathbb{Z}_p, \mathcal{W} = \mathbb{Z}_p$. Then we have the following theorem.

Theorem 7.3 *$\mathcal{SPHF}_{\mathsf{DH}}$ is a 2-smooth SPHF.*

Proof: We show that $\mathcal{SPHF}_{\mathsf{DH}}$ is projective and smooth (2-smooth).

Correctness. With the above notations, for a word $W = (u_1, u_2) = (g_1^r, g_2^r)$ we have

$$\begin{aligned} & \mathsf{Hash}(\mathsf{hk}, (W, d) \\ = \; & u_1^{\alpha_1 + d\beta_1} u_2^{\alpha_2 + d\beta_2} \\ = \; & \mathsf{hp}_1^r \mathsf{hp}_2^{dr} \\ = \; & \mathsf{ProjHash}(\mathsf{hp}, (W, r, d). \end{aligned}$$

Smoothness (2-smooth). Suppose $g_2 = g_1^\theta$. Note that $\mathsf{hp}_1 = g_1^{\alpha_1} g_2^{\alpha}, \mathsf{hp}_2 = g_1^{\beta_1} g_2^{\beta_2}$ which constraints $(\alpha_1, \alpha_2, \beta_1, \beta_2)$ to satisfy

$$\log_{g_1} \mathsf{hp}_1 = \alpha_1 + \theta\alpha_2.$$

$$\log_{g_1} \mathsf{hp}_2 = \beta_1 + \theta\beta_2.$$

Let $W_1 = (g_1^{r_1}, g_2^{r_2}), W_2 = (g_1^{r_1'}, g_2^{r_2'}) \in \mathcal{X} \backslash \mathcal{L}_{\mathsf{DH}}$ where $r_1 \neq r_2, r_1' \neq r_2'$, suppose $aux_1 = d_1 = H_1(W_1, aux_1'), aux_2 = d_2 = H_1(W_2, aux_2')$, then the hash value hv_1 of W_1, hv_2 of W_2 are as follows,

$$\mathsf{hv}_1 = \mathsf{Hash}(\mathsf{hk}, W_1, aux_1) = g_1^{r_1(\alpha_1 + d_1\beta_1)} g_2^{r_2(\alpha_2 + d_1\beta_2)},$$

$$\mathsf{hv}_2 = \mathsf{Hash}(\mathsf{hk}, W_2, aux_2) = g_1^{r'_1(\alpha_1+d_2\beta_1)} g_2^{r'_2(\alpha_2+d_2\beta_2)},$$

which also constraint $(\alpha_1, \alpha_2, \beta_1, \beta_2)$ to satisfy

$$\log_{g_1} \mathsf{hv}_1 = r_1\alpha_1 + r_2\theta\alpha_2 + r_1d_1\beta_1 + r_2d_1\theta\beta_2. \tag{7.5}$$

$$\log_{g_1} \mathsf{hv}_2 = r'_1\alpha_1 + r'_2\theta\alpha_2 + r'_1d_2\beta_1 + r'_2d_2\theta\beta_2. \tag{7.6}$$

From the above equations, we have

$$(\alpha_1, \alpha_2, \beta_1, \beta_2) \cdot \mathbf{A} = (\log_{g_1} \mathsf{hp}_1, \log_{g_1} \mathsf{hp}_2, \log_{g_1} \mathsf{hv}_1, \log_{g_1} \mathsf{hv}_2),$$

where $\mathbf{A}$ is a matrix defined as

$$\mathbf{A} = \begin{bmatrix} 1 & \theta & 0 & 0 \\ 0 & 0 & 1 & \theta \\ r_1 & \theta r_2 & r_1d_1 & \theta r_2d_1 \\ r'_1 & \theta r'_2 & r'_1d_2 & \theta r'_2d_2 \end{bmatrix}.$$

Since $(W_1, aux_1) \neq (W_2, aux_2)$ where $aux_1 = d_1 = H_1(W_1, aux'_1), aux_2 = d_2 = H_1(W_2, aux'_2)$, we have that $d_1 \neq d_2$. Furthermore, as $\theta \neq 0$, $r_1 \neq r_2$ and $r'_1 \neq r'_2$, we can obtain that the determinant of $\mathbf{A}$ is $\theta^2 \cdot (r_2 - r_1) \cdot (r'_2 - r'_1) \cdot (d_2 - d_1) \neq 0$ and hence the equation (7.6) is independent of the equation (7.5). Therefore, we have that hv_2 is perfectly indistinguishable from any element randomly chosen from $\mathbb{G}$.

7.4.2 Concrete AKE Protocol

We then show a concrete AKE protocol based on $\mathcal{SPHF}_{\mathsf{DH}}$ in Table 7.2.

Protocol Description. In the system setup phase, let $\mathbb{G}$ be a group of primer order p and $g_1, g_2 \in \mathbb{G}$. For the $\mathcal{SPHF}_{\mathsf{DH}}$, we have that $\mathcal{Y} = \mathbb{G}, \mathcal{HK} = \mathbb{Z}_p^4, \mathcal{HP} = \mathbb{G}_p^2, \mathcal{AUX} = \mathbb{Z}_p, \mathcal{W} = \mathbb{Z}_p$. We then choose a collision-resistant hash functions $H_1 : \{0,1\}^* \to \mathbb{G}$. [e] We pick strong extractors as follows. Let $\mathsf{Ext}_1 : \mathbb{Z}_p^4 \times \{0,1\}^{t_1(\ell)} \to \{0,1\}^{l_1(\ell)}$ be average-case $(4 \cdot \log p - \lambda_1, \epsilon_1)$-strong extractor, $\mathsf{Ext}_2 : \{0,1\}^{u(\ell)} \times \{0,1\}^{t_2(\ell)} \to \{0,1\}^{l_2(\ell)}$ be average-case $(u(\ell) - \lambda_2, \epsilon_2)$-strong extractor and $\mathsf{Ext}_3 : \mathbb{G} \times \{0,1\}^{t_3(\ell)} \to \{0,1\}^{l_3(\ell)}$ be average-case $(\log p - \lambda_1, \epsilon_3)$-strong extractor. Choose $\widehat{F} \leftarrow \widehat{\mathsf{F}}^{k,\sum_{\widehat{\mathsf{F}}},\mathcal{D}_{\widehat{\mathsf{F}}},\mathcal{R}_{\widehat{\mathsf{F}}}}$, $\overline{F} \leftarrow \overline{\mathsf{F}}^{k,\sum_{\overline{\mathsf{F}}},\mathcal{D}_{\overline{\mathsf{F}}},\mathcal{R}_{\overline{\mathsf{F}}}}$ and $\widetilde{F} \leftarrow \widetilde{\mathsf{F}}^{k,\sum_{\widetilde{\mathsf{F}}},\mathcal{D}_{\widetilde{\mathsf{F}}},\mathcal{R}_{\widetilde{\mathsf{F}}}}$. The system parameter is $(\mathbb{G}, p, g_1, g_2, g, H_1, \mathsf{Ext}_1, \mathsf{Ext}_2, \mathsf{Ext}_3, \widehat{F}, \overline{F}, \widetilde{F})$.

For the long-term key generation, $\mathcal{A}$ chooses $(\alpha_1, \alpha_2, \beta_1, \beta_2) \xleftarrow{\$} \mathbb{Z}_p^4$ as its long-

[e] Note that in the concrete construction, H_2 is not needed as the hash value space $\mathcal{Y} = \mathbb{G}$.

Table 7.2: The Concrete CLR-eCK secure AKE Protocol

$\mathcal{A}$		$\mathcal{B}$
$\mathsf{hk} = (\alpha_1, \alpha_2, \beta_1, \beta_2) \xleftarrow{\$} \mathbb{Z}_p^4$,		$\mathsf{hk}' = (\alpha_1', \alpha_2', \beta_1', \beta_2') \xleftarrow{\$} \mathbb{Z}_p^4$,
$\mathsf{hp} = (\mathsf{hp}_1, \mathsf{hp}_2) = (g_1^{\alpha_1} g_2^{\alpha_2}, g_1^{\beta_1} g_2^{\beta_2}) \in \mathbb{G}_p^2$,		$\mathsf{hp}' = (\mathsf{hp}_1', \mathsf{hp}_2') = (g_1^{\alpha_1'} g_2^{\alpha_2'}, g_1^{\beta_1'} g_2^{\beta_2'}) \in \mathbb{G}_p^2$,
$r_1 \xleftarrow{\$} \{0,1\}^{t_1(\ell)}, r_2 \xleftarrow{\$} \{0,1\}^{t_2(\ell)}$,		$r_1' \xleftarrow{\$} \{0,1\}^{t_1(\ell)}, r_2' \xleftarrow{\$} \{0,1\}^{t_2(\ell)}$,
$lsk_{\mathcal{A}} = \mathsf{hk}, lpk_{\mathcal{A}} = (\mathsf{hp}, r_1, r_2)$.		$lsk_{\mathcal{B}} = \mathsf{hk}', lpk_{\mathcal{B}} = (\mathsf{hp}', r_1', r_2')$.
$e \xleftarrow{\$} \{0,1\}^{u(\ell)}, t \xleftarrow{\$} \{0,1\}^{t_3(\ell)}$,		$e' \xleftarrow{\$} \{0,1\}^{u(\ell)}, t' \xleftarrow{\$} \{0,1\}^{t_3(\ell)}$,
$\widehat{lsk}_{\mathcal{A}} = \mathsf{Ext}_1(lsk_{\mathcal{A}}, r_1)$,		$\widehat{lsk}_{\mathcal{B}} = \mathsf{Ext}_1(lsk_{\mathcal{B}}, r_1')$,
$\widehat{esk}_{\mathcal{A}} = \mathsf{Ext}_2(e, r_2)$,		$\widehat{esk}_{\mathcal{B}} = \mathsf{Ext}_2(e', r_2')$,
$(r, x) = \widehat{F}_{\widehat{lsk}_{\mathcal{A}}}(e) + \overline{F}_{\widehat{esk}_{\mathcal{A}}}(r_1)$,		$(r', y) = \widehat{F}_{\widehat{lsk}_{\mathcal{B}}}(e') + \overline{F}_{\widehat{esk}_{\mathcal{B}}}(r_1')$,
$W = (u_1, u_2) = (g_1^r, g_2^r), X = g^x$,		$W' = (u_1', u_2') = (g_1^{r'}, g_2^{r'}), Y = g^y$,
Erase all state except (e, W, X, t).		Erase all state except (e', W', Y, t').
	$\xrightarrow{(\widehat{B}, \widehat{A}, W, X, t)}$	
	$\xleftarrow{(\widehat{A}, \widehat{B}, W', Y, t')}$	
Set $\mathsf{sid} = (\widehat{A}, \widehat{B}, W, X, t, W', Y, t')$		Set $\mathsf{sid} = (\widehat{A}, \widehat{B}, W, X, t, W', Y, t')$
$d = H_1(\mathsf{sid}), K_{\mathcal{A}_1} = Y^x$,		$d = H_1(\mathsf{sid}), K_{\mathcal{A}_1} = X^y$,
$K_{\mathcal{A}_2} = \mathsf{hp}_1'^r \mathsf{hp}_2'^{dr}, K_{\mathcal{A}_3} = u_1'^{\alpha_1 + d\beta_1} u_2'^{\alpha_2 + d\beta_2}$,		$K_{\mathcal{B}_2} = u_1^{\alpha_1' + d\beta_1'} u_2^{\alpha_2' + d\beta_2'}, K_{\mathcal{A}_3} = \mathsf{hp}_1^{r'} \mathsf{hp}_2^{dr'}$,
$s_{\mathcal{A}} = Ext_3(K_{\mathcal{A}_1} \oplus K_{\mathcal{A}_2} \oplus K_{\mathcal{A}_3}, t_{\mathcal{A}} \oplus t_{\mathcal{B}})$,		$s_{\mathcal{B}} = Ext_3(K_{\mathcal{B}_1} \oplus K_{\mathcal{B}_2} \oplus K_{\mathcal{B}_3}, t_{\mathcal{A}} \oplus t_{\mathcal{B}})$,
$SK_{\mathcal{A}} = \widetilde{F}_{s_{\mathcal{A}}}(\mathsf{sid})$.		$SK_{\mathcal{B}} = \widetilde{F}_{s_{\mathcal{B}}}(\mathsf{sid})$.

term secret key, computes $(\mathsf{hp}_1, \mathsf{hp}_2) = (g_1^{\alpha_1} g_2^{\alpha_2}, g_1^{\beta_1} g_2^{\beta_2})$, picks $r_1 \xleftarrow{\$} \{0,1\}^{t_1(\ell)}, r_2 \xleftarrow{\$} \{0,1\}^{t_2(\ell)}$ and sets its long-term public key as $(\mathsf{hp}_1, \mathsf{hp}_2, r_1, r_2)$. Similarly, $\mathcal{B}$ sets its long-term secret/public key pair as $((\alpha_1', \alpha_2', \beta_1', \beta_2'), (\mathsf{hp}_1', \mathsf{hp}_2', r_1', r_2'))$.

After a session is activated, $\mathcal{A}$ picks an ephemeral secret key e and the extraction key $t \xleftarrow{\$} \{0,1\}^{t_3(\ell)}$, derives (r, x) using the secret keys and sends $(\widehat{B}, \widehat{A}, W = (u_1, u_2) = (g_1^r, g_2^r), X = g^x, t)$ to $\mathcal{B}$. Simultaneously, $\mathcal{B}$ executes the same procedure and returns $(\widehat{A}, \widehat{B}, W' = (u_1', u_2') = (g_1^{r'}, g_2^{r'}), Y = g^y, t')$ to $\mathcal{A}$.

To compute the shared session key, $\mathcal{A}$ runs the ProjHash algorithm to compute the hash value of W using the witness r and the long-term public key of $\mathcal{B}$, runs the Hash algorithm to compute the hash value of W' using its long-term secret key. $\mathcal{B}$ runs the Hash algorithm to compute the hash value of W using its long-term secret key, runs the Hash algorithm to compute the hash value of W' using the witness r' and the long-term public key of $\mathcal{A}$. Note that the auxiliary input to all the hash value computation is $d = H_1(\widehat{A}, \widehat{B}, W, X, t, W', Y, t')$. Both $\mathcal{A}$ and $\mathcal{B}$ also compute the value of g^{xy}. They then finally apply the πPRF function $\widetilde{F}$ to derive the session key.

Correctness. The correctness of the protocol can be easily obtained from the correctness of $\mathcal{SPHF}_{\mathsf{DH}}$. Precisely, $u_1^{\alpha_1' + d\beta_1'} u_2^{\alpha_2' + d\beta_2'} = \mathsf{hp}_1'^r \mathsf{hp}_2'^{dr}, u_1'^{\alpha_1 + d\beta_1} u_2'^{\alpha_2 + d\beta_2} = \mathsf{hp}_1^{r'} \mathsf{hp}_2^{dr'}, X^y = Y^x = g^{xy}$.

Based on **Theorem** 7.1, **Theorem** 7.2 and **Theorem** 7.3, we have the following result for the concrete AKE protocol.

Theorem 7.4 *The concrete AKE protocol is* (λ_1, λ_2)-CLR-eCK-*secure, where* $\lambda_1 \leq$

$\min\{4\log p - 2\log(1/\epsilon_1) - l_1(\ell), \log p - 2\log(1/\epsilon_3) - l_3(\ell)\}, \lambda_2 \leq u(\ell) - 2\log(1/\epsilon_2) - l_2(\ell)$.

7.5 Chapter Summary

In this chapter, we introduced a new leakage-resilient security model for AKE protocols to overcome the limitations in the previous models. Our model is the first to allow the adversary to obtain challenge-dependent leakage on both long-term and ephemeral secret keys, and hence are strong yet meaningful compared with the previous models. We also presented a generic framework to construct efficient one-round AKE protocol that is secure under the proposed security model, as well as an efficient instantiation of the general framework under the DDH assumption. Our framework ensures the session key are private and authentic even if the adversary learns a large fraction of both the long-term secret key and ephemeral secret key and provides qualitatively stronger privacy guarantees than existing AKE protocols constructed in prior and concurrent works, since such protocols necessarily become insecure if the adversary can perform leakage attacks during the execution of session.

Part III

Conclusion and Future Work

Chapter 8

Conclusion and Future Work

In this chapter, we summarize the work presented in this thesis and put forward several directions for further research.

8.1 Conclusion

8.1.1 Secure Data Storage in Cloud Computing

Data Auditing. We gave a formal treatment on Merkle Hash Tree for secure dynamic cloud auditing. We first revisited a well-known authentication structure named Merkle Hash Tree (MHT) and demonstrated how to extend its basic version to a sequence-enforced version that allows position checking. In order to support efficient and verifiable dynamic data operations, we further proposed a variant of MHT, named rank-based MHT (rMHT) that can be used to support verifiable dynamic data auditing. We also reviewed a cloud storage data auditing protocol named Oruta and showed that the protocol is vulnerable to replace and replay attacks. We then employed the proposed rMHT to fix the security problems in Oruta without sacrificing any desirable features of the protocol. It is of independent interest to find other security applications for rMHT.

Data Deduplication. We formalized a new primitive called Block-Level Message-Locked Encryption for DLSB-deduplication of large files to achieve space-efficient storage in cloud. We also presented a concrete BL-MLE scheme that can efficiently realize our design ideas. We showed that our proposed scheme can achieve significant savings in space and bandwidth. Moreover, we also showed that our BL-MLE scheme can be easily modified to achieve efficient data auditing, which makes our scheme multi-purpose for secure cloud storage.

8.1.2 Secure Data Retrieval in Cloud Computing

Data Searching. To overcome the inherent insecurity (under inside KGA) of the conventional PEKS system, we proposed two different solutions. The first solution is a new framework, named Dual-Server Public Key Encryption with Keyword Search (DSPEKS). A new Smooth Projective Hash Function (SPHF) is then introduced and used to construct a generic DS-PEKS scheme. We also showed an efficient instantiation of the new SPHF based on the Diffie-Hellman problem, which

results in an efficient DS-PEKS scheme without pairings. As the second solution, we provided a practical and applicable treatment on (inside) off-line KGA by formalizing a new PEKS system, namely Server-Aided Public Key Encryption with Keyword Search (SA-PEKS). We introduced a universal transformation from any PEKS scheme to a secure SA-PEKS scheme, along with the first instantiation of SA-PEKS. We also showed how to securely implement the client-KS protocol with a rate-limiting mechanism against on-line KGA. The experimental results showed that our proposed scheme achieves much better efficiency while providing resistance against both off-line and on-line KGAs.

Data Transmission. We introduced a new leakage-resilient security model for AKE protocols to overcome the limitations in the previous models. Our model is the first to allow the adversary to obtain challenge-dependent leakage on both long-term and ephemeral secret keys, and hence are strong yet meaningful compared with the previous models. We also presented a generic framework to construct efficient one-round AKE protocol that is secure under the proposed security model, as well as an efficient instantiation of the general framework under the DDH assumption. Our framework ensures the session key are private and authentic even if the adversary learns a large fraction of both the long-term secret key and ephemeral secret key and provides qualitatively stronger privacy guarantees than existing AKE protocols constructed in prior and concurrent works, since such protocols necessarily become insecure if the adversary can perform leakage attacks during the execution of session.

8.2 Future Work

We put forward the following directions for further research.

1. Regarding the data deduplication, we ask whether a fully randomized BL-MLE can be constructed for lock-dependent messages [ABM+13] to obtain stronger privacy. Secondly, our proposed scheme in Chapter 4 is proven secure in the random oracle model, we ask whether it is possible to design efficient BL-MLE schemes that are proven secure in the standard model. Thirdly, our proposed scheme uses public-key techniques in tag constructions and hence is less computation-efficient than the MLE schemes, we ask if there are other more efficient ways to construct BL-MLE schemes. Lastly, it is also an interesting research problem to design BL-MLE schemes supporting variable size data blocks.

2. For the purpose of helping the data owner enjoy fine-grained access control of data stored on untrusted cloud servers, a feasible solution would be encrypting data through certain cryptographic primitive(s), and disclosing decryption

keys only to authorized users. Unauthorized users, including cloud servers, are not able to do decryption since they do not have the data decryption keys. Moreover, the new cryptographic primitive(s) needs to be able to support dynamic requests so that data owners can add or revoke access privileges to other users. Therefore, one critical issue with this branch of approaches is how to achieve the desired security goals outlined above without introducing high complexity on computation, privilege revocation and key management.

3. In terms of secure data transmission, we leave the construction of efficient AKE protocols that are secure under stronger leakage setting as the future work. More precisely, noting that the intermediate value generated (not the ephemeral secret key) during the execution of an AKE protocol might also be leaked, we ask whether a fully leakage-resilient AKE protocol can be constructed.

4. It is of independent interest to find other security applications of our proposed rMHT and LH-SPHF. We leave the constructions of new auditing protocols based on rMHT and new cryptographic schemes based on LH-SPHF as the future work.

Bibliography

[ABB+13] Michel Abdalla, Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, and David Pointcheval. Sphf-friendly non-interactive commitments. In *ASIACRYPT*, pages 214–234, 2013. 20

[ABC+02] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger Wattenhofer. FARSITE: federated, available, and reliable storage for an incompletely trusted environment. In *OSDI*, 2002. 53

[ABC+05] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. In *CRYPTO*, pages 205–222, 2005. 82, 83

[ABC+07] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proc. ACM Conf. Comput. Commun. Security*, pages 598–609, 2007. 24, 25, 26, 49, 72, 73

[ABM+13] Martín Abadi, Dan Boneh, Ilya Mironov, Ananth Raghunathan, and Gil Segev. Message-locked encryption for lock-dependent messages. In *CRYPTO*, pages 374–391, 2013. 57, 61, 65, 77, 154

[ABS14] Janaka Alawatugoda, Colin Boyd, and Douglas Stebila. Continuous after-the-fact leakage-resilient key exchange. In *ACISP*, pages 258–273, 2014. 130

[ACP09] Michel Abdalla, Céline Chevalier, and David Pointcheval. Smooth projective hashing for conditionally extractable commitments. In *CRYPTO*, pages 671–689, 2009. 89

[ADPMT08] Giuseppe Ateniese, Roberto Di Pietro, Luigi V. Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *Proc. 4th Conf. Security and Privacy in Commun. Netw.*, pages 9:1–9:10, 2008. 28

[ADR02] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars R. Knudsen, editor, *Proceedings: Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 83–107, Amsterdam, The Netherlands, April 28 - May 2 2002. Springer. 16

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009. 128, 129, 132, 133, 134

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009. 6, 128, 131, 133, 134

[AKK09] Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of storage from homomorphic identification protocols. In *ASIACRYPT*, pages 319–333, 2009. 24, 72

[AKSX04] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order-preserving encryption for numeric data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 563–574, 2004. 80

[ASB14] Janaka Alawatugoda, Douglas Stebila, and Colin Boyd. Modelling after-the-fact leakage for key exchange. In *ASIACCS*, pages 207–216, 2014. 128, 129, 130

[BBC+13a] Fabrice Ben Hamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. Efficient uc-secure authenticated key-exchange for algebraic languages. In *Public-Key Cryptography - PKC 2013*, pages 272–291, 2013. 89

[BBC+13b] Fabrice Ben Hamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New smooth projective hash functions and one-round authenticated key exchange. *IACR Cryptology ePrint Archive*, 2013:34, 2013. 89

[BBC+13c] Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New techniques for sphfs and efficient one-round PAKE protocols. In *CRYPTO*, pages 449–475, 2013. 20

[BBO07] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO*, pages 535–552, 2007. 4, 53

[BCH12] Nir Bitansky, Ran Canetti, and Shai Halevi. Leakage-tolerant interactive protocols. In *TCC*, pages 266–284, 2012. 130

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Proceedings: Advances in Cryptology - CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15, Santa Barbara, California, USA, August 18-22 1996. Springer. 12

[BCK98] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *ACM Symposium on the Theory of Computing*, pages 419–428, 1998. 133

[BCOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004. 4, 18, 53, 80, 82, 83, 84, 101, 102, 106, 116, 117, 123, 124, 125

[BGPS07] Lakshmi N. Bairavasundaram, Garth R. Goodson, Shankar Pasupathy, and Jiri Schindler. An analysis of latent sector errors in disk drives. In *SIGMETRICS*, pages 289–300, 2007. 2, 4

[BH11] Ayad F. Barsoum and M. Anwar Hasan. On verifying dynamic multiple data copies over cloud servers. Cryptology ePrint Archive, Report 2011/447, 2011. `http://eprint.iacr.org/`. 27

[Bit] Bitcasa. `http://www.bitcasa.com/`. 3, 4, 25

[BJO09] Kevin D. Bowers, Ari Juels, and Alina Oprea. HAIL: a high-availability and integrity layer for cloud storage. In *Proc. ACM Conf. Comput. Commun. Security*, pages 187–198, 2009. 24

[BKR13] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In *EUROCRYPT*, pages 296–312, 2013. 53, 55, 56, 57, 60, 61, 74

[BNPS03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-rsa-inversion problems and the secu-

rity of chaum's blind signature scheme. *J. Cryptology*, 16(3):185–215, 2003. 116, 117

[Bon98] Dan Boneh. The decision diflie-hellman problem. In Joe P. Buhler, editor, *Proceedings: Algorithmic Number Theory - ANT 1998*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63, Portland, Oregon, USA, June 21-25 1998. Springer. 11, 12

[BPV] Olivier Blazy, David Pointcheval, and Damien Vergnaud. Round-optimal privacy-preserving protocols with smooth projective hash functions. In *TCC, pages = 94–111, year = 2012,*. 89

[BR93a] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *CRYPTO*, pages 232–249, 1993. 133

[BR93b] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *Proceedings: ACM conference on Computer and communications security - CCS 1993*, pages 62–73, Fairfax, VA, USA, November 3-5 1993. ACM. 12

[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, Third VLDB Workshop, SDM*, pages 75–83, 2006. 83

[BSS06] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. On the integration of public key data encryption and public key encryption with keyword search. In *Information Security ISC*, pages 217–232, 2006. 83

[BSS08] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Public key encryption with keyword search revisited. In *Computational Science and Its Applications - ICCSA*, pages 1249–1259, 2008. 83, 84, 113

[BSW13] Elette Boyle, Gil Segev, and Daniel Wichs. Fully leakage-resilient signatures. *J. Cryptology*, 26(3):513–558, 2013. 128

[CBH05] Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. Examining indistinguishability-based proof models for key establishment protocols. In *ASIACRYPT*, pages 585–604, 2005. 133

[CDRW10] Sherman S. M. Chow, Yevgeniy Dodis, Yannis Rouselakis, and Brent Waters. Practical leakage-resilient identity-based encryption from simple assumptions. In *CCS*, pages 152–161, 2010. 128

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In Jeffrey Scott Vitter, editor, *Proceedings: ACM Symposium on the Theory of Computing - STOC 1998*, pages 209–218, Dallas, Texas, USA, May 23-26 1998. ACM. 13

[CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006*, pages 79–88, 2006. 80

[Cha83] David Chaum. Blind signature system. In *CRYPTO*, page 153, 1983. 17

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, pages 453–474, 2001. 6, 128, 133, 135

[CKW13] David Cash, Alptekin Küpçü, and Daniel Wichs. Dynamic proofs of retrievability via oblivious RAM. In *EUROCRYPT*, pages 279–295, 2013. 29

[CLW06] Giovanni Di Crescenzo, Richard J. Lipton, and Shabsi Walfish. Perfectly secure password protocols in the bounded retrieval model. In *TCC*, pages 225–244, 2006. 61

[CMY+15] Rongmao Chen, Yi Mu, Guomin Yang, Fuchun Guo, and Xiaofen Wang. A new general framework for secure public key encryption with keyword search. In *Information Security and Privacy - 20th Australasian Conference, ACISP*, pages 59–76, 2015. 106, 113, 124

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *Cryptography and Coding*, pages 360–363, 2001. 83

[Cre11] Cas Cremers. Examining indistinguishability-based security models for key exchange protocols: the case of ck, ck-hmqv, and eck. In *ASIACCS, 2011*, pages 80–91, 2011. 133

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT*, pages 45–64, 2002. 20, 140

[CS07] Giovanni Di Crescenzo and Vishal Saraswat. Public key encryption with searchable keywords based on jacobi symbols. In *INDOCRYPT*, pages 282–296, 2007. 83

[CW79] J.Lawrence Carter and Mark N. Wegman. Universalclasses of hash-functions. *Journal of Computer and System Sciences*, 18(2):143–154, April 1979. 12

[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. 53

[Ded] The pros and cons of file-level vs. block-level data deduplication technology. `http://searchdatabackup.techtarget.com/tip/The-pros-and-cons-of-file-level-vs-block-level-data-deduplication-` 51

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976. 11, 13, 15

[DHLW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Efficient public-key cryptography in the presence of key leakage. In *ASIACRYPT*, pages 613–631, 2010. 128, 129, 132, 134

[DKL09] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009. 133

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008. 19

[Dro] Dropbox. `http://www.dropbox.com/.` 2, 3, 4, 25

[Dzi06] Stefan Dziembowski. Intrusion-resilience via the bounded-storage model. In *TCC*, pages 207–224, 2006. 61

[EKPT09] Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. In *Proc. ACM Conf. Comput. Commun. Security*, pages 213–222, 2009. 29

[FOPS01] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. Rsa-oaep is secure under the rsa assumption. In Joe Kilian, editor, *Proccedings: Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 260–274, Santa Barbara, California, USA, August 19-23 2001. Springer. 12

[GD12] Gantz and D.Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. `http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf`, 2012. 3

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986. 20

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In *EUROCRYPT*, pages 524–543, 2003. 20, 21, 82

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988. 15, 16

[Goo] GoogleDrive. `http://drive.google.com/.` 2, 3, 4, 25

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, September 2008. 11

[HHPS11] Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Proofs of ownership in remote storage systems. In *Proc. ACM Conf. Comput. Commun. Security*, pages 491–500, 2011. 52, 54, 57, 74, 76, 77

[HK12] Shai Halevi and Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. *J. Cryptology*, 25(1):158–193, 2012. 20

[HL11] Shai Halevi and Huijia Lin. After-the-fact leakage in public-key encryption. In *TCC*, pages 107–124, 2011. 129, 130

[HPS10] 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. 52

[HSH+08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX Security Symposium*, pages 45–60, 2008. 6, 128, 133

[ISO] Entity authentication mechanisms-part3: Entity authentication using asymmetric techniques. *ISO/IEC IS 9789-3,*, 1993. 6, 128

[JJ07] Ari Juels and Burton S. Kaliski Jr. Pors: proofs of retrievability for large files. In *Proc. ACM Conf. Comput. Commun. Security*, pages 584–597, 2007. 24, 25, 26

[JKHL09] Ik Rae Jeong, Jeong Ok Kwon, Dowon Hong, and Dong Hoon Lee. Constructing PEKS schemes secure against keyword guessing attacks is possible? *Computer Communications*, 32(2):394–396, 2009. 84

[KBR13] Sriram Keelveedhi, Mihir Bellare, and Thomas Ristenpart. Dupless: Server-aided encryption for deduplicated storage. In *USENIX Security*, pages 179–194, 2013. 55, 120, 121, 122

[Kha06] Dalia Khader. Public key encryption with keyword search based on k-resilient IBE. In *Computational Science and Its Applications - ICCSA*, pages 298–308, 2006. 82, 83

[KK05] Vishal Kher and Yongdae Kim. Securing distributed storage: challenges, techniques, and systems. In *StorageSS*, pages 9–25, 2005. 2, 4

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007. 9, 12

[Kra03] Hugo Krawczyk. SIGMA: the 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike-protocols. In *CRYPTO*, pages 400–425, 2003. 6, 128

[KRS+03] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus: Scalable secure file sharing on untrusted storage. In *FAST*, pages 29–42, 2003. 25

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009. 130

[KV11] Jonathan Katz and Vinod Vaikuntanathan. Round-optimal password-based authenticated key exchange. In *TCC*, pages 293–310, 2011. 20

[LCY+14] Chang Liu, Jinjun Chen, Laurence T. Yang, Xuyun Zhang, Chi Yang, Rajiv Ranjan, and Kotagiri Ramamohanarao. Authorized public auditing of dynamic big data storage on cloud with efficient verifiable fine-grained updates. *IEEE Trans. on Parall. and Distrib. Syst.*, 25(9):2234–2244, 2014. 26, 28

[LKMS04] Jinyuan Li, Maxwell N. Krohn, David Mazires, and Dennis Shasha. Secure untrusted data repository (sundr). In *OSDI*, pages 121–136, 2004. 25

[LLM07] Brian A. LaMacchia, Kristin E. Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In *Provable Security*, pages 1–16, 2007. 133, 135

[LRY+14] Chang Liu, Rajiv Ranjan, Chi Yang, Xuyun Zhang, Lizhe Wang, and Jinjun Chen. Mur-dpa: Top-down levelled multi-replica merkle hash tree based secure public auditing for dynamic big data storage on cloud. *IACR Cryptology ePrint Archive*, 2014:391, 2014. 28

[Mar] R. Marvin. Google admits an android crypto prng flaw led to bitcoin heist (august 2013). `http://sdt.bz/64008`. 6, 128, 129

[MB12] Dutch T. Meyer and William J. Bolosky. A study of practical deduplication. *TOS*, 7(4):14, 2012. 4

[Mer87] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, pages 369–378, 1987. 29, 57

[Mil10] R Miller. Amazon addresses ec2 power outages. In *Data Center Knowledge*, volume 1, 2010. 2, 4

[MO] Daisuke Moriyama and Tatsuaki Okamoto. Leakage resilient eck-secure key exchange protocol without random oracles. In *ASIACCS*. 128, 129, 132, 134

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004. 133

[MVS00] Umesh Maheshwari, Radek Vingralek, and William Shapiro. How to build a trusted database system on untrusted storage. In *OSDI*, pages 135–150, 2000. 25

[MXZC14] Zhen Mo, Qingjun Xiao, Yian Zhou, and Shigang Chen. On deletion of outsourced data in cloud computing. In *2014 IEEE 7th International*

Conference on Cloud Computing, Anchorage, AK, USA, June 27 - July 2, 2014, pages 344–351, 2014. 28

[MZC12] Zhen Mo, Yian Zhou, and Shigang Chen. A dynamic proof of retrievability (por) scheme with o(logn) complexity. In *Proceedings of IEEE International Conference on Communications, ICC 2012, Ottawa, ON, Canada, June 10-15, 2012*, pages 912–916, 2012. 28

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009. 128, 129, 137

[NYMX14] Jianbing Ni, Yong Yu, Yi Mu, and Qi Xia. On the security of an efficient dynamic auditing protocol in cloud storage. *IEEE Trans. Parallel Distrib. Syst.*, 25(10):2760–2761, 2014. 27

[Odl85] Andrew M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, editors, *Proceedings: Advances in Cryptology - CRYPTO 1984*, volume 209 of *Lecture Notes in Computer Science*, pages 224–314, Paris, France, April 9-11 1985. Springer. 11

[Oka07] Tatsuaki Okamoto. Authenticated key exchange and key encapsulation in the standard model. In *ASIACRYPT*, pages 474–484, 2007. 20, 132

[Ope] Opendedup. `http://opendedup.org/`. 4

[OR07] Alina Oprea and Michael K. Reiter. Integrity checking in cryptographic file systems with constant trusted storage. In *Proc. 16th USENIX Security Symp.*, pages 183–198, 2007. 26

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In *Proc. ACM Conf. Comput. Commun. Security*, pages 196–205, 2001. 60

[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. 83

[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. 83, 101, 102

[RS92] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *Proceedings: Advances in Cryptology - CRYPTO 1991*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, California, USA, August 11-15 1992. Springer. 14

[RSK09] Hyun Sook Rhee, Willy Susilo, and Hyun-Jeong Kim. Secure searchable public key encryption scheme against keyword guessing attacks. *IEICE Electronic Express*, 6(5):237–243, 2009. 83

[SF] D. Shumow and N. Ferguson. On the possibility of a back door in the nist sp800-90 dual ec prng. `http://rump2007.cr.yp.to/15-shumow.pdf`. 6, 128, 129

[SGLM08] Mark W. Storer, Kevin M. Greenan, Darrell D. E. Long, and Ethan L. Miller. Secure data deduplication. In *StorageSS*, pages 1–10, 2008. 53

[SSP13] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical dynamic proofs of retrievability. In *Proc. ACM Conf. Comput. Commun. Security*, pages 325–336, 2013. 29

[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. 29

[SW08] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *ASIACRYPT*, pages 90–107, 2008. 24, 25, 26, 27, 72

[SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symp. Security and Privacy*, pages 44–55, 2000. 4, 53, 80

[WBDS04] Brent R. Waters, Dirk Balfanz, Glenn Durfee, and Diana K. Smetters. Building an encrypted and searchable audit log. In *NDSS*, 2004. 82, 83

[WCW+13] Cong Wang, Sherman S. M. Chow, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for secure cloud storage. *IEEE Trans. Comput.*, 62(2):362–375, 2013. 27

[WLL14] Boyang Wang, Baochun Li, and Hui Li. Oruta: Privacy-preserving public auditing for shared data in the cloud. *IEEE Trans. on Cloud Comput*, 2(1):43–56, 2014. 8, 28, 40, 43, 44, 45, 46, 47, 48, 49

[WW08] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe: the least-authority filesystem. In *StorageSS*, pages 21–26, 2008. 53

[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. Parall. and Distrib. Syst.*, 22(5):847–859, 2011. 26, 27, 28, 31, 48, 72

[XJWW13] Peng Xu, Hai Jin, Qianhong Wu, and Wei Wang. Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack. *IEEE Trans. Computers*, 62(11):2266–2277, 2013. 84, 101, 102, 103, 106, 123, 124, 125

[XZ14] Jia Xu and Jianying Zhou. Leakage resilient proofs of ownership in cloud storage, revisited. In *ACNS*, pages 97–115, 2014. 73

[YCN+10] Jinhui Yao, Shiping Chen, Surya Nepal, David Levy, and John Zic. Truststore: Making amazon s3 trustworthy with services composition. In *CCGRID*, pages 600–605, 2010. 2, 4

[YHG08] Wei-Chuen Yau, Swee-Huay Heng, and Bok-Min Goi. Off-line keyword guessing attacks on recent public key encryption with keyword search schemes. In *ATC*, pages 100–105, 2008. 83

[YMSW13] Guomin Yang, Yi Mu, Willy Susilo, and Duncan S. Wong. Leakage resilient authenticated key exchange secure in the auxiliary input model. In *ISPEC*, pages 204–217, 2013. 129, 134

[YNA+14] Yong Yu, Jianbing Ni, Man Ho Au, Hongyu Liu, Hua Wang, and Chunxiang Xu. Improved security of a dynamic remote data possession checking protocol for cloud storage. *Expert Syst. Appl.*, 41(17):7789–7796, 2014. 27

[YSPY10] Yu Yu, François-Xavier Standaert, Olivier Pereira, and Moti Yung. Practical leakage-resilient pseudorandom generators. In *CCS*, pages 141–151, 2010. 128

[YTHW10] Guomin Yang, Chik How Tan, Qiong Huang, and Duncan S. Wong. Probabilistic public key encryption with equality test. In *CT-RSA*, pages 119–131, 2010. 4, 53, 65

[YY13] Jiawei Yuan and Shucheng Yu. Proofs of retrievability with public verifiability and constant communication cost in cloud. In *SCC@ASIACCS*, pages 19–26, 2013. 24, 25

[YZYL14] Tsz Hon Yuen, Ye Zhang, Siu-Ming Yiu, and Joseph K. Liu. Identity-based encryption with post-challenge auxiliary inputs for secure cloud applications and sensor networks. In *ESORICS*, pages 130–147, 2014. 129, 130

[Zet] K. Zetter. How a crypto 'backdoor' pitted the tech world against the nsa. `http://www.wired.com/threatlevel/2013/09/nsa-backdoor/all/`. 6, 128, 129

[ZL12] Dimitrios Zissis and Dimitrios Lekkas. Addressing cloud computing security issues. *Future Generation Comp. Syst.*, 28(3):583–592, 2012. 2, 4