VERIFIABLE DELEGATED COMPUTATION ON OUTSOURCED DATA

APPROVED BY SUPERVISING COMMITTEE:

Shouhuai Xu, Ph.D., Chair

Giuseppe Ateniese, Ph.D.

Rajendra V. Boppana, Ph.D.

Kay A. Robbins, Ph.D.

Ravi Sandhu, Ph.D.

Accepted:

Dean, Graduate School

DEDICATION

This dissertation is dedicated to

my wife, Qi Li, the love of my life;

my parents, parents in law, brothers and sister, for their invaluable supports.

VERIFIABLE DELEGATED COMPUTATION ON OUTSOURCED DATA

by

QINGJI ZHENG, M. Sc.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
May 2014

ACKNOWLEDGEMENTS

It is my fortune and honor to be supervised by Dr. Shouhuai Xu, and I would like to express my special gratitude to him. Without his expertise and guidance, the dissertation cannot be accomplished. His enthusiasm on the research, sharp insights on the problems and wide range of interests and expertise inspired me not only how to approach a technical solution but also how to conduct research from scratch.

I would like to thank my collaborator, Dr. Gieseppe Ateniese, for his expert perspective and contributions to the dissertation. I must thank other committee members Dr. Rajendra V. Boppana, Dr. Kay A. Robbins and Dr. Ravi Sandhu for their helpful comments and suggestions on the dissertation.

I would like to thank my friend Xinwen Zhang for his various help, and friends at UTSA for accompanying me on the Ph.D. journey.

The research described in the dissertation was partly supported by Prof. Shouhuai Xu's NSF Grant # 1111925.

May 2014

VERIFIABLE DELEGATED COMPUTATION ON OUTSOURCED DATA

Qingji Zheng, Ph.D.
The University of Texas at San Antonio, 2014

Supervising Professor: Shouhuai Xu, Ph.D.

Cloud computing has been an outstanding computing paradigm in the literature. While promising, it also brings a range of security problems that must be adequately addressed. One class of security problem is related to untrusted cloud (i.e., cloud infrastructure vendors and cloud service providers) that might be compromised. While encrypting data outsourced to the cloud can be used to mitigate such threats, it is insufficient to tackle another difficult problem: How can cloud users trust the results which are the outputs of some computing tasks conducted by the clouds ("delegated computation") on the users' outsourced encrypted data? This problem leads to a general concept of "verifiability", which has not been investigated sufficiently. This dissertation addresses three problems in this domain, by presenting three families of provably-secure cryptographic protocols.

The first contribution deals with the problem: how to allow secure keyword search on outsourced encrypted data while complying with flexible access control policies, and assure that the cloud faithfully followed the search procedures? We introduces the solution concept of "verifiable attribute-based keyword search", which enables data owners to grant keyword search capability with respect to access control policies and data users to delegate keyword search to the cloud as long as their attributes satisfy the access control policies, and further allows data users to verify that the cloud faithfully executed the search operations.

The second contribution is to explore the problem: how can cloud users delegate the set intersection operation to the cloud on their outsourced encrypted data sets, and further verify the correctness of the intersection set returned from the cloud? We introduce the novel notion of "verifiable delegated set intersection on outsourced encrypted data", which is to delegate the set intersection operation to the cloud, while (i) not giving the decryption capability to the cloud, and

(ii) being able to hold the misbehaving cloud accountable.

The third contribution concentrates on the problem: How to achieve verifiable SQL queries on outsourced databases? We present an efficient solution to support various SQL queries that include selection, projection, join, and (weighted) aggregation queries? The solution is built on top of two building blocks: an efficient authenticated data structure to support dynamic update on outsourced databases, and newly devised homomorphic linear tag, which can efficiently verify the integrity of query results via aggregation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: INTRODUCTION

1.1 Motivation

The emergence of cloud computing enables both enterprises and individual users to use the on-demand computing resources at the affordable price. Despite various benefits, cloud computing inevitably brings in new security and privacy concerns, because the cloud (i.e., cloud infrastructure vendors or cloud service providers) might be malicious by birth or compromised, and therefore cannot be fully trusted. Cloud is not as promising as it claimed until the relevant security problems have been adequately addressed.

Specifically, cloud security problems can be classified into categories based on the trust models as follows.

- Trusted cloud model: The cloud is one who is assumed not to deviate from the protocols in question. It does not require encrypting the outsourced data and performing verification on any computation outcome.

- Semi-trusted cloud model(honest-but-curious): A semi-trusted cloud is one who follows the protocol with the exception that it keeps all its intermediate computations and attempts to infer something from the data. Whether to encrypt outsourced data depends on its sensitivity. For example, it needs no encryption for public records (e.g., government employees' salary), while it demands encryption for specific records of individuals (e.g., patients' healthcare records). There is no need to perform computation verification because the semi-trusted cloud service provider follows the protocol exactly.

- Untrusted cloud model: A malicious cloud is one who can deviate the protocol arbitrarily. Whether to encrypt outsourced data lies on its sensitivity, and it indeed needs to verify the correctness of any output from the cloud service provider since the protocol might be executed arbitrarily.

In this dissertation, we center on the security problems in the untrusted cloud model: How can cloud users assure that the cloud faithfully conducted the specified computation functions? In other words, can cloud users efficiently verify that the output of the functions has been honestly computed by the cloud on the right outsourced data?

To resolve this problem, verifiable delegated computation has been introduced and led to a large volume of prior works. Roughly these works can be classified into three categories. The first one is the architectural approach, which is to deploy trusted hardware to the server, e.g. secure processors [113,120,126] or Trusted Platform Modules (TPMs) [66], for building a secure and trusted computation environment. The second is the interactive probabilistically checkable approach (interactive proof) [51,60,86,110], where users actively challenge the server and receive the response to verify the correctness of the statement, initiated by the work about Interactive Proofs [10,59]. The last one is the non-interactive approach [16,38,46,50,53,64,83,89–92,94,97–99,101,106,125], where the server sends the proof to users with the computation output in the same round.

1.2 Dissertation Contributions

This dissertation focuses on verifiable delegated computation on outsourced data in cloud computing. More specifically, the dissertation contribution can be summarized into the three parts as follows.

1.2.1 Verifiable Attributed-based Keyword Search on Outsourced Encrypted Data

With the increasing adoption of cloud computing for data storage, assuring secure data service, in terms of data confidentiality and retrieval correctness, has been outstanding. While classical data encryption guarantees confidentiality but makes it impossible for authorized parties to search on encrypted data. Searchable encryption, on the other end, allows authorized parties to provide only specific records according to some search criteria, e.g., keywords. Unfortunately, these solutions are not suitable for the case where data owners wish to share data with users by restricting their search capability to comply with flexible access control policies. In addition, as the cloud providers might deceive users, for example, by returning partial search result or operating on partial, modified, or corrupted data, because of economic incentives or system corruption, we consider the retrieval correctness when authorized users retrieve data from the cloud. By considering these issues comprehensively, we investigate the solution of verifiable attribute-based keyword search for facilitating flexible access control on keyword search on encrypted data and supporting verification on search results. This work was presented at the 2014 IEEE INFOCOM [130] and will be presented in Chapter 2.

1.2.2 Verifiable Set Intersection on Outsourced Encrypted Data

Private set intersection protocol is an essential building block for many cryptographic applications, which enables data users to compute the set intersection without revealing the non-matching items. Cloud computing with massive storage and vast computational resource drives the desire to store the encrypted data in the cloud and also outsource such computations, which can sharply decrease the cost happening on data users. This leads to a general question: How can the cloud execute the delegated functions on outsourced encrypted data, without being given the decryption capability? This question is not trivial to be solved since the outsourced data are encrypted under different cloud users' public keys. In addition, How can cloud users verify that the cloud executed the delegated computational functions honestly? We therefore motivate and introduce the novel notion of Verifiable Delegated Set Intersection on outsourced encrypted data [128], which will be presented in Chapter 3.

1.2.3 Verifiable SQL Queries on Outsourced Dynamic Databases

As an outstanding application, third-party service providers deploy databases on top of cloud infrastructure and provide database-as-a-services, by enjoying overall benefits of cloud computing. However, outsourcing databases to the cloud also causes security concerns even for databases. In particular, for the database queirers, they need to verify the correct execution of database queries to avoid any deliberate or inadvertent misbehavior of the cloud, which is referred to query integrity. Concretely, database queirers need to verify the query integrity in the sense that (1) for any query request, the query is executed by the cloud on correct data and the returned results have not been modified; (2) the query result should embrace all data sets satisfying the SQL query; and (3) the query must be executed on the database of latest version considering frequent updates to the database. Several schemes have been proposed for query integrity. However, these solutions incur a significant overhead in practice, either from the perspective of communication or from the view of computation. Therefore, we explore the approach which can significantly reduce overhead by devising new cryptographic primitive, and consider flexible SQL queries to better fulfill the requirements of many practical systems. This work was presented at the 2012 ACM Workshop on Cloud Computing Security [129], and will be presented in Chapter 4.

Chapter 2: VERIFIABLE ATTRIBUTE-BASED SEARCH ON OUTSOURCED ENCRYPTED DATA

2.1 Introduction

Cloud computing allows data owners to use massive data storage and vast computation capabilities at a very convenient price. Despite its benefits, data outsourcing deprives data owners of direct control on their outsourced data. To alleviate such concerns, data owners should adopt encryption and encrypt their data before storing it into the cloud. However, encryption alone may severely hinder several functionalities cloud solutions have accustomed users to. For instance, it would be impossible to search on data owner's outsourced encrypted data. In addition, a data owner may not be able to grant search capabilities to data users according to a specified access control policy (e.g., stating that only certain users can search on sensitive data). While the cloud may search on encrypted data thanks to current cryptographic techniques, it is not clear how to verify whether it faithfully followed the requested search operations. As we will elaborate later, existing solutions cannot achieve these goals simultaneously.

2.1.1 Our Contribution

We propose a novel cryptographic solution by which data owners can control the search and use of its outsourced encrypted data according to its access control policy, while the legitimate data users can outsource the often costly search operations to the cloud and can verify whether the cloud has faithfully executed users' search operations. More specifically, the solution allows a data user with proper credentials (according to a data owner's access control policy) to (i) search on the data owner's outsourced encrypted data, (ii) outsource the actual search operations to the cloud, and (iii) verify whether or not the cloud has faithfully executed the user's search operations. The solution is centered on a novel cryptographic scheme, *verifiable attribute-based keyword search* (VABKS).

In addition to formally define security properties of VABKS, we present the VABKS construction in a modular fashion, by using attribute-based encryption, bloom filter, digital signature, and a newly introduced building-block called attribute-based keyword search (ABKS), which may be of independent value. Our application of bloom filter and digital signature allows data users to *efficiently* verify that the cloud honestly conducted the users' search operations on some data owner's outsourced encrypted data. Experimental evaluation shows that the VABKS (and VABKS) schemes are practical.

2.1.2 Related Work

To the best of our knowledge, no solution is adequate for what we want to achieve. Nevertheless, we briefly review the relevant techniques below.

Keyword Search over Encrypted Data. Existing solutions for keyword-based search over encrypted data can be classified into two categories: searchable encryption in the symmetric-key setting (e.g., [32, 34, 37, 43, 58, 71, 73, 74, 80, 114]) and searchable encryption in the public-key setting (e.g., [11, 14, 22, 26, 119]). In these solutions, the data owner generates some tokens that can be used by a data user to search over the data owner's encrypted data. Several variants (e.g., [24, 63, 85, 112]) have been proposed to support complex queries. Moreover [13, 43] considered searchable encryption in the multi-users setting, where the data owner can enforce an access control policy by distributing (stateful) secret keys to authorized users directly and revoke the secret keys if necessary. However, all these solutions do not solve the problem studied in the present chapter because (1) some solutions require interactions between data users and data owners (or trusted proxy, e.g., trapdoor generation entity [26]) to grant search capabilities, and (2) all these silutions (except [32]) assume that the server faithfully executed search operations. In contrast, our solution allows a data user with proper credentials to issue search tokens by which the cloud can perform keyword search on behalf of the user, *without* requiring any interaction with the data owner. Moreover, the data user can verify whether or not the cloud has faithfully executed the user's search operations. This is true even for the powerful technique called predicate encryption [76, 95], which does not offer the desired verifiability.

Attribute-Based Encryption (ABE). This technique allows entities with proper credentials to decrypt a ciphertext in question [108]. It has two variants, depending on how access control is enforced, called key-policy ABE (KP-ABE), where the decryption key is associated with an access control policy [65], and ciphertext-policy ABE (CP-ABE), where the ciphertext is associated with an access control policy [18]. ABE is a popular method for enforcing access control policies via cryptographic means, and has been enriched with various features (e.g., [35, 36, 82, 96]). In this chapter, we extend ABE with the novel feature of keyword search, namely *attribute-based keyword search*, by which keywords are encrypted with some access control policies so that only data users with proper cryptographic credentials can generate tokens that can be used to searcher over the outsourced encrypted data. This effectively prevents data owners from knowing the keywords a data user is searching for while requiring no interaction between data users and data owners/trusted authorities. This is even in contrast to [26], where data users should interact with data owners/trusted authorities to obtain search tokens obliviously.

Verifiable Keyword Search. Recently, verifiable keyword search solutions have been proposed in [16,50,100], where each keyword is represented as a root of a polynomial. It is possible to check whether a keyword is present or not by evaluating the polynomial on the keyword and verifying if its output is zero. However, these approaches work only when keywords are sent in the cleartext to the cloud, and are not suitable for our purpose because the cloud should not know the keywords involved. It is worth mentioning that the secure verifiable keyword search in the symmetric-key setting [32] can be *insecure* in the public-key setting because the attacker can infer keywords in question via an *off-line* keyword guessing attack (in lieu of the off-line dictionary attack against passwords).

2.2 Preliminaries

Let $a \leftarrow S$ denote selecting an element a from a set S uniformly at random, $||$ denote the concatenation operation and $\mathsf{string}(S)$ denote the concatenation of elements of S ordered by their hash values. Let $\mathsf{U} = \{\mathsf{at}_1, \ldots, \mathsf{at}_n\}$ be the set of attributes, based on which access control policies are specified.

2.2.1 Cryptographic Assumptions and Structure

Let p be an ℓ-bit prime, and G, G_T be cyclic groups of prime order p with generators g, g_T respectively. Let e be a bilinear map: $e : G \times G \rightarrow G_T$ satisfying: (i) $\forall a, b \leftarrow \mathbb{Z}_p$, $e(g^a, g^b) = e(g, g)^{ab}$, (ii) $e(g, g) \neq 1$, and (iii) e can be computed efficiently.

Decisional Linear Assumption (DL). Given $(g, f, h, f^{r_1}, g^{r_2}, Q)$ where $g, f, h, Q \leftarrow G, r_1, r_2 \leftarrow \mathbb{Z}_p$, this assumption says that any probabilistic polynomial-time algorithm $\mathcal{A}$ can determine if $Q \overset{?}{=} h^{r_1+r_2}$ with a negligible advantage in security parameter ℓ, where the advantage is defined as

$$|\mathsf{Pr}[\mathcal{A}(g, f, h, f^{r_1}, g^{r_2}, h^{r_1+r_2}) = 1] - \mathsf{Pr}[\mathcal{A}(g, f, h, f^{r_1}, g^{r_2}, Q) = 1]|.$$

Generic Bilinear Group [20]. Let ψ_0, ψ_1 be two random encodings of the additive group $\mathbb{Z}_p^+$, such that ψ_0, ψ_1 are injective maps from $\mathbb{Z}_p^+$ to $\{0, 1\}^m$, where $m > 3\log(p)$. Let $G = \{\psi_0(x)|x \in \mathbb{Z}_p\}$ and $G_T = \{\psi_1(x)|x \in \mathbb{Z}_p\}$. There is an oracle to compute $e : G \times G \rightarrow G_T$. G is referred to as a generic bilinear group. Let g denote $\psi_0(1)$, g^x denote $\psi_0(x)$, $e(g, g)$ denote $\psi_1(1)$, and $e(g, g)^y$ denote $\psi_1(y)$.

Pseudorandom Generator [75]. A pseudorandom generator $H : \{0, 1\}^\ell \rightarrow \{0, 1\}^m, \ell < m$, is a deterministic algorithm that takes as input an ℓ-bit seed and generates a m-bit string that cannot be distinguished from a m-bit random string by any polynormial-time algorithm (in ℓ).

Attribute-based Encryption (ABE). Let ABE be a secure attribute-based encryption scheme [18, 65], such that ABE $=$ (Setup, KeyGen, Enc, Dec) where Setup is to initialize the system parameter and master key, KeyGen is to generate credentials for users, Enc is to encrypt the data with the access control policy and Dec is to decrypt the ciphertext with the user's credentials.

Symmetric Encryption (SE) [75]. Let SE be a secure symmetric encryption, such that SE $=$ (KeyGen, Enc, Dec) where KeyGen is to generate symmetric key, Enc is to encrypt the message and Dec is to decrypt the ciphertext.

Digital Signature (Sig) **[75].** Let Sig be a secure digital signature, such that $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, where KeyGen is to generate public/private key, Sign is to generate a signature for the message and Verify is to verify whether the message is matched the signature.

2.2.2 Bloom Filter for Membership Query

A Bloom filter [19] is a data structure for succinctly representing a static set of items while allowing membership queries. A m-bit Bloom filter is an array of m bits, that are initially set to 0. It uses k independent universal hash functions $H'_1, \ldots, H'_k$ of range $\{0, \ldots, m-1\}$. For each element $w \in S = \{w_1, \ldots, w_n\}$, the bits corresponding to $H'_j(w)$ are set to 1 for $1 \leq j \leq k$. To check whether w is an element of S, it verifies if all bits corresponding to $H'_j(w)$, $1 \leq j \leq k$, are set to 1. If not, w is certainly not an element of S; otherwise, w might be a member of S with high probability (a false-positive rate should be considered). Suppose the k hash functions are perfectly random and n elements are hashed into the m-bit Bloom filter, the false positive rate is $(1-(1-\frac{1}{m})^{km})^k \approx (1-e^{-kn/m})^k$. The optimal number of hash functions minimizing the false positive rate is $k = (\ln 2)m/n$, and the minimal false positive rate is $(0.6185)^{m/n}$. A m-bit Bloom filter has two associated algorithms:

- $\mathsf{BF} \leftarrow \mathsf{BFGen}(\{H'_1, \ldots, H'_k\}, \{w_1, \ldots, w_n\})$: This algorithm generates a m-bit Bloom filter by hashing the data set $\{w_1, \ldots, w_n\}$ with $\{H'_1, \ldots, H'_k\}$.

- $\{0, 1\} \leftarrow \mathsf{BFVerify}(\{H'_1, \ldots, H'_k\}, \mathsf{BF}, w)$: This algorithm returns 1 if w is an element of S, and 0 otherwise.

2.2.3 Access Trees for Representing Access Control Policies

Access trees are convenient for representing access control policies [65]. In an access tree, a leaf is associated with an attribute and an inner node represents a threshold gate. Let num_v be the number of children of the node v, and label the children from the left to the right as 1 to num_v. Let k_v be the threshold value associated with node v, so that $1 \leq k_v \leq \mathsf{num}_v$, where $k_v = 1$ represents the OR gate and $k_v = num_v$ represents the AND gate as two special cases. Let $\mathsf{parent}(v)$ denote the parent of node v, $\mathsf{ind}(v)$ denote the label associated with node v, $\mathsf{att}(v)$ denote the attribute associated with leaf node v, $\mathsf{lvs}(\mathsf{T})$ denote the set of leaves of the access tree T, and T_v denote the subtree of T rooted at the node v (thus, $\mathsf{T}_{\mathsf{root}} = \mathsf{T}$).

Given an attribute set $\mathsf{Atts} \subseteq \mathsf{U}$, Let $F(\mathsf{Atts}, \mathsf{T}_v) = 1$ indicate that Atts satisfies the access control policy represented by subtree T_v. The evaluation of $F(\mathsf{Atts}, \mathsf{T}_v)$ can be performed iteratively as follows:

- In the case v is a leaf: If $\mathsf{att}(v) \in \mathsf{Atts}$, set $F(\mathsf{Atts}, \mathsf{T}_v) = 1$; otherwise, set $F(\mathsf{Atts}, \mathsf{T}_v) = 0$.
- In the case v is an inner node with children $v_1, \ldots, v_{\mathsf{num}_v}$: If there exists a subset $I \subseteq \{1, \ldots, \mathsf{num}_v\}$ such that $|I| \geq k_v$ and $\forall\, j \in I, F(\mathsf{Atts}, \mathsf{T}_{v_j}) = 1$, then set $F(\mathsf{Atts}, \mathsf{T}_v) = 1$; otherwise, set $F(\mathsf{Atts}, \mathsf{T}_v) = 0$.

Given access tree T, we denote the algorithm for distributing a secret s according to T as:

$$\{q_v(0) | v \in \mathsf{lvs}(\mathsf{T})\} \leftarrow \mathsf{Share}(\mathsf{T}, s).$$

The algorithm generates a polynomial q_v for each node v in T with the respective threshold value k_v in a top-down fashion according to T (for each leaf node, its threshold value is naturally 1) as follows.

- If v is the root of T ($v = \mathsf{root}$), set $q_v(0) = s$ and randomly pick $k_v - 1$ coefficients for polynomial q_v.
- Else if v is a leaf of T, set $q_v(0) = q_{\mathsf{parent}(v)}(\mathsf{ind}(v))$.
- Otherwise v is an inner node, set $q_v(0) = q_{\mathsf{parent}(v)}(\mathsf{ind}(v))$ and randomly select $k_v - 1$ coefficients for polynomial q_v.

When the algorithm stops, each leaf v is associated with a value $q_v(0)$, which is the secret share of s.

Given access tree T and a set of values $\{E_{u_1}, \ldots, E_{u_m}\}$, where $u_1, \ldots, u_m$ are leaves of T such that $F(\{\mathsf{att}(u_1), \ldots, \mathsf{att}(u_m)\}, \mathsf{T}) = 1$, $E_{u_j} = e(g,h)^{q_{u_j}(0)}$ for $1 \leq j \leq m, g, h \in G$, e is a bilinear map and $q_{u_1}(0), \ldots, q_{u_m}(0)$ are secret shares of s according to T, we denote the algorithm for reconstructing $e(g,h)^s$ as

$$e(g,h)^s \leftarrow \mathsf{Combine}(\mathsf{T}, \{E_{u_1}, \ldots, E_{u_m}\}).$$

The algorithm executes the following steps with respect to node v in a bottom-top fashion according to T.

- If $F(\{\mathsf{att}(u_1), \ldots, \mathsf{att}(u_m)\}, \mathsf{T}_v) = 0$, then continue.
- Otherwise, $F(\{\mathsf{att}(u_1), \ldots, \mathsf{att}(u_m)\}, \mathsf{T}_v) = 1$ and it executes below:
  - If v is a leaf, set $E_v = E_{u_j}(0) = e(g,h)^{q_{u_j}(0)}$ where $v = u_j$ for some j.
  - Otherwise, v is an inner node. Then, for v's children nodes $\{v_1, \cdots, v_{\mathsf{num}_v}\}$, there should exist a set of indices S, such that $|S| = k_v$, such that $j \in S, F(\{\mathsf{att}(u_1)$, …, $\mathsf{att}(u_m)\}$, $\mathsf{T}_{v_j}) = 1$. Let $\Delta_{v_j} = \prod_{l \in S, l \neq j} \frac{-j}{l-j}$ so that $E_v = \prod_{j \in S} E_{v_j}^{\Delta_{v_j}} = \prod_{j \in S} (e(g,h)^{q_{v_j}(0)})^{\Delta_{v_j}} = e(g,h)^{q_v(0)}$

When the algorithm stops, the root of T has an associated value $E_{\mathsf{root}} = e(g,h)^{q_{\mathsf{root}}(0)} = e(g,h)^s$.

2.3 Attribute-Based Keyword Search (ABKS)

This building-block technique, called attribute-based keyword search (ABKS), allows data owners to specify access control policies with respect to their data and keyword search functions when outsourcing their encrypted data to the cloud, The data users who possess attributes satisfying a data owner's access control policy can search on encrypted keywords without interacting with the data owner.

This building-block technique has two variants: Key-Policy ABKS (KP-ABKS) where the cryptographic credentials are associated to an access control policy, and Ciphertext-Policy ABKS (CP-

ABKS) where the ciphertext is associated to an access control policy. To unify the presentation, let I_{Enc} denote the input to encryption function Enc and I_{KeyGen} to denote the input to key generation function KeyGen. In the case of $\mathsf{CP\text{-}ABKS}$, I_{Enc} and I_{KeyGen} are respectively the access tree and attribute set. In the case of $\mathsf{KP\text{-}ABKS}$, I_{Enc} and I_{KeyGen} are respectively the attribute set and access tree. Let $F(I_{\mathsf{KeyGen}}, I_{\mathsf{Enc}}) = 1$ denote that I_{KeyGen} satisfies I_{Enc} in $\mathsf{CP\text{-}ABKS}$ and that I_{Enc} satisfies I_{KeyGen} in $\mathsf{KP\text{-}ABKS}$.

2.3.1 ABKS Definition and Security

Definition 1. An ABKS operates in the following model: A data owner outsources its encrypted keyword to the cloud, a data user generates the search token with respect to some keyword and retrieves encrypted data from the cloud, and a cloud provides data storage and retrieval service upon request. The ABKS consists of algorithm as follows:

- $(\mathsf{mk}, \mathsf{pm}) \leftarrow \mathsf{Setup}(1^{\ell})$: This algorithm initializes the public parameter pm and generates a master key mk.

- $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{mk}, I_{\mathsf{KeyGen}})$: Taking as input the master key mk and I_{KeyGen}, this algorithm outputs credential sk for a user.

- $\mathsf{cph} \leftarrow \mathsf{Enc}(w, I_{\mathsf{Enc}})$: Taking as input keyword w and I_{Enc}, this algorithm outputs the encrypted keyword cph.

- $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$: Taking as input credential sk and keyword w, a legitimate (or authorized) data user uses this algorithm to generates a search token tk.

- $\{0, 1\} \leftarrow \mathsf{Search}(\mathsf{cph}, \mathsf{tk})$: The cloud uses this algorithm to determine whether the encrypted keyword cph and the token tk correspond to the same keyword. Return 1 if so, and 0 otherwise.

Correctness of ABKS requires that, given $(\mathsf{mk}, \mathsf{pm}) \leftarrow \mathsf{Setup}(1^{\ell})$, $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{mk}, I_{\mathsf{KeyGen}})$ and $F(I_{\mathsf{KeyGen}}, I_{\mathsf{Enc}}) = 1$, for any w, $\mathsf{cph} \leftarrow \mathsf{Enc}(w, I_{\mathsf{Enc}})$ and $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$, then $1 \leftarrow$

$\mathsf{Search}(\mathsf{cph}, \mathsf{tk})$ always holds.

The threat model of ABKS is the following: data owners and authorized data users are trusted, whereas the cloud is semi-trusted (i.e., trusted but curious). ABKS security requires that the cloud learn nothing but the search results. Specifically, given a probabilistic polynomial-time adversary $\mathcal{A}$ modeling the semi-trusted cloud, an ABKS scheme should satisfy the following intuitive security requirements:

- Selective security against chosen-keyword attack: This notion is to capture that the adversary $\mathcal{A}$ cannot deduce any information about keyword only with keyword ciphertexts (i.e. without being any matched search tokens) in the selective security model, other than what it already knows from previous results, e.g. querying search tokens by adaptively choosing keywords. In other words, $\mathcal{A}$ cannot distinguish the encryption of two challenge keywords of its choice. Here the selective security model [30] means that $\mathcal{A}$ must determine I_{Enc} it intends to attack before the system parameters are initiated. We formalize this security property via the following selective chosen-keyword attack game .

- Keyword secrecy: In the public-key setting, it is impossible to protect the information encoded by search tokens (aka. predicate privacy [111]) due to the *keyword guessing attack*: $\mathcal{A}$ can encrypt any keyword of his choice and check whether the resulting keyword ciphertext and the target token have the same keyword. Therefore, we use a weaker notion, *keyword secrecy*, that assures the probability of $\mathcal{A}$ learning the keyword from the ciphertext as well as the search token is no more than that of one random keyword guess. We formalize this security property via the keyword secrecy game.

Selectively Chosen-Keyword Attack (SCKA) **Game**:

Setup: $\mathcal{A}$ selects a non-trivial challenge I^*_{Enc} (a trivial challenge I^*_{Enc} is one that can be satisfied by any data user who does not have any credential), and gives it to the challenger. Then the challenger runs $\mathsf{Setup}(1^\ell)$ to generate the public parameter pm and the master key mk.

Phase 1: $\mathcal{A}$ can query the following oracles for polynomially many times, and the challenger keeps a keyword list L_{kw}, which is initially empty.

- $\mathcal{O}_{\mathsf{KeyGen}}(I_{\mathsf{KeyGen}})$: If $F(I_{\mathsf{KeyGen}}, I^*_{\mathsf{Enc}}) = 1$, then abort; otherwise, the challenger returns to $\mathcal{A}$ credential sk corresponding to I_{KeyGen}.

- $\mathcal{O}_{\mathsf{TokenGen}}(I_{\mathsf{KeyGen}}, w)$: The challenger generates credential sk with I_{KeyGen}, and returns to $\mathcal{A}$ a search token tk by running algorithm $\mathsf{TokenGen}$ with inputs sk and w. If $F(I_{\mathsf{KeyGen}}, I^*_{\mathsf{Enc}}) = 1$, the challenger adds w to L_{kw}.

Challenge phase: $\mathcal{A}$ chooses two keywords w_0 and w_1, where $w_0, w_1 \notin L_{kw}$. The challenger selects $\lambda \leftarrow \{0,1\}$, computes $\mathsf{cph}^* \leftarrow \mathsf{Enc}(w_\lambda, I^*_{\mathsf{Enc}})$, and delivers cph^* to $\mathcal{A}$. Note that the requirement of $w_0, w_1 \notin L_{kw}$ is to prevent $\mathcal{A}$ from trivially guessing λ with tokens from $\mathcal{O}_{\mathsf{TokenGen}}$.

Phase 2: $\mathcal{A}$ continues to query the oracles as in Phase 1. The restriction is that $(I_{\mathsf{KeyGen}}, w_0)$ and $(I_{\mathsf{KeyGen}}, w_1)$ cannot be the input to $\mathcal{O}_{\mathsf{TokenGen}}$ if $F(I_{\mathsf{KeyGen}}, I^*_{\mathsf{Enc}}) = 1$.

Guess: $\mathcal{A}$ outputs a bit λ', and wins the game if $\lambda' = \lambda$.

Let $|\mathsf{Pr}[\lambda = \lambda'] - \frac{1}{2}|$ be the advantage of $\mathcal{A}$ winning the above SCKA game. Thus, we have

Definition 2. An ABKS scheme is *selectively secure against chosen-keyword attack* if the advantage of any $\mathcal{A}$ winning the SCKA game is negligible in security parameter ℓ.

Keyword Secrecy Game:

Setup: The challenger runs $\mathsf{Setup}(1^\ell)$ to generate the public parameter pm and the master key mk.

Phase 1: $\mathcal{A}$ can query the following oracles for polynomially many times:

- $\mathcal{O}_{\mathsf{KeyGen}}(I_{\mathsf{KeyGen}})$: The challenger returns to $\mathcal{A}$ credential sk corresponding to I_{KeyGen}. It adds I_{KeyGen} to the list L_{KeyGen}, which is initially empty.

- $\mathcal{O}_{\mathsf{TokenGen}}(I_{\mathsf{KeyGen}}, w)$: The challenger generates credential sk with I_{KeyGen}, and returns to $\mathcal{A}$ a search token tk by running algorithm $\mathsf{TokenGen}$ with input sk and w.

Challenge phase: $\mathcal{A}$ chooses a non-trivial I^*_{Enc} and gives it to the challenger. The challenger selects w^* from the message space uniformly at random and selects I^*_{KeyGen} such that $F(I^*_{\mathsf{KeyGen}}, I^*_{\mathsf{Enc}}) = 1$.

The challenger runs $\mathsf{cph} \leftarrow \mathsf{Enc}(w^*, I^*_{\mathsf{Enc}})$ and $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w^*)$ and delivers $(\mathsf{cph}, \mathsf{tk})$ to $\mathcal{A}$. We require that $\forall I_{\mathsf{KeyGen}} \in L_{\mathsf{KeyGen}}$,

$F(I_{\mathsf{KeyGen}}, I^*_{\mathsf{Enc}}) = 0$.

Guess: After guessing q distinct keywords, $\mathcal{A}$ outputs a keyword w', and wins the game if $w' = w$.

Definition 3. An ABKS scheme achieves *keyword secrecy* if the probability that $\mathcal{A}$ wins the keyword secrecy game is at most $\frac{1}{|\mathcal{M}|-q} + \epsilon$, where $\mathcal{M}$ is the keyword space, q is the number of distinct keywords that the adversary has attempted, and ϵ is a negligible in security parameter ℓ.

2.3.2 ABKS Construction

The basic idea underlying the ABKS construction is to separate each keyword ciphertext and search token into two parts: one is associated to the keyword and the other is associated to the attributes (or access control policy). If the data user's attributes satisfy the access control policy, it can determine whether the search token matches the encrypted keyword.

To further highlight the basic idea, let us consider KP-ABKS as example. Assume that $H_1 : \{0,1\}^* \rightarrow G$ is a secure hash function modeled as random oracle, and $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_p$ is a secure one-way hash function. A data user's credentials are generated by letting $t \leftarrow \mathbb{Z}_p$, $A_v = g^{q_v(0)} H_1(\mathsf{att}(v))^t$, $B_v = g^t$ for each leaf v, where g is the generator of G, $q_v(0)$ is the share of secret ac for leaf v according to access tree T. The keyword ciphertext and search token are generated as follows:

- Keyword w is encrypted into two parts: one is to "blend" keyword with randomness by letting $W' = g^{cr_1}$, $W = g^{a(r_1+r_2)} g^{bH_2(w)r_1}$ and $W_0 = g^{r_2}$ where $g^a, g^b, g^c \in G$ are public keys and $r_1, r_2 \leftarrow \mathbb{Z}_p$, and the other is associated to the attribute set Atts by letting $W_j = H_1(\mathsf{at}_j)^{r_2}$ for each $\mathsf{at}_j \in \mathsf{Atts}$. The two parts are tied to r_2.

- Given a set of credentials, a search token for keyword w is generated with two parts: one is associated to the keyword, $\mathsf{tok}_1 = (g^a g^{bH_2(w)})^s$ and $\mathsf{tok}_2 = g^{cs}$ by selecting $s \leftarrow \mathbb{Z}_p$, and the other is associated to the credentials by letting $A'_v = A^s_v$, $B'_v = B^s_v$ for each $v \in \mathsf{lvs}(\mathsf{T})$. The two parts are tied to s.

As long as the attribute set satisfies the access tree T, the cloud can use A'_v, B'_v and W_0, W_j to recover $e(g,g)^{acr_2 s}$, which can be used to test the keyword equality as elaborated below.

$\mathsf{KP\text{-}ABKS}$ **Construction.** Let ℓ be the primary security parameter. It consists of the following algorithms.

$\mathsf{Setup}(1^\ell)$: This algorithm selects a bilinear map $e : G \times G \to G_T$, such that G and G_T are cyclic groups of order p, an ℓ-bit prime. It selects $a, b, c \leftarrow \mathbb{Z}_p$ and $g \leftarrow G$, and sets

$$\mathsf{pm} = (e, g, p, g^a, g^b, g^c, G, G_T), \mathsf{mk} = (a, b, c).$$

$\mathsf{KeyGen}(\mathsf{mk}, \mathsf{T})$: This algorithm executes $\{q_v(0)|v \in \mathsf{lvs}(\mathsf{T})\} \leftarrow \mathsf{Share}(\mathsf{T}, ac)$ to obtain secret shares of ac for all leaves in T. For each leaf v in T, it picks $t \leftarrow \mathbb{Z}_p$, and computes $A_v = g^{q_v(0)} H_1(\mathsf{att}(v))^t$ and $B_v = g^t$. It sets

$$\mathsf{sk} = (\mathsf{T}, \{(A_v, B_v)|v \in \mathsf{lvs}(\mathsf{T})\}).$$

$\mathsf{Enc}(w, \mathsf{Atts})$: This algorithm selects $r_1, r_2 \leftarrow \mathbb{Z}_p$, and computes $W' = g^{cr_1}$, $W = g^{a(r_1+r_2)} g^{bH_2(w)r_1}$ and $W_0 = g^{r_2}$. For each $\mathsf{at}_j \in \mathsf{Atts}$, it computes $W_j = H_1(\mathsf{at}_j)^{r_2}$. It sets

$$\mathsf{cph} = (\mathsf{Atts}, W', W, W_0, \{W_j|\mathsf{at}_j \in \mathsf{Atts}\}).$$

$\mathsf{TokenGen}(\mathsf{sk}, w)$: This algorithm selects $s \leftarrow \mathbb{Z}_p$, and computes $A'_v = A_v^s, B'_v = B_v^s$ for each $v \in \mathsf{lvs}(\mathsf{T})$. It computes $\mathsf{tok}_1 = (g^a g^{bH_2(w)})^s$ and $\mathsf{tok}_2 = g^{cs}$. It sets

$$\mathsf{tk} = (\mathsf{tok}_1, \mathsf{tok}_2, \mathsf{T}, \{(A'_v, B'_v)|v \in \mathsf{lvs}(\mathsf{T})\})$$

$\mathsf{Search}(\mathsf{tk}, \mathsf{cph})$: Given the attribute set Atts included in cph, this algorithm selects a attribute set S satisfying the access tree T included in tk. If S does not exist, return 0; otherwise, for each $\mathsf{at}_j \in S$, it computes $E_v = e(A'_v, W_0)/e(B'_v, W_j) = e(g,g)^{sr_2 q_v(0)}$, where $\mathsf{att}(v) = \mathsf{at}_j$ for $v \in \mathsf{lvs}(\mathsf{T})$. Then it computes $e(g,g)^{sr_2 q_{\mathsf{root}}(0)} \leftarrow \mathsf{Combine}(\mathsf{T}, \{E_v | \mathsf{att}(v) \in S\})$ so that $E_{\mathsf{root}} = e(g,g)^{acsr_2}$. It returns 1 if $e(W', \mathsf{tok}_1)E_{\mathsf{root}} = e(W, \mathsf{tok}_2)$, and 0 otherwise.

Correctness of KP-ABKS can be verified as follows:

$$\begin{aligned} e(W', \mathsf{tok}_1)E_{\mathsf{root}} &= e(g^{cr_1}, (g^a g^{bH_2(w)})^s)E_{\mathsf{root}} \\ &= ee(g,g)^{acsr_1} e(g,g)^{bcsH_2(w)r_1} E_{\mathsf{root}} \\ &= e(g,g)^{acs(r_1+r_2)} e(g,g)^{bcsH_2(w)r_1} \\ e(W, \mathsf{tok}_2) &= ee(g^{a(r_1+r_2)} g^{bH_2(w)r_1}, g^{cs}) \\ &= ee(g,g)^{acs(r_1+r_2)} e(g,g)^{bcsH_2(w)r_1} \end{aligned}$$

Security of KP-ABKS is assured by the following theorems.

Theorem 1. *Assume that the DL assumption holds, the* KP-ABKS *scheme is selectively secure against chosen-keyword attack in the random oracle model.*

Proof. We show that if there is a polynomial-time adversary $\mathcal{A}$ that wins the SCKA game with advantage μ, then there is a challenger algorithm that solves the DL problem with advantage $\mu/2$. Given a DL instance $(g, h, f, f^{r_1}, g^{r_2}, Q)$, where $g, f, h, Q \leftarrow G$ and $r_1, r_2 \leftarrow \mathbb{Z}_p$, the challenger simulates the SCKA game as follows.

Setup: The challenger sets $g^a = h$ and $g^c = f$ where a and c are unknown, selects $d \leftarrow \mathbb{Z}_p$ and computes $g^b = f^d = g^{cd}$ by implicitly defining $b = cd$. Let H_2 be an one-way hash function and $\mathsf{pm} = (e, g, p, h, f^d, f)$ and $\mathsf{mk} = (d)$.

$\mathcal{A}$ selects an attribute set Atts^* and gives it to the challenger. The random oracle $\mathcal{O}_{H_1}(\mathsf{at}_j)$ is defined as follows:

- If at_j has not been queried before,
  - if $\mathsf{at}_j \in \mathsf{Atts}^*$, select $\beta_j \leftarrow \mathbb{Z}_p$, add $(\mathsf{at}_j, \alpha_j = 0, \beta_j)$ to $\mathcal{O}_{H_1}$, and return g^{β_j};

  - otherwise, select $\alpha_j, \beta_j \leftarrow \mathbb{Z}_p$, add $(\mathsf{at}_j, \alpha_j, \beta_j)$ to $\mathcal{O}_{H_1}$, and return $f^{\alpha_j} g^{\beta_j}$.

- If at_j has been queried before, retrieve (α_j, β_j) from $\mathcal{O}_{H_1}$ and return $f^{\alpha_j} g^{\beta_j}$.

Phase 1: $\mathcal{A}$ can adaptively query the following oracles for polynomially-many times and the challenger keeps a keyword list L_{kw}, which is empty initially.

$\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{T})$: $\mathcal{A}$ gives an access tree T to the challenger. If $F(\mathsf{Atts}^*, \mathsf{T}) = 1$, then the challenger aborts; otherwise, the challenger generates attributes as follows.

Define the following two procedures to determine the polynomial for each node of T:

- PolySat($\mathsf{T}_v, \mathsf{Atts}^*, \lambda_v$): Given secret λ_v, this procedure determines the polynomial for each node of T_v rooted at v when $F(\mathsf{Atts}^*, \mathsf{T}_v) = 1$. It works as follows: Suppose the threshold value of node v is k_v, it sets $q_v(0) = \lambda_v$ and picks $k_v - 1$ coefficients randomly to fix the polynomial q_v. For each child node v' of v, recursively call PolySat($\mathsf{T}_{v'}, \mathsf{Atts}^*, \lambda_{v'}$) where $\lambda_{v'} = q_v(\mathsf{Index}(v'))$.

- PolyUnsat($\mathsf{T}_v, \mathsf{Atts}^*, g^{\lambda_v}$): Given element $g^{\lambda_v} \in G$ where the secret λ_v is unknown, this procedure determines the polynomial for each node of T_v rooted at v when $F(\mathsf{Atts}^*, \mathsf{T}_v) = 0$ as follows. Suppose the threshold value of the node v is k_v. Let V be the empty set. For each child node v' of v, if $F(\mathsf{Atts}, \mathsf{T}_{v'}) = 1$, then set $V = V \bigcup \{v'\}$. Because $F(\mathsf{Atts}, \mathsf{T}_v) = 0$, then $|V| < k_v$. For each node $v' \in V$, it selects $\lambda_{v'} \leftarrow \mathbb{Z}_p$, and sets $q_v(\mathsf{Index}(v')) = \lambda_{v'}$. Finally it fixes the remaining $k_v - |V|$ points of q_v randomly to define q_v and makes $g^{q_v(0)} = g^{\lambda_v}$. For each child node v' of v,

  - if $F(\mathsf{Atts}^*, \mathsf{T}_{v'}) = 1$, then run PolySat($\mathsf{T}_{v'}, \mathsf{Atts}^*, q_v(\mathsf{Index}(v'))$), where $q_v(\mathsf{Index}(v'))$ is known to the challenger;

  - otherwise, call PolyUnsat($\mathsf{T}_{v'}, \mathsf{Atts}^*, g^{\lambda_{v'}}$), where $g^{\lambda_{v'}} = g^{q_v(\mathsf{Index}(v')}$ is known to the challenger.

With the above two procedures, the challenger runs PolyUnsat($\mathsf{T}, \mathsf{Atts}^*, g^a$), by implicitly defining $q_{\mathsf{root}}(0) = a$. Then for each $v \in \mathsf{lvs}(\mathsf{T})$, the challenger gets $q_v(0)$ if $\mathsf{att}(v) \in \mathsf{Atts}^*$, and gets $g^{q_v(0)}$ otherwise. Because $cq_v(0)$ is the secret share of ac, due to the linear property, the challenger generates credentials for each $v \in \mathsf{lvs}(\mathsf{T})$ as follows:

- If $\mathsf{att}(v) = \mathsf{at}_j$ for some $\mathsf{at}_j \in \mathsf{Atts}^*$: Select $t \leftarrow \mathbb{Z}_p$, set $A_v = f^{q_v(0)} g^{\beta_j t} = g^{cq_v(0)} H_1(\mathsf{att}(v))^t$ and $B_v = g^t$;

- If $\mathsf{att}(v) \notin \mathsf{Atts}^*$ (assuming $\mathsf{att}(v) = \mathsf{at}_j$): Select $t' \leftarrow \mathbb{Z}_p$, set $A_v = (g^{q_v(0)})^{\frac{-\beta_j}{\alpha_j}} (f^{\alpha_j} g^{\beta_j})^{t'}$ and $B_v = g^{q_v(0)\frac{-1}{\alpha_j}} g^{t'}$. Note that (A_v, B_v) is a valid credential because

$$\begin{aligned}
B_v &= g^{q_v(0)\frac{-1}{\alpha_j}} g^{t'} = g^{t' - \frac{q_v(0)}{\alpha_j}} \\
A_v &= g^{q_v(0)\frac{-\beta_j}{\alpha_j}} (f^{\alpha_j} g^{\beta_j})^{t'} \\
&= f^{q_v(0)} (f^{\alpha_j} g^{\beta_j})^{\frac{-q_v(0)}{\alpha_j}} (f^{\alpha_j} g^{\beta_j})^{t'} \\
&= f^{q_v(0)} (f^{\alpha_j} g^{\beta_j})^{t' - \frac{q_v(0)}{\alpha_j}} \\
&= g^{cq_v(0)} H_1(\mathsf{att}(v))^{t' - \frac{q_v(0)}{\alpha_j}}
\end{aligned}$$

  by implicitly letting $t = t' - \frac{q_v(0)}{\alpha_j}$. Note also that $\mathcal{A}$ cannot construct A_v and B_v without knowing α_j, β_j.

Eventually, the challenger returns $\mathsf{sk} = \{(A_v, B_v) | v \in \mathsf{lvs}(\mathsf{T})\}$ to $\mathcal{A}$.

$\mathcal{O}_{\mathsf{TokenGen}}(\mathsf{T}, w)$: The challenger runs $\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{T})$ to get $\mathsf{sk} = (\mathsf{T}, \{A_v, B_v | v \in \mathsf{lvs}(\mathsf{T})\})$, computes $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$, and returns tk to $\mathcal{A}$. If $F(\mathsf{Atts}, \mathsf{T}) = 1$, the challenger adds w to the keyword List L_{kw}.

Challenge phase: $\mathcal{A}$ chooses two keywords w_0 and w_1 of equal length, such that $w_0, w_1 \notin L_{kw}$. The challenger outputs cph^* as:

- Select $\lambda \leftarrow \{0, 1\}$.

- For each $\mathsf{at}_j \in \mathsf{Atts}^*$, set $W_j = (g^{r_2})^{\beta_j}$.

- Set $W' = f^{r_1}$, $W = Q(f^{r_1})^{dH_2(w_\lambda)}$, and $W_0 = g^{r_2}$.

- Set $\mathsf{cph}^* = (\mathsf{Atts}^*, W', W, W_0, \{W_j | \mathsf{at}_j \in \mathsf{Atts}^*\})$ and return cph^* to $\mathcal{A}$.

We note that if $Q = h^{r_1+r_2}$, then cph^* is indeed a legitimate ciphertext for keyword w_λ. The reason is that $W' = f^{r_1} = g^{cr_1}$, $W = Qf^{r_1 d H_2(w_\lambda)} = Qg^{r_1 c d H_2(w_\lambda)} = g^{a(r_1+r_2)} g^{br_1 H_2(w_\lambda)}$, $W_0 = g^{r_2}$, and for $\mathsf{at}_j \in \mathsf{Atts}^*$, $W_j = (g^{r_2})^{\beta_j} = H_1(\mathsf{at}_j)^{r_2}$.

Phase 2: $\mathcal{A}$ continues to query the oracles as in Phase 1. The only restriction is that (T, w_0) and (T, w_1) cannot be the input to $\mathcal{O}_{\mathsf{TokenGen}}$ if $F(\mathsf{Atts}^*, \mathsf{T}) = 1$.

Guess: Finally, $\mathcal{A}$ outputs a bit λ' and gives it to the challenger. If $\lambda' = \lambda$, then the challenger outputs $Q = h^{r_1+r_2}$; otherwise, it outputs $Q \neq h^{r_1+r_2}$.

This completes the simulation. In the challenge phase, if $Q = h^{r_1+r_2}$, then cph^* is a valid ciphertext of w_λ, so the probability of $\mathcal{A}$ outputting $\lambda = \lambda'$ is $\frac{1}{2} + \mu$. If Q is an element randomly selected from G, then cph^* is not a valid ciphertext of w_λ. The probability of $\mathcal{A}$ outputting $\lambda = \lambda'$ is $\frac{1}{2}$ since W is an random element in G. Therefore, the probability of the challenger correctly guessing $Q \overset{?}{=} h^{r_1+r_2}$ with the DL instance $(g, h, f, f^{r_1}, g^{r_2}, Q)$ is $\frac{1}{2}(\frac{1}{2} + \mu + \frac{1}{2}) = \frac{1}{2} + \frac{\mu}{2}$. That is, the challenger solves the DL problem with advantage $\mu/2$ if $\mathcal{A}$ wins the SCKA game with an advantage μ. □

Theorem 2. *Given the one-way hash function H_2, the* KP-ABKS *scheme achieves keyword secrecy.*

Proof. We construct a challenger that exploits the keyword secrecy game as follows:

Setup: The challenger selects $a, b, c \leftarrow \mathbb{Z}_p$, $f \leftarrow G$. Let H_2 be an one-way hash function and $\mathsf{pm} = (e, g, g^a, g^b, g^c, f)$ and $\mathsf{mk} = (a, b, c)$.

The random oracle $\mathcal{O}_{H_1}(\mathsf{at}_j)$ is simulated as follows: If at_j has not been queried before, the challenger selects $\alpha_j \leftarrow \mathbb{Z}_p$, adds $(\mathsf{at}_j, \alpha_j)$ to $\mathcal{O}_{H_1}$, and returns g^{α_j}; otherwise, the challenger retrieves α_j from $\mathcal{O}_{H_1}$ and returns g^{α_j}.

Phase 1: $\mathcal{A}$ can adaptively query the following oracles for polynomially-many times.

$\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{T})$: The challenger generates $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{T}, \mathsf{mk})$ and returns sk to $\mathcal{A}$. It adds T to the list L_{KeyGen}, which is initially empty.

$\mathcal{O}_{\mathsf{TokenGen}}(\mathsf{T}, w)$: The challenger runs $\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{T})$ to obtain $\mathsf{sk} = (\mathsf{T}, \{A_v, B_v | v \in \mathsf{lvs}(\mathsf{T})\})$, computes $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$, and returns tk to $\mathcal{A}$.

Challenge Phase: $\mathcal{A}$ selects an attribute set Atts^*. The challenger chooses an access control policy that is represented as T^* such that $F(\mathsf{Atts}^*, \mathsf{T}^*) = 1$, computes $\mathsf{sk}^* \leftarrow \mathsf{KeyGen}(\mathsf{mk}, \mathsf{T}^*)$. By taking as input Atts^* and sk^*, it selects w^* from keyword space uniformly at random, and computes cph^* and tk^* with Enc and $\mathsf{TokenGen}$. Atts^* should satisfy the requirement defined in the keyword secrecy game.

Guess: Finally, $\mathcal{A}$ outputs a keyword w' and gives it to the challenger. The challenger computes $\mathsf{cph}' \leftarrow \mathsf{Enc}(\mathsf{Atts}, w')$ and if $\mathsf{Search}(\mathsf{tk}^*, \mathsf{cph}') = 1$, then $\mathcal{A}$ wins the game.

This finishes the simulation. Suppose $\mathcal{A}$ has already attempted q distinct keywords before outputting w', we can see that the probability of $\mathcal{A}$ winning the keyword secrecy game is at most $\frac{1}{|\mathcal{M}|-q} + \epsilon$. This is because the size of the remaining keyword space is $|\mathcal{M}| - q$, and as the H_2 is an one way secure hash function, meaning deriving w^* from $H_2(w^*)$ is at most a negligible probability ϵ. Therefore, given q distinct keywords $\mathcal{A}$ has attempted, the probability of $\mathcal{A}$ winning the keyword secrecy game is at most $\frac{1}{|\mathcal{M}|-q} + \epsilon$. Thus, our scheme achieves keyword secrecy as in Definition 3. □

$\mathsf{CP\text{-}ABKS}$ **Construction.** Let ℓ be the primary security parameter. It consists of the following algorithms.

$\mathsf{Setup}(1^\ell)$: This algorithm selects a bilinear group $e : G \times G \rightarrow G_T$, such that G and G_T are cyclic groups of order p, an ℓ-bit prime. It selects $a, b, c \leftarrow \mathbb{Z}_p$ and $g \leftarrow G$, and sets

$$\mathsf{pm} = (e, g, p, g^a, g^b, g^c, G, G_T), \mathsf{mk} = (a, b, c).$$

$\mathsf{KeyGen}(\mathsf{mk}, \mathsf{Atts})$: This algorithm selects $r \leftarrow \mathbb{Z}_p$, computes $A = g^{(ac-r)/b}$. Then for each $\mathsf{at}_j \in \mathsf{Atts}$, it selects $r_j \leftarrow \mathbb{Z}_p$ and computes $A_j = g^r H_1(\mathsf{at}_j)^{r_j}$ and $B_j = g^{r_j}$. It sets

$$\mathsf{sk} = (\mathsf{Atts}, A, \{(A_j, B_j) | \mathsf{at}_j \in \mathsf{Atts}\}).$$

$\mathsf{Enc}(w, \mathsf{T})$: This algorithm selects $r_1, r_2 \leftarrow \mathbb{Z}_p$, and computes $W = g^{cr_1}$, $W_0 = g^{a(r_1+r_2)} g^{bH_2(w)r_1}$ and $W' = g^{br_2}$. It further computes secret shares of r_2 for all leaves of T by running $\{q_v(0)|v \in \mathsf{lvs}(\mathsf{T})\} \leftarrow \mathsf{Share}(\mathsf{T}, r_2)$. Then for each $v \in \mathsf{lvs}(\mathsf{T})$, it computes $W_v = g^{q_v(0)}$ and $D_v = H_1(\mathsf{att}(v))^{q_v(0)}$. It sets

$$\mathsf{cph} = (\mathsf{T}, W, W_0, W', \{(W_v, D_v)|v \in \mathsf{lvs}(\mathsf{T})\})$$

$\mathsf{TokenGen}(\mathsf{sk}, w)$: This algorithm selects $s \leftarrow \mathbb{Z}_p$, and computes $\mathsf{tok}_1 = (g^a g^{bH_2(w)})^s, \mathsf{tok}_2 = g^{cs}$ and $\mathsf{tok}_3 = A^s = g^{(acs-rs)/b}$. Then for each $\mathsf{at}_j \in \mathsf{Atts}$, it computes $A'_j = A_j^s$ and $B'_j = B_j^s$. It sets

$$\mathsf{tk} = (\mathsf{Atts}, \mathsf{tok}_1, \mathsf{tok}_2, \mathsf{tok}_3, \{(A'_j, B'_j)|\mathsf{at}_j \in \mathsf{Atts}\}).$$

$\mathsf{Search}(\mathsf{tk}, \mathsf{cph})$: Given an attribute set Atts included in tk, this algorithm selects a attribute set S that satisfies the access tree T included in cph. If S does not exist, return 0; otherwise, for each $\mathsf{at}_j \in S$, it computes $E_v = e(A'_j, W_v)/e(B'_j, D_v) = e(g, g)^{rsq_v(0)}$, where $\mathsf{att}(v) = \mathsf{at}_j$ for $v \in \mathsf{lvs}(\mathsf{T})$. Then it computes $e(g, g)^{rsq_{\mathsf{root}}(0)} \leftarrow \mathsf{Combine}(\mathsf{T}, \{E_v|\mathsf{att}(v) \in S\})$, so that $E_{\mathsf{root}} = e(g, g)^{rsr_2}$. It returns 1 if $e(W_0, \mathsf{tok}_2) = e(W, \mathsf{tok}_1)E_{\mathsf{root}}e(\mathsf{tok}_3, W')$, and 0 otherwise.

The correctness of CP-ABKS can be verified similar to that of KP-ABKS. Security of CP-ABKS is assured by the following theorems. The proof of the latter one is omitted because it is similar to that of Theorem 2.

Theorem 3. CP-ABKS *scheme is selectively secure against chosen-keyword attack in the generic bilinear group model [20].*

Proof. We show that the CP-ABE scheme is selectively secure against chosen-keyword attack in the generic bilinear group model, where H_1 is modeled as a random oracle and H_2 is a one-way hash function.

In the SCKA game, $\mathcal{A}$ attempts to distinguish $g^{a(r_1+r_2)} g^{br_1 H_2(w_0)}$ from $g^{a(r_1+r_2)} g^{br_1 H_2(w_1)}$. Given $\theta \leftarrow \mathbb{Z}_p$, the probability of distinguishing $g^{a(r_1+r_2)} g^{br_1 H_2(w_0)}$ from g^θ is equal to that of distinguishing g^θ from $g^{a(r_1+r_2)} g^{br_1 H_2(w_1)}$. Therefore, if $\mathcal{A}$ has advantage ϵ in breaking the SCKA game, then it has advantage $\epsilon/2$ in distinguishing $g^{a(r_1+r_2)}g^{br_1 H_2(w_0)}$ from g^{θ}. Thus, let us consider a modified game where $\mathcal{A}$ can distinguish $g^{a(r_1+r_2)}$ from g^{θ}. The modified SCKA game is described as follows:

Setup: The challenger chooses $a, b, c \leftarrow \mathbb{Z}_p$ and sends public parameters (e, g, p, g^a, g^b, g^c) to $\mathcal{A}$. $\mathcal{A}$ chooses an access tree T^*, which is sent to the challenger.

$H_1(\mathsf{at}_j)$ is simulated as follows: If at_j has not been queried before, the challenger chooses $\alpha_j \leftarrow \mathbb{Z}_p$, adds $(\mathsf{at}_j, \alpha_j)$ to O_{H_1} and returns g^{α_j}; otherwise the challenger returns g^{α_j} by retrieving α_j from O_{H_1}.

Phase 1: $\mathcal{A}$ can query $\mathcal{O}_{\mathsf{KeyGen}}$ and $\mathcal{O}_{\mathsf{TokenGen}}$ as follows:

$\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{Atts})$: The challenger selects $r^{(t)} \leftarrow \mathbb{Z}_p$ and computes $A = g^{(ac+r^{(t)})/b}$. For each attribute $\mathsf{at}_j \in \mathsf{Atts}$, the challenger chooses $r_j^{(t)} \leftarrow \mathbb{Z}_p$, computes $A_j = g^{r^{(t)}}g^{\alpha_j r_j^{(t)}}$ and $B_j = g^{r_j^{(t)}}$, and returns $(\mathsf{Atts}, A, \{(A_j, B_j) | \mathsf{at}_j \in \mathsf{Atts}\})$.

$\mathcal{O}_{\mathsf{TokenGen}}(\mathsf{Atts}, w)$: The challenger queries $\mathcal{O}_{\mathsf{KeyGen}}(\mathsf{Atts})$ to get $\mathsf{sk} = (\mathsf{Atts}, A, \{(A_j, B_j) | \mathsf{at}_j \in \mathsf{Atts}\})$ and returns $\mathsf{tk} = (\mathsf{Atts}, \mathsf{tok}_1, \mathsf{tok}_2, \mathsf{tok}_3, \{(A'_j, B'_j) | \mathsf{at}_j \in \mathsf{Atts}\})$ where $\mathsf{tok}_1 = (g^a g^{bH_2(w)})^s$, $\mathsf{tok}_2 = g^{cs}$, $\mathsf{tok}_3 = A^s, A'_j = A_j^s$ and $B'_j = B_j^s$ by selecting $s \leftarrow \mathbb{Z}_p$. If $F(\mathsf{Atts}, \mathsf{T}^*) = 1$, the challenger adds w to the keyword List L_{kw}.

Challenge phase: Given two keywords w_0, w_1 of equal length where $w_0, w_1 \notin L_{kw}$, the challenger chooses $r_1, r_2 \leftarrow \mathbb{Z}_p$, and computes secret shares of r_2 for each leaves in T^*. The challenger selects $\lambda \leftarrow \{0, 1\}$. If $\lambda = 0$, it outputs

$$W = g^{cr_1}, W_0 = g^{\theta}, W' = g^{br_2},$$
$$\{(W_v = g^{q_v(0)}, D_v = g^{\alpha_j q_v(0)}) | v \in \mathsf{lvs}(\mathsf{T}^*), \mathsf{att}(v) = \mathsf{at}_j\}$$

by selecting $\theta \in \mathbb{Z}_p$; otherwise it outputs

$$W = g^{cr_1}, W_0 = g^{a(r_1+r_2)}, W' = g^{br_2},$$
$$\{(W_v = g^{q_v(0)}, D_v = g^{\alpha_j q_v(0)}) | v \in \mathsf{lvs}(\mathsf{T}^*), \mathsf{att}(v) = \mathsf{at}_j\}.$$

Table 2.1: Possible terms for querying group oracle G_T

a	$r_j^{(t)}$	$s(ac+r^{(t)})/b$	cr_1
b	$r^{(t)}+\alpha_j r_j^{(t)}$	$s(r_j^{(t)})$	$q_v(0)$
c	$(ac+r^{(t)})/b$	$s(r^{(t)}+\alpha_j r_j^{(t)})$	$\alpha_j q_v(0)$
α_j	cs	$s(a+bH_2(w))$	br_2

Phase 2: This is the same as in the SCKA game.

We can see that if $\mathcal{A}$ can construct $e(g,g)^{\delta a(r_1+r_2)}$ for some g^δ that can be composed from the oracle outputs he has already queried, then $\mathcal{A}$ can use it to distinguish g^θ from $g^{a(r_1+r_2)}$. Therefore, we need to show that $\mathcal{A}$ can construct $e(g,g)^{\delta a(r_1+r_2)}$ for some g^δ with a negligible probability. That is, $\mathcal{A}$ cannot gain non-negligible advantage in the SCKA game.

In the generic group model, ψ_0 and ψ_1 are random injective maps from $\mathbb{Z}_p$ into a set of p^3 elements. Then the probability of $\mathcal{A}$ guessing an element in the image of ψ_0 and ψ_1 is negligible. Recall that $G=\{\psi_0(x)|x\in\mathbb{Z}_p\}$ and $G_T=\{\psi_1(x)|x\in\mathbb{Z}_p\}$. Hence, let us consider the probability of $\mathcal{A}$ constructing $e(g,g)^{\delta a(r_1+r_2)}$ for some $\delta\in\mathbb{Z}_p$ from the oracle outputs he has queried.

We list all terms that can be queried to the group oracle G_T in Table 2.1. Let us consider how to construct $e(g,g)^{\delta a(r_1+r_2)}$ for some δ. Because r_1 only appears in the term cr_1, δ should contain c in order to construct $e(g,g)^{\delta a(r_1+r_2)}$. That is, let $\delta=\delta' c$ for some δ' and $\mathcal{A}$ wishes to construct $e(g,g)^{\delta' ac(r_1+r_2)}$. Therefore, $\mathcal{A}$ needs to construct $\delta' acr_2$, which will use terms br_2 and $(ac+r^{(t)})/b$. Because $(br_2)(ac+r^{(t)})/b=acr_2+r^{(t)}r_2$, $\mathcal{A}$ needs to cancel $r^{(t)}r_2$, which needs to use the terms $\alpha_j, r^{(t)}+\alpha_j r_j^{(t)}$, $q_v(0)$ and $\alpha_j q_v(0)$ because $q_v(0)$ is the secret share of r_2 according to T^*. However, it is impossible to construct $r^{(t)}r_2$ with these terms because $r^{(t)}r_2$ only can be reconstructed if the attributes corresponding to $r_j^{(t)}$ of $r^{(t)}+\alpha_j r_j^{(t)}$ satisfies the access tree T^*.

Therefore, we can conclude that $\mathcal{A}$ gains a negligible advantage in the modified game, which means that $\mathcal{A}$ gains a negligible advantage in the SCKA game. This completes the proof. □

Theorem 4. *Given the one way hash function H_2, the* CP-ABKS *scheme achieves keyword secrecy.*

2.4 Verifiable Attribute-based Keyword Search (VABKS)

In the model of ABKS, the cloud is semi-trusted. VABKS achieves the goal of ABKS in the presence of possibly malicious cloud, which may cheat in the search operation.

2.4.1 System Model and Threat Model

We consider the system model as illustrated in Figure 4.1, which involves four entities: the data owner, who outsources its encrypted data which is indexed by keywords that are also encrypted but can be searched by the data users with the proper credentials; the cloud, which provides storage services and can conduct the actual keyword search operations on behalf of the data users; the data user, who is to retrieve the data owner's encrypted data (i.e. keyword ciphertext and the associated data ciphertexts) according to some keyword; the trusted authority that issues credentials to the data owners/users. We assume that the credentials are sent over authenticated private channels (which can be achieved through another layer of mechanisms).

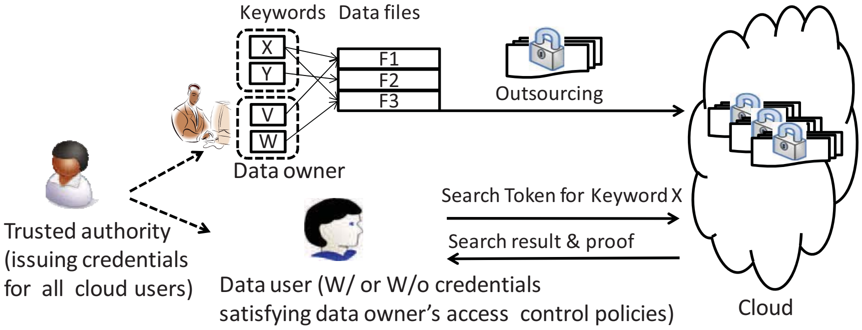


Figure 2.1: System model for verifiable attribute-based keyword search on outsourced encrypted data, where keywords X, Y and V, W have different access control policies.

The data owners are naturally trusted. We assume that both authorized and unauthorized data users are semi-trusted, meaning that they may try to derive some sensitive information of interest. We assume that the cloud is *not* trusted as it may manipulate the search operations (already implying that it manipulates the stored data).

2.4.2 VABKS Definition and Security

Let $\mathsf{D} = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS})$ denote the inverted index and the set of data files. $\mathsf{KS} = \{\mathsf{KS}_1, \ldots, \mathsf{KS}_l\}$ is a set of l keyword sets (also referred as "keyword group"), in which elements are encrypted with the same access control policy. $\mathsf{MP} = \{\mathsf{MP}(w) | w \in \cup_{i=1}^{l} \mathsf{KS}_i\}$ is the set of $\mathsf{MP}(w), w \in \cup_{i=1}^{l} \mathsf{KS}_i$, and $\mathsf{MP}(w)$ consists of the set of identifiers for identifying data files associated with keyword w. $\mathsf{FS} = \{\mathsf{F}_1, \ldots, \mathsf{F}_n\}$ is the set of n data files.

Definition 4. A VABKS scheme consists of the following algorithms:

- $(\mathsf{mk}, \mathsf{pm}) \leftarrow \mathsf{Init}(1^\ell)$: The trusted authority runs this algorithm to initialize the system.

- $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{mk}, I_{\mathsf{KeyGen}})$: The trusted authority runs this algorithm issue credentials sk to a data user/owner.

- $(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}) \leftarrow \mathsf{BuildIndex}(\{I_{\mathsf{Enc}}\}_l, \{I'_{\mathsf{Enc}}\}_n, \mathsf{D})$: A data owner runs this algorithm to encrypt $\mathsf{D} = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS})$ and obtain data ciphertexts $\mathsf{D}_{\mathsf{cph}}$, encrypted index Index and auxiliary information Au, where $\{I_{\mathsf{Enc}}\}_l$ is the set of access control policies to encrypt l keyword groups in KS and $\{I'_{\mathsf{Enc}}\}_n$ is the set of access control policies to encrypt n data files in FS.

- $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$: With credential sk, the data user runs this algorithm to issue search token tk for keyword w.

- $(\mathsf{proof}, \mathsf{rslt}) \leftarrow \mathsf{SearchIndex}(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}, \mathsf{tk})$: The cloud uses this algorithm to perform search operations on encrypted index Index on behalf of a data user. It outputs the search result rslt and a proof proof.

- $\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{sk}, w, \mathsf{tk}, \mathsf{rslt}, \mathsf{proof})$: The data user runs this algorithm to verify that $(\mathsf{rslt}, \mathsf{proof})$ is valid with respect to search token tk.

Correctness of VABKS requires that, given $(\mathsf{mk}, \mathsf{pm}) \leftarrow \mathsf{Init}(1^\ell)$, $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{mk}, I_{\mathsf{KeyGen}})$, for any keyword-based data collection D and keyword w, $(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}) \leftarrow \mathsf{BuildIndex}(\{I_{\mathsf{Enc}}\}_l,$

$\{I'_{\mathsf{Enc}}\}_n$, $\mathsf{D})$, $\mathsf{tk} \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w)$ and $(\mathsf{proof}, \mathsf{rslt}) \leftarrow \mathsf{SearchIndex}(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}, \mathsf{tk})$, then $1 \leftarrow \mathsf{Verify}(\mathsf{sk}, w, \mathsf{tk}, \mathsf{rslt}, \mathsf{proof})$ always holds.

Intuitively, security of VABKS is specified by the following four requirements against untrusted cloud $\mathcal{A}$.

- Data secrecy: Even given encrypted keywords and search tokens, $\mathcal{A}$ still cannot learn any information (in a computational sense) about encrypted data files. This definition can be formalized similar to chosen-plaintext security game where the two challenge data $\mathsf{D}_0 = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS}_0), \mathsf{D}_1 = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS}_1)$ have the same KS and MP, and $|\mathsf{FS}_0| = |\mathsf{FS}_1|$.

- Selective security against chosen-keyword attack: Without seeing the corresponding search tokens, $\mathcal{A}$ cannot guess the keyword that is encrypted. This requirement is extended from selective security against chosen-keyword attack of ABKS.

- Keyword secrecy: Even given the encrypted data files, the probability of $\mathcal{A}$ learning the keyword from a keyword ciphertext and the search tokens is no more than that of a random guess. This security definition also can be extended from the keyword secrecy of ABKS.

- Verifiability: If $\mathcal{A}$ returns an incorrect search result, then the cheating behavior can be detected with an overwhelming probability. We formalize this security property via the following verifiability game.

Verifiability Game:

Setup: The challenger runs $(\mathsf{pm}, \mathsf{mk}) \leftarrow \mathsf{Init}(1^\ell)$. $\mathcal{A}$ selects $\mathsf{D} = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS}), \{I_{\mathsf{Enc}}\}_l$ and $\{I'_{\mathsf{Enc}}\}_n$ and sends them to the challenger. The challenger runs $(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}) \leftarrow \mathsf{BuildIndex}(\{I_{\mathsf{Enc}}\}_l$, $\{I'_{\mathsf{Enc}}\}_n$, $\mathsf{D})$, and gives $(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}})$ to $\mathcal{A}$.

Phase 1: $\mathcal{A}$ can query the following oracles for polynomially many times.

- $\mathcal{O}_{\mathsf{KeyGen}}(I_{\mathsf{KeyGen}})$: The challenger returns to $\mathcal{A}$ credential sk corresponding to I_{KeyGen}.

- $\mathcal{O}_{\mathsf{TokenGen}}(I_{\mathsf{KeyGen}}, w)$: The challenger generates credential sk with I_{KeyGen}, and returns to $\mathcal{A}$ a search token tk by running algorithm TokenGen with inputs sk and w.

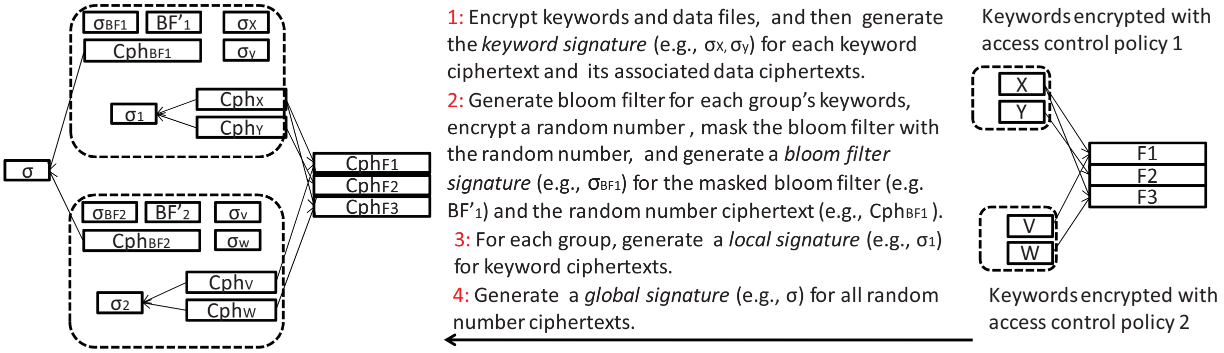
- $\mathcal{O}_{\mathsf{Verify}}(I_{\mathsf{KeyGen}}, w, \mathsf{tk}, \mathsf{rslt}, \mathsf{proof})$: The challenger generates credential sk with I_{KeyGen}, returns γ to $\mathcal{A}$ by running $\gamma \leftarrow \mathsf{Verify}(\mathsf{sk}, w, \mathsf{tk}, \mathsf{rslt}, \mathsf{proof})$.

Challenge phase: $\mathcal{A}$ selects a non-trivial challenge I^*_{Enc} and a keyword w^* and gives them to the challenger. The challenger selects I^*_{KeyGen} such that $F(I^*_{\mathsf{KeyGen}}, I^*_{\mathsf{Enc}}) = 1$, generates credential sk^* with I^*_{KeyGen} and returns to $\mathcal{A}$ a search token tk^* by running $\mathsf{tk}^* \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w^*)$.

Guess: $\mathcal{A}$ outputs $(\mathsf{rslt}^*, \mathsf{proof}^*)$ to the challenger. We say $\mathcal{A}$ wins the game if $1 \leftarrow \mathsf{Verify}(\mathsf{sk}^*, w^*,$ tk^*, rslt^*, $\mathsf{proof}^*)$ and $\mathsf{rslt}^* \neq \mathsf{rslt}$, where $(\mathsf{rslt}, \mathsf{proof})$ is produced by the challenger by running $\mathsf{SearchIndex}(\mathsf{Au}, \mathsf{Index}, \mathsf{tk}^*)$.

Definition 5. A VABKS scheme is verifiable if the advantage that any $\mathcal{A}$ wins the verifiability game is negligible in security parameter ℓ.

2.4.3 VABKS **Construction**


σBF1
BF'1
σX
CphBF1
σY
σ1
CphX
CphY
σ
CphF1
CphF2
CphF3
σBF2
BF'2
σV
CphBF2
σW
σ2
CphV
CphW
Keywords encrypted with access control policy 1
X
Y
F1
F2
F3
V
W
Keywords encrypted with access control policy 2

1: Encrypt keywords and data files, and then generate the *keyword signature* (e.g., σX, σY) for each keyword ciphertext and its associated data ciphertexts.
2: Generate bloom filter for each group's keywords, encrypt a random number, mask the bloom filter with the random number, and generate a *bloom filter signature* (e.g., σBF1) for the masked bloom filter (e.g. BF'1) and the random number ciphertext (e.g., CphBF1).
3: For each group, generate a *local signature* (e.g., σ1) for keyword ciphertexts.
4: Generate a *global signature* (e.g., σ) for all random number ciphertexts.

Figure 2.2: Basic idea for achieving verifiability, where data files F_1, F_2, F_3 were encrypted to $\mathsf{cph}_{\mathsf{F}_1}, \mathsf{cph}_{\mathsf{F}_2}, \mathsf{cph}_{\mathsf{F}_3}$, keywords X, Y were encrypted to $\mathsf{cph}_X, \mathsf{cph}_Y$ with access control policy 1, and keywords V, W were encrypted to $\mathsf{cph}_V, \mathsf{cph}_W$ with access control policy 2. Given a search token tk, for each group i, the cloud provides $(\sigma_w, \mathsf{cph}_{\mathsf{BF}_i})$ as the proof when it finds some keyword ciphertext cph_w that matches tk, and $(\mathsf{cph}_{\mathsf{BF}_i}, \mathsf{BF}'_i, \sigma_{\mathsf{BF}_i})$ otherwise.

A trivial solution is that data users download all keyword ciphertexts and check whether any of keyword ciphertexts and the search token has the same keyword, which however incurs prohibited communication and computation overhead. Instead, we let a data user verify that (1) the cloud performed search operations over all keyword ciphertexts, and (2) the cloud honestly returned

$\underline{\mathsf{Init}}(1^\ell)$: Given security parameter ℓ, the attribute authority chooses k universal hash functions $H'_1, \ldots, H'_k$, which are used to construct m-bit Bloom filter. Let $H : \{0,1\}^\ell \rightarrow \{0,1\}^m$ be a secure pseudorandom generator, SE be a secure symmetric encryption scheme, ABE be a secure ABE scheme and ABKS be a secure ABKS scheme. It executes $(\mathsf{ABE.pm}, \mathsf{ABE.mk}) \leftarrow \mathsf{ABE.Setup}(1^\ell)$ and $(\mathsf{ABKS.pm}, \mathsf{ABKS.mk}) \leftarrow \mathsf{ABKS.Setup}(1^\ell)$. It sets $\mathsf{pm} = (\mathsf{ABE.pm}, \mathsf{ABKS.pm}, H'_1, \ldots, H'_k)$ which is public known, and $\mathsf{mk} = (\mathsf{ABE.mk}, \mathsf{ABKS.mk})$.

$\underline{\mathsf{KeyGen}}(\mathsf{mk}, I_{\mathsf{KeyGen}})$: The attribute authority runs $\mathsf{ABE.sk} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.mk}, I_{\mathsf{KeyGen}})$ and $\mathsf{ABKS.sk} \leftarrow \mathsf{ABKS.KeyGen}(\mathsf{ABKS.mk}, I_{\mathsf{KeyGen}})$, sets $\mathsf{sk} = (\mathsf{ABE.sk}, \mathsf{ABKS.sk})$, and sends sk to the data owner/user over an authenticated private channel.

$\underline{\mathsf{BuildIndex}}(\{I_{\mathsf{Enc}}\}_l, \{I'_{\mathsf{Enc}}\}_n, \mathsf{D})$: The data owner runs $(\mathsf{Sig.sk}, \mathsf{Sig.pk}) \leftarrow \mathsf{Sig.KeyGen}(1^\ell)$, keeps Sig.sk private and makes Sig.pk public. Given $\mathsf{D} = (\mathsf{KS} = \{\mathsf{KS}_1, \ldots, \mathsf{KS}_l\}, \mathsf{MP} = \{\mathsf{MP}(w) | w \in \cup_{i=1}^{l} \mathsf{KS}_i\}, \mathsf{FS} = \{\mathsf{F}_1, \ldots, \mathsf{F}_n\})$, the data owner executes as follows:

1. Encrypt the data file with hybrid encryption: $\forall \mathsf{F}_j \in \mathsf{FS}$, it generates the ciphertext $\mathsf{cph}_{\mathsf{F}_j} = (\mathsf{cph}_{\mathsf{sk}_j}, \mathsf{cph}_{\mathsf{SE}_j})$ by running $\mathsf{SE.sk}_j \leftarrow \mathsf{SE.KeyGen}(1^\ell)$, $\mathsf{cph}_{\mathsf{SE}_j} \leftarrow \mathsf{SE.Enc}(\mathsf{SE.sk}_j, \mathsf{F}_j)$, and $\mathsf{cph}_{\mathsf{sk}_j} \leftarrow \mathsf{ABE.Enc}(I'_{\mathsf{Enc}j}, \mathsf{SE.sk}_j)$.

2. Encrypt each keyword and generate keyword signature: Given $\mathsf{KS}_i, 1 \leq i \leq l$, for each $w \in \mathsf{KS}_i$, it runs $\mathsf{cph}_w \leftarrow \mathsf{ABKS.Enc}(I_{\mathsf{Enc}_i}, w)$, sets $\mathsf{MP}(\mathsf{cph}_w) = \{\mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} | \mathsf{ID}_{\mathsf{F}_j} \in \mathsf{MP}(w)\}$, and generates $\sigma_w \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, \mathsf{cph}_w || \mathsf{string}(\{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} \in \mathsf{MP}(\mathsf{cph}_w)\}))$, where $\mathsf{ID}_{\mathsf{F}_j}$ and $\mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}}$ are identities for identifying data file F_j and data ciphertext $\mathsf{cph}_{\mathsf{F}_j}$, respectively.
3. Generate a bloom filter, a bloom filter signature and a local signature for each group KS_i: Let $\mathsf{BF}_i \leftarrow \mathsf{BFGen}(\{H'_1, \ldots, H'_k\}, \mathsf{KS}_i)$, $\mathsf{cph}_{\mathsf{BF}_i} \leftarrow \mathsf{ABE.Enc}(I_{\mathsf{Enc}_i}, \mathsf{M})$ by randomly selecting M from the message space of ABE, compute $\mathsf{BF}'_i = H(\mathsf{M}) \bigotimes \mathsf{BF}_i$ and generate $\sigma_{\mathsf{BF}_i} \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, \mathsf{BF}' || \mathsf{cph}_{\mathsf{BF}_i})$. Let $\sigma_i \leftarrow \mathsf{Sig.Sign}(\mathsf{Sig.sk}, \mathsf{string}(\{\mathsf{cph}_w | w \in \mathsf{KS}_i\}))$.
4. Generate the global signature: Let $\sigma = \mathsf{Sig.Sign}(\mathsf{Sig.sk}, \mathsf{cph}_{\mathsf{BF}_1} || \ldots || \mathsf{cph}_{\mathsf{BF}_l})$.
5. Let $\mathsf{Au} = (\sigma, \sigma_1, \ldots, \sigma_l, \mathsf{cph}_{\mathsf{BF}_1}, \ldots, \mathsf{cph}_{\mathsf{BF}_l}, \sigma_{\mathsf{BF}_1}, \ldots, \sigma_{\mathsf{BF}_l}, \{\sigma_w | w \in \cup_{i=1}^{l} \mathsf{KS}_i\})$, $\mathsf{Index} = (\{\mathsf{cph}_w | w \in \cup_{i=1}^{l} \mathsf{KS}_i\}, \{\mathsf{MP}(\mathsf{cph}_w) | w \in \cup_{i=1}^{l} \mathsf{KS}_i\})$ and $\mathsf{D}_{\mathsf{cph}} = (\{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{F}_j \in \mathsf{FS}\})$.

$\underline{\mathsf{TokenGen}}(\mathsf{sk}, w)$: Given credentials sk, a data user generates search token $\mathsf{tk} \leftarrow \mathsf{ABKS.TokenGen}(\mathsf{ABKS.sk}, w)$.

$\underline{\mathsf{SearchIndex}}(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}, \mathsf{tk})$: Let rslt be an empty set and $\mathsf{proof} = (\sigma)$ initially. The cloud enumerates $\prod_i = \{\mathsf{cph}_w | w \in \mathsf{KS}_i\}, 1 \leq i \leq l$, which are the keyword ciphertexts with respect to the same access control policy:

1. For each $\mathsf{cph}_w \in \prod_i$, it runs $\gamma \leftarrow \mathsf{ABKS.Search}(\mathsf{cph}_w, \mathsf{tk})$. If $\gamma = 0$, then it continues to next keyword ciphertext in $\prod_i$; otherwise it adds the tuple $(\mathsf{cph}_w, \{\mathsf{cph}_{\mathsf{F}_j} | \mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} \in \mathsf{MP}(\mathsf{cph}_w)\})$ to rslt and $(\sigma_w, \mathsf{cph}_{\mathsf{BF}_i})$ to proof.
2. If there exist no $\gamma = 1$ after enumerating all cph_w in $\prod_i$, then add $(\mathsf{BF}'_i, \mathsf{cph}_{\mathsf{BF}_i}, \sigma_{\mathsf{BF}_i})$ to proof.

Figure 2.3: Init, KeyGen, BuildIndex, TokenGen and searchindex in VABKS construction.

the search result for each group, which could be either null or one keyword ciphertext and its associated data ciphertexts. The basic idea is further illustrated in Figure 2.2. More specifically, the data owner uses the signatures and bloom filters as follows:

- The first type of signature, called *keyword signature*, is generated for each keyword ciphertext and its associated data ciphertexts. This prevents the cloud from returning incorrect data ciphertexts in question as the search result.

- For each group, one bloom filter is built from its keywords. This allows data users to *efficiently* check that the queried keyword was indeed not in the keyword group when the cloud returns a null search result, *without* downloading all keyword ciphertexts from the cloud. A random number is selected and encrypted with the same access control policy as keywords. The random number masks the bloom filter for preserving keyword privacy. The second type of signature, called *bloom filter signature*, is generated for the masked bloom filter and the random number ciphertext for assuring their integrity.

- The third type of signature, called *global signature*, is obtained by signing random number ciphertexts of all groups. It allows a data user to verify integrity of all random number ciphertexts.

- The fourth type of signature, called *local signature*, is generated for all keyword ciphertexts within the same group. This signature is to validate integrity of keyword ciphertexts within the group.

The VABKS scheme is depicted in Figure 2.3 and Figure 2.4, making use of a signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, a symmetric encryption scheme $\mathsf{SE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, an ABE scheme $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, both of which are used to encrypt data files. The VABKS scheme is extended from an ABKS scheme $\mathsf{ABKS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{TokenGen}, \mathsf{Search})$, which is to encrypt keywords. Here ABE and ABKS are either ciphertext-policy or key-policy, meaning that we have two variants of VABKS.

$\underline{\mathsf{Verify}}(\mathsf{sk}, w, \mathsf{tk}, \mathsf{proof}, \mathsf{rslt})$: The data user verifies the search result from the cloud as follows:

1. Verify integrity of random number ciphertexts: $\gamma = \mathsf{Sig.Verify}(\mathsf{Sig.pk}, \sigma, \mathsf{cph}_{\mathsf{BF}_1}||\ldots||\mathsf{cph}_{\mathsf{BF}_l})$. If $\gamma = 0$, then return 0; otherwise, continue to execute the following.

2. For $i = 1, \ldots, l$, it executes as follows to verify that the cloud indeed returned correct result for each group:

   Case 1: If $(\mathsf{cph}_w, \{\mathsf{cph}_{\mathsf{F}_j}|\mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} \in \mathsf{MP}(\mathsf{cph}_w)\}) \in \mathsf{rslt}$, where cph_w corresponds to the same access control policy as what is specified by $\mathsf{cph}_{\mathsf{BF}_i}$, then it runs $\gamma \leftarrow \mathsf{ABKS.Search}(\mathsf{cph}_w, \mathsf{tk})$ and $\gamma' \leftarrow \mathsf{Sig.Verify}(\mathsf{Sig.pk}, \sigma_w, \mathsf{cph}_w||\mathsf{string}(\{\mathsf{cph}_{\mathsf{F}_j}|\mathsf{ID}_{\mathsf{cph}_{\mathsf{F}_j}} \in \mathsf{MP}(\mathsf{cph}_w)\}))$ to verify whether or not cph_w matches $\mathtt{tk}$ and all the associated data ciphertexts are returned by the cloud. If either $\gamma = 0$ or $\gamma' = 0$, then return 0, otherwise it continues to $i = i + 1$.

   Case 2: If $(\mathsf{BF}'_i, \mathsf{cph}_{\mathsf{BF}_i}, \sigma_{\mathsf{BF}_i}) \in \mathsf{rslt}$, then it continues to verify integrity of the masked Bloom filter by running $\gamma' \leftarrow \mathsf{Sig.Verify}(\mathsf{Sig.pk}, \sigma_{\mathsf{BF}_i}, \mathsf{BF}'_i||\mathsf{cph}_{\mathsf{BF}_i})$. If $\gamma' = 0$, return 0; otherwise it executes below:

   - If the data user is authorized, it computes $\mathsf{M} \leftarrow \mathsf{ABE.Dec}(\mathsf{ABE.sk}, \mathsf{cph}_{\mathsf{BF}_i})$, $\mathsf{BF}_i = H(\mathsf{M}) \bigotimes \mathsf{BF}'_i$. It executes $\delta \leftarrow \mathsf{BFVerify}(\{H'_1, \ldots, H'_k\}, \mathsf{BF}_i, w)$ to check whether w is present in the keyword group represented by BF_i.
     - If $\delta = 0$, meaning w is not present in the keyword group represented by BF_i, then it continues to $i = i + 1$.
     - If $\delta = 1$, it downloads $\prod_i = \{\mathsf{cph}_w|w \in \mathsf{KS}_i\}$ and σ_i from the cloud, and runs $\eta \leftarrow \mathsf{Sig.Verify}(\mathsf{Sig.pk}, \sigma_i, \mathsf{string}(\{\mathsf{cph}_w|w \in \mathsf{KS}_i\}))$. If $\eta = 0$, it returns 0; otherwise it runs $\tau \leftarrow \mathsf{ABKS.Search}(\mathsf{cph}_w, \mathsf{tk})$ by enumerating cph_w in $\prod_i$. If there exists some $\tau = 1$ after enumerating all cph_w (meaning there exists some cph_w matches tk), it returns 0; otherwise it continues to $i = i + 1$.
   - If the data user is unauthorized, then it continues to $i = i + 1$ because $\mathsf{cph}_{\mathsf{BF}_i}$ cannot be decrypted.

   Case 3: Return 0 since the verification should be executed in either case above if the search result is correct.

3. Return 1 if all tuples in the search result have been verified, and 0 otherwise.

Figure 2.4: Verify algorithm in VABKS construction.

In algorithm Verify of Figure 2.4, note that when an authorized data user verifies a null search result for the group $\{\mathsf{cph}_w|w \in \mathsf{KS}_i\}$ with respect to a search token that corresponds to keyword w', it can happen that $1 \leftarrow \mathsf{BFVerify}(\{H'_1, \ldots, H'_k\}, \mathsf{BF}_i, w')$ but the matched keyword ciphertext was not stored in the cloud due to the false-positive of the Bloom filter. To validate the search result in this case, algorithm Verify has to download $\{\mathsf{cph}_w | w \in \mathsf{KS}_i\}$, and checks one by one. We stress that this does not incur any significant unnecessary communications from the perspective of amortization because we can set the false-positive rate as low as possible by choosing appropriate m and k (i.e., the “wasted" bandwidth communication and computation cost are proportional to this false-positive rate). For example, in our experiment we set the false-positive rate to be 4.5×10^{-9}.

2.4.4 Security Analysis

We show that the VABKS scheme satisfies the security requirements with the following theorems.

We show that if there exists polynomial time algorithm $\mathcal{A}$ breaking VABKS's data secrecy, then it breaks the assumption that CPA-secure $\mathsf{ABE}$and CPA-secure SE. This is formally achieved via Theorem 5.

Theorem 5. *If the* ABE *and* SE *are secure against chosen-plaintext attack, the* VABKS *scheme achieves data secrecy.*

Proof. We show that if there exists a polynomial-time algorithm $\mathcal{A}$ breaks VABKS's data secrecy with the advantage ρ, then we can break either CPA security for ABE or CPA security for SE with the advantage $\frac{\rho}{n^2}$ where n is the number of data files to be encrypted.

The challenger proceeds the conventional CPA security game with $\mathcal{A}$. In the challenge phase, suppose $\mathcal{A}$ presents two data collections $\mathsf{D}_0 = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS}_0 = \{\mathsf{F}_{01}, \ldots, \mathsf{F}_{0n}\})$, $\mathsf{D}_1 = (\mathsf{KS}, \mathsf{MP}, \mathsf{FS}_1 = \{\mathsf{F}_{11}, \ldots, \mathsf{F}_{1n}\})$, $\{I_{\mathsf{Enc}}\}_l$ and $\{I'_{\mathsf{Enc}}\}_n$. The challenge selects $\lambda \leftarrow \{0, 1\}$ and encrypts FS_λ with the ABE and $\{I'_{\mathsf{Enc}}\}_n$.

Now let us consider the advantage of $\mathcal{A}$ correctly guessing λ. As we know, given two messages, the advantage of distinguishing which message was encrypted by the hybrid encryption of ABE and SE is equal. Therefore, given two sets of data files FS_0 and FS_1, if the advantage of distinguishing which data set was encrypted is ρ, then the advantage of distinguishing which data file was encrypted is $\frac{\rho}{n^2}$ by selecting one data file from FS_0 and one from FS_1.

Therefore, we can see that if $\mathcal{A}$ breaks VABKS's data secrecy of with a non-negligible advantage ρ, then the advantage of breaking CPA security for ABE or CPA security for SE is $\frac{\rho}{n^2}$

–a non-negligible probability, which contracts the assumption that ABE is CPA-secure and SE is CPA-secure. □

We show that if there exists polynomial time algorithm $\mathcal{A}$ breaking VABKS's selective security against chosen keyword attack, then it breaks the assumption that ABKS achieves the selective security against chosen keyword attack, given that ABE is CPA-secure and H is a secure pseudorandom generator. This is formally achieved via Theorem 6.

Theorem 6. *If* ABE *is secure against chosen-plaintext attack,* H *is a secure pseudorandom generator and the* ABKS *is selectively secure against chosen keyword attack, the* VABKS *scheme is selectively secure against chosen-keyword attack.*

Proof. We show that if there exists a polynomial-time algorithm $\mathcal{A}$ breaks the selective security against chosen-keyword attack of ABKS with the advantage ρ, then we can break the selective security against chosen-keyword attack game of ABKS with the advantage of $\frac{\rho}{l^2}$, given that ABE is CPA-secure and H is a secure pseudorandom generator.

The challenger proceeds selective security against chosen-keyword attack game with $\mathcal{A}$. In the challenge phase, suppose $\mathcal{A}$ presents two data collections $\mathsf{D}_0 = (\mathsf{KS}_0 = \{\mathsf{KS}_{01}, \ldots, \mathsf{KS}_{0l}\}, \mathsf{MP}, \mathsf{FS})$, and $\{I'_{\mathsf{Enc}}\}_n$. The challenge selects $\lambda \leftarrow \{0,1\}$ and encrypts KS with ABKS, and generates $\mathsf{BF}'_i, \mathsf{cph}_{\mathsf{BF}_i}$ and σ_i for each keyword group.

Since ABE is CPA-secure and H is a secure pseudorandom generator, the probability of $\mathcal{A}$ inferring λ via $\mathsf{BF}'_i, \mathsf{cph}_{\mathsf{BF}_i}$ is negligible. Then let us consider the advantage of $\mathcal{A}$ correctly guessing λ from keyword ciphertexts. As we know, given two keywords, the advantage of distinguishing which keyword was encrypted by ABKS is equal. Therefore, given two keyword sets KS_0 and KS_1, if the advantage of distinguishing which keyword set was encrypted is ρ, then the advantage of distinguishing which keyword was encrypted is $\frac{\rho}{l^2}$ by selecting one keyword from KS_0 and one from KS_1.

Therefore, we can see that if $\mathcal{A}$ breaks VABKS's selective security against chosen-keyword attack with a non-negligible advantage ρ, then the advantage of breaking ABKS's selective security against chosen-keyword attack is $\frac{\rho}{l^2}$ –a non-negligible probability, which contracts the assumption that ABKS achieve selective security against chosen-keyword attack, given that ABE is CPA-secure and H is a secure pseudorandom generator. □

We show that if there exists polynomial time algorithm $\mathcal{A}$ breaking keyword secrecy of the VABKS, then it breaks the assumption that ABKS achieves keyword secrecy, given that ABE is CPA-secure and H is a secure pseudorandom generator. This is formally achieved via Theorem 7.

Theorem 7. *If* ABE *is secure against chosen-plaintext attack, H is a secure pseudorandom generator and the* ABKS *achieves keyword secrecy, the* VABKS *scheme also achieves keyword secrecy.*

Proof. We show that if there exists polynomial time algorithm $\mathcal{A}$ breaking VABKS'skeyword secrecy, then it breaks the assumption that ABKS achieves keyword secrecy.

Suppose $\mathcal{A}$ presents a data collection $\mathsf{D} = (\mathsf{KS} = \{\mathsf{KS}_1, \ldots, \mathsf{KS}_l\}, \mathsf{MP}, \mathsf{FS})$, $\{I_{\mathsf{Enc}}\}_l$ and $\{I'_{\mathsf{Enc}}\}_n$. The challenger simulates the keyword secrecy game, where the keyword space consists of keywords specified by FS. We can see that the probability of $\mathcal{A}$ inferring the keyword from a search token and corresponding keyword ciphertext is equal to that of ABKS. Therefore, if in $\mathsf{VABKS}\mathcal{A}$ guesses the keyword from the search token and corresponding keyword ciphertext with the probability more than $\frac{1}{|\mathcal{M}|-q} + \epsilon$ after guessing q distinct keywords, then the probability of guessing the keyword from the search token and keyword ciphertext in ABKS is more than $\frac{1}{|\mathcal{M}|-q} + \epsilon$ after guessing q distinct keywords, which contracts the assumption that ABKS achieves keyword secrecy. □

We show that if there exists polynomial time algorithm $\mathcal{A}$ breaking verifiability of the VABKS then it breaks the Sig's unforgeability. This is formally achieved via Theorem 8.

Theorem 8. *If* Sig *is a secure signature, the* VABKS *construction achieves the verifiability.*

Proof. We show that under the assumptions that Sig is unforgeable, any polynomial-time adversary $\mathcal{A}$ presents an incorrect search result and succeeds in the verification with negligible probability.

The challenger proceeds the verifiability game, where $\mathcal{A}$ provides the keyword-based data $\mathsf{D} = (\mathsf{KS} = \{\mathsf{KS}_1, \ldots, \mathsf{KS}_l\}, \mathsf{MP} = \{\mathsf{MP}(w) | w \in \cup_{i=1}^{l} \mathsf{KS}_i\}, \mathsf{FS} = \{\mathsf{F}_1, \ldots, \mathsf{F}_n\})$, $\{I_{\mathsf{Enc}}\}_l$ and $\{I'_{\mathsf{Enc}}\}_n$. The challenger runs $(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}}) \leftarrow \mathsf{BuildIndex}(\{I_{\mathsf{Enc}}\}_l, \{I'_{\mathsf{Enc}}\}_n, \mathsf{D})$, and gives $(\mathsf{Au}, \mathsf{Index}, \mathsf{D}_{\mathsf{cph}})$ to $\mathcal{A}$.

In the challenge phase, with w^* and I^*_{Enc} from $\mathcal{A}$, the challenger selects I^*_{KeyGen} such that $F(I^*_{\mathsf{KeyGen}}, I^*_{\mathsf{Enc}}) = 1$ where I^*_{Enc} is selected by $\mathcal{A}$, generates credential sk^* with I^*_{KeyGen} and returns to $\mathcal{A}$ a search token tk^* by running $\mathsf{tk}^* \leftarrow \mathsf{TokenGen}(\mathsf{sk}, w^*)$. $\mathcal{A}$ returns $(\mathsf{rslt}^*, \mathsf{proof}^*)$ to the challenger.

Suppose that $(\mathsf{rslt}^*, \mathsf{proof}^*)$ succeeds in the verification. That is, $1 \leftarrow \mathsf{Verify}(\mathsf{sk}^*, w^*, \mathsf{tk}^*, \mathsf{rslt}^*, \mathsf{proof}^*)$. Let us consider the probability of $\mathcal{A}$ cheating with incorrect search result.

First, we claim that the global signature σ and random keyword ciphertexts $\mathsf{cph}_{\mathsf{BF}_1}, \ldots, \mathsf{cph}_{\mathsf{BF}_l}$ are included in proof^* without being manipulated; otherwise we can break the unforgeability of Sig.

Second, let us consider the search result within each group with respect to access control policies, i.e. $i = 1, \ldots, l$:

- If there exists no keyword ciphertext matched the search token tk^*, then we claim that $\mathcal{A}$ cannot cheat the challenger with some keyword ciphertext and data ciphertexts in order to make $\mathsf{VABKS.Verify}$ output 1. The reason is that $\mathcal{A}$ cannot forge a keyword signature σ_w for the keyword ciphertext and data ciphertexts; otherwise, we can break the unforgeability of Sig.

- If there exists a keyword ciphertext matched the search token tk^*, then we claim that $\mathcal{A}$ cannot cheat the challenger with a null search result in order to make $\mathsf{VABKS.Verify}$ output 1. Suppose $\mathcal{A}$ returns a null result and the proof $(\mathsf{BF}'_i, \mathsf{cph}_{\mathsf{BF}_i}, \sigma_{\mathsf{BF}_i})$. Since BF'_i cannot be manipulated due to σ_{BF_i}, the unmasked bloom filter indicates that w^* is a member within the group. The challenger downloads $\mathsf{cph}_{w_1}, \ldots, \mathsf{cph}_{w_{|\mathsf{KS}_i|}}$ and σ_i without being manipulated; otherwise we break the Sig's unforgeability. Then the challenger can conduct the search operation with each keyword ciphertext, and VABKS.Verify will output 0. That is, if there exists keyword ciphertext matched the search token, $\mathcal{A}$ returns a null result, then it cannot make VABKS.Verify output 1.

To sum up, in order to make VABKS.Verify output 1, $\mathcal{A}$ has to faithfully execute search operations and return the search result honestly; otherwise, we will break Sig's unforgeability. □

2.5 Performance Evaluation

We evaluate efficiency of the ABKS schemes in terms of both asymptotic complexity and actual execution time, and efficiency of the VABKS scheme in terms of actual execution time. We do not consider the asymptotic complexity of VABKS because it uses multiple building-blocks (e.g., the signature and ABE scheme) that can be instantiated with any secure ones. Asymptotic complexity is measured in terms of four kinds of operations: H_1 denotes the operation of mapping a bit-string to an element of G, Pair denotes the pairing operation, E denotes the exponentiation operation in G, and E_T denotes the exponentiation operation in G_T. We ignore multiplication and hash operations (other than H_1) because they are much more efficient than the above operations [3].

We implemented ABKS and VABKS in JAVA based on the Java Pairing Based Cryptography library (jPBC) [3]. In our implementations, we instantiated the bilinear map with Type A pairing ($\ell = 512$), which offers a level of security that is equivalent to 1024-bit DLOG [3]. For both CP-VABKS and KP-VABKS, we instantiated the symmetric encryption scheme with AES-CBC, and the signature scheme with DSA signature scheme provided by JDK1.6. We instantiated ABKS, ABE with CP-ABKS, CP-ABE [18] for CP-ABKS, and KP-ABKS, KP-ABE [65] for KP-VABKS, respectively. Finally, we set the example access control policy as “at_1 AND . . . AND at_N."

2.5.1 Efficiency of ABKS

Asymptotic Complexity of the ABKS Schemes. Table 2.2 describes the asymptotic complexities of the ABKS schemes. We observe that in the CP-ABKS scheme, the complexity of KeyGen is almost the same as that of Enc. In the KP-ABKS scheme, KeyGen is more expensive than Enc. In both schemes, the two Search algorithms incur almost the same cost.

Table 2.2: Asymptotic complexities of CP-ABKS and KP-ABKS, where S is the number of a data user's attributes and N is the number of attributes that are involved in a data owner's access control policy (i.e., the number of leaves in the access tree).

KP-ABKS	KeyGen	$3N\mathsf{E} + NH_1$	$2N\lvert G\rvert$
	Enc	$(S+4)\mathsf{E} + SH_1$	$(S+3)\lvert G\rvert$
	TokenGen	$(2N+2)\mathsf{E}$	$(2N+2)\lvert G\rvert$
	Search	$(2S+2)\mathsf{Pair} + S\mathsf{E}_T$	
		complexity	output size
CP-ABKS	KeyGen	$(2S+2)\mathsf{E} + SH_1$	$(2S+1)\lvert G\rvert$
	Enc	$(2N+4)\mathsf{E} + NH_1$	$(2N+3)\lvert G\rvert$
	TokenGen	$(2S+4)\mathsf{E}$	$(2S+3)\lvert G\rvert$
	Search	$(2N+3)\mathsf{Pair} + N\mathsf{E}_T$	

ABKS **Performance.** To evaluate the performance of the ABKS schemes, we ran the experiments on a client machine with Linux OS, 2.93GHz Intel Core Duo CPU (E7500), and 2GB RAM. We varied N, the number of attributes that are involved in the example access control policy, from 1 to 50 with step length 10. We ran each experiment for 10 times to obtain the average execution time. Table 2.3 shows the execution time of the two ABKS schemes.

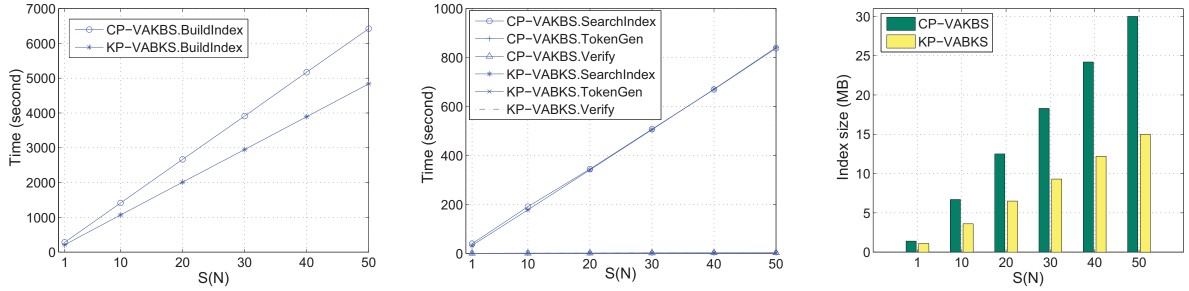
Table 2.3: Execution time (second) of the algorithms in the KP -ABKS and CP -ABKS schemes, where N is the number of attributes involved in the example access control policy. The number of data user's attributes is also set to N, namely $S = N$ in the experiments.

KP-ABKS	KeyGen	0.088	0.786	1.539	2.316	3.081	3.863
	Enc	0.108	0.539	1.016	1.492	1.983	2.434
	TokenGen	0.073	0.331	0.627	0.917	1.211	1.504
	Search	0.049	0.275	0.480	0.711	0.947	1.182
	S(N)	1	10	20	30	40	50
CP-ABKS	KeyGen	0.107	0.686	1.275	1.901	2.525	3.151
	Enc	0.121	0.681	1.304	1.923	2.546	3.169
	TokenGen	0.088	0.349	0.673	0.932	1.228	1.513
	Search	0.061	0.329	0.493	0.728	0.97	1.202

From Table 2.3 we observe that for both schemes, the keyword encryption algorithm Enc (run by the data owner) is more expensive than that of the keyword search algorithm Search (run by the cloud) with the same N. However, the keyword encryption algorithm is executed only once for each keyword, whereas the keyword search algorithm will be performed as many times as needed. Furthermore, we advocate the burden on the data users be further eased by outsourcing the keyword search operation to the cloud (i.e., taking advantage of the cloud's computational resources).

2.5.2 Efficiency of VABKS with Real Data

To demonstrate the feasibility of VABKS in practice, we evaluated it with real data, which consists of 2,019 distinct keywords extracted from 670 PDF documents (papers) from the ACM Digital Library with a total size of 778.1MB. We set $k = 28$ and $m = 10KB$ for Bloom filter so that $\frac{m}{n} = \frac{10*8*1024}{2019} \approx 40$ and the false-positive rate is around 4.5×10^{-9}. We vary the access control policy ranging from 1 to 50 attributes with step-length 10. In each experiment, we encrypted all keywords with the same access control policy. The algorithms run by the data owner and the data users (i.e. BuildIndex, TokenGen and Verify) were executed on a client machine with Linux OS, 2.93GHz Intel Core Duo CPU (E7500), and 2GB RAM. The algorithm run by the cloud (i.e., SearchIndex) was executed on a server machine (a laptop) with Windows 7, Intel i5 2.60GHz CPU, and 8GB RAM.



(a) BuildIndex

(b) TokenGen, SearchIndex and Verify

(c) Size of index and auxiliary information

Figure 2.5: Performance of the CP-VABKS and KP-VABKS schemes, where N is the number of attributes involved in the example access control policy. The number of data user's attributes is also set to N, namely $S = N$ in the experiments.

Figure 2.5a shows the execution time of BuildIndex that was run by the data owner. We observe that with the same attribute/policy complexity, CP-VABKS is more costly than that of KP-VABKS when running algorithm BuildIndex. Figure 2.5b plots the execution time of the algorithms run by the data user and the cloud. We simulated that algorithm SearchIndex needs to conduct search operations on 1,010 keyword ciphertexts to find the matched keyword ciphertext. We observe that the execution time of TokenGen and Verify is really small, when compared with keyword search algorithm SearchIndex. This again confirms that the data user should outsource keyword search operations to the cloud. Figure 2.5c plots the size of index and auxiliary information, including 2,019 keyword ciphertexts, bloom filter and signatures. We also see that CP-VABKS consumes around two times more storage space than KP-VABKS with the same attribute/policy complexity. These discrepancies should serve as a factor when deciding whether to use CP-VABKS or KP-VABKS in practice.

2.6 Chapter Summary

We have introduced a novel cryptographic solution called verifiable attribute-based keyword search, for secure cloud computing on outsourced encrypted data. The solution achieves the following: Data owners can control the search and use of their outsourced encrypted data according to their access control policies, while authorized data users can outsource the often costly search operations to the cloud and forces the cloud to faithfully execute the search operations. Performance evaluation shows that the new tool is practical. Our study focused on static data, and one interesting open problem for future research is to accommodate dynamic data.

Chapter 3: VERIFIABLE DELEGATED SET INTERSECTION ON OUTSOURCED ENCRYPTED DATA

3.1 Introduction

Cloud computing allows users to outsource their data to the cloud, but the data privacy issue often makes them reluctant to do so. It is therefore natural to encrypt the outsourced data and delegate the heavy-duty computational tasks on the outsourced encrypted data to the cloud. This leads to a general question: How can the cloud execute the delegated functions on outsourced encrypted data, without being given the decryption capability? Although Fully Homomorphic Encryption (FHE) [25,56,57] is promising to tackle this problem, it is not practical enough for applications that involve a large volume of data [81]. Moreover, FHE in general does not solve another important problem: How can we force the cloud to execute the delegated computational functions honestly? This calls for solutions that can hold the misbehaving cloud accountable.

In this chapter, we consider the problem of Verifiable Delegated Set Intersection on outsourced encrypted data (VDSI), which can be seen as the cloud version of the well investigated problem of Private Set Intersection (PSI) [12, 41, 88]. In the setting of PSI, two parties jointly compute the intersection of their private data sets such that they learn the intersection set but nothing else (the sizes of their private data sets may or may not be deemed as confidential [8]).

In the setting of VDSI, two cloud users, Alice and Bob, outsource their encrypted private data sets to the cloud. They would like to conduct the set intersection operation on their plaintext data sets. The straightforward solution would be for them to download their outsourced ciphertexts, decrypt the ciphertexts locally, and then execute a commodity two-party set intersection protocol. The straightforward solution is not practical, especially when the outsourced data sets are large and when they use wireless systems such as smartphones. Another drawback of this solution is that both Alice and Bob must participate simultaneously. For these reasons, Alice and Bob would prefer delegating the set intersection operation to the cloud, while being able to hold the misbehaving cloud accountable. Note that it is realistic to assume that the cloud is untrusted because it has the incentive not to honestly execute the protocols (e.g., for saving resources or shortening service response time). Moreover, the cloud may have been compromised and the attacker may return Alice and Bob with misleading results.

3.1.1 Our Contribution

We initiate the investigation of a novel notion called VDSI, a useful primitive for delegating the set intersection operation on outsourced ciphertexts to the untrusted cloud. In contrast to the straightforward solution mentioned above, VDSI solves the problem by enabling the cloud to compute the set intersection, but *without* giving the decryption capability to the cloud. As such, VDSI can be seen as a special-purpose homomorphic cryptographic system for use in cloud computing. Since the cloud is untrusted and possibly malicious, VDSI allows Alice and Bob to verify whether the cloud has faithfully computed the delegated set intersection protocol or not.

Specifically, we formally define security properties of VDSI, and present a concrete VDSI scheme. The scheme is based on two ideas: (i) using proxy re-encryption to enable the cloud to compare equality of plaintexts corresponding to two ciphertexts that are encrypted using different public keys; (ii) using a novel variant of cryptographic accumulator, which can be used to verify the membership of *multiple* elements through a single examination and may be of independent value, to allow the cloud to show the correctness of the resulting intersection set.

Our VDSI scheme has two appealing features. First, it does not require the participation of Alice and Bob, because the cloud conducts the delegated computing. Second, it is much more efficient than the straightforward solution mentioned above. Suppose Alice's (Bob's) data set has n (m) elements, and the intersection set has k elements. Our solution only incurs $O(k)$ computational and communication costs on Alice and Bob, for decrypting and verifying the results received from the cloud. This means that our solution is optimal (up to a constant factor). In contrast, the straightforward solution incurs $O(m + n)$ computational and communication costs on Alice and Bob. Note that it is possible that $m + n >> k$. Experimental evaluation confirms that the VDSI scheme is practical,

We believe that the novel concept of VDSI will inspire many fruitful studies. For example, our solution only achieves a “weak" version of the *function output secrecy* property, which allows the untrusted cloud to launch a *plaintext guessing* attack against Alice’s and Bob’s private data (i.e., the success probability depends on the size of the plaintext space). It is an outstanding open problem to settle down whether or not this weak guarantee is inherent to the problem that VDSI aims to solve; if not, we need to design a better solution that is immune to this attack. Another outstanding open problem is to enforce fine-grained access control over the delegated set intersection operation, which may or may not need to be traded from the verifiability.

3.1.2 Related Work

To the best of our knowledge, this is the first work that considers the PSI problem in the cloud computing setting, where cloud users not only outsource their private data but also outsource their set intersection operations, while being able to hold the dishonest cloud vendors accountable for not faithfully executing the delegated operations. Nevertheless, there are prior studies on related problems.

Private Set Intersection. The PSI (private set intersection) problem was initiated in [52] and has become an essential building-block for many applications. Many variants of PSI [8, 28, 41, 44, 47, 67–70, 79] have been proposed, with various features (e.g., preventing a malicious party from choosing arbitrary inputs [28,41], hiding the sizes of the inputs [8]). There have been schemes that aim to reduce the computational and communication complexities (e.g., the RSA-OPRF-based protocol [42], the garbled circuit protocol [69], and the garbled bloom filter protocol [47]). Among the state-of-the-art PSI solutions, the most efficient PSI protocol incurs $O(m + n)$ computational and communication complexities, where m and n are sizes of the respective data sets [47]. We note that [72, 77, 78] considered the problem of server-aided private set intersection, where cloud users share some secrets with each other to preprocess data sets at the time of outsourcing their data. Such collaborative preprocessing is not needed in the setting of VDSI. Finally, a recent work [31] studies verifiable complex set operations over outsourced plaintext data sets (i.e., the outsourced data is not encrypted). In contrast, we consider outsourced computing on outsourced encrypted data.

Public Key Encryption with Equality Test. The problem of public key encryption with equality test is to decide whether two ciphertexts that are encrypted using two different public keys correspond to the same plaintext or not [4, 29, 118, 123]. In order to enforce access control over the equality test operation, a variant of the problem is to allow the data owners to authorize who can perform the equality test on the outsourced encrypted data [116]. These protocols do not consider the requirement of verifiability on the equality test results, which is crucial to VDSI in the present chapter.

Verifiable Computation. How to securely and efficiently delegate the computation of a function to a remote server has been under active research [17, 33, 40, 50, 54, 55, 61, 102, 107, 117]. In these solutions, the data owner pre-processes the inputs to the delegated function in question before outsourcing the data to the cloud, and the cloud needs to prove the correctness of the outcome of a function execution. However, these solutions do not solve the problem studied in this chapter because (i) the input to their functions is from a single source and known to the delegator in advance, and (ii) some solutions do not consider privacy of the input. In contrast, our model has the following characteristics: the inputs to the delegated functions include other data owners' private data sets, which are not known to the delegators in advance. Finally, [127] considered the notion of verifiable private multi-party computation, but not in the setting of outsourcing data and functions to the cloud.

3.2 Cryptographic Preliminaries

Let $(e, g, G, G_T, p) \leftarrow \mathsf{MapGen}(1^\ell)$ denote that the bootstrapping algorithm MapGen generates a bilinear map $e : G \times G \rightarrow G_T$, where G and G_T are cyclic groups of order p which is an ℓ-bit prime, g is a generator of G, and the bilinear map e satisfies (i) for $a, b \in \mathbb{Z}_p$, $e(g^a, g^b) = e(g, g)^{ab}$, (ii) $e(g, g)$ is non-degenerate, and (iii) e can be efficiently computed. The bilinear map e is one-way, i.e. the probability of a probabilistic polonomial algorithm inverting e is negligible, which holds when G and G_T are instantiated with Weil or Tate pairing over MNT curves [21]. Table **??** summaries the notions for the algorithms and parameters in the VDSI scheme, multi-accumulator scheme and the signature scheme Sig.

Bilinear q-strong Diffie-Hellman assumption (q-SDH) [21]. For given $(e, g, G, G_T, p) \leftarrow \mathsf{MapGen}(1^\ell)$, and $g^\alpha, g^{\alpha^2}, \dots, g^{\alpha^q}$ where $\alpha \overset{R}{\leftarrow} \mathbb{Z}_p$ and q is bounded by a polynomial in ℓ, there exists no probabilistic polynomial-time algorithm $\mathcal{A}$ that can compute $(s, e(g, g)^{1/(\alpha+s)})$ where $s \in \mathbb{Z}_p$ with a non-negligible probability in ℓ. The probability is defined over the random choices of the parameters and random coins used by $\mathcal{A}$.

Decisional Linear assumption (DL) [21]. For given $(e, g, G, G_T, p) \leftarrow \mathsf{MapGen}(1^\ell)$, and $(f, h, g^{r_1}, f^{r_2}, Q)$ where $f, h, Q \overset{R}{\leftarrow} G$ and $r_1, r_2 \overset{R}{\leftarrow} \mathbb{Z}_p$, there exists no probabilistic polynomial-time algorithm $\mathcal{A}$ that can determine $Q \overset{?}{=} h^{r_1+r_2}$ with a non-negligible advantage, where "advantage" is defined as

$$\Pr[\mathcal{A}(g, f, h, g^{r_1}, f^{r_2}, Q) = 1] - \Pr[\mathcal{A}(g, f, h, g^{r_1}, f^{r_2}, h^{r_1+r_2}) = 1],$$

and the probability is defined over the random choices of the parameters and random coins used by $\mathcal{A}$.

Unforgeable Digital Signature. Let $\mathsf{Sig} = (\mathsf{sigKeyGen}, \mathsf{sigSign}, \mathsf{sigVerify})$ be a secure signature scheme, where sigKeyGen generates a pair of public and private keys, sigSign generates a signature for a message, and sigVerify determines if a message matches a signature. Any signature scheme satisfying the standard definition of unforgeability under adaptive chosen-message attacks [62] is sufficient for the purpose of this chapter.

Table 3.1: Notations for algorithms and parameters in the VDSI, multi-accumulator and the signature scheme Sig.

Notation	Description
$\mathsf{D}_a, \mathsf{C}_a$	Alice's data set and its encryption form
$\mathsf{D}_b, \mathsf{C}_b$	Bob's data set and its encryption form
Setup, KeyGen, Enc, Dec, AuGen, SetOp, Verify	algorithms of the VDSI
pm, sk, pk, si, au, rslt, proof	parameters of the VDSI
acKeyGen, acGen, acProve, acVerify	algorithms of the multi-accumulator
acSk, acPk, acDig, acRslt, acWit	parameters of the multi-accumulator
sigKeyGen, sigSign, sigVerify	algorithms of signature scheme Sig
sigSk, sigPk, σ	parameters of the signature scheme

3.3 VDSI Model and Definition

3.3.1 System Model

Figure 4.1 illustrates the *system model* of VDSI (verifiable delegated set intersection operations on outsourced encrypted data). The system has four entities: a trusted third party, a cloud, and two cloud users (i.e., data owners) referred to as Alice and Bob. The trusted third party is responsible for initializing system public parameters used by the cloud and cloud users. Alice and Bob can be either individuals or organizations that outsource their private data sets, denoted by D_a and D_b, to the cloud in encrypted form, denoted by C_a and C_b, respectively. Alice and Bob want to compute the intersection set $\mathsf{D}_a \cap \mathsf{D}_b$, by delegating the set intersection operation to the cloud but *without* giving the cloud the capability to decrypt C_a and C_b.

Remark. The above system model can be easily extended to accommodate the following more general scenarios. First, rather than letting Alice and Bob outsource their encrypted data to the same cloud, they can outsource their encrypted data to two different clouds (dubbed *storage clouds*). Second, rather than letting (one of) the storage cloud(s) conduct the delegated computation on C_a and C_b, another cloud (dubbed *computing* cloud) or any other third party can be used for this purpose. The extension is trivial and omitted.

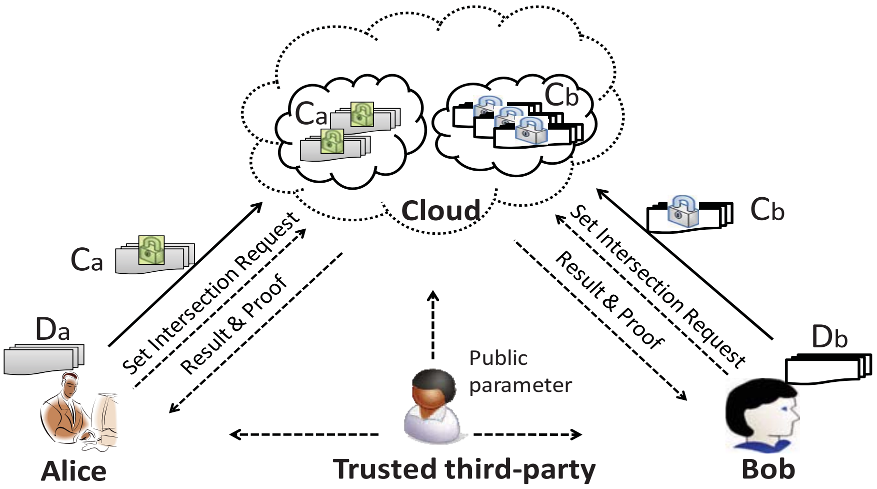


Figure 3.1: VDSI system model: data owners Alice and Bob encrypt their data sets (denoted by D_a and D_b), using their respective public keys, outsource to the cloud the resulting ciphertexts (denoted by C_a and C_b), and delegate the computation of $\mathsf{D}_a \cap \mathsf{D}_b$ to the cloud but without giving it the capability to decrypt C_a and C_b.

3.3.2 Threat Model and Basic Idea of Defense

We assume that the cloud users (i.e., data owners) are honest-but-curious, meaning that they act according to the protocols and use their real data sets as inputs to the protocols, but are curious about each other's private data. However, the cloud is possibly malicious. This means that the cloud can attempt to breach the secrecy of the data outsourced to the cloud, manipulate the integrity of the outsourced data, and deviate from the protocols arbitrarily. The cloud may be controlled by an attacker, who also has control over all the communication channels. This means that denial-of-service attack is inevitable and should be addressed orthogonally by another layer of defense. We will use “the attacker" and “the cloud" interchangeably.

The basic idea for defending against the possibly malicious cloud is to ask the cloud to generate a proof, which shows that it has faithfully executed the delegated set intersection operation. By “examining" the result and proof returned by the cloud, Alice and/or Bob can verify whether or not the cloud has faithfully executed the delegated set intersection operations on C_a and C_b or not.

3.3.3 VDSI Function Definition

In order to simplify the description of both the definition and the concrete scheme that will be presented later, we assume that there is an authenticated user-to-cloud private communication channel, which is used by a cloud user to send some secret information to the cloud (e.g., the secret information that allows the cloud to conduct the delegated set intersection operation). This assumption does not impose any significant restriction because in the case the cloud is controlled by the attacker, the secret information is given to the attacker any way. In practice, the channel can be readily realized by encrypting the secret information under the cloud's public key.

With loss of generality, denote by Alice's plaintext data set $\mathsf{D}_a = \{d_{a,0}, \ldots, d_{a,n}\}$ and Bob's plaintext data set $\mathsf{D}_b = \{d_{b,0}, \ldots, d_{b,m}\}$. Alice (Bob) outsources her (his) encrypted version of D_a (D_b), denoted by C_a (C_b), to the cloud. Alice and Bob want to compute $\mathsf{D}_a \cap \mathsf{D}_b$ by delegating the computation to the cloud, but without giving the cloud capability to decrypt C_a and C_b.

Definition 6. A VDSI scheme has seven algorithms:

- $\mathsf{pm} \leftarrow \mathsf{Setup}(1^\ell)$: Given security parameter ℓ, the trusted third party runs this algorithm to bootstrap the public parameters pm.

- $(\mathsf{pk}_a, \mathsf{sk}_a) \leftarrow \mathsf{KeyGen}(\mathsf{pm})$: Alice runs this randomized algorithm to generate a pair of public and private keys $(\mathsf{pk}_a, \mathsf{sk}_a)$, where pk_a is made public and sk_a is kept secret by Alice. Similarly, we denote Bob's pair of public and private keys by $(\mathsf{pk}_b, \mathsf{sk}_b)$.

- $(\mathsf{C}_a, \mathsf{si}_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a)$: Alice runs this encryption algorithm to encrypt her data set D_a to ciphertext C_a, which is outsourced to the cloud, and some secret information si_a, which is kept secret by Alice. Bob can generate $(\mathsf{C}_b, \mathsf{si}_b)$ similarly.

- $\{\mathsf{D}', \perp\} \leftarrow \mathsf{Dec}(\mathsf{sk}_a, \mathsf{rslt}_a)$: Alice runs this decryption algorithm to decrypt ciphertext rslt_a, which is the output of the delegated set intersection operation conducted by the cloud on ciphertexts C_a and C_b, to obtain the intersection set $\mathsf{D}' = \mathsf{D}_a \cap \mathsf{D}_b$. In the case the decryption fails, the algorithm outputs $\perp$ instead. Note this decryption algorithm also can be used to decrypt C_a.

- $\mathsf{au}_a \leftarrow \mathsf{AuGen}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{pk}_b)$: In order to allow the cloud to conduct the set intersection operation on the outsourced ciphertexts C_a and C_b, Alice runs this algorithm to generate some auxiliary information au_a, which is sent to the cloud through the authenticated user-to-cloud private communication channel (see justification above), where si_a is the Alice's secret information generated by $\mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a)$. Similarly, Bob can generate and send au_b to the cloud, where $\mathsf{au}_b \leftarrow \mathsf{AuGen}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{pk}_a)$.

- $\{(\mathsf{rslt}_a, \mathsf{proof}_a), (\mathsf{rslt}_b, \mathsf{proof}_b)\} \leftarrow \mathsf{SetOp}(\mathsf{C}_a,\ \mathsf{au}_a,\ \mathsf{C}_b,\ \mathsf{au}_b)$: This is the delegated set intersection operation run by the cloud. Depending on the application, the cloud may return $(\mathsf{rslt}_a, \mathsf{proof}_a)$ and $(\mathsf{rslt}_b, \mathsf{proof}_b)$ respectively to Alice and Bob, or return $(\mathsf{rslt}_a, \mathsf{proof}_a)$ to Alice or $(\mathsf{rslt}_b, \mathsf{proof}_b)$ to Bob who requested the delegated set intersection operation, where proof_a and proof_b are proofs that can show that the cloud has faithfully executed the SetOp protocol.

- $\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{rslt}_a, \mathsf{proof}_a)$: Alice runs this algorithm to verify whether rslt_a is faithfully generated by the cloud according to the SetOp protocol. If so (with output 1), Alice calls the $\mathsf{Dec}(\mathsf{sk}_a, \mathsf{rslt}_a)$ algorithm to decrypt rslt_a; otherwise (with output 0), the cloud is cheating.

We say a VDSI scheme is correct if the following holds:

$$\Pr\left[\begin{array}{l} \mathsf{pm} \leftarrow \mathsf{Setup}(1^{\ell}), \\ (\mathsf{pk}_a, \mathsf{sk}_a) \leftarrow \mathsf{KeyGen}(\mathsf{pm}), (\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(\mathsf{pm}), \\ \forall\, \mathsf{D}_a, \mathsf{D}_b, (\mathsf{C}_a, \mathsf{si}_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a),\ \ (\mathsf{C}_b, \mathsf{si}_b) \leftarrow \mathsf{Enc}(\mathsf{pk}_b, \mathsf{D}_b), \\ \ \mathsf{au}_a \leftarrow \mathsf{AuGen}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{pk}_b),\ \ \mathsf{au}_b \leftarrow \mathsf{AuGen}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{pk}_a), \\ \ \{(\mathsf{rslt}_a, \mathsf{proof}_a), (\mathsf{rslt}_b, \mathsf{proof}_b)\} \leftarrow \mathsf{SetOp}(\mathsf{C}_a, \mathsf{au}_a, \mathsf{C}_b, \mathsf{au}_b) : \\ \ 1 \leftarrow \mathsf{Verify}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{rslt}_a, \mathsf{proof}_a), \\ \ 1 \leftarrow \mathsf{Verify}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{rslt}_b, \mathsf{proof}_b), \\ \ \mathsf{D}_a \cap \mathsf{D}_b = \mathsf{Dec}(\mathsf{sk}_a, \mathsf{rslt}_a) = \mathsf{Dec}(\mathsf{sk}_b, \mathsf{rslt}_b) \end{array}\right] = 1.$$

3.3.4 VDSI Security Definition

Informally, VDSI aims to achieve the following security properties against the afore-discussed threat model. We consider three security properties: *outsourced data secrecy*, *function output secrey* and *verifiability*, which are formally defined below. Let ϵ be a negligible function in security parameter ℓ. We consider a probabilistic polynomial-time (in ℓ) adversary $\mathcal{A}$ controlling the cloud.

Outsourced Data Secrecy: Similar to security against chosen-plaintext attack, this property means that the attacker $\mathcal{A}$ cannot breach secrecy of the outsourced data, unless that $\mathcal{A}$ is given the respect auxiliary information.

Definition 7. (outsourced data secrecy) A VDSI scheme achieves *outsourced data secrecy* if the following holds:

$$\left|\Pr\left[\begin{array}{l} \mathsf{pm} \leftarrow \mathsf{Setup}(1^{\ell}), (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pm}), \\ (\mathsf{D}_0, \mathsf{D}_1) \leftarrow \mathcal{A}^{\mathsf{Enc}}(\mathsf{pk}), s.t. |\mathsf{D}_0| = |\mathsf{D}_1|, \\ \lambda \overset{R}{\leftarrow} \{0, 1\}, (\mathsf{C}_\lambda, \mathsf{si}_\lambda) \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{D}_\lambda), \\ \lambda' \leftarrow \mathcal{A}^{\mathsf{Enc}}(\mathsf{pk}, \mathsf{C}_\lambda, \mathsf{D}_0, \mathsf{D}_1) : \lambda = \lambda' \end{array}\right] - \frac{1}{2}\right| \leq \epsilon$$

This property is necessary but not sufficient because it only assures the secrecy of outsourced data when $\mathcal{A}$ is not given the delegated set intersection operation capability. The following property, function output secrecy, is used to capture the secrecy of oursouced data after $\mathcal{A}$ is granted the capability (i.e, $\mathcal{A}$ is given the auxiliary information au).

Function Output Secrecy: This property means that $\mathcal{A}$ cannot breach secrecy of the resulting intersetction set $\mathsf{D}_a \cap \mathsf{D}_b$. Ideally, given a target ciphertext and the auxiliary information, $\mathcal{A}$ cannot learn the plaintext with a non-negligible probability.

Definition 8. (function output secrecy) A VDSI scheme achieves *function output secrecy* if

$$\Pr\left[\begin{array}{l} \mathsf{pm} \leftarrow \mathsf{Setup}(1^{\ell}), \\ (\mathsf{pk}_a, \mathsf{sk}_a) \leftarrow \mathsf{KeyGen}(\mathsf{pm}), (\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(\mathsf{pm}), \\ \forall \mathsf{D}_a, \mathsf{D}_b, (\mathsf{C}_a, \mathsf{si}_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a), (\mathsf{C}_b, \mathsf{si}_b) \leftarrow \mathsf{Enc}(\mathsf{pk}_b, \mathsf{D}_b), \\ \forall \mathsf{cph} \in (\mathsf{C}_a \cup \mathsf{C}_b), \\ \mathsf{au}_a \leftarrow \mathsf{AuGen}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{pk}_b), \mathsf{au}_b \leftarrow \mathsf{AuGen}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{pk}_a), \\ \{d_1, \ldots, d_q\} \leftarrow \mathcal{A}^{\mathsf{Enc},\mathsf{SetOp},\mathsf{Verify}}(\mathsf{pk}_a, \mathsf{au}_a, \mathsf{C}_a, \mathsf{pk}_b, \mathsf{au}_b, \mathsf{C}_b, \mathsf{cph}) : \\ \exists i \in [1, q], d_i = d \end{array}\right] \leq f(\ell, q, |\mathcal{M}|),$$

where q is the maximum number of guessing against cph, d is the plaintext with respect to cph, and $\mathcal{M}$ is the plaintext domain.

Remark. Ideally, we want $f(\ell, q, |\mathcal{M}|)$ to be a negaligible function in ℓ as well. Unfortunately, we are only able to construct a scheme that achieves $f(\ell, q, |\mathcal{M}|) = \frac{q}{|\mathcal{M}|} + \epsilon$, which is a non-negligible function in ℓ because $|\mathcal{M}|$ would not be exponentially in ℓ. The intuition behind $\frac{q}{|\mathcal{M}|}$ is that $\mathcal{A}$ can launch a plaintext-guessing attack (in a way similar to the *online dictionary attack* against passwords), which is specific to our scheme that will be presented later. Designing a VDSI scheme that achieves negligible $f(\ell, q, |\mathcal{M}|)$ in ℓ is left as an open problem for future research. Nevertheless, our definition is general enough to accommodate that scenario.

Verifiability: This property means that any $\mathcal{A}$ not faithfully executing the SetOp protocol is bound to be caught.

Definition 9. (verifiability) A VDSI scheme is *verifiable* if

$$\Pr\left[\begin{array}{l}
\mathsf{pm} \leftarrow \mathsf{Setup}(1^{\ell}),\\
(\mathsf{pk}_a, \mathsf{sk}_a) \leftarrow \mathsf{KeyGen}(\mathsf{pm}), (\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(\mathsf{pm}),\\
(\mathsf{D}_a, \mathsf{D}_b) \leftarrow \mathcal{A}^{\mathsf{Enc},\mathsf{AuGen},\mathsf{SetOp},\mathsf{Verify}}(\mathsf{pk}_a, \mathsf{pk}_b),\\
(\mathsf{C}_a, \mathsf{si}_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a), (\mathsf{C}_b, \mathsf{si}_b) \leftarrow \mathsf{Enc}(\mathsf{pk}_b, \mathsf{D}_b),\\
\mathsf{au}_a \leftarrow \mathsf{AuGen}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{pk}_b),\\
\mathsf{au}_b \leftarrow \mathsf{AuGen}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{pk}_a),\\
\{(\mathsf{rslt}_a, \mathsf{proof}_a), (\mathsf{rslt}_b, \mathsf{proof}_b)\} \leftarrow \mathcal{A}^{\mathsf{Enc},\mathsf{SetOp},\mathsf{Verify}}\\
\qquad (\mathsf{pk}_a, \mathsf{au}_a, \mathsf{D}_a, \mathsf{C}_a, \mathsf{si}_a, \mathsf{pk}_b, \mathsf{au}_b, \mathsf{D}_b, \mathsf{C}_b, \mathsf{si}_b) :\\
1 \leftarrow \mathsf{Verify}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{rslt}_a, \mathsf{proof}_a) \wedge 1 \leftarrow \mathsf{Verify}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{rslt}_b, \mathsf{proof}_b) \wedge\\
(\mathsf{Dec}(\mathsf{sk}_a, \mathsf{rslt}_a) \neq \mathsf{Dec}(\mathsf{sk}_b, \mathsf{rslt}_b) \vee \mathsf{Dec}(\mathsf{sk}_a, \mathsf{rslt}_a) \neq (\mathsf{D}_a \cap \mathsf{D}_b))
\end{array}\right] \leq \epsilon$$

3.4 Building-Block: multi-accumulator

A cryptography accumulator is a primitive for a *verifier* to examine the membership of an element with respect to a (static or dynamic) data set. The examination is based on some public data and membership proof provided by a *prover*. In a single-accumulator scheme, each membership proof allows a verifier to examine the membership of a *single* element with respect to a data set. The idea of single-accumulator has been studied extensively (see, e.g., [27,45,93]). We introduce the idea of multi-accumulator by which, each membership proof allows a verifier to examine the membership of *multiple* elements with respect to a data set. in the context of set intersection operations, multi-accumulator allows Alice (Bob) to verify, via a single examination, that $\mathsf{D}_a \cap \mathsf{D}_b \subseteq \mathsf{D}_a$ ($\subseteq \mathsf{D}_b$).

3.4.1 Function and Security Definitions

Suppose Alice has a data set acD_a and outsources it to the cloud (as the prover). Bob (as the verifier) has a dataset acD_b and queries the cloud for $\mathsf{acD}_a \cap \mathsf{acD}_b$.

Definition 10. A multi-accumulator scheme has the following algorithms:

- $(\mathsf{acSk}, \mathsf{acPk}) \leftarrow \mathsf{acKeyGen}(1^\ell)$: The trusted third party runs this algorithm to generate a pair of public and private key $(\mathsf{acPk}, \mathsf{acSk})$.

- $\mathsf{acDig}_a \leftarrow \mathsf{acGen}(\mathsf{acPk}, \mathsf{acD}_a)$: Alice runs this algorithm to generate a digest acDig for acD_a, which is outsourced to the cloud. Similarly, Bob can generate acDig_b with respect to acD_b.

- $(\mathsf{acRslt}, \mathsf{acWit}) \leftarrow \mathsf{acProve}(\mathsf{acPk}$, acD_b, $\mathsf{acD}_a)$: Given the data set acD_b from Bob, the cloud runs this algorithm to generate $\mathsf{acRslt} = (\mathsf{acD}_b \cap \mathsf{acD}_a)$ with an accompanying witness acWit for this fact.

- $\{0,1\} \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_b$, acRslt, acWit, $\mathsf{acDig}_a)$: Bob runs this algorithm to examine if $\mathsf{acRslt} = \mathsf{acD}_b \cap \mathsf{acD}_a$, where acDig_b is the digest with respect to acD_b and acDig is the digest with respect to acD_a. If so, output 1; otherwise, output 0.

A $\mathsf{multi\text{-}accumulator}$ scheme is correct if

$$\Pr\left[\begin{array}{l} \forall\ \mathsf{acD}_a, \mathsf{acD}_b \\ (\mathsf{acSk}, \mathsf{acPk}) \leftarrow \mathsf{acKeyGen}(1^\ell), \\ \mathsf{acDig}_b \leftarrow \mathsf{acGen}(\mathsf{acPk}, \mathsf{acD}_b), \mathsf{acDig}_a \leftarrow \mathsf{acGen}(\mathsf{acPk}, \mathsf{acD}_a), \\ (\mathsf{acRslt}, \mathsf{acWit}) \leftarrow \mathsf{acProve}(\mathsf{acPk}, \mathsf{acD}_b, \mathsf{acD}_a) : \\ 1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_b, \mathsf{acRslt}, \mathsf{acWit}, \mathsf{acDig}_a) \end{array}\right] = 1.$$

A $\mathsf{multi\text{-}accumulator}$ scheme is secure if a malicious probabilistic polynomial-time prover $\mathcal{A}$ can cheat the honest verifier without being caught. Let ℓ be a security parameter and ϵ be a negligible function in ℓ. Formally, we have:

Definition 11. A $\mathsf{multi\text{-}accumulator}$ scheme is secure if

$$\Pr\left[\begin{array}{l}(\mathsf{acPk},\mathsf{acSk}) \leftarrow \mathsf{acKeyGen}(1^{\ell}),\\ \mathsf{acD}_a \leftarrow \mathcal{A}^{\mathsf{acProve},\mathsf{acVerify}}(\mathsf{acPk}), \mathsf{acDig}_a \leftarrow \mathsf{acGen}(\mathsf{acSk},\mathsf{acD}_a),\\ (\mathsf{acD}_b,\mathsf{acRslt},\mathsf{acWit}) \leftarrow \mathcal{A}^{\mathsf{acProve},\mathsf{acVerify}}(\mathsf{acPk},\mathsf{acD}_a):\\ \mathsf{acDig}_b \leftarrow \mathsf{acGen}(\mathsf{acPk},\mathsf{acD}_b),\\ 1 \leftarrow \mathsf{acVerify}(\mathsf{acPk},\mathsf{acDig}_b,\mathsf{acRslt},\mathsf{acWit},\mathsf{acDig}_a),\\ \mathsf{acRslt} \neq \mathsf{acD}_b \cap \mathsf{acD}_a\end{array}\right] \leq \epsilon.$$

3.4.2 Construction based on Bilinear Map

A $\mathsf{multi\text{-}accumulator}$ scheme can be based on a $\mathsf{single\text{-}accumulator}$ scheme that supports both membership and non-membership proofs, as follows: the cloud generates a witness for each element of acD_b showing the element is a member or non-member of acD_a and simply puts them together as the witness for $\mathsf{acRslt} = \mathsf{acD}_b \cap \mathsf{acD}_a$. However, this straightforward approach is costly because both the computational and communication complexities are linear to $|\mathsf{acD}_b|$.

We present a $\mathsf{multi\text{-}accumulator}$ scheme, where the size of the witness is constant (i.e., independent of $|\mathsf{acD}_b|$). The proposed $\mathsf{multi\text{-}accumulator}$ scheme is extended from the $\mathsf{single\text{-}accumulator}$ scheme due to [45,93], while adapting the basic idea underlying [104] as follows: Suppose Alice's data set is $\mathsf{acD}_a = \{d_{a,1},\ldots,d_{a,n}\}$, Bob's data set is $\mathsf{acD}_b = \{d_{b,1},\ldots,d_{b,m}\}$, and $\mathsf{acRslt} = \mathsf{acD}_a \cap \mathsf{acD}_b$. We can encode acD_a via polynomial $R(x) = \prod_{t\in\mathsf{acD}_a}(x+t)$, encode acD_b via polynomial $W(x) = \prod_{t\in\mathsf{acD}_b}(x+t)$, encode the intersection set acRslt via polynomial $T(x) = \prod_{t\in\mathsf{acRslt}}(x+t)$, and encode the subset $\mathsf{acD}_b - \mathsf{acRslt}$ via polynomial $Q(x) = \prod_{t\in(\mathsf{acD}_b-\mathsf{acRslt})}(x+t)$. These polynomials satisfy the following: (i) $T(x)Q(x) = W(x)$, (ii) $T(x)$ is a divisor of $R(x)$, and (iii) $Q(x)$ is co-prime to $R(x)$. For the special case $\mathsf{acRslt} = \emptyset$, the three conditions also hold since $T(x) = 1$, $Q(x) = W(x) = \prod_{t\in\mathsf{acD}_b}(x+t)$ and $R(x) = \prod_{t\in\mathsf{acD}_a}(x+t)$. Therefore, based on this idea, the $\mathsf{multi\text{-}accumulator}$ scheme allows the cloud to show the correctness of the intersection set, which can be either empty or non-empty. It can be constructed as follows:

- $\mathsf{acKeyGen}(1^\ell)$: Let $(e, g, G, G_T, p) \leftarrow \mathsf{MapGen}(1^\ell)$, set $\alpha \overset{R}{\leftarrow} \mathbb{Z}_p$ and $\mathsf{acPk} = (g^\alpha,\ g^{\alpha^2},\ \ldots, g^{\alpha^q})$, $\mathsf{acSk} = (\alpha)$, where q is bounded by a polynomial in security parameter ℓ.

- $\mathsf{acGen}(\mathsf{acPk}, \mathsf{acD}_a)$: Given Alice's data set $\mathsf{acD}_a = \{d_{a,1}, \ldots, d_{a,n}\} \in \mathbb{Z}_p^n$ where $n \leq q$, compute its digest as
$$\mathsf{acDig}_a = g^{\prod_{i=1}^{n}(d_{a,i}+\alpha)}.$$

- $\mathsf{acProve}(\mathsf{acPk}, \mathsf{acD}_b, \mathsf{acD}_a)$: Given Bob's data set $\mathsf{acD}_b = (d_{b,1}, \ldots, d_{b,m}) \in \mathbb{Z}_p^m$ where $m \leq q$, compute $\mathsf{acRslt} = \mathsf{acD}_b \cap \mathsf{acD}_a$, and generate a witness as follows:
    - Let $T'(x) = \prod_{t \in (\mathsf{acD}_a - \mathsf{acRslt})}(x + t)$ and compute $g^{T'(\alpha)}$ by substituting x with α.
    - Let $Q(x) = \prod_{t \in (\mathsf{acD}_b - \mathsf{acRslt})}(x+t)$ and $R(x)$= $\prod_{t \in \mathsf{acD}_a}(x+t)$, and find two polynomials $Q'(x), R'(x)$ such that $Q(x)Q'(x)$ + $R(x)R'(x)$ = $1 \mod p$ by taking advantage of $\gcd(Q(x), R(x)) = 1$. Compute $(g^{Q(\alpha)}, g^{Q'(\alpha)}, g^{R'(\alpha)})$ by substituting x with α.

  Set $\mathsf{acRslt} = \mathsf{acD}_b \cap \mathsf{acD}_a$ and $\mathsf{acWit} = (g^{Q(\alpha)},\ g^{Q'(\alpha)}, g^{R'(\alpha)}, g^{T'(\alpha)})$.

- $\mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_b, \mathsf{acRslt}, \mathsf{acWit}, \mathsf{acDig}_a)$: Given acWit and acRslt from the prover, the verifier proceeds as follows:
    1. If $\mathsf{acRslt} \neq \emptyset$, compute $g^{T(\alpha)}$ according to $T(x) = \prod_{t \in \mathsf{acRslt}}(x + t)$. Otherwise, let $T(x) = 1$ and $g^{T(\alpha)} = g$.
    2. If $e(g^{Q(\alpha)}, g^{T(\alpha)})) \neq e(\mathsf{acDig}_b, g)$, return 0; otherwise, proceed to next step.
    3. If $e(g^{T(\alpha)}, g^{T'(\alpha)}) \neq e(\mathsf{acDig}_a, g)$, return 0; otherwise, proceed to next step.
    4. If $e(g^{Q(\alpha)}, g^{Q'(\alpha)})e(\mathsf{acDig}_a, g^{R'(\alpha)}) \neq e(g, g)$, return 0; otherwise, return 1.

Correctness of the multi-accumulator scheme can be verified easily. We describe its asymptotic efficiency in Table 3.2. It is worth noting that (i) the witness generated by algorithm $\mathsf{acProve}$ only consists of four group elements, meaning that the complexity is independent of $k = |\mathsf{acD}_b \cap \mathsf{acD}_a|$, and (ii) the computational complexity of algrorithm $\mathsf{acVerify}$ is linear to $k = |\mathsf{acD}_b \cap \mathsf{acD}_a|$.

Now we prove its security.

Table 3.2: Asymptotical efficiency of the multi-accumulator scheme, where Exp denotes the exponentiation operation, Pairing denotes the pairing operation, $n = |\mathsf{acD}_a|$, $m = |\mathsf{acD}_b|$ and $k = |\mathsf{acD}_a \cap \mathsf{acD}_b|$.

	acGen	acProve	acVerify
Computation	$n\mathsf{Exp}$	$(n+m)\mathsf{Exp}$	$k\mathsf{Exp} + 7\mathsf{Pairing}$
Output Size	$\|G\|$	$4\|G\|$	N/A

Theorem 9. *Assume that the q-SDH assumption holds, the* multi-accumulator *scheme is secure with respect to Definition 11.*

Proof. We show that if there is an adversary $\mathcal{A}$ that can break the multi-accumulator scheme with a non-negligible probability, there is an algorithm $\mathcal{B}$ that can break the q-SDH assumption with a non-negligible probability.

Suppose $\mathcal{B}$ is given a challenge instance $(g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^q})$, where $\alpha \overset{R}{\leftarrow} \mathbb{Z}_p$ and α is unknown. $\mathcal{B}$ simulates the multi-accumulator scheme for $\mathcal{A}$, according to the game implied by Definition 11. For $\mathsf{acD}_a = \{d_{a,1}, \ldots, d_{a,n}\}$ with digest $\mathsf{acDig}_a = g^{\prod_{t \in \mathsf{acD}_a}(\alpha+t)}$, suppose $\mathcal{A}$ returns acD_b, acRslt and $\mathsf{acWit} = (g^{Q(\alpha)}, g^{Q'(\alpha)}, g^{R'(\alpha)}, g^{T'(\alpha)})$. If $\mathcal{A}$ breaks the security of the multi-accumulator with non-negligible probability, then there exists acD_b, acRslt and acWit such that the followings hold: (i) $1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_b, \mathsf{acRslt}, \mathsf{acWit}, \mathsf{acDig}_a)$ where $\mathsf{acDig}_b \leftarrow \mathsf{acGen}(\mathsf{acPk}, \mathsf{acD}_b)$, and (ii) $\mathsf{acRslt} \neq \mathsf{acD}_b \cap \mathsf{acD}_a$, and then $\mathcal{B}$ can break the q-SDH assumption by presenting a tuple $(t', e(g,g)^{1/(\alpha+t')})$ where $t' \in \mathbb{Z}_p$.

First, we claim that $\forall t \in \mathsf{acRslt}$, it holds that $t \in \mathsf{acD}_b$. To prove this, suppose there exists $t' \in \mathsf{acRslt}$ but $t' \notin \mathsf{acD}_b$, meaning that polynomial $\prod_{t \in \mathsf{acD}_b}(x+t)$ cannot be divided by $(x + t')$. Therefore, polynomial $\prod_{t \in \mathsf{acD}_b}(x+t)$ can be represented (in polynomial-time) as $U(x)(x + t') + \lambda$ where $\lambda \neq 0$, and $U(x)$ is a polynomial of degree $|\mathsf{acD}_b| - 1$. On the other hand, $1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_b, \mathsf{acRslt}, \mathsf{acWit}, \mathsf{acDig}_a)$ implies the following:

$$e(g^{Q(\alpha)}, g^{\prod_{t \in \mathsf{acRslt}}(\alpha+t)}) = e(\mathsf{acDig}_b, g) = e(g^{\prod_{t \in \mathsf{acD}_b}(\alpha+t)}, g).$$

By substituting $\prod_{t \in \mathsf{acD}_b}(\alpha+t)$ with $U(\alpha)(\alpha+t') + \lambda$ in the right-hand of the above equation, we have

$$e(g^{Q(\alpha)}, g^{\prod_{t\in\mathsf{acRslt}}(\alpha+t)}) \;=\; e(g^{U(\alpha)(\alpha+t')+\lambda}, g).$$

This leads to:

$$e(g^{Q(\alpha)}, g^{\prod_{t\in\mathsf{acRslt}, t\neq t'}(\alpha+t)}) \;=\; e(g^{U(\alpha)+\frac{\lambda}{\alpha+t'}}, g).$$

Therefore,

$$e(g,g)^{\frac{1}{\alpha+t'}} = (e(g^{Q(\alpha)}, g^{\prod_{t\in\mathsf{acRslt}, t\neq t'}(\alpha+t)})e(g,g)^{-U(\alpha)})^{\frac{1}{\lambda}}.$$

That is, if (i) $\mathcal{A}$ breaks the $\mathsf{multi\text{-}accumulator}$ scheme with a non-negligible probability and (ii) $\exists t' \in \mathsf{acRslt}$ such that $t' \notin \mathsf{acD}_b$, then $\mathcal{B}$ can break the q-SDH assumption by outputting (t', $e(g,g)^{1/(\alpha+t')}$) with a non-negligible probability.

Second, we claim that $\forall t \in \mathsf{acRslt}$ it holds that $t \in \mathsf{acD}_a$. This can be proved similarly.

Third, we claim that $\forall t \in (\mathsf{acD}_b - \mathsf{acRslt})$, it holds that $t \notin \mathsf{acD}_a$. To prove this, suppose there exists $t' \in (\mathsf{acD}_b - \mathsf{acRslt})$ but $t' \in \mathsf{acD}_a$. This means that there exists polynomials

$$Q'(x)(x+t') \prod_{t\in(\mathsf{acD}_b-\mathsf{acRslt})}^{t\neq t'} (x+t) + R'(x)(x+t') \prod_{t\in\mathsf{acD}_a}^{t\neq t'} (x+t) = 1,$$

which means

$$Q'(x) \prod_{t\in(\mathsf{acD}_b-\mathsf{acRslt})}^{t\neq t'} (x+t) + R'(x) \prod_{t\in\mathsf{acD}_a}^{t\neq t'} (x+t) = \frac{1}{(x+t')}.$$

On the other hand, $1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acD}_b, \mathsf{acRslt}, \mathsf{acWit}, \mathsf{acDig}_a)$ implies:

$$e(g^{Q'(\alpha)}, g^{\prod_{t\in(\mathsf{acD}_b-\mathsf{acRslt})}(\alpha+t)})e(g^{R'(\alpha)}, g^{\prod_{t\in\mathsf{acD}_a}(\alpha+t)}) = e(g,g).$$

Therefore, we have

$$e(g,g)^{\frac{1}{(\alpha+t')}} = e(g^{Q'(\alpha)}, g^{\prod_{t\in(\mathsf{acD}_b-\mathsf{acRslt}),t\neq t'}(\alpha+t)})e(g^{R'(x)}, g^{\prod_{t\in\mathsf{acD}_a,t\neq\mathsf{acD}_a}(\alpha+t)}).$$

That is, if (i) $\mathcal{A}$ breaks the multi-accumulator scheme with a non-negligible probability and (ii) there exists $t' \in (\mathsf{acD}_b - \mathsf{acRslt})$ such that $t' \in \mathsf{acD}_a$, then $\mathcal{B}$ can break the q-SDH assumption by outputting $(t', e(g,g)^{1/(\alpha+t')})$ with a non-negligible probability.

To sum up, since $\forall t \in \mathsf{acRslt}$ it holds that $t \in \mathsf{acD}_b$ and $t \in \mathsf{acD}_a$, and $\forall t' \in (\mathsf{acD}_b - \mathsf{acRslt})$ it holds that $t' \notin \mathsf{acD}_a$, we conclude that $\mathsf{acRslt} = \mathsf{acD}_b \cap \mathsf{acD}_a$. Therefore, the multi-accumulator scheme is secure with respect to Definition 11. □

3.5 The VDSI Scheme

Basic Ideas. In order to attain a VDSI scheme, we need to resolve two issues: (i) How can we enable the cloud to compare the equality of two ciphertexts that are encrypted under two different public keys pk_a and pk_b, respectively? (ii) How can we enable the cloud to generate a proof for showing that it has faithfully executed the SetOp protocol, ideally without using zero-knowledge proof for the sake of better efficiency?

To resolve the above (i), we adopt the idea of proxy re-encryption as follows: Alice can generate a re-key and send it to the cloud, which can use the re-key to transform ciphertext C_a (encrypted under Alice's public key pk_a) into an intermediate form, say T_a. Similarly, the cloud can transform ciphertext C_b (encrypted under Bob's public key pk_b) into the same kind fo intermediate form, denoted by T_b. Then, the cloud can "compare" T_a and T_b to determine whether they correspond to the same plaintext or not. More specifically, a data item $d_{a,i} \in \mathsf{D}_a$ is encrypted using $\mathsf{pk}_a = (g^{\beta_a}, g^{\gamma_a})$ as $(g^{r_2}, g^{\gamma_a r_1}, d_{a,i}g^{\beta_a(r_1+r_2)})$, where $r_1, r_2 \overset{R}{\leftarrow} \mathbb{Z}_p$. Alice can give the re-key $\mathsf{rk}_a = g^{\beta_a/\gamma_a}$, rather than her private key $\mathsf{sk}_a = (\beta_a, \gamma_a)$ to the cloud, which now can transform the ciphertext into

$$\frac{e(d_{a,i}g^{\beta_a(r_1+r_2)}, g)}{e(g^{\gamma_a r_1}, g^{\beta_a/\gamma_a})e(g^{r_2}, g^{\beta_a})} = e(d_{a,i}, g).$$

Similarly, for data item $d_{b,i} \in \mathsf{D}_b$, the cloud can transform the corresponding ciphertext into $e(d_{b,i}, g)$. If $d_{a,i} = d_{b,i}$, then $e(d_{a,i}, g) = (d_{b,i}, g)$. While this method is sufficient to allow the cloud to determine whether the two ciphertexts correspond to the same plaintext or not, it does not achieve the desired semantic security because the cloud can launch the *plaintext guess* attack against elements $d_{a,i}$ and $d_{b,i}$. We have tried without success to eliminate this attack while preserving the other properties (especially the *verifiability*). We therefore leave it as an open problem.

To resolve the above issue (ii), we observe that the cloud, as illustrated above, can generate $e(d_{a,i}, g)$ for each $d_{a,i} \in \mathsf{D}_a$ and $e(d_{b,i}, g)$ for each $d_{b,i} \in \mathsf{D}_b$. As a result, the cloud can use the multi-accumulator scheme to generate a proof as follows: For Bob, let $e(d_{a,i}, g)$'s as acD_a and $e(d_{b,i}, g)$'s as acD_b, the cloud applies $\mathsf{acProve}$ to generate witness acWit_b for showing $\mathsf{acRslt} = \mathsf{acD}_a \cap \mathsf{acD}_b$ is the correct intersection set with respect to $e(d_{a,i}, g)$'s and $e(d_{b,i}, g)$'s. Given witness acWit_b, digest acDig_b of $e(d_{b,i}, g)$'s and digest acDig_a of $e(d_{a,i}, g)$, Bob can verify the correctness of acRslt, which can be computed from the returned intersection set of C_a and C_b. Similarly, the cloud can generate witness acWit_a for Alice, who can then conduct the same kind of verification. That is, by using the multi-accumulator scheme in section 3.4, the cloud users can verify the correctness of the intersection set, which contains zero or more common elements.

3.5.1 The Scheme

The scheme is a modular construction based on (i) a secure multi-accumulator scheme $\mathsf{Ac} = (\mathsf{acKeyGen}, \mathsf{acGen}, \mathsf{acProve}, \mathsf{acVerify})$ such as the one described in Section 3.4, and (ii) a secure digital signature scheme $\mathsf{Sig} = (\mathsf{sigKeyGen}, \mathsf{sigSign}, \mathsf{sigVerify})$. The digital signature scheme is used to authenticate the encryption form of the accumulator digest, which assures that the cloud cannot manipulate it without being detected. Specifically, the scheme is described as follows:

Setup(1^ℓ): Given security parameter ℓ, the trusted third party runs $(e, g, G, G_T, p) \leftarrow \mathsf{MapGen}(1^\ell)$. Let $H : G_T \rightarrow \mathbb{Z}_p$ be a collision-resistant hash function. The trusted third party also runs

$(\mathsf{acPk}, \mathsf{acSk}) \leftarrow \mathsf{KeyGen}(1^\ell)$ and sets the public parameter as

$$\mathsf{pm} = (\mathsf{acPk}, e, p, g, G, G_T).$$

KeyGen(pm): Alice runs $(\mathsf{sigPk}_a, \mathsf{sigSk}_a) \leftarrow \mathsf{sigKeyGen}(1^\ell)$, selects $\beta_a, \gamma_a \overset{R}{\leftarrow} \mathbb{Z}_p$, and sets

$$\mathsf{sk}_a = (\beta_a, \gamma_a, \mathsf{sigSk}_a), \quad \mathsf{pk}_a = (g^{\beta_a}, g^{\gamma_a}, \mathsf{sigPk}_a).$$

Similarly, Bob generates $\mathsf{sk}_b = (\beta_b, \gamma_b, \mathsf{sigSk}_b)$ and $\mathsf{pk}_b = (g^{\beta_b}, g^{\gamma_b}, \mathsf{sigPk}_b)$.

Enc($\mathsf{pk}_a, \mathsf{D}_a$): Alice, with $\mathsf{D}_a = (d_{a,1}, \ldots, d_{a,n})$ where $d_{a,i} \in G$ for $1 \leq i \leq n$, executes as follows:

- Select $d_{a,0} \overset{R}{\leftarrow} G$ (for a security purpose that will be elaborated later).
- For $0 \leq i \leq n$, select $r_{i1}, r_{i2} \overset{R}{\leftarrow} \mathbb{Z}_p$ and compute

$$\mathsf{cph}_{a,i} = (g^{r_{i2}}, g^{\gamma_a r_{i1}}, d_{a,i} g^{\beta_a(r_{i1}+r_{i2})}).$$

- For $0 \leq i \leq n$, let $T_i = H(e(d_{a,i}, g))$ and compute $\mathsf{acDig}_a \leftarrow \mathsf{acGen}(\mathsf{acPk}, \{T_0, \ldots, T_n\})$.
- Set $\mathsf{C}_a = \{\mathsf{cph}_{a,0}, \ldots, \mathsf{cph}_{a,n}\}$ and $\mathsf{si}_a = \mathsf{acDig}_a$.

Similarly, Bob, with $\mathsf{D}_b = (d_{b,1}, \ldots, d_{b,m})$ where $d_{b,i} \in G$ for $1 \leq i \leq m$, can obtain $\mathsf{C}_b = \{\mathsf{cph}_{b,0}, \ldots, \mathsf{cph}_{b,m}\}$ and $\mathsf{si}_b = \mathsf{acDig}_b$.

Dec($\mathsf{sk}_a, \mathsf{rslt}_a$): Given the cloud-generated ciphertext intersection set $\mathsf{rslt}_a = \{\mathsf{cph}_{a,j}, \ldots, \mathsf{cph}_{a,k}\}$ where $1 \leq j, k \leq n$, Alice decrypts ciphertexts $\mathsf{cph}_{a,i}$ for $j \leq i \leq k$ as follows:

$$d_{a,i} = d_{a,i} g^{\beta_a(r_{i1}+r_{i2})}) / (g^{r_{i2}})^{\beta_a} (g^{\gamma_a r_{i1}})^{\beta_a/\gamma_a}.$$

The decryption of rslt_a is $\mathsf{D}_a \cap \mathsf{D}_b = \{d_{a,j}, \ldots, d_{a,k}\}$. Note that this algorithm can also be used to decrypt C_a *without* involving any delegated set operations. In this case, the integrity of C_a can be easily assured by acDig_a since the plaintexts of C_a should be accumulated to acDig_a.

Similarly, Bob can decrypt the cloud-generated ciphertext intersection set $\mathsf{rslt}_b = \{\mathsf{cph}_{b,j}, \ldots, \mathsf{cph}_{b,k}\}$ where $1 \leq j, k \leq m$ to obtain $\mathsf{D}_a \cap \mathsf{D}_b$.

AuGen($\mathsf{sk}_a, \mathsf{si}_a, \mathsf{pk}_b$): Given private key sk_a, Alice generates re-key $\mathsf{rk}_a = (g^{\beta_a/\gamma_a})$. Alice encrypts the secret information si_a using Bob's public key pk_b to obtain ciphertext $\mathsf{cph}_B = (g^{r_2}, g^{\gamma_b r_1}, \mathsf{acDig}_a g^{\beta_b(r_1+r_2)})$, where $r_1, r_2 \overset{R}{\leftarrow} \mathbb{Z}_p$. Then, Alice runs $\sigma_a \leftarrow \mathsf{sigSign}(\mathsf{sigSk}_a, \mathsf{cph}_B)$ to obtain a signature σ_a on message cph_B. Finally, Alice sets $\mathsf{au}_a = (\mathsf{rk}_a, \mathsf{cph}_B, \sigma_a)$.

Similarly, Bob can generate $\mathsf{au}_b = (\mathsf{rk}_b, \mathsf{cph}_A, \sigma_b)$.

SetOp($\mathsf{C}_a, \mathsf{au}_a, \mathsf{C}_b, \mathsf{au}_a$): Given $\mathsf{C}_a = \{\mathsf{cph}_{a,0}, \ldots, \mathsf{cph}_{a,n}\}$, $\mathsf{C}_b = \{\mathsf{cph}_{b,0}, \ldots, \mathsf{cph}_{b,m}\}$, $\mathsf{au}_a = (\mathsf{rk}_a = g^{\beta_a/\gamma_a}, \mathsf{cph}_B, \sigma_a)$, and $\mathsf{au}_b = (\mathsf{rk}_b = g^{\beta_b/\gamma_b}, \mathsf{cph}_A, \sigma_b)$, the cloud executes as follows:

- Transform ciphertexts $\mathsf{cph}_{a,i}$ for $0 \leq i \leq n$ into

  $$T_{a,i} = \frac{e(d_{a,i} g^{\beta_a(r_{i1}+r_{i2})}, g)}{e(g^{\gamma_a r_{i1}}, g^{\beta_a/\gamma_a}) e(g^{r_{i2}}, g^{\beta_a})} = e(d_{a,i}, g),$$

  and compute $T_a = \{H(T_{a,0}), \ldots, H(T_{a,n})\}$.

- Transform ciphertexts $\mathsf{cph}_{b,i}$ for $1 \leq i \leq m$ into

  $$T_{b,i} = \frac{e(d_{b,i} g^{\beta_b(r_{i1}+r_{i2})}, g)}{e(g^{\gamma_b r_{i1}}, g^{\beta_b/\gamma_b}) e(g^{r_{i2}}, g^{\beta_b})} = e(d_{b,i}, g)$$

  and compute $T_b = \{H(T_{b,0}), \ldots, H(T_{b,m})\}$.

- Generate the intersection set rslt_a and a proof with respect to C_a as follows: Run $(\mathsf{acRslt}, \mathsf{acWit}_a) \leftarrow \mathsf{acProve}(\mathsf{acPk}, T_a, T_b)$ and set

  $$\mathsf{rslt}_a = \{\mathsf{cph}_{a,i} | H(A_{a,i}) \in \mathsf{acRslt}\},$$
  $$\mathsf{proof}_a = (\mathsf{acWit}_a, \mathsf{cph}_A, \sigma_b).$$

- Generate the intersection set rslt_b and a proof with respect to C_b as follows: Run $(\mathsf{acRslt}, \mathsf{acWit}_b) \leftarrow$

$\mathsf{acProve}(\mathsf{acPk}, T_b, T_a)$ and set

$$\mathsf{rslt}_b = \{\mathsf{cph}_{b,i} | H(A_{b,i}) \in \mathsf{acRslt}\},$$

$$\mathsf{proof}_b = (\mathsf{acWit}_b, \mathsf{cph}_B, \sigma_a).$$

Verify($\mathsf{sk}_a, \mathsf{si}_a, \mathsf{rslt}_a, \mathsf{proof}_a$**)**: Given rslt_a and proof_a, Alice verifies that the cloud faithfully executed the SetOp protocol as follows:

- Verify the integrity of cph_A by running $\mathsf{sigVerify}(\mathsf{sigPk}_b, \mathsf{cph}_A, \sigma_b)$. If it outputs 0, then return 0; otherwise, proceed to next step.
- Decrypt cph_A using private key sk_a according to

$$\mathsf{acDig}_b = \mathsf{acDig}_b g^{\beta_a(r_1+r_2)})/(g^{r_2})^{\beta_a}(g^{\gamma_a r_1})^{\beta_a/\gamma_a}.$$

- If rslt_a is not empty, decrypt rslt_a to obtain the plaintexts and compute $Y_a = \{e(d_{a,i}, g) | \mathsf{cph}_{a,i} \in \mathsf{rslt}_a\}$. Otherwise, let $Y_a = \emptyset$.
- Run $\mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_a, Y_a, \mathsf{acWit}_a, \mathsf{acDig}_b)$. If it outputs 0, then return 0; otherwise, return 1.

If the algorithm returns 1, **Dec(**$\mathsf{sk}_a, \mathsf{rslt}_a$**)** is called to obtain $\mathsf{D}_a \cap \mathsf{D}_b$.

Similarly, Bob can run the same algorithm to verify that the cloud does not cheat.

Remark: why using $d_{a,0}$ and $d_{b,0}$?

Since Alice needs to know the accumulator digest acDig_b for the sake of verifying the correctness of rslt_a, we need to assure that Alice cannot use acDig_b to infer useful information about D_b. This is achieved by "blending" the accumulator digest with the randomness, namely the hash value of the randomly selected $d_{b,0}$. This eliminates the usage of zero-knowledge proofs [5, 48], while assuring no useful information is leaked.

Remark. Our VDSI scheme only offers coarse-grained access control in the following sense. Suppose Alice and Bob allow the cloud to conduct the delegated set intersection operation on C_a and C_b, and Alice and Carlos allow the cloud to conduct the delegated set intersection operation on C_a and C_c. Then, the cloud is able to conduct the set intersection operation on C_b and C_c without the authorization from Bob and Carlos. It is a future work to enforce fine-grained access control in VDSI.

3.5.2 Security Analysis

Correctness of the VDSI scheme can be examined easily. In what follows, we focus on its security properties.

Theorem 10. *Under the DL assumption, the scheme achieves* outsourced data secrecy *(Definition 7).*

Proof. The proof strategy is: we first show that the VDSI scheme achieves *outsourced data secrecy* when the challenge data set $|\mathsf{D}_0| = |\mathsf{D}_1| = 1$, which is then used as a "building-block" to show that the VDSI scheme achieves *outsourced data secrecy* when $|\mathsf{D}_0| = |\mathsf{D}_1| = n$.

First, we show that given $|\mathsf{D}_0| = |\mathsf{D}_1| = 1$, the VDSI scheme achieves *outsourced data secrecy* under the DL assumption. To prove that, we show that if there is a probabilistic polynomial-time adversary $\mathcal{A}$ that can break *outsourced data secrecy* with a non-negligible probability, then there is an algorithm $\mathcal{B}$ that can break the DL assumption with a non-negligible probability.

Suppose $\mathcal{B}$ is given a DL instance $(f, g, h, g^{r_1}, f^{r_2}, Q)$, where $f, g, Q \overset{R}{\leftarrow} G$, $r_1, r_2 \overset{R}{\leftarrow} \mathbb{Z}_p$ and r_1, r_2 are unknown to $\mathcal{B}$. Now $\mathcal{B}$ can simulate the game as follows:

Setup: $\mathcal{B}$ treats $f = g^\gamma$ and $h = g^\beta$ for some unknown γ and β, runs $(\mathsf{acPk}, \mathsf{acSk}) \leftarrow \mathsf{acKeyGen}(1^\ell)$ and $(\mathsf{sigPk}, \mathsf{sigSk}) \leftarrow \mathsf{sigKeyGen}(1^\ell)$, sets $\mathsf{pm} = (\mathsf{acPk}, e, p, g, G, G_T)$ and $\mathsf{pk} = (\mathsf{sigPk}, f, g, h)$ and $\mathsf{sk} = (\mathsf{sigSk})$, and finally sends public key pk and public parameter pm to adversary $\mathcal{A}$.

Phase 1: $\mathcal{A}$ can query the following oracle polynomially-many (in ℓ) times:

- $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pk}, \mathsf{D})$: Given $\mathsf{D} = \{d_1, \ldots, d_n\}$, $\mathcal{B}$ encrypts d_i as $\mathsf{cph}_i = (g^{r_{i2}}, f^{r_{i1}}, d_i h^{(r_{i1}+r_{i2})})$ where

$r_{i1}, r_{i2} \xleftarrow{R} \mathbb{Z}_p$, sets $\mathsf{C} = \{\mathsf{cph}_1, \ldots, \mathsf{cph}_n\}$, runs $\mathsf{acDig} \leftarrow \mathsf{acGen}(\mathsf{acPk}, \{H(e(d_i, g)), \ldots, H(e(d_n, g))\})$, and returns C to $\mathcal{A}$.

Challenge: $\mathcal{A}$ outputs two data sets $(\mathsf{D}_0 = \{d_0\}, \mathsf{D}_1 = \{d_1\})$ where $|\mathsf{D}_0| = |\mathsf{D}_1| = 1$ and $\mathsf{D}_0 \neq \mathsf{D}_1$. $\mathcal{B}$ selects $\lambda \xleftarrow{R} \{0, 1\}$, and computes $\mathsf{cph}_\lambda = (g^{r_1}, f^{r_2}, d_\lambda Q)$. Let cph be the ciphertext of a selected random value unkown to $\mathcal{A}$, $\mathsf{C} = \{\mathsf{cph}, \mathsf{cph}_\lambda\}$, $\mathsf{acDig} \leftarrow \mathsf{acGen}(\mathsf{acPk}, \{H(e(d_\lambda, g))\})$. $\mathcal{B}$ returns C to $\mathcal{A}$.

Phase 2: $\mathcal{A}$ can query the oracle the same as Phase 1.

Guess: Lastly, $\mathcal{A}$ outputs λ'. If $\lambda' = \lambda$, then the challenge outputs $h^{(r_1+r_2)} = Q$; otherwise, it outputs $h^{(r_1+r_2)} \neq Q$.

We can see that if $Q = h^{(r_1+r_2)}$, then cph_λ is the valid ciphertext. In this case the probability of $\mathcal{A}$ outputting $\lambda = \lambda'$ is $\frac{1}{2}+\mu$. If Q is a random element from G, then the probability of $\mathcal{A}$ outputting $\lambda = \lambda'$ is $\frac{1}{2}$. Therefore, the probability of $\mathcal{B}$ correctly guessing $Q \overset{?}{=} h^{(r_1+r_2)}$ is $\frac{1}{2}(\frac{1}{2}+\mu+\frac{1}{2}) = \frac{1}{2}+\frac{\mu}{2}$. That is, given the challenge data sets $|\mathsf{D}_0| = |\mathsf{D}_1| = 1$, if $\mathcal{A}$ can break outsourced data secrecy of the proposed scheme with non-negligible advantage μ, then there exists a challenger breaking the DL assumption with non-negligible advantage $\frac{\mu}{2}$.

In what follows we show that given arbitrary size of data sets, e.g. $|\mathsf{D}_0| = |\mathsf{D}_1| = n$ (or m), the proposed scheme achieves outsourced data secrecy. Given the data sets $\mathsf{D}_0 = (d_1, ..., d_n)$ and $\mathsf{D}_1 = (d'_1, ..., d'_n)$ from adversary $\mathcal{A}$, let $\mathsf{C}^{(i)}$ denote the encryption form of $(d_1, \ldots, d_i, d'_{i+1}, \ldots, d'_n)$, so that $\mathsf{C}^{(n)}$ is the encryption form of D_0 and $\mathsf{C}^{(0)}$ is the encryption form of D_1. We simulate the challenge phase with an additional adversary $\mathcal{A}'$ as follows:

- $\mathcal{A}'$ selects an index $i \xleftarrow{R} [1, n]$ and presents (d_i, d'_i) to $\mathcal{B}$. The challenger returns cph_i by encrypting d_i if $\lambda = 0$ and d'_i otherwise.

- $\mathcal{A}'$ encrypts $(d_1, \ldots, d_{i-1})$ and $(d'_{i+1}, \ldots, d'_n)$, and sends $(\mathsf{cph}_1, \ldots, \mathsf{cph}_n)$ to $\mathcal{A}$. $\mathcal{A}'$ outputs λ' that is output by $\mathcal{A}$.

We can see that $\mathcal{A}'$ sends to $\mathcal{A}$ the ciphertexts $\mathsf{C}^{(i)}$ when $\lambda = 0$ and $\mathsf{C}^{(i-1)}$ when $\lambda = 1$. Now let's consider the probability of $\mathcal{A}'$ winning the security game, and have (we denote as $\mathcal{A}(\mathsf{C}^{(i)})$ the guess of $\mathcal{A}$ with ciphertexts $\mathsf{C}^{(i)}$)

$$\begin{aligned}
\Pr[\mathcal{A}'\text{outputs } 0|\lambda = 0] &= \sum_{i=1}^{n} \frac{1}{n} \Pr[\mathcal{A}(\mathsf{C}^{(i)}) = 0] \\
\Pr[\mathcal{A}'\text{outputs } 1|\lambda = 1] &= \sum_{i=1}^{n} \frac{1}{n} \Pr[\mathcal{A}(\mathsf{C}^{(i-1)}) = 1]
\end{aligned}$$

Therefore, the probability of $\mathcal{A}'$ winning the selective security game is

$$\begin{aligned}
&\frac{1}{2}\Pr[\mathcal{A}'\text{outputs } 0|\lambda = 0] + \frac{1}{2}\Pr[\mathcal{A}'\text{outputs } 1|\lambda = 1] \\
&= \sum_{j=1}^{n} \frac{1}{2n} \Pr[\mathcal{A}(\mathsf{C}^{(j)}) = 0] + \sum_{j=1}^{n} \frac{1}{2n} \Pr[\mathcal{A}(\mathsf{C}^{(j-1)}) = 1] \\
&= \frac{n}{2n} + \frac{1}{2n} \Pr[\mathcal{A}(\mathsf{C}^{(n)}) = 0] + \frac{1}{2n} \Pr[\mathcal{A}(\mathsf{C}^{(0)}) = 1] \\
&= \frac{n}{2n} + \frac{1}{n}(\frac{1}{2} \Pr[\mathcal{A}(\mathsf{C}^{(n)}) = 0] + \frac{1}{2} \Pr[\mathcal{A}(\mathsf{C}^{(0)}) = 1]) \\
&\leq \frac{1}{2} + \epsilon
\end{aligned}$$

Here ϵ is negligible probability of the advantage of $\mathcal{A}'$ winning the security game to guess λ. Therefore, the probability of $\mathcal{A}$ distinguishing $\mathsf{C}^{(0)}$ and $\mathsf{C}^{(n)}$ is

$$\frac{1}{2} \Pr[\mathcal{A}(\mathsf{C}^{(n)}) = 0] + \frac{1}{2} \Pr[\mathcal{A}(\mathsf{C}^{(0)}) = 1] \leq \frac{1}{2} + n\epsilon$$

That is, the advantage of $\mathcal{A}$ distinguishing $\mathsf{C}^{(0)}$ and $\mathsf{C}^{(n)}$ is at most $n\epsilon$, which is negligible with respect to ℓ. Therefore, we show that the proposed scheme achieves outsourced data secrecy under the DL assumption. □

Theorem 11. *Given that the bilinear map e is one-way, the* VDSI *scheme achieves the function output secrecy property (Definition 8).*

Proof. We show that given a ciphertext and corresponding re-key, any probabilistic polynomial-time adversary $\mathcal{A}$ infers the plaintext with the probability $\frac{q}{|\mathcal{M}|} + \epsilon$ at most if $\mathcal{A}$ has q times to guess the plaintexts, where $\mathcal{M}$ is the plaintext space.

Setup: The challenger $\mathcal{B}$ runs $\mathsf{pm} \leftarrow \mathsf{Setup}(1^{\ell})$ and makes pm publicly known.

Challenge: The challenger runs $\mathsf{KeyGen}(\mathsf{pm})$ to obtain $\mathsf{sk}_a = (\mathsf{sigSk}_a, \beta_a, \gamma_a)$ and $\mathsf{pk}_a = (\mathsf{sigPk}_a, g^{\beta_a}, g^{\gamma_a})$ for Alice, and runs $\mathsf{KeyGen}(\mathsf{pm})$ to obtain $\mathsf{sk}_b = (\mathsf{sigSk}_b, \beta_b, \gamma_b)$, $\mathsf{pk}_b = (\mathsf{sigPk}_b, g^{\beta_b}, g^{\gamma_b})$ for Bob. $\mathcal{B}$ selects two data sets $\mathsf{D}_b, \mathsf{D}_b$, elements of which are randomly selected from the plaintext space $\mathcal{M}$, then runs $(\mathsf{C}_a, \mathsf{si}_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a)$ and $(\mathsf{C}_b, \mathsf{si}_b) \leftarrow \mathsf{Enc}(\mathsf{pk}_b, \mathsf{D}_b)$, $\mathsf{au}_a \leftarrow \mathsf{AuGen}$ $(\mathsf{sk}_a, \mathsf{si}, \mathsf{pk}_b)$ and $\mathsf{au}_b \leftarrow \mathsf{AuGen}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{pk}_a)$, and returns C_a, C_b, pk_a, pk_b, au_a, au_b to $\mathcal{A}$.

Phase 1: $\mathcal{A}$ can query the following oracles polynomially many times.

- $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pk}_a, \mathsf{D}'_a)$: Given the data set D'_a, the challenger runs $(\mathsf{C}'_a, \mathsf{si}'_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}'_a)$ and returns $\mathsf{C}'_a, \mathsf{si}'_a$ to $\mathcal{A}$.

- $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pk}_b, \mathsf{D}'_b)$: Given the data set D'_b, the challenger runs $(\mathsf{C}'_b, \mathsf{si}'_b) \leftarrow \mathsf{Enc}(\mathsf{pk}_b, \mathsf{D}'_b)$ and returns $\mathsf{C}'_b, \mathsf{si}'_b$ to $\mathcal{A}$.

- $\mathcal{O}_{\mathsf{Verify}}(\mathsf{pk}_a, \mathsf{rslt}_a, \mathsf{proof}_a)$: The challenger runs $\mathsf{Verify}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{rslt}_a, \mathsf{proof}_a)$ and returns the output to $\mathcal{A}$, where si_a is the secret information with respect to the data set D_a .

- $\mathcal{O}_{\mathsf{Verify}}(\mathsf{pk}_b, \mathsf{rslt}_b, \mathsf{proof}_b)$: The challenger runs $\mathsf{Verify}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{rslt}_b, \mathsf{proof}_b)$ and returns the output to $\mathcal{A}$, where si_b is the secret information with respect to the data set D_b .

Guess: The challenger selects a ciphertext cph from $\mathsf{C}_a \cup \mathsf{C}_b$ uniformly at random. $\mathcal{A}$ has q times to guess the keyword with respect to cph.

We can see that given the ciphertexts $\mathsf{C}_a, \mathsf{C}_b$ and $\mathsf{au}_a, \mathsf{au}_b$, $\mathcal{A}$ can only get the values $e(d_{a,i}, g)$ for $d_{a,i} \in \mathsf{D}_a$ and $e(d_{b,i}, g)$ for $d_{b,i} \in \mathsf{D}_b$. Therefore, as the bilinear map e is an one-way function, $\mathcal{A}$ inverts the plaintext from the pairing value with negligible probability ϵ. The only way of inferring the plaintext with respect to cph is with brute-force method by enumerating possible elements within the plaintext space. Hence, given that A has q times to guess plaintexts, the probability of $\mathcal{A}$ outputting correct plaintext is $\frac{q}{|\mathcal{M}|} + \epsilon$. □

Theorem 12. *Assume that* Sig *is an unforgeable signature scheme,* Ac *is a secure* $\mathsf{multi\text{-}accumulator}$ *scheme and H is a collision resistance hash function, the* VDSI *scheme achieves the verifiability property (Definition 9).*

Proof. We show that if there exists a probabilistic polynomial-time adversary $\mathcal{A}$ breaking the verifiability of the VDSI scheme (i.e. presenting an incorrect intersection result and succeeding in the verification with non-negligible probability), there exists an algorithm $\mathcal{B}$ breaking the assumptions that Sig is a secure signature scheme, Ac is a secure multi-accumulator scheme or H is a collision resistant hash function. $\mathcal{B}$ proceeds as follows.

Setup: $\mathcal{B}$ runs $\mathsf{pm} \leftarrow \mathsf{Setup}(1^{\ell})$ and makes pm public known. It then runs $\mathsf{KeyGen}(\mathsf{pm})$ to obtain $\mathsf{sk}_a = (\mathsf{sigSk}_a, \beta_a, \gamma_a)$ and $\mathsf{pk}_a = (\mathsf{sigPk}_a, g^{\beta_a}, g^{\gamma_a})$, runs $\mathsf{KeyGen}(\mathsf{pm})$ to obtain $\mathsf{sk}_b = (\mathsf{sigSk}_b, \beta_b, \gamma_b)$, $\mathsf{pk}_b = (\mathsf{sigPk}_b, g^{\beta_b}, g^{\gamma_b})$, and returns $\mathsf{pk}_a, \mathsf{pk}_b$ to $\mathcal{A}$.

Phase 1: $\mathcal{A}$ can query the following oracles polynomially many times.

- $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pk}_a, \mathsf{D}_a)$: Given the data set D_a, $\mathcal{B}$ runs $(\mathsf{C}_a, \mathsf{si}_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a)$ and returns $\mathsf{C}_a, \mathsf{si}_a$ to $\mathcal{A}$.

- $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pk}_b, \mathsf{D}_b)$: Given the data set D_b, $\mathcal{B}$ runs $(\mathsf{C}_b, \mathsf{si}_b) \leftarrow \mathsf{Enc}(\mathsf{pk}_b, \mathsf{D}_b)$ and returns $\mathsf{C}_b, \mathsf{si}_b$ to $\mathcal{A}$.

- $\mathcal{O}_{\mathsf{AuGen}}(\mathsf{pk}_a, \mathsf{D}_a, \mathsf{pk}_b)$: $\mathcal{B}$ runs $\mathsf{au}_a \leftarrow \mathsf{AuGen}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{pk}_b)$ and returns au_a to $\mathcal{A}$.

- $\mathcal{O}_{\mathsf{AuGen}}(\mathsf{pk}_b, \mathsf{D}_b, \mathsf{pk}_a)$: $\mathcal{B}$ runs $\mathsf{au}_b \leftarrow \mathsf{AuGen}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{pk}_a)$ and returns au_a to $\mathcal{A}$.

- $\mathcal{O}_{\mathsf{Verify}}(\mathsf{pk}_a, \mathsf{rslt}_a, \mathsf{proof}_a)$: $\mathcal{B}$ runs $\mathsf{Verify}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{rslt}_a, \mathsf{proof}_a)$ and returns the output to $\mathcal{A}$, where si_a is the secret information with respect to the data set D_a .

- $\mathcal{O}_{\mathsf{Verify}}(\mathsf{pk}_b, \mathsf{rslt}_b, \mathsf{proof}_b)$: $\mathcal{B}$ runs $\mathsf{Verify}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{rslt}_b, \mathsf{proof}_b)$ and returns the output to $\mathcal{A}$, where si_b is the secret information with respect to the data set D_b .

Challenge: $\mathcal{A}$ selects $\mathsf{D}_a, \mathsf{D}_b$ of its choice, and sends them to $\mathcal{B}$. $\mathcal{B}$ runs $(\mathsf{C}_a, \mathsf{si}_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a)$

and $(\mathsf{C}_b, \mathsf{si}_b) \leftarrow \mathsf{Enc}(\mathsf{pk}_b, \mathsf{D}_b)$, $\mathsf{au}_a \leftarrow \mathsf{AuGen}\ (\mathsf{sk}_a, \mathsf{si}_a, \mathsf{pk}_b)$ and $\mathsf{au}_b \leftarrow \mathsf{AuGen}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{pk}_a)$, and returns $\mathsf{C}_a, \mathsf{si}_a, \mathsf{au}_a, \mathsf{C}_b, \mathsf{si}_b, \mathsf{au}_b$ to $\mathcal{A}$, where $\mathsf{si}_a = (\mathsf{acDig}_a)$ and $\mathsf{si}_b = (\mathsf{acDig}_b)$.

Phase 2: $\mathcal{A}$ can query the oracles the same as Phase 1.

Guess: $\mathcal{A}$ outputs $(\mathsf{rslt}_a, \mathsf{proof}_a), (\mathsf{rslt}_b, \mathsf{proof}_b)$ to $\mathcal{B}$.

This completes the simulation. First let us consider the verification for $(\mathsf{rslt}_a, \mathsf{proof}_a)$. Note that cph_A specifed by proof_a cannot be manipulated, otherwise it breaks the unforgeability of Sig. B decrypts cph_A to obtain acDig_b. In addition, $\mathcal{B}$ decrypts $\mathsf{Dec}(\mathsf{sk}_a, \mathsf{rslt}_a)$, and obtains $T_a = \{H(e(d'_{a,i}, g)) | \mathsf{cph}_{a,i} \in \mathsf{rslt}_a\}$ where $d'_{a,i}$ is the plaintext with respect to $\mathsf{cph}_{a,i}$.

Suppose $T = \{H(e(d_{a,i}, g)) | d_{a,i} \in \mathsf{D}_a\} \cap \{H(e(d_{b,i}, g)) |\ d_{b,i} \in \mathsf{D}_b\}$. If $\mathcal{A}$ breaks the verifiability with $(\mathsf{rslt}_a, \mathsf{proof}_a)$, then at least one of the following cases should hold:

Case 1:

$$1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_a, T_a, \mathsf{acWit}_a, \mathsf{acDig}_b)$$
$$1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_a, T, \mathsf{acWit}_a, \mathsf{acDig}_b)$$
$$T_a = T$$
$$\exists d'_{a,i} \neq d_{a,i}, s.t. H(e(d'_{a,i}, g)) = H(e(d_{a,i}, g))$$

Case 2:

$$1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_a, T_a, \mathsf{acWit}_a, \mathsf{acDig}_b)$$
$$1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_a, T, \mathsf{acWit}_a, \mathsf{acDig}_b)$$
$$T \neq T_a$$

If $\mathcal{A}$ breaks the verifiability with $(\mathsf{rslt}_a, \mathsf{proof}_a)$ with respect to case 1, then it breaks the assumption that H is collision resistant. This is because $d'_{a,i} \neq d_{a,i}$ leads to $e(d'_{a,i}, g)) \neq e(d_{a,i}, g)$ while $H(e(d'_{a,i}, g)) = H(e(d_{a,i}, g))$.

If $\mathcal{A}$ breaks the verifiability with $(\mathsf{rslt}_a, \mathsf{proof}_a)$ with respect to case 2, then it breaks the security of the multi-accumulator scheme by presenting $\mathsf{acRslt} = T_a$, which is different from T.

Therefore, we prove that $\mathcal{A}$ breaks the verifiability of VDSI scheme with respect to $(\mathsf{rslt}_a, \mathsf{proof}_a)$ in a negligible probability under the assumptions that Sig is unforgeable, H is collision resistant and Ac is a secure multi-accumulator scheme. Similarly, we can prove that $\mathcal{A}$ breaks the verifiable of VDSI scheme with respect to $(\mathsf{rslt}_b, \mathsf{proof}_b)$ in a negligible probability. □

3.6 Performance Evaluation

3.6.1 Asymptotic Complexity

Table 3.3: Asymptotic complexity for VDSI scheme, where Exp denotes the exponentiation operation, Pairing denotes the pairing operation, $n = |\mathsf{acD}_a|$, $m = |\mathsf{acD}_b|$ and $k = |\mathsf{D}_a \cap \mathsf{D}_b|$.

Algorithm	Computational Complexity
Enc	$3n\mathsf{Exp} + n\mathsf{Pairing} + \mathsf{acGen}$
Dec	$2n\mathsf{Exp}$
AuGen	$4\mathsf{Exp} + \mathsf{sigSign}$
SetOp	$3(n+m)\mathsf{Pairing} + 2\mathsf{acProve}$
Verify	$2(k+1)\mathsf{Exp} + k\mathsf{Pairing} + \mathsf{acVerify} + \mathsf{sigVerify}$

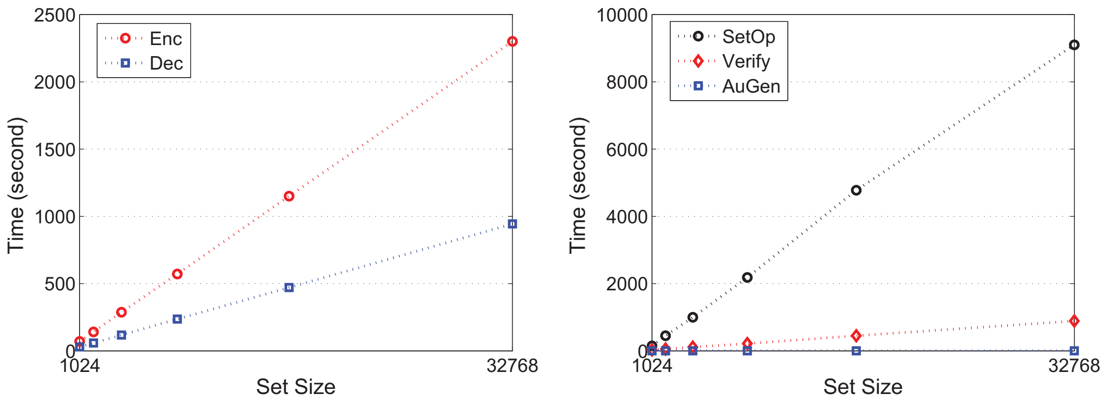
Table 3.3 describes the asymptotic complexity of algorithms in the VDSI scheme. While algorithm SetOp is more costly when compared with the other algorithms, it is worth noting that algorithm SetOp is executed by the cloud rather than by the users.

Table 3.4: Asymptotic performance comparison for the VDSI scheme and the straightforward solution. We assume that the straightforward solution adopting the encryption and decryption algorithms of the VDSI scheme. Here n is the size of Alice's data set, m is the size of Bob's data set, k is the size of set intersection, and Comp(PSI) and Comm(PSI) denotes the respective computation and communication complexity of the private set intersection protocol. Note that Comp(PSI) and Comm(PSI) are both linear to the size of data sets $(m+n)$ for the state-of-the-art solution [47].

		VDSI solution				Straightforward solution		
Phase		Alice	Bob	Cloud		Alice	Bob	Cloud
Data outsourcing	Computation	$O(n)$	$O(m)$	N/A		$O(n)$	$O(m)$	N/A
	Communication	$O(n)$	$O(m)$	$O(n+m)$		$O(n)$	$O(m)$	$O(n+m)$
Set operation	Computation	$O(k)$	$O(k)$	$O(m+n)$		$O(n)$+Comp(PSI)	$O(m)$+Comp(PSI)	N/A
	Communication	$O(k)$	$O(k)$	$O(k)$		$O(n)$ +Comm(PSI)	$O(m)$ +Comm(PSI)	$O(n+m)$

Table 3.4 summarizes the communication and computational overhead incurred by the VDSI

solution, which is grouped into two phases: (i) data outsourcing phase, during which the cloud users outsource their encrypted private data sets to the cloud (i.e. Enc); (ii) set operation phase, during which the cloud users attain the intersection set (i.e. AuGen, SetOp and Verify). We compare the overhead with its counterpart that is incurred by the straightforward solution, namely that Alice and Bob download their outsourced encrypted data from the cloud, decrypt their data, and then run a PSI protocol to jointly compute the intersection set. From the perspective of cloud users, we observe that the VDSI scheme outperforms the straightforward solution in both communication and computational complexities. Assume that the data sets have been stored in the cloud, the VDSI scheme only incurs $O(k)$ computational complexity to obtain the intersection set (including the cost for verification) and $O(k)$ communication complexity for returning the intersection set to the user, where k is the size of intersection set. This means that the VDSI scheme is optimal (up to a constant factor). In contrast, the straightforward solution incurs $O(m+n)$ in computational and communication overhead where m and n are the sizes of the two data sets. The advantage of VDSI scheme is most substantial when $k << m$ or $k << n$.



(a) Performance of Enc and Dec

(b) Performance of AuGen, SetOp and Verify

Figure 3.2: Performance of VDSI, where Alice and Bob outsource their data sets of the same size (i.e., $m = n$), algorithms Enc, Dec, AuGen and Verify run on the Client Machine, and algorithm SetOp runs on the Server Machine.

3.6.2 Performance Evaluation

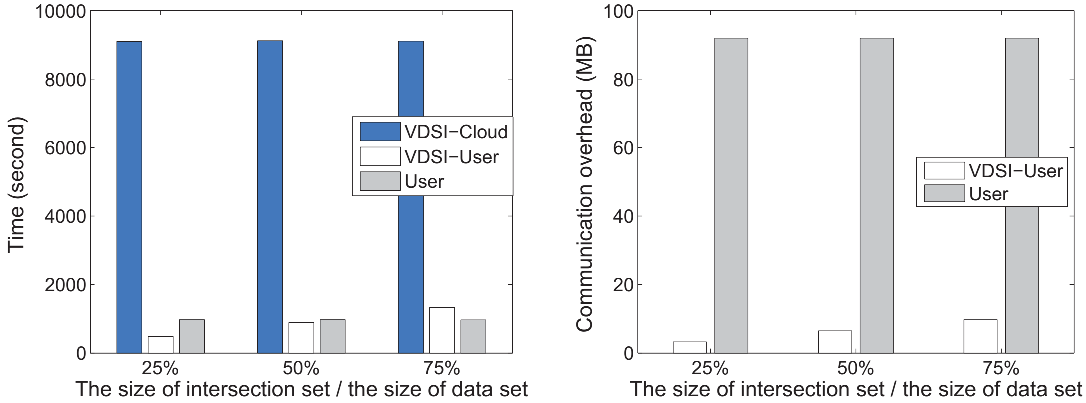
Implementation We implemented the VDSI scheme in JAVA based on the Java Pairing Based Cryptography library (jPBC) [3]. In our implementation, we instantiated the bilinear map with Type A pairing ($\ell = 512$), which offers a level of security that is comparable to 1024-bit DLOG [3]. We instantiated the signature scheme with the DSA signature scheme provided by JDK1.6. We varied the set size (m and n) from 2^{10} to 2^{15}. The algorithms run by the cloud users (i.e., Enc, Dec, AuGen and Verify) were executed on a CLIENT MACHINE with Linux OS, 2.93GHz Intel Core Duo CPU (E7500), and 2GB RAM. The algorithm run by the cloud (i.e., SetOp) was executed on a SERVER MACHINE with Linux OS, 4 processors of 2.40GHz Intel Xeon CPU, and 8GB RAM.

Evaluation and result In our experiments we set the same size for data sets owned by the two cloud users, i.e., $m = n$, and set the size of intersection set $k = n/2$. For algorithms Enc, Dec, AuGen and Verify, we evaluated each algorithm's execution time for both cloud users and treat their average execution time as the real execution time. Figure 3.2a plots the execution time of Enc and Dec that are run by the cloud users. We observe that the execution times for both algorithms are almost linear to the size of data sets. We also can see that the execution of Enc is more expensive than that of Dec. However, Enc is executed only once when the cloud user outsources data sets. Figure 3.2b shows the execution time of SetOp (run by the cloud) and the execution time of AuGen and Verify (run by the cloud users). We observe that the execution time of AuGen and Verify is much more smaller than that of the algorithm SetOp. This suggests that cloud users should leverage the cloud's computation resources by delegating set intersection operations.

Performance Comparison In order to understand the benefit and limitation of the VDSI solution, we compare it with the straightforward solution, where data sets are encrypted by the algorithm Enc and Dec of the VDSI, and the private set operations between two data users are performed by the protocol (Java version) in [47], which is the most efficient PSI protocol in the literature.

We ran the experiments on the same SERVER MACHINE with Linux OS, 4 processors of

2.40GHz Intel Xeon CPU, and 8GB RAM, with the data sets each consisting of 32768 elements. We vary the size of the intersection set as 25%, 50% and 75% of the size of the data set respectively, and compare the communication and computation overhead in the set operation phase (we did not compare the cost of the data outsourcing phase because they are the same for both solutions), which is shown in Figure 3.3. From Figure 3.3a we observe that the computation overhead in the VDSI solution decreases when the size of the intersection set decreases. However, the computation overhead in the straightforward solution remains the same regardless the size of the intersection set. We also can see in Figure 3.3b that the communication overhead for the data users in the VDSI solution is linear to the size of intersection set, and is much less than that of the straightforward solution. This advantage can become more substantial when the size of the intersection size is far less than the size of data set owned by the data users. We note that while the straightforward solution can use other efficient encryption schemes (e.g., symmetric encryption) to encrypt/decrypt data sets to achieve higher efficiency, it cannot reduce the communication cost that is linear to the size of data set. Therefore, our VDSI solution is extremely suitable for computing intersection set whose size is far less than that of the data sets.



(a) Computation overhead.

(b) Communication overhead.

Figure 3.3: Performance comparison between our VDSI solution and the straightforward solution, where each data user outsourced the data set of 32768 elements. We vary the size of the intersection set with 25%, 50% and 75% of the size of the data set respectively. VDSI-User and VDSI-Cloud denote the costs spent by the cloud and each data user in the VDSI solution and User represents the cost spent by each data user in the straightforward solution.

3.6.3 Improvement with Parallelization

In our proposed scheme, the execution of Enc, Dec, SetOp and Verify can be implemented more efficiently with parallelization, because operations related to elements of data sets are independent. In practice, we implemented the algorithms by using multiple threads to compute independent operations (e.g. encrypting elements of the data sets, decrypting ciphertexts, and transforming ciphertexts into a value of G_T). In the parallelization version, we created 4 threads and ran the algorithms Enc, Dec, SetOp and Verify on the SERVER MACHINE with Linux OS, 4 processors of 2.40GHz Intel Xeon CPU, and 8GB RAM. To understand the efficiency gain of parallelization, we also ran the algorithms without parallelization on the same SERVER MACHINE. Figure 3.4 shows the performance comparison, which indicates that the algorithms using parallelization are about 2 times faster than their counterparts that do not use parallelization. This means that our scheme can leverage the multi-core architecture, and that our scheme is suitable for delegating set intersection over outsourced large data sets.

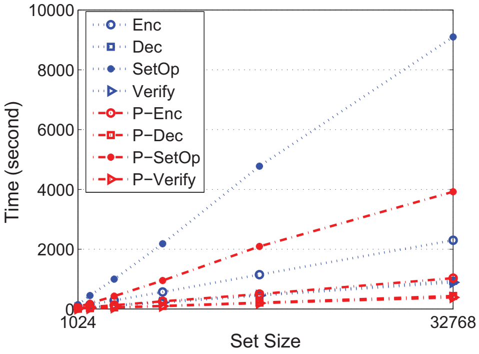


Figure 3.4: Performance comparison for algorithms Enc, Dec, AuGen and Verify executed on SERVER MACHINE. The algorithms with prefix "P-" were implemented with parallelization, and the algorithms without prefix were implemented without parallelization.

3.7 Chapter Summary

We have introduced the novel notion of VDSI, which allows two users to outsource to the cloud their encrypted data sets as well as the set intersection operation on ciphertexts. This is achieved without giving the cloud the capability to decrypt the encrypted data, while enabling the users to hold the misbehaving cloud accountable.

Our study brings interesting and challenging open problems for future research. In addition to the ones mentioned in the chapter (e.g., incorporating fine-grained access control, if possible), we need to design the same kinds of solutions for other set operations.

Chapter 4: VERIFIABLE SQL QUERIES ON OUTSOURCED DYNAMIC DATABASES

4.1 Introduction

Outsourcing databases to the cloud has been a popular trend in cloud computing because cloud users, which can be either home users or enterprise users, can access and process the outsourced databases from anywhere and at any time, and enjoy the benefits such as cost saving, on-demand self-service, resource elasticity, etc. In spite of those appearling advantages, outsourcing databases to the cloud also raises security concerns (even for non-confidential databases) , which hinders the potential cloud users from outsourcing their databases to the cloud.

One of the security concerns is that cloud users who query the outsourced databases may wonder whether or not the queries had been executed faithfully by the cloud servers. This is because large-scale cloud infrastructures exposes vast vulnerability surfaces, ranging from hardware/software failures, operation errors and even malicious attacks, and the cloud infrastructure providers have the incentive not to honestly execute the protocols (e.g., for saving resources or better service response time). This naturally leads to the question that how can database queriers assure that the cloud server honestly executed the SQL queries? Another security concerns is related to data privacy for outsourced confidential databases, which can be accessed by either insiders (e.g., database administrators) or potential outside attackers. The importance of assuring the privacy of outsourced private data has led to legislations, such as EU Data Protection Directive [1] and U.S. HIPAA [2]. Therefore, This naturally leads to the advanced question: How can assure that the cloud faithfully executed the SQL queries on outsourced databases?

4.1.1 Our Contribution

We present a novel solution for verifiable SQL queries on outsourced dynamic databases (VQDDB). In particular, by comparing with the current state-of-the-art solutions that supports selection / projection / join queries simultaneously (other works only support a single type of queries and are thus limited in scope and applicability), our solution can be characterized from three perspectives:

- From the perspective of functionality, our solution supports four kinds of queries — selection, projection, join, and aggregate. As we will discuss in detail, the two state-of-the-art solutions [83, 99] support selection, projection and join queries but do not support aggregate queries (see Table 4.2). Moreover, our solution supports more *flexible* join queries, in the sense that they do not have to be defined with respect to pre-defined keyword attributes (more on this later).

- From the perspective of security, our solution is provably secure as long as the two underlying building-blocks are secure with respect to standard security definitions. This is thanks to our "compiler"-like modular construction. The idea is to start from relatively simple building blocks and *compile* them into a full-fledged scheme. As long as the building blocks are secure, the complete scheme is also secure under the same assumptions. Clearly, building blocks can be later substituted with improved versions without affecting the security of the final scheme.

- From the perspective of efficiency, our solution is characterized as follows. The efficiency of our mechanism mainly comes from a new cryptographic primitive, dubbed Homomorphic Linear Tag (HLT), which is weaker than the Homomorphic Linear Authenticator introduced in [7] and may be of independent value. Let m be the number of attributes and n be the number of tuples.

  1. Our solution incurs $O(n)$ storage complexity at the cloud side, in contrast to $O(mn)$ of [83, 99].
  2. For projection query, our solution incurs $O(n)$ modular exponentiations at the querier side. This is not as efficient as $O(n)$ hash operations of [83], but is in sharp contrast to $O(nk)$ exponentiation operations on bilinear map of [99] where $k \leq m$ is the number

of attributes in the projection operation. Our solution incurs $O(n+m)$ communication complexity, which is in sharp contrast to $O((m-k)n)$ attribute values of [83] where $k \leq m$ is the number of attributes in the projection operation, and is the same as in [99].

3. For selection query, our solution incurs $O(n)$ exponentiations at the querier side, which is not as efficient as $O(n)$ hash operations of [83], but is more efficient than $O(n)$ exponentiation operations on bilinear map of [99]. Our solution incurs communication of $O(n)$ tags, which is less efficient than $O(\log n)$ values of [83], and comparable to $O(n)$ of [99].
4. For join query with respect to two tables of n tuples and m attributes, our solution incurs $O(n)$ modular exponentiations at the querier side, which is not as efficient as $O(n \log n)$ hash operations of [83], but more efficient than $O(n)$ exponentiation operations on bilinear map of [99]. Our solution incurs the communication complexity of $O(n+m)$ tags, which is much more efficient than $O(n(\log n))$ hash values of [83], and comparable to $O(n)$ of [99].

4.1.2 Related Work

The problem of achieving verifiable SQL query in the context of outsourced databases has attracted a fair amount of attention. There are two main approaches to this problem. The *tree-based* approach mainly used Merkle hash tree [87] or its variants to index search keys [46, 64, 83, 89, 90, 94, 97, 125]. Roughly speaking, this approach leads to logarithmic complexity in terms of both communication and verification. To enable efficient verification, Goodrich et al [64] used Merkle hash tree to maintain signatures at multiple hash tree levels. The state-of-art solution regarding tree-based approach is due to Li et al [83], which proposed Merkle B-tree and Embedded Merkle B-tree in order to reduce the I/O operations.

The *signature-based* approach mainly used the signature aggregation technique [23, 91] to aggregate the validity evidence of query answers [91, 92, 98, 99]. Roughly speaking, this approach can lead to low (even constant) communication complexity, but may require special treatment to handle more powerful (e.g., projection) queries and often leads to large storage and computational complexities. The state of the art solution is due to [99], which will be compared with our solution in Section 4.6. Essentially, [99] used aggregate signatures to dramatically reduce the proof size. It signed each attributes so that only a single signature will be returned as the proof for projection query. On the other hand, it applied a chaining signing technique to build the index for the search key to facilitate range queries. To deal with dynamic updates, a certified bitmap must be published at every update period which complicates the verification of a query result. In summary, [99] incurs a large storage and communication overhead, but it also involves complex and expensive operations, such as exponentiations and pairing operations.

In addition to the above, there are works on authenticating the answers to aggregate queries using authenticated prefix-sums trees [84], authenticating the answers to join queries [124], and on authenticating count query with respect to multi-dimensional data in a privacy-preserving fashion [122]. These solutions do not apply to our specific class of problems and do not provide the solution we are after. For example, the scheme presented in [122] does not support selection nor projection queries. Finally, probabilistic integrity assurance was also investigated in [121].

Another related topic is outsourced verifiable computation [6, 39, 53], attaining operation-sensitive verification on general functionalities. However, they are inadequate to meet other functionalities, e.g., public verifiability and dynamic updates. Certification of data structure was studied in [115], where solutions are introduced for general query types.

4.2 Problem Formulation

4.2.1 System Model

We consider the system model of Verifiable SQL Queries on Outsourced Dynamic Databases (VQDDB) illustrated in Figure 4.1, which consists of three participants: a cloud server, which provides storage services and can perform SQL queries on outsourced databases on behalf of data owners, a data owner, who outsources its plaintext database to the cloud and updates the databases as necessary, and a data querier (e.g. business partners of the data owner), who issues SQL queries towards the outsourced databases and receives query answers from the cloud. More specifically, the data owner owns a relational database D, which consists of multiple tables having multiple tuples and multiple attributes. After the database D was outsourced to the cloud, the data owner can update D by interacting with the cloud and should be able to verify that the cloud honestly executed the updates. The data querier can issue any SQL query qry to the cloud, receive query answers from the cloud, and verify that the query qry is faithfully executed with respect to the database D.

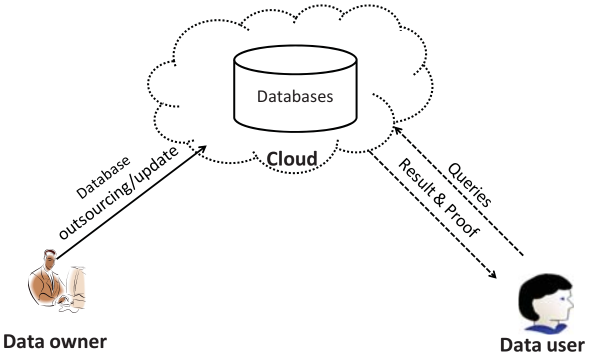


Figure 4.1: System model of verifiable SQL queries on outsourced dynamic database, where the data owner outsources the databases to the cloud and can update them as necessary, and the data queirer can query the databases stored in the cloud and retrieve query answers from the cloud.

We assume that the data owner and data user are trusted. The cloud server is not trusted since it may manipulate the SQL queries on stored databases, which implicitly implies that the cloud may manipulate the outsourced databases as well the answer of SQL queries.

4.2.2 Functional and Security Definitions

We present the functional and security definitions of verifiable SQL queries on outsourced dynamic database (VQDDB), which was somewhat inspired by the definitions of Authenticated Data Structures that allow verifiable queries on dynamic sets [103, 105].

Definition 12. (VQDDB) A scheme for verifiable SQL queries on outsourced dynamic database consists of the following algorithms:

- KeyGen: This algorithm is run by the data owner to generate a pair of private and public keys $(\mathsf{sk}, \mathsf{pk})$ by taking as input the primary security parameter ℓ. It can be denoted as

$$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^{\ell}).$$

- SetUp: This algorithm is executed by a data owner O before outsourcing its database D to the cloud server. By taking as input the private key sk and the database D, this algorithm outputs some cryptographic auxiliary information au and state information state. Both D and au will be outsourced to the cloud server and state will be made public (so as to allow third parties to verify the query answers). It can be denoted as

$$(\mathsf{state}, \mathsf{au}, D) \leftarrow \mathsf{SetUp}(\mathsf{sk}, D)$$

- Update: This protocol is executed between a data owner O and the cloud server S to perform update operations, the detail of which is described by upd. By taking as input the private key sk and the current state information state, the data owner interacts with the cloud server, which takes as input the stored database D and the cryptographic auxiliary information au. The data owner O updates its state information to state' with respect to the update information upd, and the server obtains au' and D' by updating the stored database accordingly. It can be denoted as

$$(\mathsf{au}', \mathsf{state}', D') \leftarrow (O(\mathsf{sk}, \mathsf{state}, \mathsf{upd}) \leftrightarrow S(\mathsf{au}, D))$$

- $\mathsf{QueryandVrfy}$: This protocol is executed between a data querier Q, who issues a SQL query qry, and the server S, which answers the query with the result rslt and a proof prf. The data querier verifies the result rslt with corresponding prf, and outputs reject if rslt is not valid with respect to the query qry and the state state; otherwise, the data user accepts rslt and prf.

It can be denoted as

$$\{(\mathsf{reject}), (\mathsf{accept}, \mathsf{rslt}, \mathsf{prf})\} \leftarrow (Q(\mathsf{pk}, \mathsf{qry}, \mathsf{state}) \leftrightarrow S(\mathsf{au}, D))$$

Correctness of the VQDDB scheme requires that, for any honest cloud server, given $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\ell)$, $(\mathsf{state}, \mathsf{au}, D) \leftarrow \mathsf{SetUp}(\mathsf{sk}, D)$ and $(\mathsf{au}_i, \mathsf{state}_i, D_i) \leftarrow (O(\mathsf{sk}, \mathsf{state}_{i-1}, \mathsf{upd}_{i-1}) \leftrightarrow S(\mathsf{au}_{i-1}, D_{i-1})), 1 \leq i \leq n$ where n is a polynomial number, the following always holds that for any query qry

$$(\mathsf{accept}, \mathsf{rslt}, \mathsf{prf}) \leftarrow (Q(\mathsf{pk}, \mathsf{qry}, \mathsf{state}_n) \leftrightarrow S(\mathsf{au}_n, D_n))$$

Intuitively, a VQDDB scheme should satisfy the security requirement – soundness – even in the presence of a probabilistic polynomial time adversary (i.e. the cloud server). The intuition of the soundness is to assure that the cloud cannot return incorrect query answer without being detected by the data querier since the cloud server is untrusted. Specifically, we say a VQDDB scheme is sound if for any query qry on database D, the cloud server can not return an *incorrect* rslt such that

$$(\mathsf{accept}, \mathsf{rslt}, \mathsf{prf}) \leftarrow (Q(\mathsf{pk}, \mathsf{qry}, \mathsf{state}_n) \leftrightarrow S(\mathsf{au}_n, D_n)).$$

Formally, we define the security game between the adversary $\mathcal{A}$ and the challenger as follows:

Setup: The challenger runs $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\ell)$ and gives pk to the adversary $\mathcal{A}$.

Phase 1: $\mathcal{A}$ interacts with the challenger as follows:

- $\mathcal{A}$ makes oracle access to SetUp, by presenting a database D_0. The challenger computes

$$(\mathsf{state}_0, \mathsf{au}_0, D_0) \leftarrow \mathsf{SetUp}(\mathsf{sk}, D_0),$$

  and gives $\mathsf{state}_0, \mathsf{au}_0$ to $\mathcal{A}$. The challenger makes state_0 public.

- $\mathcal{A}$ is allowed to ask for updating D_0 adaptively with $\mathtt{upd}_i$, $n \leq i \geq 1$. The challenger interacts with $\mathcal{A}$ to compute

$$(\mathsf{au}_i, \mathsf{state}_i, D_i) \leftarrow (O(\mathsf{sk}, \mathsf{state}_{i-1}, \mathsf{upd}_{i-1}, \mathsf{au}_{i-1}, D_{i-1}) \leftrightarrow S(\mathsf{au}_{i-1}, D_{i-1})).$$

Challenge: $\mathcal{A}$ outputs a query qry and a query result rslt with proof prf. We say $\mathcal{A}$ wins the game if

$$(\mathsf{accept}, \mathsf{rslt}, \mathsf{prf})\} \leftarrow (Q(\mathsf{pk}, \mathsf{qry}, \mathsf{state}_n) \leftrightarrow S(\mathsf{au}_n, D_n))$$

for some $k \geq 0$ and $\mathsf{rslt} \neq \mathsf{localRst}$, where $\mathsf{localRst} \leftarrow \mathsf{LocalQuery}(\mathsf{qry}, D_n)$ is produced by the challenger that faithfully executes query qry on database D_n

Definition 13. Let $\Lambda = (\mathsf{KeyGen}, \mathsf{SetUp}, \mathsf{Update}, \mathsf{QueryandVrfy})$ be a VQDDB scheme and $\mathcal{A}$ be a probabilistic polynomial-time adversary. We say that Λ is sound if any polynomial-time algorithm $\mathcal{A}$ can win the security game with at most a negligible probability.

4.3 Building-Block I: Authenticated Data Structure on Outsourced Ordered Data (AuthDS)

In this section, we introduce a building block that is to assure verifiable range query on an ordered data set, which is outsourced to the server. This building-block is called Authenticated Data Structure on Outsourced Ordered Data (AuthDS), the definition and security requirement of which are similar to those of VQDDB.

4.3.1 Definition of AuthDS

Definition 14. (AuthDS) The scheme of authenticated data structure on outsourced ordered data is operated in the following model: The data owner owns an ordered data set E and preprocesses it before outsourcing it to the cloud server. The data querier can query the cloud server to retrieve data that falls within the range $[a, b]$ and be able to verify that the returned is correct with respect to the outsourced ordered data set and the range query. More specifically, an AuthDS scheme consists of the following algorithms, which are similar to those in Definition 12:

- KeyGen: This key generation algorithm generates the public/private key as KeyGen in Definition 12.

- SetUp: This setup algorithm is the same as SetUp in Definition 12, except that the database is replaced with an ordered set E.

- Update: This update protocol proceeds is the same as Update in Definition 12, except that the update operations are element insertion/deletion/update on the ordered data set E.

- $\mathsf{QueryandVrfy}$: This query protocol is the same as $\mathsf{QueryandVrfy}$ in Definition 12, except that it only supports range query $\mathsf{qry}(a, b)$ that asks for all elements in the interval $[a, b]$.

The correctness of AuthDS can be defined similar to that of VQDDB scheme.

Definition 15. (soundness of AuthDS) Given an AuthDS scheme, $\Lambda = (\mathsf{KeyGen}, \mathsf{SetUp}, \mathsf{Update}, \mathsf{QueryandVrfy})$, we consider the security game as in Definition 13, except that (i) the initial database is replaced with an ordered set E, (ii) the update operation is element insertion, deletion or update on the ordered data set, and (iii) the queries are only range queries $\mathsf{qry}(a, b)$ that ask for elements in the interval $[a, b]$. We say that Λ is sound if any polynomial-time algorithm $\mathcal{A}$ can win the game with at most a negligible probability.

4.3.2 Construction and Analysis of AuthDS: Merkle B-Tree

Now we describe an AuthDS scheme, which is a Merkle B-tree (MB-tree) and has been extensively studied in [83, 90]. Merkle B-tree applies the basic idea of Merkle tree on a B^+ tree structure, where the operations on Merkle B-tree (e.g., insertion and deletion) are similar to those on B^+ tree. The primary advantage of B^+ tree is that it has a large fan-out, which can reduce the number of I/O operations when searching for an element [83]. Let $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a secure signature scheme. Let E be an ordered set. The Merkle B-tree scheme consists of algorithms as follows:

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\ell)$: This algorithm runs $\mathsf{Sig.KeyGen}(1^\ell)$ to obtain a pair of private and public keys $(\mathsf{sk}, \mathsf{pk})$.

- $(\mathsf{state}, \mathsf{au}) \leftarrow \mathsf{SetUp}(\mathsf{sk}, E)$: This algorithm outputs a succinct signature which can be used for verification. The structure of Merkle B-tree $\mathcal{T}$ is similar to B^+ tree, where the leaves store elements in the ordered set E, and the values of internal nodes are computed from the concatenation of the values of their children via an appropriate hash function. The root of the tree will be signed to produce the state information, denoted by $\mathsf{state} = \mathsf{Sig.Sign}(\mathcal{T})$ and $\mathsf{au} = \mathcal{T}$.

- Update: The update protocol fulfills update operations. For simplicity, we consider the example of the replacement operation while assuming that the replacement preserves the order of the elements. We refer to [83] for details about the insertion and deletion operations. Suppose upd = "update the element E_i to E_i'". Upon receiving upd from the data owner, the server updates E to E' by replacing E_i with E_i', and updates $\mathcal{T}$ to $\mathcal{T}'$. The server provides a proof, a path of E_i in $\mathcal{T}$, namely a sequence including values of the nodes from E_i to the root of MB-tree as well as the values of these nodes' siblings. The data owner can hash the path of E_i from the bottom to the top and verify whether the root is valid with respect to state state or not. If so, the data owner updates the path from the bottom to the top by replacing E_i with E_i', which will result in a new root, signs the new root, and sets $\mathsf{state}' = \mathsf{Sig.Sign}(\mathcal{T}')$; otherwise, the data owner aborts.

- $\mathsf{QueryandVrfy}$: Given a range query $\mathsf{qry}(a, b)$, the server outputs a proof prf showing that rslt contains all elements in $[a, b]$.

  - If rslt is empty, which means there exists some s, such that $E_s <= a, b <= E_{s+1}$. The server returns the proof prf including two paths: a path of E_s and a path of E_{s+1}. The querier hashes each path from bottom to the top, and verify whether the roots match the state state, and E_s is neighbor to E_{s+1}. If so, the querier returns the null set rslt, prf, and accept. Otherwise, abort.

- If rslt is not null, suppose the query result is $(E_s, \ldots, E_t), s \leq t$. The server returns the proof prf including two paths: one path of the left-most neighbor leaf of E_s, and the other path of the right-most leaf of E_t. Then the querier uses prf and the result rslt to construct a B^+ tree, and verifies whether the root of the this B^+ tree is valid for $\mathsf{state} = \mathsf{Sig.Sign}(\mathcal{T})$. If so, the querier returns $(\mathsf{rslt}, \mathsf{prf}, \mathsf{accept})$; otherwise, the querier aborts.

Theorem 13. *Assuming that* Sig *is a secure signature scheme and the hash function is collision resistant, the Merkle B-tree scheme is sound with respect to Definition 15.*

4.4 Building Block II: Homomorphic Linear Tag (HLT)

Now we present the second building block, called Homomorphic Linear Tag (HLT) scheme. Intuitively, HLT offers the following property: Given messages $M_1, \ldots, M_n$ and their respective tags $\sigma_1, \ldots, \sigma_n$ generated by HLT, then for coefficients $c_1, \ldots, c_n$ in a pre-defined coefficient space, the aggregate message $M = \sum_{i=1}^{n} c_i M_i$ can be verified via the aggregate tag σ of $\sigma_1, \ldots, \sigma_n$ and the coefficients $c_1, \ldots, c_n$. HLT can be divided into two types according to the necessary requirement for verifiability:

- Publicly verifiable HLT: It allows anyone (*without knowing any secret*) to conduct the verification of aggregated messages and corresponding tags. In order to allow any third party to verify the integrity of query answers, this type of HLT is needed for the purpose of the present chapter.

- Privately verifiable HLT: It allows someone *who knows the relevant secret* to conduct the verification. Putting this into the context of the present chapter, this type of HLT can be used to allow the data owner (but not third parties) to verify the integrity of query answers. Therefore, this type of HLT will not be discussed further in the chapter.

The concept of HLT was inspired by the notion of Homomorphic Linear Authenticator (HLA), which was formally introduced in [9]. The difference between them is that HLT is weaker than

HLA because HLT only considers attacks that do not attempt to tamper the individual tags (which is dealt with by another layer of protection for free, namely by the first building-block); whereas, HLA explicitly accommodates attacks that aim to tamper the individual tags. This makes it possible to construct HLT schemes that are more efficient that their HLA counterparts. It is worthwhile to point out the following feature of HLT and HLA: the aggregated message M and the aggregated tag σ are sufficient to allow the verifier to test their validity *without* knowing the individual messages $M_1, \ldots, M_n$. This is not the case for aggregate signatures [23], batch RSA [49], and condensed RSA [91], which are not sufficient for the purpose of HLT or HLA.

4.4.1 Definitions of HLT

Definition 16. (publicly verifiable HLT) A publicly verifiable HLT scheme consists of the following algorithms:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\ell)$: This algorithm takes as input a security parameter ℓ, and outputs a pair of public and private keys $(\mathsf{pk}, \mathsf{sk})$. It may optionally specify a coefficient domain $\mathcal{C}$ and a message space $\mathcal{M}$.

- $\sigma_i \leftarrow \mathsf{TagGen}(\mathsf{sk}, M_i)$: This algorithm takes as input the private key sk and a message $M_i \in \mathcal{M}$, and outputs a tag σ_i for M_i.

- $\sigma \leftarrow \mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$: This linear aggregation algorithm takes as input a vector of tags $\vec{\mathsf{Tag}} = (\sigma_1, \ldots, \sigma_n)$ with respect to a vector of messages $\vec{M} = (M_1, \ldots, M_n)$ and a vector of coefficients $\vec{c} = (c_1, \ldots, c_n)$. It outputs an aggregate tag σ with respect to the aggregated message $M = \sum_{i=1}^{n} c_i M_i$.

- $\{0, 1\} \leftarrow \mathsf{Vrfy}(\mathsf{pk}, M', \sigma')$: This deterministic algorithm takes as input the public key pk, a candidate message M', and a tag σ'. It outputs 1 if σ' is valid with respect to M', and outputs 0 otherwise.

We require a HLT scheme to be correct, meaning that any faithfully aggregated message M and tag σ are always accepted as valid. Formally, this means that for $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\ell)$, $\vec{M} = (M_1, \ldots, M_n) \in \mathcal{M}^n$, $\vec{\mathsf{Tag}} = (\sigma_1, \ldots, \sigma_n)$ where $\sigma_i \leftarrow \mathsf{TagGen}(\mathsf{sk}, M_i)$ for $1 \leq i \leq n$, and $\vec{c} = (c_1, \ldots, c_n) \in \mathcal{C}^n$, then $\sigma \leftarrow \mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$ implies $1 \leftarrow \mathsf{Vrfy}(\mathsf{pk}, \sum_{i=1}^n c_i M_i, \sigma)$.

Informally, the security of HLT requires that for any tag σ, there is no probabilistic polynomial time adversary that can present M' such that $1 \leftarrow \mathsf{Vrfy}(\mathsf{pk}, M', \sigma)$ in the follwing cases: (i) $M' \neq M$ if σ is the tag of message M; or (ii) $M' \neq \sum c_i M_i$ if σ is the aggregated tag with respect to messages $M_1, \ldots, M_n$ with coefficients $c_1, \ldots, c_n$. Formally, we define the secuity game between the adversary $\mathcal{A}$ and the challenger as follows:

Setup: The challenger runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\ell)$ and gives pk to $\mathcal{A}$. The optional coefficient domain $\mathcal{C}$ and the message space $\mathcal{M}$ are specified by KeyGen.

Phase 1: $\mathcal{A}$ can make the oracle query to TagGen and HLTAgg as follows:

- $\mathcal{A}$ may make oracle queries to TagGen by adaptively selecting $M_1, \ldots, M_n$ from the message space $\mathcal{M}$ to obtain the HLT tag for $1 \leq i \leq n$. The challenger computes $\sigma_i \leftarrow \mathsf{TagGen}(\mathsf{sk}, M_i)$ for $1 \leq i \leq n$ and returns σ_i to $\mathcal{A}$. The challenger keeps the lists of messages and tags: $(M_1, \ldots, M_n)$ and $(\sigma_1, \ldots, \sigma_n)$.

- $\mathcal{A}$ may make oracle queries to HLTAgg by selecting a vector of coefficients $\vec{c} = (c_1, \ldots, c_n)$, obtain the aggregate tag σ, and run Vrfy with the aggregate tag σ and the aggregated message $\sum_{i=1}^n c_i M_i$. This can be performed polynomially many times.

Challenge: Eventually, $\mathcal{A}$ selects a vector of coefficients $\vec{c} = (c_1, \ldots, c_n)$, where $c_i \in \mathcal{C}$, and some $M' \in \mathcal{M}$. We say $\mathcal{A}$ wins the game if $M' \neq \sum_{i=1}^n c_i M_i$ and $1 \leftarrow \mathsf{Vrfy}(\mathsf{pk}, M', \sigma)$, where $\sigma \leftarrow \mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$ was computed by the challenger, where $\vec{\mathsf{Tag}} = (\sigma_1, \ldots, \sigma_n)$ corresponds to the message vector $(M_1, \ldots, M_n)$ that can be identified by the coefficient vector $\vec{c} = (c_1, \ldots, c_n)$ provided by the adversary $\mathcal{A}$.

Definition 17. Let $\Lambda = (\mathsf{KeyGen}, \mathsf{TagGen}, \mathsf{HLTAgg}, \mathsf{Vrfy})$ be a HLT scheme and $\mathcal{A}$ be a probabilistic polynomial-time adversary. We say Λ is secure if no probabilistic polynomial-time algorithm $\mathcal{A}$ can win the game with a non-negligible probability in the security parameter ℓ.

From the security game, we observe that the adversary $\mathcal{A}$ is only allowed to manipulate the messages $M_1, \ldots, M_n$ but not the tags. This further explains why HLT is weaker than the aforementioned HLA (Homomorphic Linear Authenticator) [7, 9, 109], where the adversary is allowed to manipulate *both* messages and tags. This can be captured by the following Lemma:

Lemma 1. *Any secure HLA scheme as defined in [9] is also a secure* HLT *scheme as defined above.*

4.4.2 Construction and Analysis of HLT

We present a HLT scheme whose security is based on the Discrete Logarithm (DLOG) problem. The scheme consists of the following algorithms.

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\ell)$: This algorithm generates private and public keys as follows:

    1. Let q be a ℓ-bit prime and p be another large prime such that $q|(p-1)$.
    2. Select v_1 and v_2 uniformly at random from Z_p^* such that the order of v_1 and v_2 is q.
    3. Select s_{j1}, s_{j2} uniformly at random from Z_q^* and set $z_j = v_1^{-s_{j1}} v_2^{-s_{j2}} \mod p$, for $1 \leq j \leq m$.
    4. Let $\mathsf{sk} = \{(s_{11}, s_{12}), \ldots, (s_{m1}, s_{m2})\}$ and $\mathsf{pk} = \{v_1, v_2, z_1, \ldots, z_m\}$.
    5. The coefficient domain $\mathcal{C}$ is $[0, q)$ and the message space is $\mathcal{M} = [0, q)^m$.

- $\sigma_i \leftarrow \mathsf{TagGen}(\mathsf{sk}, M_i)$: For $M_i \in \mathcal{M}$, the tag σ_i is computed by selecting r_1, r_2 uniformly at random from Z_q^* and setting:

$$\begin{aligned} x &= v_1^{r_1} v_2^{r_2} \mod p, \\ y_1 &= r_1 + \sum_{j=1}^{m} M_i[j] s_{j1} \mod q, \\ y_2 &= r_2 + \sum_{j=1}^{m} M_i[j] s_{j2} \mod q. \end{aligned}$$

Let $\sigma_i = (x, y_1, y_2)$.

- $\sigma \leftarrow \mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$: Given tags $\vec{\mathsf{Tag}} = (\sigma_1, \ldots, \sigma_n)$ with $\sigma_i = (x_i, y_{i1}, y_{i2})$, and $\vec{c} = (c_1, \ldots, c_n)$, the aggregate tag $\sigma = (x, y_1, y_2)$ is computed as:

$$\begin{aligned} x &= \prod_{i=1}^{n} x_i^{c_i} \mod p, \\ y_1 &= \sum_{i=1}^{n} c_i y_{i1} \mod q, \\ y_2 &= \sum_{i=1}^{n} c_i y_{i2} \mod q. \end{aligned}$$

- $\{0,1\} \leftarrow \mathsf{Vrfy}(\mathsf{pk}, M, \sigma)$: To verify that M is valid with respect to tag σ, check whether:

$$x \stackrel{?}{=} v_1^{y_1} v_2^{y_2} \prod_{j=1}^{m} z_j^{M[j]} \mod p.$$

If it holds, return 1; otherwise, return 0.

Given $M = \sum_{i=1}^{n} c_i M_i$ and the aggregated tag σ, the correctness of HLT can be verified as follows:

$$
\begin{aligned}
v_1^{y_1} v_2^{y_2} \prod_{j=1}^{m} z_j^{M[j]} &= v_1^{\sum_{i=1}^{n} c_i y_{i1}} v_2^{\sum_{i=1}^{n} c_i y_{i1}} \prod_{j=1}^{m} z_j^{\sum_{i=1}^{n} c_i M_i[j]} \\
&= \prod_{i=i}^{n} v_1^{c_i y_{i1}} \prod_{i=i}^{n} v_2^{c_i y_{i2}} \prod_{j=1}^{m} z_j^{\sum_{i=1}^{n} c_i M_i[j]} \\
&= \prod_{i=i}^{n} (v_1^{c_i y_{i1}} v_2^{c_i y_{i2}} \prod_{j=1}^{m} z_j^{c_i M_i[j]}) \\
&= \prod_{i=i}^{n} x_i^{c_i} = x
\end{aligned}
$$

Theorem 14. *Assuming DLOG problem is hard, the* HLT *scheme is secure according to Definition 17.*

Proof. Let $M_1, \ldots, M_n$ be the messages adaptively selected by $\mathcal{A}$ and $\sigma_1 = (x_1, y_{11}, y_{12}), \ldots, \sigma_n = (x_n, y_{n1}, y_{n2})$ be the corresponding tags generated by the challenger. Assume the adversary wins the security game with a non-negligible probability. That is, it outputs a vector of coefficients $\vec{c} = \{c_1, \ldots, c_n\}$ and a message $M' \in \mathcal{M}$, such that $M' \neq M = \sum_{i=1}^{n} c_i M_i$ but $1 \leftarrow \mathsf{Vrfy}(\mathsf{pk}, M', \sigma)$, where $\sigma \leftarrow \mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$, and $\vec{\mathsf{Tag}} = (\sigma_1, \ldots, \sigma_n)$. We show that if $\mathcal{A}$ wins the security game with a non-negligible probability, then we can solve the DLOG problem: given v_1, v_2 randomly selected from Z_p^*, find $\log_{v_2}(v_1)$.

Suppose $\sigma = (x, y_1, y_2)$. Since $1 \leftarrow \mathsf{Vrfy}(pk, M', \sigma)$, we have

$$x = v_1^{y_1} v_2^{y_2} \prod_{j=1}^{m} z_j^{M'[j]}.$$

On the other hand, as $\sigma \leftarrow \mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$, we have

$$x = v_1^{y_1} v_2^{y_2} \prod_{j=1}^{m} z_j^{M[j]},$$

where $M = \sum_{i=1}^{n} c_i M_i$. Therefore, we have

$$\prod_{j=1}^{m} z_j^{M'[j]} = \prod_{j=1}^{m} z_j^{M[j]},$$

namely

$$\prod_{j=1}^{m} z_j^{M'[j]-M[j]} = 1.$$

As $M' \neq M$, let $\Delta M[j] = M'[j] - M[j]$ for $1 \leq j \leq m$. Since $z_j = v_1^{-s_{j1}} v_2^{-s_{j2}}$, we have

$$v_1^{\sum_{j=1}^{m} -s_{j1}\Delta M[j]} v_2^{\sum_{j=1}^{m} -s_{j2}\Delta M[j]} = 1.$$

We claim that $\sum_{j=1}^{m} -s_{j1}\Delta M[j] \mod q = 0$ with negligible probability because s_{j1} for $1 \leq j \leq n$ are kept secret. Then we have

$$v_1 = v_2^{\frac{\sum_{j=1}^{m} s_{j2}\Delta M[j]}{\sum_{j=1}^{m} -s_{j1}\Delta M[j]}}.$$

□

HLT **Performance** As stated in Lemma 1, any secure HLA scheme is also a secure HLT scheme. Now we show that HLT constructions can be significantly more efficient than HLA schemes. Specifically, we compare our HLT with two HLA schemes presented in [7, 109]. We use comparable parameters that offer the same level of security. Specifically, the parameter q is 140-bit and p is 512-bit in our HLT scheme, p is 160-bit in [109] and N is 1024-bit in [7]. We consider n messages, namely $M_i = (M_i[1], \ldots, M_i[m])$ for $1 \leq i \leq n$, and compare the costs of the respective operations.

As shown in Table 4.1, the HLA scheme presented in [109] has the shortest tag but incurs the most expensive computation. Recall that exponentiations and multiplications in pairing groups are much less efficient than those in integer groups (e.g., the cost of one pairing is about that of 6-20

Table 4.1: Performance comparison between the HLT and HLA, where Ex denotes exponentiation and Mu denotes multiplication.

	HLT	HLA [109]	HLA [7]
assumption	DLOG	CDH	Factoring
pairing-based?	No	Yes	No
tag size	790 bits	160 bits	1024 bits
tagGen	$2n$ Ex+ mnMu	mnEx+ mnMu	mnEx+ mnMu
verify (single)	mEx	2Pairing + mEx	mEx
verify (aggregate)	mEx+ mn Mu	2Pairing + $(m+n)$Ex+ mn Mu	$(m+n)$Ex+mn Mu
tagAggregate	nEx+ $2n$ Mu	nEx+ n Mu	nEx+ n Mu

exponentiations [15]).

4.5 Verfiable SQL Queries on Outsourced Dynamic Databases

In this section, we begin with a discussion on the high level idea. Then, we present the main construction and analyze its security. Efficiency evaluation and comparison to the state-of-the-art solutions is deferred to Section 4.6.

4.5.1 High Level Idea

The state-of-the-art solutions to the verifiable query problems fall into two approaches. The first approach is *tree-based* [83]. This approach incurs the least computational complexity because of the hash operations, but $O(n \log n)$ communication overhead. The second approach is *signature-based* [99]. This approach incurs high computational complexity of $O(kn)$ exponentiation operations on bilinear map, and communication complexity of $O(n)$ bitmaps (a small constant bits). Both approaches incur $O(mn)$ extra storage complexity in the cloud.

Our solution is based on a third approach. It substantially reduces the extra complexity in the cloud side from $O(mn)$ to $O(n)$. It achieves a balanced trade-off between computational and communication communications. Specifically, it is less efficient than the tree-based solution in terms of computational complexity but substantially more efficient than the tree-based solution in terms of communication complexity. It is also substantially more efficient than the signature-based solution in terms of computational complexity but less efficient than the signature-based solution in terms of communication complexity. Perhaps more importantly, our solution can accommodate aggregate queries, while they do not [83, 99].

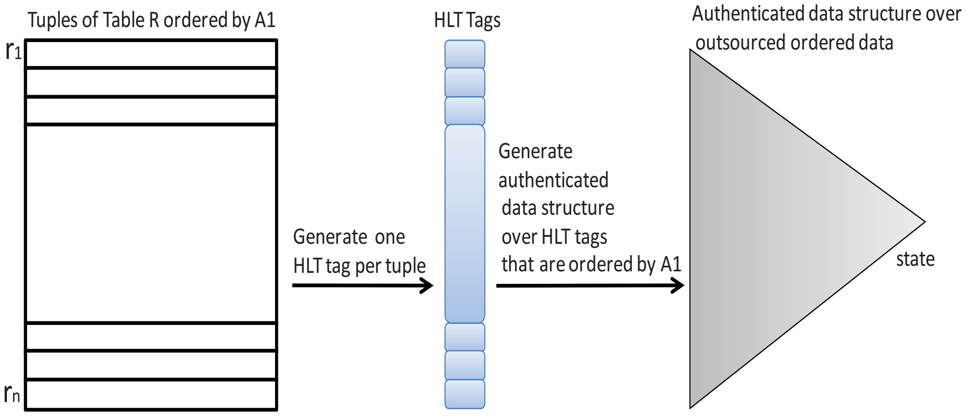

Tuples of Table R ordered by A1
r1
rn
Generate one HLT tag per tuple
HLT Tags
Generate authenticated data structure over HLT tags that are ordered by A1
Authenticated data structure over outsourced ordered data
state

Figure 4.2: High-level idea of our solution that are composed by the two building blocks AuthDS and HLT: Given the table R with tuples ordered by A_1, it generates a HLT tag σ for each tuple, and then constructs the AuthDS over the tuples of $(r.A_1, \sigma)$ that are ordered by $r.A_1$.

The high-level idea of our solution is straight-forward as shown in Figure 4.2: The HLT scheme generates a tag for each tuple in the table, so that we can use the AuthDS scheme to build the authenticated data structured ove tags, which are ordered by the search key A_1. Intuitively, the AuthDS scheme provides two functionalities: one is to verify the correntness of the range query and the other is to guarantee the tag integrity, addressing the problem of preventing HLT tags from being manipulated. By that, the HLT scheme can efficiently verify the integrity of the tuples by aggregating these tuples and tags. The promising performance benefit comes from the HLT scheme, due to the fact that only one aggregate tuple is needed to verify the integrity of (parts of) tuples, which is critical for the projection query as its query result only contains a portion of attributes from all tuples.

4.5.2 Proposed Construction

Let R be a table of n tuples with schema $(A_1, \ldots, A_m)$. Assume A_1 is the search key and $r_1, \ldots, r_n$ are tuples ordered by A_1. Let L and U be the lower and upper bound of the search key A_1. Let Λ_{RS} be an AuthDS scheme and Λ_{HLT} be an HLT scheme, s.t. $\Lambda_{\mathsf{RS}} =$ (KeyGen, SetUp, QueryandVrfy, Update) and $\Lambda_{\mathsf{HLT}} =$ (KeyGen, Tag, Vrfy, HLTAgg). The VQDDB can be constructed as follows:

- KeyGen: Given the primary security parameter ℓ, the data owner obtains two secondary security parameters ℓ_1 and ℓ_2, and generates the private/public keys $(\mathsf{sk}, \mathsf{pk})$,

  1. Compute $(\Lambda_{\mathsf{RS}}.\mathsf{sk}, \Lambda_{\mathsf{RS}}.\mathsf{pk}) \leftarrow \Lambda_{\mathsf{RS}}.\mathsf{KeyGen}(1^{\ell_1})$
  2. Compute $(\Lambda_{\mathsf{HLT}}.\mathsf{sk}, \Lambda_{\mathsf{HLT}}.\mathsf{pk}) \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{KeyGen}(1^{\ell_2})$
  3. Let $\mathsf{sk} = \{\Lambda_{\mathsf{RS}}.\mathsf{sk}, \Lambda_{\mathsf{HLT}}.\mathsf{sk}\}$ and $\mathsf{pk} = \{\Lambda_{\mathsf{RS}}.\mathsf{pk}, \Lambda_{\mathsf{HLT}}.\mathsf{pk}\}$
  4. $\Lambda_{\mathsf{HLT}}.\mathsf{KeyGen}$ specifies the coefficient domain $\mathcal{C}$ and the message space $\mathcal{M}$, s.t. $(r_i.A_1$, ..., $r_i.A_m) \in \mathcal{M}$ for $r_i \in R, 1 \leq i \leq n$.

- SetUp: The data owner takes as inputs the private key sk and the table R, and obtains state and au as follows:

  1. Let r_0 and r_{n+1} be two tuples added at both ends of table R in order to facilitate range query, where $r_0.A_1 = L$ and $r_{n+1}.A_1 = U$.
  2. Compute $\sigma_i \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{TagGen}(\Lambda_{\mathsf{HLT}}.\mathsf{sk}, r_i)$, for tuple $r_i, 0 \leq i \leq n+1$.
  3. Let E_{RS} be the ordered set, s.t. $E_{\mathsf{RS}} = \{(r_0.A_1, \sigma_0), \ldots, (r_n.A_1, \sigma_{n+1})\}$ ordered by A_1, and compute $(\mathsf{state}_{\mathsf{RS}}, \mathsf{au}_{\mathsf{RS}}, E_{\mathsf{RS}}) \leftarrow \Lambda_{\mathsf{RS}}.\mathsf{SetUp}(\Lambda_{\mathsf{RS}}.\mathsf{sk}, E_{\mathsf{RS}})$. Note here values $r_i.A_1, 0 \leq i \leq n+1$ are used to implicitly specify the order of the tags and efficiently locate the target tags within the authenticated data structure.
  4. Let $\mathsf{state} = \mathsf{state}_{\mathsf{RS}}, \mathsf{au} = (\mathsf{au}_{\mathsf{RS}}, E_{\mathsf{RS}})$. R and au will be outsourced to the server. Note that the data owner makes state public and maintains R and au optionally.

- Update: The data owner updates the table R with the update information upd and asks the server to update the stored table accordingly. Suppose upd is "update table R set $A_3 = d$ where $r.A_1 = a$" where a, d are constant values:

1. The data owner runs $\mathsf{QueryandVrfy}$ with selection query "select * from R where $r.A_1 = a$" to get the tuple r (Note that $\mathsf{QueryandVrfy}$ can guarantee that the returned tuple is correct, which is presented shortly).

2. Let r' be the tuple of r by setting $A_3 = d$, and the data owner computes $\sigma' \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{TagGen}(\Lambda_{\mathsf{HLT}}.\mathsf{sk}, r)$.

3. Let $\mathsf{upd}_{\mathsf{RS}}$ be "update $(r.A_1, \sigma)$ with $(r'.A_1, \sigma')$" where σ is the HLT tag of tuple r . The data owner, taking as input $\mathsf{upd}_{\mathsf{RS}}, \Lambda_{\mathsf{RS}}.\mathsf{sk}$ and $\mathsf{au}_{\mathsf{RS}}$, runs the $\Lambda_{\mathsf{RS}}.\mathsf{Update}$ protocol with the server, who takes as input $\mathsf{upd}_{\mathsf{RS}}$ and $\mathsf{au}_{\mathsf{RS}}$. Eventually, the data owner outputs the new $\mathsf{state}'_{\mathsf{RS}}$ and $\mathsf{au}'_{\mathsf{RS}}$, and updates E_{RS} to E'_{RS} where $E'_{\mathsf{RS}} = \{\ldots, (r'.A_1, \sigma'), \ldots, \}$. The server also updates $\mathsf{au}_{\mathsf{RS}}$ to $\mathsf{au}'_{\mathsf{RS}}$ and E_{RS} to E'_{RS} correspondingly.

4. The data owner delivers upd to the server, so that the server updates the table R to R'.

In the following we present the construction of $\mathsf{QueryandVrfy}$ protocol according to different types of query: selection/ projection / join /aggregate queries. Recall that $\mathsf{state} = \mathsf{state}_{\mathsf{RS}}$ and $\mathsf{au} = (\mathsf{au}_{\mathsf{RS}}, E_{\mathsf{RS}})$.

$\mathsf{QueryandVrfy}$ **on Selection Query** As shown in Figure 4.3, the basic idea to assure verifiable selection query is that the querier can use the authenticated data structure to ensure that the HLT tags and its corresponding search keys satisfy the condition clause (aka. range query) and use the HLT scheme to verify the integrity of query result. Especially, assume that the selection query issued by the querier is qry ="select * from R where $A_1 \geq a$ and $A_1 \leq b$". Suppose that the result rslt is not null, saying $\mathsf{rslt} = \{r_s, \ldots, r_t\}, 1 \leq s \leq t \leq n$, s.t. $r_{s-1}.A_1 < a \leq r_s.A_1$ and $r_t.A_1 \leq b < r_{t+1}.A_1$. The protocol executed by the cloud and the querier proceeds as follows:

1. The server: Let $\mathsf{rslt} = \{r_s, \ldots, r_t\}$ and $\mathsf{prf} = \{(r_s.A_1, \sigma_s), \ldots, (r_t.A_1, \sigma_t)\}$, and send rslt, prf to the querier.

2. The querier: Run the protocol $\Lambda_{\mathsf{RS}}.\mathsf{QueryandVrfy}$ with the server for the range query (a, b) to assure that $\{(r_s.A_1, \sigma_s), \ldots, (r_t.A_1, \sigma_t)\}$ is the correct answer. If the output is reject, then

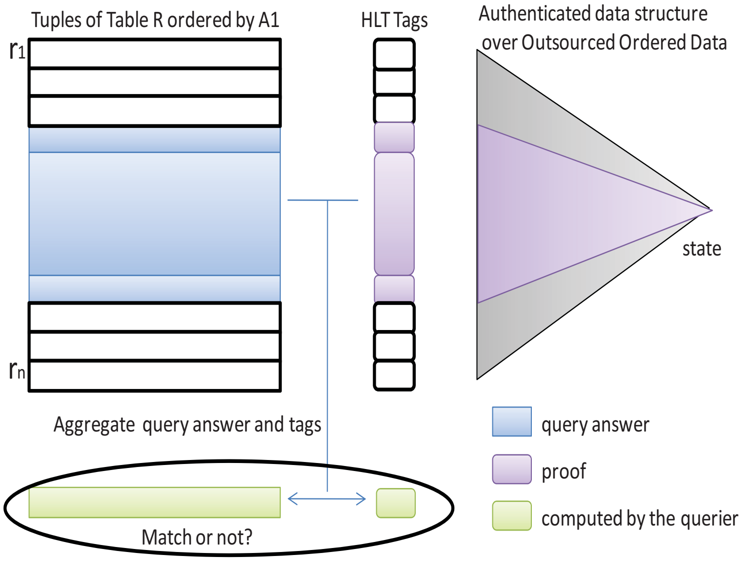

Tuples of Table R ordered by A1
HLT Tags
Authenticated data structure over Outsourced Ordered Data
r1
rn
state
Aggregate query answer and tags
Match or not?
query answer
proof
computed by the querier

Figure 4.3: Idea for assuring verifiable selection query: Given the SQL query qry= "select * from R where $A_1 \geq a$ and $A_1 \leq b$', the AuthDS is used to guarantee that the server honestly returned all tuples $(r.A_1, \sigma)$ where $a \leq r.A_1 \leq b$ (implying the integrity of HLT tag σ), and the HLT is used to verify that the integrity of the query answer.

abort.

3. The querier: Select a vector of coefficients $\vec{c} = (c_s, \ldots, c_t)$ randomly, compute $\sigma \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$ where $\vec{\mathsf{Tag}} = (\sigma_s, \ldots, \sigma_t)$, and verify $\Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.\mathsf{pk}, \sum_{i=s}^{t} c_i r_i, \sigma)$. If the output is 1, then return accept and rslt, otherwise return reject.

If the result rslt is null, saying there exist two consecutive tuples $(r_s.A_1, \sigma_s)$ and $(r_{s+1}, \sigma_{s+1}), 0 \leq s \leq n$, such that $r_s.A_1 < a, b < r_{s+1}.A_1$. The querier runs the protocol $\Lambda_{\mathsf{RS}}.\mathsf{QueryandVrfy}$ on the range query (a, b), and verifies that no tuple is located in the interval $((r_s.A_1, \sigma_s), (r_{s+1}.A_1, \sigma_{s+1}))$.

QueryandVrfy **on Projection Query** The basic idea of assuring verifiable projection query is shown in Figure **??**: The querier use the authenticated data structure to assure the integrity of HLT tags, and use the HLT tags to verify the integrity of queried attributes together with the aggregate unqueried attributes computed by the server. More concretely, assume that the projection query issued by the querier is qry ="select $A_1, \ldots, A_k$ from R" $(k \geq 1)$. The protocol executed by the server and the querier proceeds as follows:

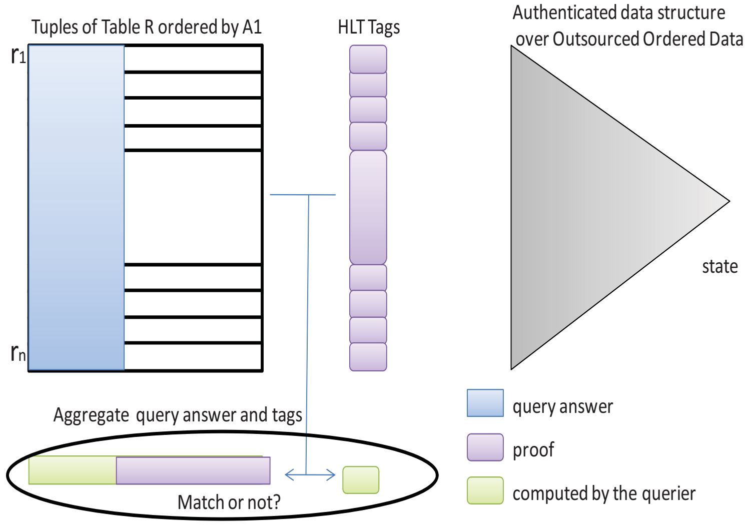

Tuples of Table R ordered by A1
r1
rn
HLT Tags
Authenticated data structure
over Outsourced Ordered Data
state
Aggregate query answer and tags
Match or not?
query answer
proof
computed by the querier

Figure 4.4: Idea for assuring verifiable projection query: The server returns all the queried attributes as the query answer, and aggregates all unquried attributes, which are presented as the proof, as well as all tuples $(r.A_1, \sigma)$. The AuthDS assures the integrity of all tuples $(r.A_1, \sigma)$ and the HLT guarantees the integrity of the query answer.

1. The server: Let $\mathsf{rslt} = \{(r_i.A_1, \ldots, r_i.A_k), 1 \le i \le n\}$ and have $\{(r_1, \sigma_1), \ldots, (r_n, \sigma_n)\}$ be part of prf.

2. The querier: Run the protocol $\Lambda_{\mathsf{RS}}.\mathsf{QueryandVrfy}$ with the server for the range query (L, U) to assure that $\{(r_1, \sigma_1), \ldots, (r_n, \sigma_n)\}$ is the correct answer. If the output is reject, then abort.

3. The querier: Select a vector of coefficients $\vec{c} = (c_1, \ldots, c_n)$ randomly and send it to the server.

4. The server: Compute $r.A_j = \sum_{i=1}^{n} c_i r_i.A_j$ for $j = k+1, \ldots, m$, and send it to the querier as part of prf.

5. The querier: Compute $r.A_j = \sum_{i=1}^{n} c_i r_i.A_j$, for $j = 1, \ldots, k$ from $\mathsf{rslt} = \{(r_i.A_1,$ …, $r_i.A_k), 1 \le i \le n\}$, and the aggregate tag $\sigma = \Lambda_{\mathsf{HLT}}.\mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$, where $\vec{\mathsf{Tag}} = \{\sigma_1, \ldots, \sigma_n\}$.

6. The querier: Compute $\Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.\mathsf{pk}, M, \sigma)$ where $M = (r.A_1, \ldots, r.A_m)$. If the output is 1, then return accept and rslt, otherwise return reject.

$\mathsf{QueryandVrfy}$ **on Join Query** The idea of achieving verifiable join query is quite straightforward: Given the join query, The verifiabile projection query can be used to assure that the query answer contains all tuples satisfying the join condition. To be specific, we run the verifiable projection query to obtain attributes related to the join condition, locally identify corresponding tuples satisfying the join condition, and then can use the HLT scheme to assure that tuples of the query answer are intact. Suppose P is another table with schema $(B_1, \ldots, B_m)$, which is processed by algorithm SetUp before being outsourced to the server.. For convenience, assume P has n tuples, B_1 is the search key of table P and A_2, B_2 are respective primary key for the table B, P, uniquely identifying the tuples respectively. Assume the join query issued by the querier is is qry =“select $R.^*, U.^*$ from R, P where $R.A_s = P.B_t$” ($1 \leq s, t \leq m$). The protocol executed by the cloud and the querier proceeds as follows:

1. The server: Return the tuples in R and P satisfying $R.A_s = P.B_t$, denoted as R^* and P^*. Let $\mathsf{rslt} = (R^*, P^*)$.

2. The querier: Execute the projection queries, qry =“select A_2, A_s from R” and qry' =“select B_2, B_t from P” respectively. After succeeding in executing the projection queries, it obtains the tuples $\{(r_i.A_2, r_i.A_s, \sigma_i), 0 \leq i \leq n+1\}$ and $\{(p_j.B_2, p_j.B_t, \sigma'_j), 0 \leq j \leq n+1\}$.

3. The querier: Identify tuples s.t. $R.A_s = P.B_t$ from $\{(r_i.A_2, r_i.A_s, \sigma_i), 1 \leq i \leq n\}$ and $\{(p_j.B_2, p_j.B_t, \sigma'_j), 1 \leq j \leq n\}$. Suppose $\alpha \subseteq \{1, \ldots, n\}$, and $\beta \subseteq \{1, \ldots, n\}$ be two set of indexes, such that $a \in \alpha, b \in \beta, r_a.A_s = p_b.B_t$. It obtains two sets of tuples, $\{(r_i.A_2, r_i.A_s, \sigma_i) | i \in \alpha\}$ and $\{(p_j.B_2, p_j.B_t, \sigma'_j), | j \in \beta\}$. Make sure that R^* has the same size with $\{(r_i.A_2, r_i.A_s, \sigma_i), i \in \alpha\}$, and P^* has the same size with $\{(r_j.B_2, p_j.B_t, \sigma'_j), j \in \beta\}$. Otherwise abort.

4. The querier: Select a vector of coefficients $\vec{c} = (c_1, \ldots, c_{|\alpha|})$ randomly, obtain σ by aggregating the tags $\{\sigma_i, i \in \alpha\}$, and compute $\Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}$ with $\vec{c}$, σ, R^* and $\Lambda_{\mathsf{HLT}}.\mathsf{pk}$. Similar action was taken towards P^*. If both output 1, then return accept, as well as the query result, otherwise return reject.

$\mathsf{QueryandVrfy}$ **on Aggregate Query** The idea of assuring verifiable aggregate query is analogue to that of selection query, where the authenticator data structure is used to assure the integrity of HLT tags and their corresponding search keys, and the HLT tags can be further used to assure the correctness of the aggregate value. More specifically, assume the aggregate query issued by the querier is qry ="select SUM(A_2) from R where $A_1 \geq a$ and $A_1 \leq b$". Suppose $1 \leq s \leq t \leq n$, $r_{s-1}.A_1 < a \leq r_s.A_1$ and $r_t.A_1 \leq b < r_{t+1}.A_1$. The protocol executed by the cloud and the querier proceeds as follows if the query result is not null (meaning there exist some tuples whose A_1 falls in the range of a and b):

1. The server: Compute $r.A_j = \sum_{i=s}^{t} r_i.A_j$ for $j = 1, \ldots, m$. Let rslt $= \{r.A_2\}$ and $\{(r_s.A_1, \sigma_s), (r_t.A_1, \sigma_t), r.A_1, r.A_3, \cdots, r.A_m\}$ as part of proof prf.

2. The querier: Make sure $r_{s-1}.A_1 < a \leq r_s.A_1$ and $r_t.A_1 \leq b < r_{t+1}.A_1$, otherwise abort. Run the protocol $\Lambda_{\mathsf{RS}}.\mathsf{QueryandVrfy}$ on the range query (σ_s, σ_t). If the output is reject, then abort, otherwise, it obtains the range query result $\mathsf{rslt}_{\mathsf{RS}} = (\sigma_s, \ldots, \sigma_t)$ and $\mathsf{prf}_{\mathsf{RS}}$ which should contain $\sigma_{s-1}, \sigma_{t+1}$.

3. The querier: Run the protocol $\Lambda_{\mathsf{RS}}.\mathsf{QueryandVrfy}$ with the server for the range query (a, b) to assure that $\{(r_s.A_1, \sigma_s), \ldots, (r_t.A_1, \sigma_t)\}$ is the correct answer. If the output is reject, then abort.

4. The querier: Compute $\sigma \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$, where $\vec{c}$ is a vector of 1s and $\vec{\mathsf{Tag}} = (\sigma_s, \ldots, \sigma_t)$. Then it compute $\Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.\mathsf{pk}, M, \sigma)$ where $M = \{r.A_1, \cdots, r.A_m\}$. If the output is 1, then return accept and rslt, otherwise return reject.

In the case that the queried result is null, then the querier and the server proceeds the same as the case in selection query.

Remark 1: It is worth noting that for selection/projection/join query, we use randomly selected the coefficient vector $\vec{c}$ to prevent *aggregate attack*. To see this, let us consider the case without using $\vec{c}$, namely $\vec{c}$ is composed of 1s. The server has $r'_i = r_i, s - 1 \leq i \leq t + 1$, and manipulates two tuples $r_e, r_{e+1}, s \leq e \leq t-1$, to obtain $r'_e = (r_e.A_1, r_e.A_2 + 1, r_e.A_3, \ldots)$ and $r'_{e+1} = (r_e.A_1, r_e.A_2 - 1, r_e.A_3, \ldots)$, which makes $\sum_{i=s}^{t} r_i = \sum_{i=s}^{t} r'_i$. With that, the server could make $\Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}$ output 1 with manipulated $\{r'_s, \ldots, r'_t\}$.

Remark 2: Our solution toward the aggregate query only supports the SUM queries and the weighted SUM queries. On the other hand, our solution supports more *flexible* join queries, in the sense that they do not have to be defined with respect to pre-defined keyword attributes

4.5.3 Security Analysis

Theorem 15. *Assume Λ_{RS} is a secure authenticated data structure scheme for range query and Λ_{HLT} is a secure homomorphic linear tag scheme. The proposed scheme for verifiable SQL queries on outsourced dynamic database achieves the soundness regarding the selection/ projection/ join/ aggregate query.*

The basic idea to prove the soundness is if, for any probabilistic polynomial time adversary, it can break the soundness of the proposed solution, we can break the security of either Λ_{RS} or Λ_{HLT}.

Proof. The challenger executes $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\ell)$, keeping sk privately and giving pk to the adversary. Let $\mathsf{state}, \mathsf{au}$ and R be the latest version of state information, auxiliary information and the table, after one execution of SetUp and polynomially many executions of Update between the adversary and the challenger, where $\mathsf{state} = \mathsf{state}_{\mathsf{RS}}, \mathsf{au} = (\mathsf{au}_{\mathsf{RS}}, E_{\mathsf{RS}})$ and $E_{\mathsf{RS}} = \{(r_0.A_1, \sigma_0), \ldots, (r_{n+1}.A_1, \sigma_{n+1})\}$. We prove that the proposed solution is sound with respect to each query separately.

Soundness of Selection Query Assume the adversary chooses a selection query qry and gives out the query result rslt and the proof prf which wins the security game. Namely, given qry =“select * from R where $A_1 \geq a$ and $A_1 \leq b$”, $\mathsf{rslt} = \{r'_s, \ldots, r'_t\}$ and $\mathsf{prf} = \{\mathsf{rslt}_{\mathsf{RS}}, \mathsf{prf}_{\mathsf{RS}}\}$, where $\mathsf{rslt}_{\mathsf{RS}} = \{(r'_s.A_1, \sigma'_s), \ldots, (r'_t.A_1, \sigma'_t)\}$ and the tuple $(\mathsf{rslt}_{\mathsf{RS}}, \mathsf{prf}_{\mathsf{RS}})$ is the result and proof for the range query (a, b) by running the $\Lambda_{\mathsf{RS}}.\mathsf{QueryandVrfy}$. Suppose the challenger randomly selects the coefficient vector $\vec{c} = (c_s, \ldots, c_t)$, and computes $\sigma' \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$, where $\vec{\mathsf{Tag}}' = (\sigma'_s, \ldots, \sigma'_t)$

As the adversary wins the game, the following should hold:

$$\begin{aligned}(\mathsf{accept}, \mathsf{rslt}_{\mathsf{RS}}, \mathsf{prf}_{\mathsf{RS}}) \leftarrow (Q(\Lambda_{\mathsf{RS}}.\mathsf{pk}, \mathsf{qry}(a, b), \mathsf{state}_{\mathsf{RS}}) \leftrightarrow S(\mathsf{au}_{\mathsf{RS}}, E_{\mathsf{RS}}))\\ 1 \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.\mathsf{pk}, \sum_{i=s}^{t} c_i r'_i, \sigma')\\ \mathsf{rslt} \neq \mathsf{localRst}\end{aligned}$$

where $\mathsf{localRst} \leftarrow \mathsf{LocalQuery}(\mathsf{qry}, R)$ computed by the challenger.

First, we claim that $\mathsf{rslt}_{\mathsf{RS}} = \{(r'_s.A_1, \sigma'_s), \ldots, (r'_t.A_1, \sigma'_t)\}$ is the correct query result with respect to the range query (a, b). Otherwise, we can break the soundness of ADS for range query. That is, $(\sigma'_s, \ldots, \sigma'_t)$ are exact the HLT tags with respect to the tuples whose search keys (A_1 values) fall into the range (a, b).

Suppose $\mathsf{localRst} = \{r_s, \ldots, r_t\}$. We now claim that $r_i = r'_i$ for $i = s, \ldots, t$. The challenger executes $\sigma \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}})$ where $\vec{\mathsf{Tag}} = (\sigma'_s, \ldots, \sigma'_t)$. Therefore, we have $\sigma = \sigma'$.

Because $(\sigma_s, \ldots, \sigma_t)$ correspond to the tuples $\{r_s, \ldots, r_t\}$, we have

$$1 \quad \leftarrow \quad \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.\mathsf{pk}, \sum_{i=u}^{w} c_i r_i, \sigma) \tag{4.1}$$

So if the adversary wins the security game, then

$$1 \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.\mathsf{pk}, \sum_{i=s}^{t} c_i r'_i, \sigma') \tag{4.2}$$

Therefore, with Eq. 4.1 and Eq. 4.2, if $r'_i \neq r_i$ for $i = s, \ldots, t$, then $\sum_{i=s}^{t} c_i r_i \neq \sum_{i=s}^{t} c_i r'_i$, which break the security of HLT. That is, $\mathsf{localRst} = \mathsf{rslt}$, so that it contracts the assumption that the adversary wins the game.

To sum up, we can see that the soundness of selection query is assured given the assumption that Λ_{RS} and Λ_{HLT} are secure as their definitions.

Soundness of Projection Query Assume the adversary chooses a projection queryqry and

$A_1 \leq b$" and gives out the query result rslt and the proof prf, which wins the security game. Namely, given qry ="select $A_1, \ldots, A_k$ from R" ($k \geq 1$), the adversary sends the query result $\mathsf{rslt} = \{(r'_i.A_1, \ldots, r'_i.A_k), 1 \leq i \leq n\}$ to the challenger. It also sends to the challenger $\mathsf{rslt}_{\mathsf{RS}} = \{((r'_1.A_1, \sigma'_1), \ldots, (r'_n.A_1, \sigma'_n)\}$, which is part of the proof prf.

After receiving rslt, the challenger randomly selects $\vec{c} = (c_1, \ldots, c_n)$ and sends it to the adversary. The challenger computes $r'.A_j = \sum_{i=1}^{n} c_i r'_i.A_j, j = 1, \ldots, k$ and aggregates the tags by $\sigma' \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}}')$, where $\vec{\mathsf{Tag}}' = (\sigma'_1, \ldots, \sigma'_n)$.

When receiving $\vec{c}$, the adversary computes $r'.A_j = \sum_{i=1}^{n} c_i r'_i.A_j$ for $j = k+1, \ldots, m$, and lets $r'.A_{k+1}, \ldots, r'.A_m$ be part of proof prf. If the adversary wins the game, the following should hold:

$$\begin{aligned}(\mathsf{accept}, \mathsf{rslt}_{\mathsf{RS}}, \emptyset) \leftarrow (Q(\Lambda_{\mathsf{RS}}.\mathsf{pk}, \mathsf{qry}(L,U), \mathsf{state}_{\mathsf{RS}}) \leftrightarrow S(\mathsf{au}_{\mathsf{RS}}, E_{\mathsf{RS}}))\\ 1 \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.\mathsf{pk}, (r'.A_1, \ldots, r'.A_m), \sigma')\\ \mathsf{rslt} \neq \mathsf{localRst}\end{aligned}$$

where $\mathsf{localRst} \leftarrow \mathsf{LocalQuery}(\mathsf{qry}, R)$ computed by the challenger locally.

First, we claim that $\mathsf{rslt}_{\mathsf{RS}} = \{(r'_1.A_1, \sigma'_1), \ldots, (r'_n.A_1, \sigma'_n)\}$ is the correct query result with respect to the range query $\mathsf{qry}(L,U)$. Otherwise, we can break the soundness of ADS for range query. That is, $(\sigma'_s, \ldots, \sigma'_t)$ are exact the HLT tags with respect to the tuples whose search keys (A_1 values) fall into the range (a, b).

Then we claim that $\mathsf{rslt} = \mathsf{localRst}$ where $\mathsf{localRst} = \{(r_1.A_1, \ldots, r_1.A_k)$, ..., $(r_n.A_1, \ldots$, $r_n.A_1)\}$. To prove that, we have

$$1 \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.\mathsf{pk}, (\sum_{i=1}^{n} r_i.A_1, \ldots, \sum_{i=1}^{n} r_i.A_k, \ldots, r'.A_m), \sigma') \tag{4.3}$$

and if the adversary wins the games, then

$$1 \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.\mathsf{pk}, (r'.A_1, \ldots, r'.A_k, \ldots, r'.A_m), \sigma') \tag{4.4}$$

If $\mathsf{rslt} \neq \mathsf{localRst}$, then there exist some $j, 1 \leq j \leq k$, s.t. $\sum_{i=1}^{n} c_i r_i.A_j \neq r'.A_j$. That is , if $\mathsf{rslt} \neq \mathsf{localRst}$, we can break the security of HLT scheme with Eq. 4.3 and Eq. 4.4. Therefore, $\mathsf{rslt} = \mathsf{localRst}$, contradicting the assumption that the adversary wins the game.

To sum up, we can see that the soundness of projection query is assured given the assumption that Λ_{RS} and Λ_{HLT} are secure as their definitions.

Soundness of Join Query It is readily to prove soundness for join query, because the soundness for projection query holds and HLT is secure.

Soundness of Aggregate Query Assume the adversary chooses an aggregate query qry and gives out the query result rslt and the proof prf in order to win the security game. Namely, given qry="select SUM(A_2) from R where $A_1 \geq a$ and $A_1 \leq b$", the adversary computes $r'.A_j = \sum_{i=s}^{t} r_i.A_j$ for $j = 1, \ldots, m$, and return the query result $\mathsf{rslt} = r'.A_2$, and $\mathsf{prf} = \{r'.A_1, r'.A_3, \ldots, r'.A_m$, $\mathsf{rslt}_{\mathsf{RS}}$, $\mathsf{prf}_{\mathsf{RS}}\}$, where $\mathsf{rslt}_{\mathsf{RS}} = \{(r'_s.A_1, \sigma'_s), \ldots, (r'_t.A_1, \sigma'_t)\}$ and $(\mathsf{rslt}_{\mathsf{RS}}\mathsf{prf}_{\mathsf{RS}})$ is the result and proof for the range query (a, b) by running the $\Lambda_{\mathsf{RS}}.\mathsf{QueryandVrfy}$.

Assume the challenger computes $\sigma' \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{HLTAgg}(\vec{c}, \vec{\mathsf{Tag}}')$, where $\vec{\mathsf{Tag}}' = (\sigma'_s, \ldots, \sigma'_t)$ and $\vec{c}$ is 1s. If the adversary wins the game,the following should hold:

$$\begin{aligned} (\mathsf{accept}, \mathsf{rslt}_{\mathsf{RS}}, \mathsf{prf}_{\mathsf{RS}}) \leftarrow (Q(\Lambda_{\mathsf{RS}}.\mathsf{pk}, \mathsf{qry}(a, b), \mathsf{state}_{\mathsf{RS}}) \leftrightarrow S(\mathsf{au}_{\mathsf{RS}}, E_{\mathsf{RS}})) \\ 1 \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.\mathsf{pk}, (r'.A_1, \ldots, r'.A_m), \sigma') \\ \mathsf{rslt} \neq \mathsf{localRst} \end{aligned}$$

where $\mathsf{localRst} \leftarrow \mathsf{LocalQuery}(\mathsf{qry}, R)$ computed by the challenger.

First, we claim that $\mathsf{rslt}_{\mathsf{RS}} = \{(r'_s.A_1, \sigma'_s), \ldots, (r'_t.A_1, \sigma'_t)\}$ is the correct query result with respect to the range query (a, b). Otherwise, we can break the soundness of ADS for range query. That is, $(\sigma'_s, \ldots, \sigma'_t)$ are exact the HLT tags with respect to the tuples whose search keys (A_1 values) fall into the range (a, b).

Then we prove that $\mathsf{rslt} = \mathsf{localRst}$. Namely, $r'.A_2 = r.A_2$.

Since $r.A_1, \ldots, r.A_m$ is the aggregate message for $r_s, \ldots, r_t$ and σ is the corresponding aggregate tag, so we have

$$1 \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.\mathsf{pk}, (r'.A_1, r.A_2, \ldots, r'.A_m), \sigma)$$

If the adversary wins the security game, it should

$$1 \leftarrow \Lambda_{\mathsf{HLT}}.\mathsf{Vrfy}(\Lambda_{\mathsf{HLT}}.\mathsf{pk}, (r'.A_1, r'.A_2, \ldots, r'.A_m), \sigma')$$

If $r'.A_2 \neq r.A_2$, then we can break the security of HLT. Therefore, $r'.A_2 = r.A_2$, which contradicts the assumption that the adversary wins the game.

To sum up, we can see that the soundness of aggregate query is assured given the assumption that Λ_{RS} and Λ_{HLT} are secure as their definitions.

□

4.6 Performance Evaluation

4.6.1 Asymptotic Performance Analysis

We compare the asymptotic performance of our solution and that of the two state-of-the-art solutions [83,99], which are the only one supporting selection / projection / join queries in the dynamic database simultaneously.

As shown in Table 4.2, our solution is more expressive because it additionally supports aggregate queries, such as: “select SUM(A_2) from R where $A_1 > a$." Moreover, our solution allows join query with respect to arbitrary attributes, such as: “select $R.^*$, $P.^*$ from R, P where $R.A_3 = P.B_4$" *without* requiring $R.A_3$ and $P.B_4$ to be search keys. Whereas, this type of join queries cannot be handled by the state-of-the-art solutions [83,99].

For pre-processing the database before outsourcing it to the cloud, our solution is more efficient than Pang et al. [99], and as efficient as Li et al. [83]. In addition, our solution incurs the least extra storage complexity. To see this, we compare the three solutions with parameters in Table 4.1. Our

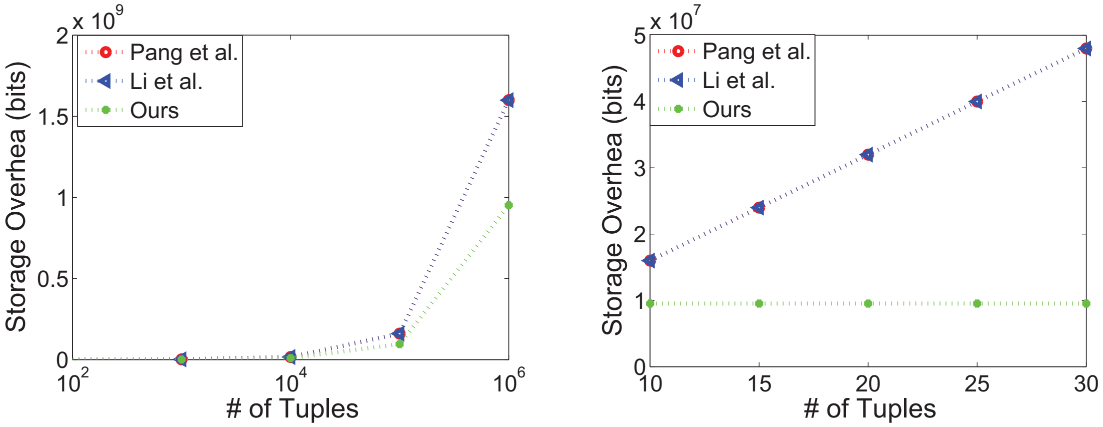
Table 4.2: Asymptotic performance comparison, where Hash is 160 bits, Sig is 1024 bits, AggSig= 160 bits, Tag= 792 bits, Bitmap is a small constant, Ex denotes modular exponentiation, Mu denotes modular multiplication Pairing denotes pairing operation, m is the number of attributes of the table, k is the number of attributes in projection query, attribute is an attribute value in R, λ is the number of attributes to be aggregated, R^* denotes unmatched tuples in R, and assume $|R| = |P| = n$ in join query. Note that our solution supports aggregate queries and more flexible join queries, and we do not count the basic search operation in comparison.

		Li et al. [83]	Pang et al. [99]	Our solution
Functions		Selection, Projection, Join	Selection, Projection, Join	Selection, Projection, Join, Aggregate
Technique Technique		Merkle-based Hash Tree	Aggregate Signature with Chaining	Merkle-based Hash Tree and HLT
Security		Sound	Sound	Sound
Data PreProcessing		$O(n)$Hash	$O(mn)$Ex	$O(n)$Hash+ $O(n)$Ex
Storage Overhead		$O(mn)$Hash	$O(mn)$AggSig	$O(n)$Hash + $O(n)$Tag
Selection	Computation$_S$	N/A	$O(n)$Mu	N/A
	Communication	$O(\log n)$Hash	$O(n)$Bitmap	$O(\log n)$Hash $+ O(n)$Tag
	Computation$_V$	$O(n)$Hash	$O(n)$Ex	$O(\log n)$Hash $+ O(n)$Ex
Projection	Computation$_S$	N/A	$O(kn)$Mu	$O(n)$Mu
	Communication	$O((m-k)n)$attribute	$O(n)$Bitmap	$O(n+m)$Tag
	Computation$_V$	$O(n)$Hash	$O(kn)$Ex	$O(n)$Ex
Join	Computation$_S$	N/A	$O(n)$Mu	$O(n)$Mu
	Communication	$O(n \log(n))$Hash $+ R^*$	$O(n)$Bitmap $+ R^*$	$O(n+m)$Tag
	Computation$_V$	$O(n \log(n))$Hash	$O(n)$Ex	$O(n)$Ex
Aggregate	Computation$_S$	N/A	N/A	N/A
	Communication	N/A	N/A	$O(\log n)$Hash $+ O(\lambda)$Tag
	N/A	N/A	N/A	$O(\log n)$Hash $+ O(\lambda)$Ex
Update	Computation$_S$	$O(\log n)$Hash	N/A	$O(\log n)$Hash
	Communication	O(1)	O(1)	O(1)
	Computation$_O$	$O(\log n)$Hash	$O(m)$Ex	$O(\log n)$Hash

solution is storage-space more efficient than [83, 99] as long as the number of attributes is greater than Tag/Hash, which is often the case. Moreover, as shown in Figure 4.5, the storage-space requirement in our solution is independent of the number of attributes; in contrast, the storage-space complexity of [83, 99] increases linearly with respect to the number of attributes.

For selection queries, projection queries and join queries, our solution incurs $O(\log n)$Hash plus $O(n)$Ex, $O(n)$Ex and $O(n)$Ex at the querier side in order to aggregate HLT tags. However, our solution still outperforms [99], which requires $O(n)$Ex, $O(kn)$Ex and $O(n)$Ex on bilinear group respectively.

For projection queries and join queries, our solutions incurs $O(n + m)$ tags. In contrast, [83] requires $O((m - k)n)$ attribute values for projection queries, and $O(n \log n)$ hash values plus



(a) $m = 10$ (i.e. each tuple has 10 attributes)

(b) $n = 10^4$ (i.e. the table has 10^4 tuples)

Figure 4.5: Comparison of storage overhead between our VQDDB scheme and the state-of-the-art solutions [83, 99].

those unmatched tuples in R for join queries. Although [99] only requires $O(n)$ certified bitmap (recording updated tuples in on update period) for projection queries, it requires at least $O(n)$ certified bitmap plus those unmatched tuples in R for join queries. It is due to the fact that [83, 99] have to fetch at least one table (either R or P) for join queries.

For aggregate queries, regardless of the number of attributes the querier wants to aggregate, the computational and communication complexities are the same. For example, "select SUM$(A_1), \ldots$, SUM(A_k) from R where $A_1 > a$ and $A_1 < b$" and "select SUM(A_1) from R where $A_1 > a$ and $A_1 < b$", our solution incurs the same complexity.

4.6.2 Implementation

We have implement the cryptographic components (i.e. HLT scheme and AuthDS scheme) in Java. In our implementation, we instantiated the algebraic group of the HLT with the default Schnorr group provided by the DSA signature where p is 512-bit and q is 160-bit. We also instantiated the Merkle B tree of the AuthDS with the SHA-1 hash function. We measure the performance of the algorithms/protocols in the VQDDB on a machine with Linux OS, 2.93GHz Intel Core Duo CPU (E7500), and 2GB RAM. Specifically, we evaluated the performance mainly with two kinds of synthetic datasets stored in the MySQL databases:

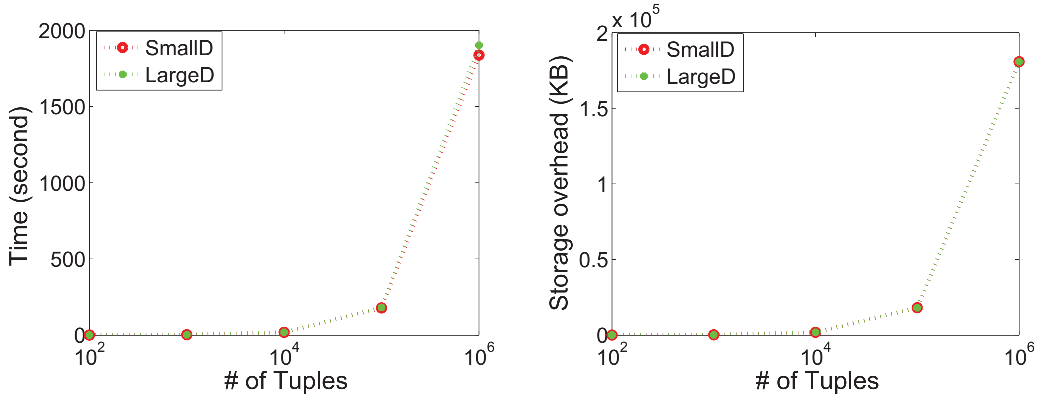
- SmallD: The tuples in this dataset have 10 attributes, and the number of tuples increases from 10^3 to 10^6 with scale 10.
- LargeD: The tuples in this dataset have 30 attributes, and the number of tuples increases from 10^3 to 10^6 with scale 10.

Each tuple has a search key (i.e., the attribute for the range query) generated uniformly at random from the domain of $[1, 10^9]$ and a primary key (i.e. the attribute uniquely identifying the tuple), and each attribute occupies 18 bytes.

4.6.3 Performance Evaluation

Setup Performance Figure 4.6 shows the execution time and storage overhead when running the SetUp algorithm, which should be executed by the data owner. Figure 4.6a illustrates the execution time of SetUp algorithm, measuring the cost of reading tuples from the databases, generating HLT tags, building Merkle B tree and storing them in the disk. We can see that although the tuples in LargeD has 20 more attributes than that of SmallD, the execution cost underlying the same number of tuples is quite close. The reason is that when generating HLT tag, exponentiation operation plays a dominant role comparing with hashing and multiplication operations (our experiment show that the cost of one exponentiation is approximate to the cost of 100 multiplications). Figure 4.6b shows that the VQDDB scheme incurs the same storage overhead regardless the number of attributes in each tuples. This confirms that the VQDDB scheme costs storage overhead independent of the number of attributes of the tuple.

Update Performance To understand the performance of Update protocol between the data owner and the cloud server, we conducted the expriments for the INSERT, UPDATE and DELETE queries, each of which is to update one tuple (i.e., insert/update/delete one tuple). We measured the average number of (update) queires that can be completed by the data owner within one second. Figure 4.7a shows the results. We can see that when the number of the tuples stored in the



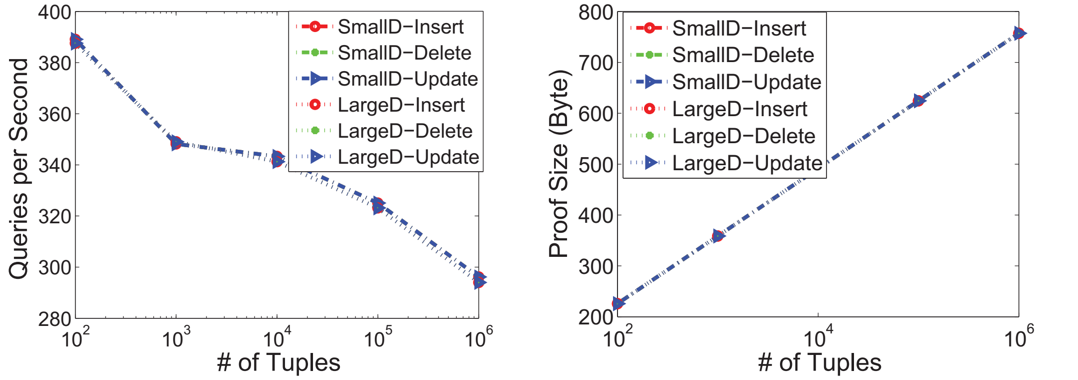
(a) Execution time of SetUp algorithm

(b) Storage overhead

Figure 4.6: The performance of SetUp algorithm run by the data onwer.

database increases, the number of queries that can be completed per second will decrease. The reason is that the data owner has to spend more time to validate the proofs. Figure 4.7b shows the communication overhead between the data owner and the cloud server. Because we runs the data owner and the cloud server on the same machine and cannot simulate additional network latency, we serialized the proof (provided by the cloud server) to the disk so that we can measure the communication overhead exactly. We can see that the communication overhead is propositional to the number of tuples stored in the databases as it is dominated by the number of hash values in the Merkle B tree. We also can observe that given the same number of tuples stored in the databases, the communication overhead is the same regardless the dataset SmallD and LargeD. That is, the communication overhead is independent of the number of attributes in the tuples.

Query-Verification Performance We evaluated the performance of the QueryandVrfy protocol for the respect selection/ projection/ join/ aggregate queries. For the selection and aggregate queries, we set 20 percent of the tuples in the table satisfying the condition clauses (i.e., 20% search keys are in the range $[a, b]$). The projection query is to select 5 attributes of all tuples (i.e., select $A_1, \ldots, A_5$ from R). For join queries, the involved two tables have the same number of tuples and we set 20 percent of the entire tuples matched each other. Similar to the evaluation of Update protocol, we evaluated the size of proofs that are transferred between the querier and the server without

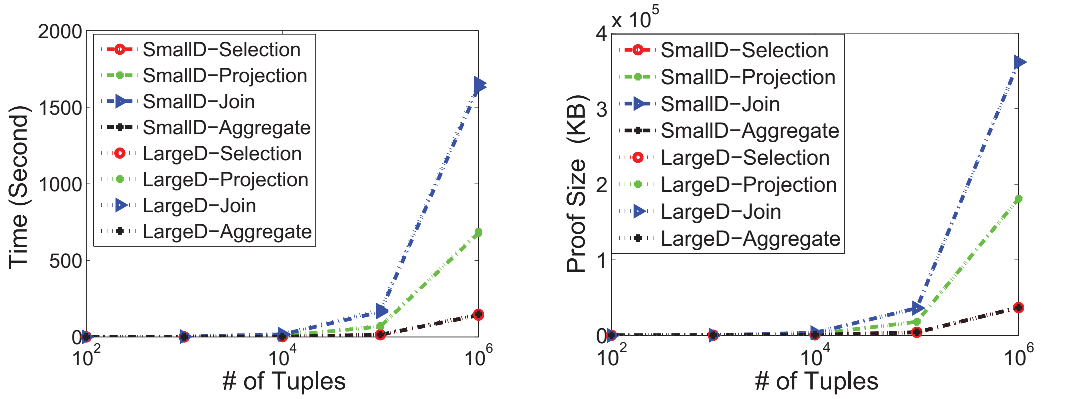


(a) Number of (update) queries that can be completed by the data owner within one second.

(b) Communication overhead for Update protocol

Figure 4.7: The performance of Update protocol between the data owner and the cloud server.

simulating the network latency.



(a) Performance of QueryandVrfy protocol executed by the querier.

(b) Communication overhead for QueryandVrfy protocol

Figure 4.8: The performance of QueryandVrfy protocol between the querier and the cloud server.

Fig 4.8a shows the execution time spent by the querier to validate proofs for respect queries. We can see that the validation costs for the data sets SmallD and LargeD are almost the same with respect to the four kinds of queries, which means that the execution time of validating proofs is almost independent of the number of attributes in the tuples. We also can observe that the verification costs for selection and aggregate queries are far less when comparing with join and projection queries. The reason is that the verification for selection and aggregate queries only need to aggregate 20 percent of HLT tags while for join and projection queries they involve the entire HLT tags. Fig 4.8b shows the communication overhead. Note that the communication overhead for validating selection/ projection/ aggregate queries is linear to the number of tuples satisfying the conditional clause, whereas the communication overhead for the join query is related to the number of tuples, regardless of the size of result for the join query.

4.7 Chapter Summary

We presented an efficient VQDDB solution to the problem of query integrity in the setting of outsourced dynamic relational database. The VQDDB allows a querier, the data owner or a third party, to verify that its queries were faithfully executed by the cloud server. Compared with the state-of-the-art solutions, our solution is not only more powerful by additionally supporting aggregate queries (in addition to selection, projection, and join queries), but also more efficient by eliminating a logarithmic or even linear multiplication factor (depending on the type of the queries) from the overall cost.

Our solution still incurs linear complexity since we always requires the tags, whose tuples satisfy the condition clause in the SQL query, being returned to the queriers. An outstanding direction for future research is to address the following open problem: Can we attain verifiable SQL queries with logarithmic (or constant) complexity?

Chapter 5: CONCLUSION

5.1 Summary

In this dissertation, we present the studies on “verifiable delegated computation”, which enable cloud users to verify the correctness of the delegated computation results on outsourced data.

We first introduce the concept of “verifiable attribute-based keyword search on outsourced encrypted data”, which allows data users to conduct secure keyword search on outsourced encrypted data while complying with flexible access control policies, and assures that the cloud faithfully followed the search procedures. We propose the concrete solution, which enables data owners to grant keyword search capability with respect to access control policies and data users to delegate keyword search to the cloud, and further allows data users to verify that the cloud faithfully executed the search operations.

Secondly, we initiate the problem of “verifiable set intersection on outsouced encrypted data”, which allows cloud users to delegate the set intersection operation to the cloud on their outsourced encrypted data sets, and further verify the correctness of the intersection set returned from the cloud. We present a solution, which is based on two ideas: (i) using proxy re-encryption to enable the cloud to compare equality of plaintexts corresponding to two ciphertexts that are encrypted using different public keys; (ii) using a novel variant of cryptographic accumulator, which can be used to verify the membership of multiple elements through a single examination and may be of independent value, to allow the cloud to show the correctness of the resulting intersection set.

Finally, we study the problem of “verifiable SQL queries on outsourced dynamic database”, which allows database queriers verify the correctness of query result returned from the cloud on outsourced dynamic databases? We propose an efficient solution to support multiple kinds of SQL queries that include selection, projection, join, and (weighted) aggregation queries. The solution is built on top of two building blocks: an authenticated data structure to support dynamic update on outsourced databases, and newly devised homomorphic linear tag, which can efficiently verify the integrity of query results via aggregation.

5.2 Future Work

Support Data Dynamic. In the problem of "verifiable attribute-based keyword search on outsourced encrypted data", the proposed solution only supports the static data set, i.e. without supporting adding new keyword and deleting keywords. The reason is that one of the building-blocks, bloom filter, cannot support keyword deletion. One of the future work is to construct solutions that allow data owners to dynamically maintain outsourced keywords and data files.

Support Fine-grained Access Control. In the problem of "verifiable delegated set intersection on outsourced encrypted data", the proposed solution only offers coarse-grained access control in the following sense. Suppose Alice and Bob allow the cloud to conduct the delegated set intersection operation on C_a and C_b, and Alice and Carlos allow the cloud to conduct the delegated set intersection operation on C_a and C_c. Then, the cloud is able to conduct the set intersection operation on C_b and C_c without the authorization from Bob and Carlos. One of the future work is to enforce fine-grained access control while achieving the goals.

More Efficient Solution for Verifiable SQL Queries. In the problem of "verifiable SQL queries on outsourced dynamic databases", the proposed solution still incurs linear complexity since we always requires the tags, whose tuples satisfy the condition clause in the SQL query, being returned to the queriers. One of the future work is to seek more efficient approach to attain verifiable SQL queries with logarithmic (or constant) complexity?

BIBLIOGRAPHY

[1] Data protection directive in eu. http://ec.europa.eu/justice/policies/privacy/docs/95-46-ce/dir1995-46_part1_en.pdf.

[2] The health insurance portability and accountability act.

[3] The java pairing based cryptography library. http://gas.dia.unisa.it/projects/jpbc/.

[4] Martín Abadi, Dan Boneh, Ilya Mironov, Ananth Raghunathan, and Gil Segev. Message-locked encryption for lock-dependent messages. In *CRYPTO (1)*, pages 374–391, 2013.

[5] Tolga Acar, Sherman S. M. Chow, and Lan Nguyen. Accumulators and u-prove revocation. In *Financial Cryptography*, pages 189–196, 2013.

[6] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul Spirakis, editors, *Automata, Languages and Programming*, volume 6198 of *Lecture Notes in Computer Science*, pages 152–163. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-14165-2_14.

[7] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proc. of ACM CCS*, pages 598–609, 2007.

[8] Giuseppe Ateniese, Emiliano De Cristofaro, and Gene Tsudik. (if) size matters: Size-hiding private set intersection. In *Public Key Cryptography*, pages 156–173, 2011.

[9] Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of storage from homomorphic identification protocols. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, pages 319–333, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] L Babai. Trading group theory for randomness. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, STOC '85, pages 421–429, New York, NY, USA, 1985. ACM.

[11] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Public key encryption with keyword search revisited. In *Proc. of ICCSA*, pages 1249–1259, 2008.

[12] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Countering gattaca: efficient and secure testing of fully-sequenced human genomes. In *ACM Conference on Computer and Communications Security*, pages 691–702, 2011.

[13] Feng Bao, Robert H. Deng, Xuhua Ding, and Yanjiang Yang. Private query on encrypted data in multi-user settings. In *Proc. of ISPEC*, pages 71–85, 2008.

[14] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *Proc. of CRYPTO*, pages 535–552, 2007.

[15] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *ACM Conference on Computer and Communications Security*, pages 390–399, 2006.

[16] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *Proc. of CRYPTO*, pages 111–131, 2011.

[17] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, pages 111–131, 2011.

[18] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Proc. of IEEE S&P*, pages 321–334, 2007.

[19] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[20] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Proc. of EUROCRYPT*, pages 440–456, 2005.

[21] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *In proceedings of CRYPTO ï¿œ04, LNCS series*, pages 41–55. Springer-Verlag, 2004.

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

[23] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Proceedings of the 22nd international conference on Theory and applications of cryptographic techniques*, EUROCRYPT'03, pages 416–432, Berlin, Heidelberg, 2003. Springer-Verlag.

[24] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Proc. of TCC*, pages 535–554, 2007.

[25] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

[26] Jan Camenisch, Markulf Kohlweiss, Alfredo Rial, and Caroline Sheedy. Blind and anonymous identity-based encryption and authorised private searches on public key encrypted data. In *Proc. of PKC*, pages 196–214, 2009.

[27] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '02, pages 61–76, London, UK, UK, 2002. Springer-Verlag.

[28] Jan Camenisch and Gregory M. Zaverucha. Private intersection of certified sets. In *Financial Cryptography*, pages 108–127, 2009.

[29] Sébastien Canard, Georg Fuchsbauer, Aline Gouget, and Fabien Laguillaumie. Plaintext-checkable encryption. In *CT-RSA*, pages 332–348, 2012.

[30] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT*, pages 207–222, 2004.

[31] Ran Canetti, Omer Paneth, Dimitrios Papadopoulos, and Nikos Triandopoulos. Verifiable set operations over outsourced databases. Cryptology ePrint Archive, Report 2013/724, 2013. `http://eprint.iacr.org/`.

[32] Qi Chai and Guang Gong. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In *Proc. of ICC*, pages 917–922, 2012.

[33] Qi Chai and Guang Gong. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In *ICC*, pages 917–922, 2012.

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

[35] Melissa Chase. Multi-authority attribute based encryption. In *Proc. of TCC*, pages 515–534, 2007.

[36] Melissa Chase and Sherman S.M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *Proc. of ACM CCS*, pages 121–130, 2009.

[37] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In *Proc. of ASIACRYPT*, pages 577–594, 2010.

[38] Kai-Min Chung, Yael Kalai, and Salil Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Proceedings of the 30th annual conference on Advances in cryptology*, CRYPTO'10, pages 483–501, Berlin, Heidelberg, 2010. Springer-Verlag.

[39] Kai-Min Chung, Yael Kalai, and Salil Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Proceedings of the 30th annual conference on Advances in cryptology*, CRYPTO'10, pages 483–501, Berlin, Heidelberg, 2010. Springer-Verlag.

[40] Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501, 2010.

[41] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear computational and bandwidth complexity. *IACR Cryptology ePrint Archive*, 2009:491, 2009.

[42] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography*, pages 143–159, 2010.

[43] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proc. of*, pages 79–88, 2006.

[44] Dana Dachman-Soled, Tal Malkin, Mariana Raykova 0001, and Moti Yung. Efficient robust private set intersection. In *ACNS*, pages 125–142, 2009.

[45] Ivan Damgård and Nikos Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. *IACR Cryptology ePrint Archive*, 2008:538, 2008.

[46] Premkumar Devanbu, Michael Gertz, Charles Martel, and Stuart G. Stubblebine. Authentic data publication over the internet. *J. Comput. Secur.*, 11(3):291–314, April 2003.

[47] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: An efficient and scalable protocol. Cryptology ePrint Archive, Report 2013/515, 2013. `http://eprint.iacr.org/`.

[48] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *J. Cryptol.*, 1(2):77–94, August 1988.

[49] Amos Fiat. Batch rsa. In *Proceedings on Advances in cryptology*, CRYPTO '89, pages 175–185, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[50] Dario Fiore and Rosario Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *Proc. of ACM CCS*, pages 501–512, 2012.

[51] Lance Fortnow and Carsten Lund. Interactive proof systems and alternating time-space complexity. In *Selected papers of the 8th annual symposium on Theoretical aspects of computer science*, STACS '91, pages 55–73, Essex, UK, 1991. Elsevier Science Publishers Ltd.

[52] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, pages 1–19, 2004.

[53] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *Proceedings of the 30th annual conference on Advances in cryptology*, CRYPTO'10, pages 465–482, Berlin, Heidelberg, 2010. Springer-Verlag.

[54] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[55] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, pages 626–645, 2013.

[56] Craig Gentry. Computing arbitrary functions of encrypted data. *Commun. ACM*, 53(3):97–105, March 2010.

[57] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 465–482, Berlin, Heidelberg, 2012. Springer-Verlag.

[58] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/2003/216/`.

[59] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, February 1989.

[60] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 113–122, New York, NY, USA, 2008. ACM.

[61] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

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

[63] Philippe Golle, Jessica Staddon, and Brent Waters. Secure conjunctive keyword search over encrypted data. In *Proc. of ACNS*, pages 31–45, 2004.

[64] Michael Goodrich, Roberto Tamassia, and Nikos Triandopoulos. Super-efficient verification of dynamic outsourced databases. In Tal Malkin, editor, *Topics in Cryptology ï¿œC CT-RSA 2008*, volume 4964 of *Lecture Notes in Computer Science*, pages 407–424. Springer Berlin / Heidelberg, 2008.

[65] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proc. of ACM CCS*, pages 89–98, 2006.

[66] Trusted Computing Group. Trusted platform module main specification,version 1.2, revision 103,. Technical report, July 2007.

[67] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. *J. Cryptology*, 23(3):422–456, 2010.

[68] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In *Public Key Cryptography*, pages 312–331, 2010.

[69] Y. Huang, D. Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? *19th Network and Distributed Security Symposium*, 2012.

[70] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *TCC*, pages 577–594, 2009.

[71] Seny Kamara and Kristin Lauter. Cryptographic cloud storage. In *Proc. of FC*, pages 136–149, 2010.

[72] Seny Kamara, Payman Mohassel, Mariana Raykova, and Saeed Sadeghian. Server-aided private set intersection: Scaling to million element sets, 2013. `http://research.microsoft.com/pubs/194141/sapsi.pdf`.

[73] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Cs2: A searchable cryptographic cloud storage system. Microsoft Technical Report, 2011. `http://research.microsoft.com/apps/pubs/?id=148632`.

[74] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proc. of ACM CCS*, pages 965–976, 2012.

[75] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[76] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Proc. of EUROCRYPT*, pages 146–162, 2008.

[77] Florian Kerschbaum. Collusion-resistant outsourcing of private set intersection. In *SAC*, pages 1451–1456, 2012.

[78] Florian Kerschbaum. Outsourced private set intersection using homomorphic encryption. In *ASIACCS*, pages 85–86, 2012.

[79] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *CRYPTO*, pages 241–257, 2005.

[80] Kaoru Kurosawa and Yasuhiro Ohtaki. Uc-secure searchable symmetric encryption. In *Proc. of FC*, pages 285–298. Springer Berlin / Heidelberg.

[81] Kristin Lauter, Michael Naehrig, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? Cryptology ePrint Archive, Report 2011/405, 2011. `http://eprint.iacr.org/`.

[82] Allison B. Lewko and Brent Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In *Proc. of CRYPTO*, pages 180–198, 2012.

[83] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 121–132, New York, NY, USA, 2006. ACM.

[84] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Authenticated index structures for aggregation queries. *ACM Trans. Inf. Syst. Secur.*, 13(4):32:1–32:35, December 2010.

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

[86] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, October 1992.

[87] Ralph C. Merkle. A certified digital signature. In *Proceedings on Advances in cryptology*, CRYPTO '89, pages 218–238, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[88] Ghita Mezzour, Adrian Perrig, Virgil Gligor, and Panos Papadimitratos. Privacy-preserving relationship path discovery in social networks. In *Cryptology and Network Security*, volume 5888 of *Lecture Notes in Computer Science*, pages 189–208. Springer Berlin Heidelberg, 2009.

[89] Kyriakos Mouratidis, Dimitris Sacharidis, and Hweehwa Pang. Partially materialized digest scheme: an efficient verification method for outsourced databases. *The VLDB Journal*, 18(1):363–381, January 2009.

[90] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Providing authentication and integrity in outsourced databases using merkley hash trees. In *UCI-SCONCE Technical Report*.

[91] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. *Trans. Storage*, 2(2):107–138, May 2006.

[92] Maithili Narasimha and Gene Tsudik. Authentication of outsourced databases using signature aggregation and chaining. In *Proceedings of the 11th international conference on Database Systems for Advanced Applications*, DASFAA'06, pages 420–436, Berlin, Heidelberg, 2006. Springer-Verlag.

[93] Lan Nguyen. Accumulators from bilinear pairings and applications. In *Proceedings of the 2005 international conference on Topics in Cryptology*, CT-RSA'05, pages 275–292, Berlin, Heidelberg, 2005. Springer-Verlag.

[94] Glen Nuckolls. Verified query results from hybrid authentication trees. In *Proceedings of the 19th annual IFIP WG 11.3 working conference on Data and Applications Security*, DBSec'05, pages 84–98, Berlin, Heidelberg, 2005. Springer-Verlag.

[95] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In *Proc. of ASIACRYPT*, pages 214–231, 2009.

[96] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *Proc. of CRYPTO*, pages 191–208, 2010.

[97] Bernardo Palazzi, Maurizio Pizzonia, and Stefano Pucacco. Query racing: fast completeness certification of query results. In *Proceedings of the 24th annual IFIP WG 11.3 working conference on Data and applications security and privacy*, DBSec'10, pages 177–192, Berlin, Heidelberg, 2010. Springer-Verlag.

[98] HweeHwa Pang, Arpit Jain, Krithi Ramamritham, and Kian-Lee Tan. Verifying completeness of relational query results in data publishing. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 407–418, New York, NY, USA, 2005. ACM.

[99] HweeHwa Pang, Jilian Zhang, and Kyriakos Mouratidis. Scalable verification for outsourced dynamic databases. *Proc. VLDB Endow.*, 2(1):802–813, August 2009.

[100] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. Cryptology ePrint Archive, Report 2011/587, 2011. `http://eprint.iacr.org/`.

[101] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *Proceedings of the 10th theory of cryptography conference on Theory of Cryptography*, TCC'13, pages 222–242, Berlin, Heidelberg, 2013. Springer-Verlag.

[102] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *TCC*, pages 222–242, 2013.

[103] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Authenticated hash tables. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 437–448, New York, NY, USA, 2008. ACM.

[104] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal verification of operations on dynamic sets. Cryptology ePrint Archive, Report 2010/455, 2010. `http://eprint.iacr.org/`.

[105] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal verification of operations on dynamic sets. In Phillip Rogaway, editor, *Advances in Cryptology ï¿œC CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 91–110. Springer Berlin / Heidelberg, 2011.

[106] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: verifiable computation from attribute-based encryption. In *Proceedings of the 9th international conference on Theory of Cryptography*, TCC'12, pages 422–439, Berlin, Heidelberg, 2012. Springer-Verlag.

[107] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, pages 422–439, 2012.

[108] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Proc. of EUROCRYPT*, pages 457–473, 2005.

[109] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '08, pages 90–107, Berlin, Heidelberg, 2008. Springer-Verlag.

[110] Adi Shamir. Ip = pspace. *J. ACM*, 39(4):869–877, October 1992.

[111] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *Proc. of TCC*, pages 457–473, 2009.

[112] Elaine Shi, John Bethencourt, Hubert T.-H. Chan, Dawn Xiaodong Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *Proc. of IEEE S&P*, pages 350–364, 2007.

[113] Sean W. Smith, Elaine R. Palmer, and Steve Weingart. Building a high-performance, programmable secure coprocessor. In *Computer Networks*, pages 831–860, 1999.

[114] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proc. of IEEE S&P*, pages 44–, 2000.

[115] Roberto Tamassia and Nikos Triandopoulos. Certification and authentication of data structures. In *AMW*, 2010.

[116] Qiang Tang. Towards public key encryption scheme supporting equality test with fine-grained authorization. In *ACISP*, pages 389–406, 2011.

[117] Brian Thompson, Stuart Haber, William G. Horne, Tomas Sander, and Danfeng Yao. Privacy-preserving computation and verification of aggregate queries on outsourced databases. In *Privacy Enhancing Technologies*, pages 185–201, 2009.

[118] Boyang Wang, Ming Li, S.S.M. Chow, and Hui Li. Computing encrypted cloud data efficiently under multiple keys. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 504–513, Oct 2013.

[119] Brent R. Waters, Dirk Balfanz, Glenn Durfee, and Diana K. Smetters. Building an encrypted and searchable audit log. In *Proc. of NDSS*, 2004.

[120] Yuanfeng Wen, Yuanfeng Wen, Jonghyuk Lee, Ziyi Lu, Qingji Zheng, Weidong Shi, Shouhuai Xu, and Taeweon Suh. Multi-processor architectural support for protecting virtual

machine privacy in untrusted cloud environment. In *Proceedings of 2013 ACM International Conference on Computing Frontiers*, CF'13, 2013.

[121] Min Xie, Haixun Wang, Jian Yin, and Xiaofeng Meng. Integrity auditing of outsourced data. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 782–793. VLDB Endowment, 2007.

[122] Jia XU and Ee-Chien CHANG. Authenticating aggregate range queries over multidimensional dataset. Cryptology ePrint Archive, Report 2010/050, 2010. `http://eprint.iacr.org/`.

[123] 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.

[124] Yin Yang, Dimitris Papadias, Stavros Papadopoulos, and Panos Kalnis. Authenticated join processing in outsourced databases. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 5–18, New York, NY, USA, 2009. ACM.

[125] Yin Yang, Stavros Papadopoulos, Dimitris Papadias, and George Kollios. Spatial outsourcing for location-based services. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 1082–1091, Washington, DC, USA, 2008. IEEE Computer Society.

[126] Bennet Yee. Using secure coprocessors. Technical report, Carnegie Mellon University, 1994.

[127] Lan Zhang, Xiang-Yang Li, Yunhao Liu, and Taeho Jung. Verifiable private multi-party computation: Ranging and ranking. In *INFOCOM*, pages 605–609, 2013.

[128] Qingji Zheng and Shouhuai Xu. Verifiable delegated set intersection operations on outsourced encrypted data. Cryptology ePrint Archive, Report 2014/178, 2014. `http://eprint.iacr.org/`.

[129] Qingji Zheng, Shouhuai Xu, and Giuseppe Ateniese. Efficient query integrity for outsourced dynamic databases. In *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*, CCSW '12, pages 71–82, New York, NY, USA, 2012. ACM.

[130] Qingji Zheng, Shouhuai Xu, and Giuseppe Ateniese. Vabks: Verifiable attribute-based keyword search over outsourced encrypted data. Cryptology ePrint Archive, Report 2013/462, 2013. `http://eprint.iacr.org/`.

VITA

Qingji Zheng was born in Fujian province, China. He received his B.S. degree from Beihang University in 2006, and M.SC. degree from Shanghai Jiaotong University in 2009. He started his Ph.D. study at University of Texas at San Antonio from September 2009.