Authenticating Aggregate Queries over Set-Valued Data with Confidentiality
Xu, Cheng; Chen, Qian; HU, Haibo; XU, Jianliang; Hei, Xiaojun

Published in:
IEEE Transactions on Knowledge and Data Engineering

DOI:
10.1109/TKDE.2017.2773541

Published: 01/04/2018

Citation for published version (APA):
Authenticating Aggregate Queries over Set-Valued Data with Confidentiality

Cheng Xu, Qian Chen, Haibo Hu, Jianliang Xu, Xiaojun Hei

Abstract—With recent advances in data-as-a-service (DaaS) and cloud computing, aggregate query services over set-valued data are becoming widely available for business intelligence that drives decision making. However, as the service provider is often a third-party delegate of the data owner, the integrity of the query results cannot be guaranteed and is thus imperative to be authenticated. Unfortunately, existing query authentication techniques either do not work for set-valued data or they lack data confidentiality. In this paper, we propose authenticated aggregate queries over set-valued data that not only ensure the integrity of query results but also preserve the confidentiality of source data. As many aggregate queries are composed of multiset operations such as set union and subset, we first develop a family of privacy-preserving authentication protocols for primitive multiset operations. Using these protocols as building blocks, we present a privacy-preserving authentication framework for various aggregate queries and further optimize their authentication performance. Security analysis and empirical evaluation show that our proposed privacy-preserving authentication techniques are feasible and robust under a wide range of system workloads.

Index Terms—Query Authentication, Aggregate Queries, Set-Valued Data, Merkle Hash Tree

1 INTRODUCTION

ReCENT advances in data-as-a-service (DaaS) and cloud computing have been driving data from different sources into massive repositories for queries. A typical example is Big-Query [1] in the Google Cloud Platform, which uses SQL-like queries to analyze big datasets. The results of these queries, particularly aggregate queries, in turn serve as sources for data analytics. For example, business intelligence executives can thus make critical, million-dollar decisions such as investing in new business opportunities. As such, ensuring the integrity of query results from the service provider is crucial. Similar requirements also exist in many other domains such as scientific research and government policy. For example, personal genomics analysis (e.g., 23andMe and the Personal Genome Project (PGP) at Harvard Medical School [2]) is based on aggregate queries on large genome datasets, the integrity of whose results is vital. Below is one example:

<table>
<thead>
<tr>
<th>PID</th>
<th>ZIP</th>
<th>Mut-Genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>95014</td>
<td>A-C130R, P-I696M</td>
</tr>
<tr>
<td>P2</td>
<td>20482</td>
<td>H-C282Y, F-P12A, R-G1886S</td>
</tr>
<tr>
<td>P3</td>
<td>95014</td>
<td>A-C130R, U-G71R, W-R611H</td>
</tr>
<tr>
<td>P5</td>
<td>20134</td>
<td>A-C130R, P-P12A, R-G1886S, S-E288V</td>
</tr>
<tr>
<td>P6</td>
<td>17868</td>
<td>C-R102G, C-G1886S</td>
</tr>
<tr>
<td>P7</td>
<td>55410</td>
<td>C-R102G, C-Q1334H, S-E288V</td>
</tr>
<tr>
<td>P8</td>
<td>20852</td>
<td>C-R102G, P-P12A, R-G1886S, K-T220M</td>
</tr>
</tbody>
</table>

TABLE 1: Set-Valued Genome Dataset

Example 1. Aggregate Queries on PGP Data. Table 1 shows a sample genome dataset, where PID is participant ID, ZIP is ZIP code, and Mut-Genes is a sensitive set-valued attribute that records the mutation genes of each participant. Users (e.g., medical doctors) may be interested in the following aggregate queries:

- **Q1**: Find the most common gene in the district of Cupertino, CA (ZIP: 95014).
- **Q2**: Count the number of participants who carry the gene ‘R-G1886S’.
- **Q3**: Find the most frequent genes with supports $\geq 3$ in ZIPS 20***.

The corresponding query results are: `{‘A-C130R’}`, 4, and `{‘P-P12A’, ‘R-G1886S’}`, respectively.

Unfortunately, there is no guarantee of the service provider returning correct and complete results, for various reasons that could include service outages, hack attacks, or even corporate dishonesty. In the field of query processing, authentication on query results has been studied by a large body of literature [3]–[10]. The fundamental concept is that the data owner signs a well-designed authenticated data structure (ADS), based on which the service provider then constructs a cryptographic proof for each query and returns it along with the query results for the query client to verify. However, the prior research cannot be applied to aggregate queries on set-valued genomics or business data as exemplified above for two reasons. First, most of the research has been focused on relational or spatial data and their associated query types, whereas set-valued data have their own unique operations and aggregate query types. Second, most of the research ignores data confidentiality — they assume that the service provider is willing to and capable of sharing a substantial amount of source data to the query client for result verification. In the genomics analysis, business intelligence, and many other cases, however, the source data cannot be disclosed to the query client, either because they are private assets of the data owner, or...
because they may contain sensitive personal information such as individual genome sequences.

As set-valued data are ubiquitous and sometimes indispensable in describing real-world problems for analytics, in this paper we study authenticated aggregate query services over set-valued data with confidentiality preservation. We assume that a dataset consists of one sensitive set-valued attribute (e.g., mutation-gene set) and multiple non-sensitive attributes (e.g., ZIP code and age). As illustrated in Example 1, an aggregate query is defined as a query whose result is derived from aggregates of data. In this paper, each aggregate query consists of two phases: a filtering phase that filters data by a range selection on non-sensitive attributes, followed by an aggregating phase that aggregates on the sensitive set-valued attribute. For broader applicability, we model the sensitive set-valued attribute as a multiset, i.e., a set that allows duplicate elements. This enables advanced data analytics such as frequent itemset mining in market basket analysis [11], inverted indexes in web search and auto-completion [12], and web log mining [13], all of which require a collection data type that supports duplicate elements.

Because aggregate queries exhibit unique properties that are unseen in those queries previously studied for authentication [3–5], the design of the ADS and its related authentication techniques face great challenges. First, due to the dynamics of aggregate queries (e.g., each query in Example 1 selects a different group of participants), it is impossible to pre-compute the query results and endorse them in advance. Second, aggregate queries over set-valued data are largely supported by multiset operations, for which existing authentication techniques, mostly designed for relational queries, cannot be applied. Third, to safeguard source data confidentiality, the query client must be able to verify the query results without learning any single set of sensitive values. To address these challenges, we propose PA², a privacy-preserving authentication framework for aggregate queries. It consists of an authenticated index Merkle Grid tree (MG-tree), a family of privacy-preserving authentication protocols for primitive multiset operations, and a series of aggregate query processing algorithms. We further propose several optimizations to reduce the computation and communication overheads for both the server and the query client. In summary, our contributions made in this paper are as follows:

- To the best of our knowledge, this is the first work that addresses both integrity and confidentiality for aggregate queries over set-valued data.
- We propose a privacy-preserving authentication framework and we develop a set of privacy-preserving authentication protocols and algorithms for various aggregate queries.
- We provide formal security analysis and cost models for the proposed authentication protocols and algorithms.
- We propose optimizations to enhance the performance of query authentication algorithms.
- We perform extensive performance evaluation on real-world datasets. Empirical results demonstrate the feasibility and robustness of our proposals.

The rest of this paper is organized as follows. Section 2 reviews the related work on aggregate queries and authenticated query processing. Section 3 formally defines the problem and its security model. Section 4 introduces some related cryptographic constructs. Section 5 proposes the PA² framework for privacy-preserving authentication of aggregate queries based on a family of verifiable multiset operations. Security analysis and performance optimization techniques are given in Sections 6 and 7. Section 8 presents our experimental results and is followed by a conclusion.

2 RELATED WORK

Aggregate Queries over Set-Valued Data. Set-valued data, in which a set of values is associated with an individual, is common in data analytics applications ranging from market basket analysis, to web log mining, to healthcare research. Özsoyölü et al. were the first to extend relational algebra and aggregate functions to set-valued data [14]. They defined algebraic expressions such as set union, set difference, and aggregation. Since then, data analytics over set-valued data has been extensively studied in various application domains. The most well-known application is association rule mining [15]. Given a database of sales transactions, each containing a subset of items in the product universe, the objective is to find rules such as “a user buying item(s) X will probably buy item(s) Y“. Although various algorithms are proposed, a fundamental problem in this application is to efficiently compute two aggregate values, namely, the support and confidence of X [11]. The former is the number of transactions that contain X, whereas the latter is those in the former that also contain Y. With the boom of web search and online advertisement, query log and click stream have become new sources of set-valued data. A variety of aggregate queries have been proposed on these sources for tasks such as website clustering and frequent item identification and counting [16]. Recently, graphs have become another new source of set-valued data. In particular, social networks have contributed various social relations, such as “friend/unfriend,” “follow,” “post/tag,” and user access rights, to set-valued data [17]. Aggregating such data for social network analysis and recommendation has been intensively studied [18].

Authenticated Query Processing. A large body of research on authenticated query processing has been carried out to verify the integrity of query results against an untrusted service provider. Most of the existing works focus on relational and spatial data [3–6], [19]. Their query authentication schemes are based on two basic techniques, namely, digital signature and Merkle hash tree (MHT). The former is a public-key message authentication scheme based on asymmetric cryptography. A digital signature is produced for each data value by the data owner using a private key. A verifier can verify the authenticity of a value using the owner’s public key and the value’s signature. To further ensure the completeness of query results, signature chaining was proposed to correlate the signatures of adjacent values [3]. Signature chaining is simple, but it requires each value to be signed and thus is not scalable to large datasets. MHT, on the other hand, requires only a single signature on an index tree [20]. Each entry in a leaf node is assigned a digest based on its data value, and each entry in an internal node is assigned a digest derived from its child nodes. The data owner signs the root digest of the MHT, which can be used to verify any subset of data values. The MHT has been widely adapted to various index structures. Typical examples include the Merkle B-tree for relational data [4], the Merkle R-tree for spatial data [5], [21], the authenticated inverted index for text data [22], and the authenticated prefix tree for multi-source data [9].

More recently, privacy-preserving query authentication techniques have been studied for location-based range and top-k queries [23], [24]. The core idea is to enable the verification of comparison results on private values. However, these previous works consider relational/spatial data only, not set-valued data.
which are the focus of this paper. As we will see later in this paper, aggregate queries over set-valued data heavily involve primitive multiset operations such as set union and subset. This renders the existing privacy-preserving authentication techniques that are based on private value comparisons inapplicable.

On the other hand, there are only a few query authentication works over set-valued data. For primitive set operations, Papamanthou et al. proposed efficient verification protocols for set union, set intersection, and set difference operations [25]. Canetti et al. extended it by allowing data updates and monolithic verification of hierarchical set operations [26]. Papadopoulos et al. studied efficient high-dimensional range query authentication [27]. Dong et al. [28] developed a result verification protocol for frequent itemset mining. Yet, none of these previous studies has considered the data confidentiality requirement. For example, the solution in [28] allows the query client to learn which transactions contain the frequent itemsets, thus breaching data privacy. Protecting the confidentiality of source data requires designing new cryptographic constructs on primitive multiset operations. The preliminary work in [29] supports multiset sum operations only. Papadopoulos et al. proposed a zero-knowledge accumulator for set-valued data and common primitive operations [30]. However, it is computationally expensive and does not support multisets. Recently, another zero-knowledge set accumulator is proposed by Zhang et al. It is expressive but still lacks of the support for multiset operations [31]. Furthermore, how to support authenticated aggregate queries based on verifiable multiset operations while preserving data confidentiality is another new challenge. Table 2 summarizes the differences between our work and the previous works.

Table 2: Comparison with prior works

<table>
<thead>
<tr>
<th>reference</th>
<th>set operations</th>
<th>multiset operations</th>
<th>aggregate queries</th>
<th>privacy preserving</th>
</tr>
</thead>
<tbody>
<tr>
<td>[25], [26], [27]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[29]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[30], [31]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>our work</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

3 Problem Definition

A data owner (DO) owns a dataset \( D = \{ o_1, o_2, \ldots, o_n \} \). Each object \( o_i \) is represented by \( A_i = \{ X_i \} \), where \( A_i \) is a set of non-sensitive attributes, and \( X_i \) is a sensitive multiset of features (hereafter called feature set). The DO outsources \( D \) to a third-party service provider (SP), together with an authenticated data structure (ADS) signed with the DO’s private key. Based on this, the SP provides aggregate query services to clients (e.g., Q1 and Q2, and Q3 in Example 1).

Multiset Operations. A multiset is a generalization of a set in which elements are allowed to occur more than once [32]. The number of occurrences is called the multiplicity of an element. For instance, in multiset \( \{ a, a, b \} \), \( a \) has a multiplicity of 2 and \( b \) has a multiplicity of 1. The multiset is order insensitive, so it can be represented as a set of pairs \( (x, \eta) \) where \( x \) is an element and \( \eta \) is its multiplicity. The above multiset \( \{ a, a, b \} \) can thus be rewritten as \( \{ (a, 2), (b, 1) \} \). The most important operations in a multiset are union and sum, denoted as \( \cup \) and \( \oplus \), respectively. The union operation on multisets is exactly the same as that on regular sets — it simply unifies two multisets and sets the multiplicity of each element to 1. For example, given multisets \( X_1 = \{ (a, 2), (b, 1) \} \), \( X_2 = \{ (b, 1), (c, 2) \} \), we have \( X_1 \cup X_2 = \{ (a, 1), (b, 1), (c, 1) \} \). In contrast, the sum operation sets the multiplicity of each element as the sum of multiplicities in the original multisets. For instance, given the same multisets as above, we have \( X_1 \oplus X_2 = \{ (a, 2), (b, 2), (c, 2) \} \).

Aggregate Queries. Since the results of aggregate queries are derived from aggregates of data, our study mainly focuses on the following primitive aggregate queries: \( \text{max/min, count, sum, top}-k \), and frequent feature query (FFQ). More specifically, an aggregate query can be expressed in the form of \( Q = (q, \{ \alpha, \beta \}, \{ \alpha, \beta \}) \), where \( q \) is the aggregate operator, \( \{ x_i \} \) is the queried feature (which is only needed for \text{count and sum} ), and \( \{ \alpha, \beta \} \) specifies the selection range on the non-sensitive attributes. Consider a sample query \( Q = (q, \{ x_i \}, \{ \alpha, \beta \}) \) in Figure 1.

Fig. 1: Example of Aggregate Queries

- \( q = \text{sum or count} \): This sums or counts the multiplicities of \( \{ x_i \} \) in all selected objects. Supposing \( \{ x_i \} = \{ a \} \), the \text{sum} and \text{count} results are \( (a, 6) \) and \( (a, 4) \), respectively.\(^2\)
- \( q = \text{max or min} \): This selects the feature with the maximum or minimum summed multiplicity in all selected objects. If \( q = \text{max} \), the result is \( (a, 6) \); if \( q = \text{min} \), the result is \( (b, 1) \).
- \( q = \text{top}-k \): This selects the top\(-k\) features with the largest summed multiplicities in all selected objects. Supposing \( k = 3 \), the result is \( (a, 6), (c, 4), (d, 3) \).
- \( q = \text{FFQ} \): This selects the features whose summed multiplicities in all selected objects are no less than a threshold \( \delta \). Supposing \( \delta = 4 \), the result is \( (a, 6), (c, 4) \).

The aggregate queries in Example 1 can be reduced to the above aggregate queries as follows. Q1 is simply a \text{max} query, i.e., \( (\text{max}, \cdot, [95014, 95014]) \). Q2 is a \text{count} query, i.e., \( (\text{count}, \{ R\text{-G188651}, \cdot, [0000, 999999] \}) \). Q3 is an FFQ query, \( (\text{FFQ}_3, \cdot, [20000, 29999]) \), which finds the set of genes whose sum result is no less than 3.

Threat Model and Problem Statement. We consider two potential security threats: 1) the SP could provide unfaithful query execution, thereby returning incorrect or incomplete query results; and 2) data privacy could be breached if sensitive source data are disclosed to the query client. Thus, the authentication problem we are investigating is for the query client to verify that the SP executes \( Q \) faithfully in terms of the following conditions: 1) the candidate objects are correctly selected and no objects in the selection range are skipped; 2) the returned features and multiplicities are not tampered with; and 3) the query result satisfies the aggregation semantics. The confidentiality requirement in this problem is to protect the objects’ (sensitive) feature sets against

1. Extension to other more advanced aggregate queries such as \text{average} and \text{confidence} will be discussed in Section 5.2.3.
2. The \text{sum} and \text{count} results will be the same if there are no duplicate elements in the feature sets.
the query client. That is, the client cannot infer the features (as well as their multiplicities) of any single object beyond what is implied from the query result.

If neither efficiency nor confidentiality is a concern, authenticating an aggregate query can work as follows. The SP returns a verification object (VO) to the client, along with the query result. As a naive solution, the VO may include the non-sensitive attributes and sensitive features of all objects in $\mathbb{D}$ and a signature of $\mathbb{D}$. The client uses the VO to verify the soundness and completeness of the results by testing the following conditions:

- None of the objects in $\mathbb{D}$ is tampered with.
- All candidate objects are in $[\alpha, \beta]$ and no objects in $[\alpha, \beta]$ are missing.
- The features and multiplicities of the candidate objects are correct.
- The result satisfies the aggregation semantics of $g$.

However, the verification cost of this naive solution is prohibitively high because the entire dataset has to be returned. Moreover, verifying the last two conditions without disclosing sensitive feature sets requires privacy-preserving protocols. To address these issues, we propose an efficient privacy-preserving authentication framework based on verifiable multiset operations, the preliminaries of which are introduced in the next section.

4 Preliminaries

This section gives some preliminaries on cryptographic constructs and integrity assurance.

Cryptographic Hash Function. A cryptographic hash function $h(\cdot)$ accepts an arbitrary-length string as its input and returns a fixed-length bit string. It is collision resistant, i.e., it is difficult to find two different messages $m_1$ and $m_2$ such that $h(m_1) = h(m_2)$. Classic cryptographic hash functions include MD5 and SHA-1.

Bilinear-Map (BM) Accumulator. This maps a multiset to a single value for ease of processing. Let $\mathbb{G}$ be a cyclic multiplicative group of order $p$. A BM accumulator is a function of a multiset $X$ of $n$ elements in the cyclic group $\mathbb{Z}_p$ [33]. It returns an accumulative value of $X$:

$$acc(X) = g^{P(X)} = g^{\prod_{x \in X}(x+s)},$$

where $g$ is a group generator of $\mathbb{G}$, $s \in \mathbb{Z}_p = \mathbb{Z}_p \setminus \{0\}$ is a random secret, and $P(X) = \prod_{x \in X}(x+s)$.

One useful property of $acc(X)$ is that even without knowing $s$, $acc(X)$ can still be computed by $X$ and $g$, $g^s$, $g^{s^2}$, etc. through polynomial interpolation. As for its security, it has been proved in [34] that the accumulative function $acc(\cdot)$ is collision resistant.

Randomized BM Accumulator. The above accumulative value $acc(X)$ is deterministic for a fixed multiset $X$. As such, an adversary can determine with high confidence that two multisets are the same if they happen to have the same accumulative value. To enhance confidentiality, we propose randomizing the $acc$ value of $X$ as

$$acc(X) = g^{P(X)\cdot r_X},$$

where $r_X$ is a random value hidden from the query client but disclosed to the SP. It is worth noting that this randomization does not affect the original properties of a BM accumulator. We further prove in Section 6 that the randomized $acc$ values are indistinguishable under chosen plaintext attack.

Bilinear Pairing. This maps a pair of elements in two groups to a single element in a third group. Let $\mathbb{G}_1$ be another cyclic multiplicative group with the same order $p$. We can find a bilinear mapping $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$ which has the following properties:

1) Bilinearity: If $u, v \in \mathbb{G}$ and $e(u, v) \in \mathbb{G}_1$, then $e(u^a, v^b) = e(u, v)^{ab}$ for any $u, v$.
2) Non-degeneracy: $e(g, g) \neq 1$.
3) Computability: Given $u, v \in \mathbb{G}$, it is easy to compute $e(u, v)$.

Bilinear $q$-Strong Diffie-Hellman (DH) Assumption. This assumption shows that bilinear pairing is appropriate for multiset operation authentication as it is hard to forge. Let $(\mathbb{G}, \mathbb{G}_1, e, g)$ be a bilinear pairing. This assumption says that as long as $s \in \mathbb{Z}_p^*$ is secret, even given all elements $g, g^s, \ldots, g^{sk}$ in $\mathbb{G}$, no probabilistic polynomial-time (PPT) algorithm can derive $e(g, g)^{s/(a+s)}$ for any $x \in \mathbb{Z}_p$ with a probability higher than a negligible value [35].

In essence, this bilinear $q$-strong DH assumption extends the regular DH assumption on $g$ and $\mathbb{G}$ to $e(g, g)$ and $\mathbb{G}_1$. This assumption will be used as a foundation in our security analysis.

Set Operation Authentication. Based on the above, two result verification protocols have been introduced in [25], [26] to authenticate the following operations on two sets $X_1$ and $X_2$: (i) $X_1 \subseteq X_2$ and (ii) $X_1 \cap X_2 = \emptyset$.

For (i), the server computes a witness value $W = acc(X_2 - X_1)$ and returns it to the query client. The client then verifies $X_1 \subseteq X_2$ by checking:

$$e(acc(X_1), W) \equiv e(acc(X_2), g).$$

For (ii), according to the extended Euclidean algorithm, there are two polynomials $Q_1, Q_2$ such that

$$Q_1 \cdot P(X_1) + Q_2 \cdot P(X_2) = 1.$$  

As such, the server prepares $F_1 = g^{Q_1}, F_2 = g^{Q_2}$, and then the client verifies it by checking:

$$e(F_1, acc(X_1)) \cdot e(F_2, acc(X_2)) \equiv e(g, g).$$

Though these two protocols are privacy-preserving in nature and can be extended to multisets, we have yet to design privacy-preserving authentication protocols for other multiset operations such as sum and union, which will be covered in Section 5.1.

5 PA²: Privacy-Preserving Authentication Framework for Aggregate Queries

In this section, we present the complete framework that can authenticate various aggregate queries while preserving data confidentiality. Fig. 2 illustrates both the query and authentication flow charts of the framework. The framework consists of two phases: candidate object selection and aggregate query processing. Recall that given an aggregate query $Q = (q, \{x_i\}, [\alpha, \beta])$, the SP first selects the candidate objects within the range $[\alpha, \beta]$, and then computes the aggregate values of these candidate objects with regard to the queried feature $x_i$. Along with query processing, the server also constructs the verification objects (VOs) for both phases. Once the client receives the query results and VOs, it can authenticate the correctness of the entire query following the verification flow, which is the opposite of the SP’s VO construction flow. In what follows, we present the detailed procedure of each phase in this framework, starting with the second phase for clarity reasons.

5.1 Privacy-Preserving Authentication Protocols on Multiset Operations

Before running into the detailed procedure of authenticating aggregate queries, we present five core privacy-preserving authen-
The key idea is to leverage the randomized bilinear-map accumulator \( \text{acc}() \) (i.e., Eq. (1)) introduced in Section 4 to hash a multiset into a fixed-length value with collision resistance. With the bilinear pairing function \( e(\cdot, \cdot) \) (also introduced in Section 4), the client can verify the accumulated value of the output multiset without learning the content of each input multiset. This distinguishes the accumulator function from other cryptographic hash functions such as SHA-1.

In the rest of this section, we introduce the five core privacy-preserving authentication protocols on multiset operations, namely, \( \text{subset}, \text{sum}, \text{empty}, \text{union}, \) and \( \text{times} \). For ease of presentation, we mark below any unverified value computed by the SP with \( * \), while all other unmarked values are trusted or already verified by the candidate object selection protocol (to be explained in Section 5.3). We also assume that the seeds \( g, g^2, \ldots, g^k \) are public and that the random values \( r_X, s \) are known to the DO and SP but hidden from the client.

**Subset:** \( \text{subset}(X_i, X_j) \). Given two multisets \( X_i, X_j \), it returns the accumulative value \( \text{acc}(X_j - X_i) \). Note that if \( X_i \not\subseteq X_j \), the SP cannot compute a correct value of \( \text{acc}(X_j - X_i) \). Hence, if \( \text{acc}(X_j - X_i) \) is verified as correct, the client is assured that \( X_i \subseteq X_j \). The detailed protocol is as follows:

i. The SP computes the accumulative value \( \text{acc}(X_j - X_i)* \) based on \( X_i, X_j \), the random values \( r_X, r_{X_j} \), and the public seeds \( g, g^2, \ldots, g^k \) (thanks to the nice property of \( \text{acc} \) as mentioned in Section 4), and sends it to the client together with \( \text{acc}(X_i) \) and \( \text{acc}(X_j) \).

ii. The client verifies the correctness of \( \text{acc}(X_j - X_i)* \) by checking the following condition, where we exploit the property of bilinear pairing function \( e(\cdot, \cdot) \) as mentioned in Section 4.

\[
e(\text{acc}(X_i), \text{acc}(X_j - X_i)) \neq e(\text{acc}(X_i), g).
\]

**Sum:** \( \text{sum}(\{X_1, \ldots, X_n\}) \). Given a set of multisets \( \{X_1, \ldots, X_n\} \), it returns the accumulative value \( \text{acc}(S) \) of the sum set \( S = \bigcup_{i=1}^n \{X_i\} \). The detailed protocol is as follows:

i. The SP computes the accumulative values \( \text{acc}(X_1 \cup X_2)*, \text{acc}(X_1 \cup X_2 \cup X_3)*, \ldots, \text{acc}(S)* \), and sends them to the client together with \( \text{acc}(X_1), \ldots, \text{acc}(X_n) \).

ii. The client verifies the correctness of \( \text{acc}(S)* \) by checking the following equations one by one:

\[
\begin{align*}
e(\text{acc}(X_1), \text{acc}(X_2)) & \neq e(\text{acc}(X_1 \cup X_2), g) \\
e(\text{acc}(X_1 \cup X_2)*, \text{acc}(X_3)) & \neq e(\text{acc}(X_1 \cup X_2 \cup X_3)*, g) \\
& \vdots \\
e(\text{acc}(\bigcup_{i=1}^{n-1} X_i)*, \text{acc}(X_n)) & \neq e(\text{acc}(S)*, g)
\end{align*}
\]

**Empty:** \( \text{empty}(\{X_1, \ldots, X_n\}) \). Given a set of multisets \( \{X_1, \ldots, X_n\} \), it verifies whether \( \cap \{X_i\} = \emptyset \). According to the extended Euclidean algorithm, if \( \cap \{X_i\} = \emptyset \), there exist \( n \) polynomials \( Q_i \) such that \( \sum_{i=1}^n Q_i \cdot P(X_i) = 1 \). The detailed protocol is as follows:

i. The SP computes \( n \) values \( F_i = g^{Q_i/X_i} \), and sends \( F_1*, \ldots, F_n*, \text{acc}(X_1), \ldots, \text{acc}(X_n) \) to the client.

ii. The client verifies the result is empty if the following equation holds:

\[
\prod_{i=1}^n e(\text{acc}(X_i), F_i^*) = e(g, g).
\]

**Union:** \( \text{union}(\{X_1, \ldots, X_n\}) \). Given a set of multisets \( \{X_1, \ldots, X_n\} \), it returns the accumulative value \( \text{acc}(U) \) of the union set \( U = \bigcup_{i=1}^n X_i \). This operation is more complicated than the \( \text{sum} \) operation. Denote \( \tilde{X} \) as the set version of a multiset \( X \). The client needs to verify two conditions:

1. Deflation checking: \( \tilde{X}_i \subseteq U \cap \tilde{X}_2 \subseteq U \cap \cdots \tilde{X}_n \subseteq U \). This is to prevent the SP from deliberately missing any object;

2. Inflation checking: \( (U - \tilde{X}_1) \cap (U - \tilde{X}_2) \cap \cdots (U - \tilde{X}_n) = \emptyset \). This is to prevent the SP from deliberately adding any non-result object.

We can authenticate them based on the above \( \text{subset} \) and \( \text{empty} \) protocols as follows:

i. Use protocols \( \text{subset}(\tilde{X}_1, U), \text{subset}(\tilde{X}_2, U), \ldots, \text{subset}(\tilde{X}_n, U) \) to verify the first condition. This not only authenticates \( \tilde{X}_i \subseteq U \), but also provides the client the correct \( \text{acc}(U - \tilde{X}_i) \).

ii. The SP computes a special hash value \( h_e(U)^* \) of the union set \( U \), and by verifying this value the client is assured that the SP knows the pre-image \( U \) besides \( \text{acc}(U)^* \).

iii. Use protocol \( \text{empty}(\{U - \tilde{X}_1\}, \{U - \tilde{X}_2\}, \cdots, \{U - \tilde{X}_n\}) \) to verify the second condition.

In step (ii), \( h_e(\cdot) \) is an extractable collision resistant hash (ECRH) function that has one unique property — extractability [36]. By presenting a correct \( h_e(U)^* \) value, the SP can prove that it knows \( U \). Inspired by [36], we design \( h_e(\cdot) \) for a multiset \( X = \{x_1, x_2, \ldots, x_t\} \) as follows:

\[
h_e(X) = (h_1, h_2) = (\prod_{i \in [t]} g^{s_{a_i}}, \prod_{i \in [t]} g^{o_{a_i}}), \]

where \( s, o \in \mathbb{Z}_p^* \) are secret values only known to the DO of \( X \), and \( a_i \) is the \( i^\text{th} \) coefficient for polynomial \( P(X) \cdot r_X \). Before any other party can compute or verify \( h_e(X) \), the DO releases \( g, g^2, \ldots, g^k, g^o, g^{o_2}, \ldots, g^{o_k} \in \mathbb{G} \) to the public, where \( k \geq |X| \), the cardinality of \( X \). Then any part, including the client, can verify whether a hash value \( h_e(X) \) is correct by checking the following equations:

\[
e(h_e(X), h_1, o^g) = e(h_e(X), h_2, g),
\]

\[
h_e(X), h_1 \neq \text{acc}(X).
\]

**Times:** \( \text{times}(X, t) \). Given a multiset \( X \) and a coefficient \( t \), it returns the accumulative value \( \text{acc}(\cdot \cdot X) \) of \( t \cdot X \), which raises the multiplicity of each element in \( X \) by \( t \). For example, if \( X = \{(a, 2), (b, 3)\} \), then \( 3 \cdot X = \{(a, 6), (b, 9)\} \). A straightforward solution is to use the \( \text{sum} \) protocol directly as \( \text{sum}(\{X, X, \ldots, X\}) \). Here we give a more efficient solution using \( \text{sum} \) on the binary components. Let \( t = (b_0, b_1, \ldots, b_d, 2^d) \), the binary form of \( t \). The client only needs to execute \( \text{sum}(\{b_0 \cdot X, \ldots, b_d \cdot 2^d \cdot X\}) \), where \( b_i = 1 \). For instance, if
$t = 5 = (101)_2$, then the client only executes $\text{sum}(\{X, 4 \cdot X\})$.

The detailed protocol is as follows:

i. Let $d = \lfloor \log_2(t) \rfloor$, the SP computes $\text{acc}(2 \cdot X)^*, \cdots, \text{acc}(2^d \cdot X)^*, \text{acc}(t \cdot X)^*$, and sends them to the client together with $\text{acc}(X)$.

ii. The client first verifies the values $\text{acc}(2 \cdot X)^*, \cdots, \text{acc}(2^d \cdot X)^*$ by checking the following equations:

\[
\begin{cases}
    e(\text{acc}(X), \text{acc}(X)) = e(\text{acc}(2 \cdot X)^*, g) \\
    e(\text{acc}(2 \cdot X)^*, \text{acc}(2 \cdot X)^*) = e(\text{acc}(4 \cdot X)^*, g) \\
    \vdots \\
    e(\text{acc}(2^d-1 \cdot X)^*, \text{acc}(2^d-1 \cdot X)^*) = e(\text{acc}(2^d \cdot X)^*, g)
\end{cases}
\]

iii. The client then verifies $\text{acc}(t \cdot X)^*$ according to the binary form of $t(b_0b_1 \cdots b_d)$.

We will show in the cost analysis that this protocol significantly improves the performance over the straightforward $\text{sum}$ solution.

5.2 Privacy-Preserving Authentication Algorithms on Aggregate Queries

Next, we present the detailed procedures of the second phase, i.e., authenticating aggregate queries while preserving data confidentiality. Here, we assume that the output from the first phase of candidate object selection has been verified (as explained in Section 5.3). In what follows, $\{X_1, \cdots, X_m\}$ are the feature sets of $m$ candidate objects, and $S = \biguplus \{X_i\}$ is the sum set. We first study the $\text{sum}/\text{count}$ and $\text{max}/\text{min}/\text{FFQ}/\text{top-k}$ aggregate queries and then extend them to other more advanced aggregate queries.

5.2.1 Sum/Count Query

The output of a $\text{sum}$ query $\text{sum}(x_q)$ is $\eta_q$, the sum of multiplicities of feature $x_q$ in all candidate objects. To align with other multiset queries, we define its result as $R = \{(x_q, \eta_q)\}$. For example, in Figure 1, after selecting the candidate objects $\{a_0, a_0, a_0, \cdots, a_1\}$, the query $Q = (\text{sum}, \{a\}, \{a, [\alpha, \beta]\})$ returns the sum of multiplicities of feature $a$, i.e., $R = \{(a, 6)\}$.

To verify this result $R$, we design Algorithm 1 for the client to check the following two conditions:

- Inflation checking (Lines 1–3): $R \subseteq S$. This is to prevent the SP from deliberately increasing the multiplicity of the result.
- Deflation checking (Line 4): $(S - R) \cap R = \emptyset$. This is to prevent the SP from deliberately decreasing the multiplicity of the result.

**Algorithm 1 PA$^2$ Sum ($\{R, X_1, \cdots, X_m\}$)**

1. Obtain $R, \text{acc}(X_1), \cdots, \text{acc}(X_m)$ from the SP (these $\text{acc}$ values are authenticated by the MG-tree, to be detailed in Section 5.3) and compute $\text{acc}(R)$ locally;
2. Execute $\text{sum}(\{X_1, \cdots, X_m\})$ to get verified $\text{acc}(S)$;
3. Execute $\text{sub}(R, S)$ to get verified $\text{acc}(S - R)$; // implies $R \subseteq S$;
4. Execute $\text{empty}(S - R, R)$ to verify $(S - R) \cap R = \emptyset$.

In the running example, the sum set of the candidate objects’ features is $S = \{(a, 6), (b, 1), (c, 4), (d, 3), (e, 2)\}$ and the union set is $U = \{(a, 1), (b, 1), (c, 1), (d, 1), (e, 1)\}$. So the client needs to perform:

- Inflation checking: $\{(a, 6)\} \subseteq \{(a, 6), (b, 1), (c, 4), (d, 3), (e, 2)\}$;
- Deflation checking: $\{(b, 1), (c, 4), (d, 3), (e, 2)\} \subseteq \{(a, 6)\}$;
- Completeness checking: $\{(b, 1), (c, 4), (d, 3), (e, 2)\} \subseteq \{(b, 6), (c, 6), (d, 6), (e, 6)\}$.

The min query is similar except that we verify $(S - R) \supseteq \tau \cdot (U - \hat{R})$ in the completeness checking.

The FFQ query returns the features whose multiplicities are no lower than a threshold $\delta$. This query is naturally supported as it is a subroutine of the $\text{max}$ query (by replacing the threshold $\tau$ with $\delta$ in the completeness checking).

Similarly, the top-$k$ query can be verified by exploiting the routine of $\text{max}$ query and replacing the top-$1$ multiplicity $\tau$ with the top-$k$ multiplicity.

5.2.2 Max/Min/FFQ/Top-k Query

The output of a $\text{max}$ query is the feature with the highest (i.e., top-1) multiplicity. Let $\tau$ denote this multiplicity, and thus the query is equivalent to searching for any feature whose multiplicity is no less than $\tau$. Formally, the query result $R = \pi(\{(x_i, \eta_i)|x_i \in S \land \eta_i \geq \tau\})$, where $\pi()$ randomly selects one feature when there are ties. For example, in Figure 1, the query $Q = (\text{max}, [\alpha, \beta])$ returns the result $R = \{(a, 6)\}$. To verify this result, we design Algorithm 2 for the client to check the following three conditions:

- Inflation checking (Lines 1–3): $R \subseteq S$.
- Deflation checking (Line 4): $(S - R) \cap R = \emptyset$.
- Completeness checking (Lines 5–8): $(S - R) \subseteq \tau \cdot (U - \hat{R})$, where $\hat{R}$ is the set version of multiset $R$, e.g., $R = \{(a, 6)\}$ and $\hat{R} = \{(a, 1)\}$. This is to prevent the SP from deliberately missing any feature whose multiplicity is larger than $\tau$.

**Algorithm 2 PA$^2$ Max ($\{R, X_1, \cdots, X_m\}$)**

1. Obtain $R, \text{acc}(X_1), \cdots, \text{acc}(X_m)$ from the SP (these $\text{acc}$ values are authenticated by the MG-tree, to be detailed in Section 5.3) and compute $\text{acc}(R)$ and $\text{acc}(\hat{R})$ locally;
2. Execute $\text{sum}(\{X_1, \cdots, X_m\})$ to get verified $\text{acc}(S)$;
3. Execute $\text{sub}(R, S)$ to get verified $\text{acc}(S - R)$; // implies $R \subseteq S$;
4. Execute $\text{empty}(S - R, R)$ to verify $(S - R) \cap R = \emptyset$;
5. Execute $\text{union}(\{X_1, \cdots, X_m\})$ to get verified $\text{acc}(U)$;
6. Execute $\text{sub}(R, U)$ to get verified $\text{acc}(U - \hat{R})$;
7. Execute $\text{times}(U - \hat{R}, \tau)$ to verify $\text{acc}(\tau \cdot (U - \hat{R}))$;
8. Execute $\text{sub}(S - R, \tau \cdot (U - \hat{R}))$ to verify $(S - R) \subseteq \tau \cdot (U - \hat{R})$.

In the running example, the sum set is $S = \{(a, 6), (b, 1), (c, 4), (d, 3), (e, 2)\}$ and the union set is $U = \{(a, 1), (b, 1), (c, 1), (d, 1), (e, 1)\}$. So the client needs to perform:

- Inflation checking: $\{(a, 6)\} \subseteq \{(a, 6), (b, 1), (c, 4), (d, 3), (e, 2)\}$;
- Deflation checking: $\{(b, 1), (c, 4), (d, 3), (e, 2)\} \subseteq \{(a, 6)\}$;
- Completeness checking: $\{(b, 1), (c, 4), (d, 3), (e, 2)\} \subseteq \{(b, 6), (c, 6), (d, 6), (e, 6)\}$.

The $\text{min}$ query is similar except that we verify $(S - R) \supseteq \tau \cdot (U - \hat{R})$ in the completeness checking.

The $\text{FFQ}$ query returns the features whose multiplicities are no lower than a threshold $\delta$. This query is naturally supported as it is a subroutine of the $\text{max}$ query (by replacing the threshold $\tau$ with $\delta$ in the completeness checking).

Similarly, the top-$k$ query can be verified by exploiting the routine of $\text{max}$ query and replacing the top-$1$ multiplicity $\tau$ with the top-$k$ multiplicity.

3. In the actual implementation, the deflation check can be omitted for max, FFQ and top-$k$ queries. This is because it is implied by the following completeness check.
indicating the candidate object selection. Without loss of generality, we assume that the objects are indexed by a grid structure on their non-sensitive attributes. Thus, we propose Merkle Grid tree (MG-tree), an authenticated data structure (ADS) for the DO to construct and sign. Note that our technique can be easily extended to other multi-dimensional index structures such as R-tree and k-d tree.

Figure 3a shows a multi-layer grid system, which partitions the space of non-sensitive attributes recursively into multiple levels of grid cells until each cell contains no more than two objects. The bounding box of each cell is called a grid box, and is denoted by \( gb \). Figure 3b shows the corresponding MG-tree for the objects in Figure 3a. Every node \( N_i \) corresponds to a non-empty grid cell. Its grid box is denoted by \( gb \), and has a digest (denoted by \( dig_i \)) that is computed from its \( C \) child entries, \( c_1, \ldots, c_C \), as follows:

**Definition 1 (Digest for Non-Leaf Node).** Let \( h(\cdot) \) be a cryptographic hash function and \( \cdot^* \) denote string concatenation. The digest of a non-leaf node is defined as:

\[
dig_i = h(gb_i | dig_{c_1} | \cdots | gb_{c_C} | dig_{c_C}).
\]

For example, in Figure 3b, the digest of node \( N_2 \), \( dig_2 = h(gb_0 | dig_6 | gb_7 | dig_7) \).

Now we define the digest of a leaf node based on the \( acc \) values of leaf entries.

**Definition 2 (Digest for Leaf Node).** The digest of a leaf node is defined as:

\[
dig_i = h(h(acc(X_i_1)) | \cdots | h(acc(X_i_{c_i}))),
\]

where \( acc(X_i) \) is the accumulative value of the feature set of object \( X_i \) (defined in Eq. (1)).

On the SP side, the processing of a range query \( [\alpha, \beta] \) starts from the root node. If a non-leaf node intersects the query range \( [\alpha, \beta] \), it will be branched, i.e., its subtree is further explored. On the other hand, if a leaf node intersects the query range \( [\alpha, \beta] \), all its entries will be returned as query results. Figure 3a illustrates an example where \( N_0, N_1, \ldots, N_{10} \) are nodes and \( o_1, o_2, \ldots, o_{12} \) are objects, i.e., leaf entries. Since the root node \( N_0 \) intersects with the range \( [\alpha, \beta] \), it will be branched and then non-leaf nodes \( N_2, N_4 \) are also branched. After that, leaf nodes \( N_7, N_9, N_10 \) intersect the query range, and all the leaf entries \( \{o_5, o_6, o_9, o_{10}, o_{11}, o_{12}\} \) are returned as results, i.e., the selected candidate objects for the next phase of aggregate query processing.

To guarantee the correctness of the selected candidate objects, the query authentication protocol on the MG-tree is as follows:

- The SP prepares a VO and sends it to the client. The VO includes: (i) the grid boxes of all intersected leaf nodes and the accumulative \( acc \) values of their leaf entries; (ii) the grid boxes and the digests of all other visited (but not intersected by the query) nodes; (iii) the signature of the root’s digest. For example, in Figure 3b, the VO (marked as grey) includes \( \{gb_5, dig_5, gb_6, dig_6, gb_7, acc(X_5), acc(X_6), gb_8, \ldots\} \).

4. In fact, the digest also includes \( h(acc(X_{i_1})) | \cdots | h(acc(X_{i_{c_i}})) \), where \( X_{i_1} \) is the set version of a multiset \( X_i \). These \( acc \) values are needed for the \( union(-) \) protocol in Section 5.1 but omitted for clarity of presentation.

5. For simplicity, in this paper we assume that the query range aligns with the grid boxes. Otherwise, the definition can be easily extended to include \( A_i \) in the digests of leaf nodes.
\[ \text{dig}_8, \text{gb}_9, \text{acc}(X_8), \text{acc}(X_{10}), \text{gb}_{10}, \text{acc}(X_{11}), \text{acc}(X_{12}), \text{sig}(\text{dig}_9) \].

- The client first verifies the correctness of the returned results, that is, the grid boxes of all the result nodes are intersected with the query range while all the others are not. Then, the client verifies whether all this information is genuine by restoring the root digest. Specifically, the client recursively rebuilds the digests of leaf nodes and non-leaf nodes up to the root level. Finally, the client checks whether the restored digest matches the signature from the DO. If so, the client verifies that all the returned information is genuine. For example, in Figure 3b, the client first verifies \( \text{gb}_7, \text{gb}_9, \text{gb}_{10} \) are intersected with the query range \([\alpha, \beta]\), while \( \text{gb}_5, \text{gb}_6, \text{gb}_8 \) are not. Next, the client rebuilds \( \text{dig}_7 = h(h(\text{acc}(X_5))| h(\text{acc}(X_8))), \text{dig}_9 = h(h(\text{acc}(X_9))| h(\text{acc}(X_{10}))), \text{dig}_{10} = h(h(\text{acc}(X_{11}))| h(\text{acc}(X_{12})) \), and then \( \text{dig}_1 = h(\text{gb}_8 \text{dig}_8), \text{dig}_2 = h(\text{gb}_9 \text{dig}_9) \), \( \text{dig}_3 = h(\text{gb}_8 \text{dig}_8) \), \( \text{dig}_4 = h(\text{gb}_9 \text{dig}_9) \). And finally, the client restores \( \text{dig}_9 = h(\text{gb}_9 \text{dig}_9) \) and checks whether it matches \( \text{sig}^{-1}(\text{dig}_9) \).

### 5.4 Cost Analysis

In this section, we analyze the performance of PA² framework. Specifically, we derive cost models for the privacy-preserving authentication protocols on multiset operations, the aggregate query authentication algorithms, and the MG-tree, respectively.

#### 5.4.1 Cost Model for Authentication Protocols on Multiset Operations

Table 3 summarizes the time costs for the SP and client, as well as the VO size for the five core privacy-preserving authentication protocols on multiset operations (Section 5.1). In this table, \( n \) is the total number of objects, \( t \) is the second operand for \( \text{times}(\cdot, \cdot) \), \( C_e \) is the time cost of a bilinear pairing operation, \( C_{gp} \) is the time cost of a multiplication operation in cyclic multiplicative group \( G \), and \( \tau \) is the second operand for \( \text{times}(\cdot, \cdot) \).

#### 5.4.2 Cost Model for Aggregate Query Authentication Algorithms

Based on the above analysis, we now derive the cost models for various aggregate queries. Let \( m \) be the number of candidate objects, \( |S| \) be the size of the sum set, \( |R| \) be the size of set \( R \), \( |X| \) be the size of the feature set for the \( i \)-th candidate object, and \( \tau \) be the top-\( k \)-th (top-1 for a max query) multiplicity.

**Count/Sum Aggregate Query.** The time cost for the SP is:

\[
C_{\text{count}}^S = C_{\text{sum}}^S + C_{\text{sub}}^S + C_{\text{empty}}^S = \left( \sum_{i=1}^m (i \cdot |X_i| + |S| - |R| \right) \]

\[+ \max(|S - R|, |R|)) \cdot \log_2(p) \cdot C_{gp}.\]

The time cost for the client is:

\[
C_{\text{count}}^C = C_{\text{sum}}^C + C_{\text{sub}}^C + C_{\text{empty}}^C + |R| \cdot \log_2(p) \cdot C_{gp} = (2m + 1) \cdot C_e + |R| \cdot \log_2(p) \cdot C_{gp}.
\]

The VO size, i.e., the communication cost, is:

\[
M_{\text{count}} = M_{\text{sum}} + M_{\text{sub}} + M_{\text{empty}} = (4m + 3) \cdot M_{\text{acc}}.
\]

**Max/Min/FFQ/Top-k Aggregate Query.** The time cost for the SP is:

\[
C_{\text{max}}^S = C_{\text{sum}}^S + 3C_{\text{sub}}^S + C_{\text{empty}}^S + C_{\text{union}}^S + C_{\text{times}}^S \]

\[= \left( \sum_{i=1}^m (i \cdot |X_i| + (m + 2 + 2\tau)|U| + \max(|S - R|, |R|) \right) \]

\[+ \max_{i=1}^n |X_i| - (1 + \tau)|R| - \sum_{i=1}^n |X_i| \cdot \log_2(p) \cdot C_{gp}.\]

The time cost for the client is:

\[
C_{\text{max}}^C = C_{\text{sum}}^C + 3C_{\text{sub}}^C + C_{\text{empty}}^C + C_{\text{union}}^C + C_{\text{times}}^C \]

\[+ (|R| + |R|) \cdot \log_2(p) \cdot C_{gp} = (3m + 6 + 2\log_2(\tau)) \cdot C_e + (|R| + |R|) \cdot \log_2(p) \cdot C_{gp}.
\]

The VO size, i.e., the communication cost, is:

\[
M_{\text{max}} = M_{\text{sum}} + 3M_{\text{sub}} + M_{\text{empty}} + M_{\text{union}} + M_{\text{times}} = (7m + 14 + 2\log_2(\tau)) \cdot M_{\text{acc}}.
\]

A key observation from the above analysis is that both the client time cost and the VO size are determined only by the result set size and are independent of the number of sensitive features.

#### 5.4.3 Cost Model for Candidate Object Selection

We next analyze the cost of the candidate object selection. Our analysis starts with the MG-tree size, based on which we derive the VO construction and verification costs for a range selection. Without loss of generality, the data space is a \( d \)-dimensional unit space \([0, 1]^d\), and we assume each non-sensitive attribute independently follows a uniform distribution. As such, the MG-tree partitions each dimension equally and the height of the tree can be approximated as \( w = \lceil \log_2(\frac{d}{f}) \rceil \), where \( n \) is the total number of objects and \( f \) is the minimum number of objects in a leaf node.

**MG-tree size.** Each tree node stores a grid box (represented by two \( d \)-dimensional corner points) and a digest. As such, the size of a node, in terms of bytes, is:

\[
M_N = 2 \cdot d \cdot 4 + M_h,
\]

where \( M_h \) is the size of a hash value.

Besides the tree nodes, the server stores all leaf entries (i.e., objects). Let \( M_{\text{acc}} \) denote the size of an accumulative value; then the total size of an MG-tree is:

\[
M_{\text{MG}} = 2 \cdot n \cdot M_{\text{acc}} + \sum_{i=0}^{w-1} \min(n, 2^{i \cdot d}) \cdot M_N.
\]

**Construction cost of VO.** According to [37], the probability that two random rectangles \( R_1, R_2 \) overlap is:

\[
Pr_{\text{overlap}}(R_1, R_2) = \prod_{j=1}^{d} (R_1.L_j + R_2.L_j),
\]

where \( R_1.L_j \) is the length of \( R_1 \) in dimension \( j \). Since both the MG-tree partition and object distribution are uniform, the lengths of an intermediate entry at level \( i \) and a leaf entry are \( 2^{-i} \) and \( \sqrt[4]{k}/n \), respectively. Putting them into Eq. (2), the numbers of visited tree nodes \( N_n \) and visited leaf entries \( N_e \) are:

\[
N_n = \sum_{i=0}^{w-1} 2^i \cdot d \cdot \prod_{j=1}^{d} (2^{-i} + Q.L_j),
\]

\[
N_e = n \cdot \prod_{j=1}^{d} \sqrt[4]{\frac{k}{n}} + Q.L_j,
\]

respectively. Let \( C_\varnothing \) denote the time cost of accessing a single entry or a single node. The cost of constructing the VO is modeled as:

\[
C_{\text{pre}} = (N_n + N_e) \cdot C_\varnothing.
\]
6 SECURITY ANALYSIS

In this section, we perform a security analysis on our PA² algorithms for aggregate queries.

6.1 Security of PA² Algorithms

The correctness of our PA² algorithms is defined in the natural way and is omitted. For soundness, it can be formally defined as follows:

Definition 3 (Soundness). The PA² authentication algorithms for aggregate queries are sound if for all PPT adversaries, the success probability is negligible in the following experiment:

- The adversary picks a dataset \( D \).
- Run the ADS generation on \( D \) and forward to the adversary.
- The adversary outputs a query \( Q \), a result \( R \), and a VO.

We say the adversary succeeds if the VO passes the result verification and \( R \neq Q(D) \).

We now show that the PA² authentication algorithms indeed satisfy the desired security requirements.

Theorem 1. The PA² authentication algorithms for aggregate queries guarantee the correctness and soundness of the query results under the bilinear \( q \)-strong Diffie-Hellman assumption.

Proof. The theorem is proved by proving the security of each multiset operation: \( sub(\cdot, \cdot), empty(\cdot), sum(\cdot), union(\cdot), \) and \( times(\cdot, \cdot) \). See Appendix A in the Supplemental Material for more detailed proofs.

6.2 Privacy Guarantee on Sensitive Features

Next, we analyze the privacy guarantee on the sensitive features in our PA² framework. Specifically, we show that the accumulative value of a feature set does not disclose any feature information to the client. Formally, it is stated as follows:

Definition 4 (IND-CPA). The augmented accumulative value \( acc(X) \) for a multiset \( X \) is indistinguishable if for all PPT adversaries, there exists a negligible probability \( neg(q) \), s.t.

\[
Pr[\text{Adv} \rightarrow \{X_1, X_2, st\}; b \rightarrow \{0, 1\}] = \frac{1}{2} + neg(q).
\]

Before showing the indistinguishability of accumulative values, we start with the following lemma on the group \( \mathbb{G} \).

Lemma 1. For any random value \( r \leftarrow \mathbb{Z}_p \), \( g^r \) has an equal probability of being any element in \( \mathbb{G} \) (with an order of \( p \)). Formally, for any \( \hat{g} \in \mathbb{G} \),

\[
Pr[g^r = \hat{g}] = 1/p.
\]

Proof. Let \( \log_g() \) denote the discrete logarithm of base \( g \) in group \( \mathbb{G} \). We have

\[
Pr[g^r = \hat{g}] = Pr[r = \log_g(\hat{g})] = 1/p.
\]

Now we show a theorem on the security of accumulative values.

Theorem 2. The augmented accumulative value \( acc(X) \) for a multiset \( X \) is indistinguishable under chosen plaintext attack (CPA).

Proof. Since the augmented accumulative value \( acc(X) = g^p(X) = (g^p)^{rx} = g^{p(X) \cdot rx} \), according to the above lemma, \( acc(X) \) has an equal probability of being any element in \( \mathbb{G} \). As such, the verifier learns nothing about \( p(X) \) and any element \( x_i \in X \).

Further, since \( rx \) is random for each \( X \), \( acc(X) \) is indistinguishable under CPA.

It is worth noting that by achieving indistinguishability on a single accumulative value, multiple accumulative values are automatically guaranteed to be indistinguishable. Therefore, the above theorem effectively proves the security of the authentication algorithms developed for various primitive aggregate queries, as only (indistinguishable) accumulative values are disclosed for the sensitive features throughout the authentication process, which protects the confidentiality of source data.

6.3 Discussion on Privacy Guarantee

It is noteworthy that during the authentication of aggregate queries, the results of primitive aggregate queries are disclosed to the client, which might be a source of privacy leak. In an extreme case, if there is only one candidate object in the selection phase, the query result could directly reveal the features of that object. Further, the difference of two query results could reveal the features of the difference of the two candidate object sets. As more aggregate queries are issued and each may consist of multiple primitive aggregate queries, it becomes easier to find smaller difference of candidate object sets. To mitigate such privacy leak, inspired by the bucketization approach [38], we impose a minimum granularity on the MG-tree leaf nodes for the non-sensitive attributes. Specifically, when building the MG-tree, the DO enforces each leaf node (i.e., a bucket) to contain a minimum number of objects that can satisfy a given privacy metric (e.g., \( k \)-anonymity or \( t \)-closeness [39, 40]). As such, the data space
is partitioned into disjoint buckets, and upon query processing, a bucket is the smallest granule for range selection. With this enforcement, no query result can reveal whether it is contributed by a specific object in a bucket, and the latter becomes the smallest granule of result privacy leak, no matter how many aggregate queries are issued. Formally, the results of aggregate queries and the associated authentication process achieve the following bucket-wise indistinguishability privacy:

**Definition 5 (Bucket-wise Indistinguishability).** Given a bucket $B$ and an object $o \in B$, if an aggregate query result $R$ is from a candidate object set $O \supseteq B$, then any adversary can determine whether $o$ contributes to $R$ with a success rate no higher than that of a random guess.

### 7 Optimizations

This section introduces three optimization techniques for our PA² authentication framework. They are orthogonal to each other, and therefore can be combined to further boost the performance.

#### 7.1 Optimized MG-Tree

One performance bottleneck is that the SP has to use the costly exponentiation operation rather than modular exponentiation to compute the accumulative values because it does not know the secret value of $s$. To alleviate this, we let the DO pre-compute and store some intermediate accumulative values in the MG-tree for the SP. For example, in Figure 3, to authenticate the query $Q = (\text{max}, \angle, [\alpha, \beta])$, the VO needs to compute the accumulative $S$ value of $S = X_5 \cup X_6 \cup X_9 \cup \cdots \cup X_{12}$. To do so, the SP has to compute five intermediate accumulative values in turn: $\text{acc}(X_5 \cup X_6), \text{acc}(X_5 \cup X_6 \cup X_9 \cup \cdots \cup X_{12})$. However, if the DO has already computed $\text{acc}(S_2)$ and $\text{acc}(S_4)$ for the two intermediate nodes $N_7$ and $N_4$, the SP can simply compute $\text{acc}(S) = \text{sum}(\{S_7, S_4\})$.

As the DO knows the value of $s$, computing such intermediate accumulative values is efficient. Nonetheless, to alleviate the SP’s storage cost, we only compute and store the accumulative values of $\text{sum}$ and $\text{union}$ operations as they are the most time-consuming operations. To accommodate these intermediate values in the MG-tree, we redesign the digest for non-leaf nodes as follows:

**Definition 6 (Optimized Digest for Non-Leaf Node).** Let $C$ be the number of child entries of node $N_i$, $S_i$ be the sum set of child entries’ union sets, and $U_i$ be the union set of child entries’ union sets; the digest of node $N_i$ is defined as:

$$
\text{dig}_i = h(g_{\text{do}}, h(\text{acc}(S_i)|\text{acc}(U_i))|\text{dig}_c) \cdots \text{| \ s | | g_{\text{do}}, h((\text{acc}(C_c)|\text{acc}(U_c))|\text{dig}_c)
$$

Based on this optimized MG-tree, the query $Q = (\text{max}, \angle, [\alpha, \beta])$ no longer needs to traverse down to the child nodes if the current tree node is fully contained in the query range $[\alpha, \beta]$. As such, the candidate object selection and PA² algorithms on the optimized MG-tree are revised as follows:

- The SP prepares the VO, which includes (i) the grid boxes, the accumulative values of sum and union sets, and the digests of those fully contained tree nodes; (ii) the grid boxes, the hash values for the accumulative values of sum and union sets, and the digests of all boundary nodes; (iii) the signature of the tree root’s digest. For example, in Figure 4, for the same query $Q = (\text{max}, \angle, [\alpha, \beta])$, the VO (marked as grey) includes $\{g_{\text{do}}, h(\text{acc}(S_i)|\text{acc}(U_i))|\text{dig}_c\}, g_{\text{do}}, h(\text{acc}(S_i)|\text{acc}(U_i))|\text{dig}_c, g_{\text{do}}, h(\text{acc}(S_i)|\text{acc}(U_i))|\text{dig}_c, g_{\text{do}}, h(\text{acc}(S_i)|\text{acc}(U_i))|\text{dig}_c, g_{\text{do}}, h(\text{acc}(S_i)|\text{acc}(U_i))|\text{dig}_c$.

- The client still verifies the VO by reconstructing the root digest of the MG-tree recursively. First, the client verifies that the grid boxes are fully inside or outside the query range. Then, the client rebuilds the digest for each node. Finally, the client checks whether the restored root digest matches the signed one. For example, the client verifies $g_{\text{do}}, h(\text{acc}(S_i)|\text{acc}(U_i))|\text{dig}_c$ for the two intermediate nodes.

- Next, the client proceeds to the PA² algorithms with input $\{R, S_1, U_1, \cdots, S_m, U_m\}$. The difference is that the accumulative values of sum and union sets are computed from the intermediate sum and union sets. That is, for a sum operation, the client computes $\text{acc}(S) = \text{sum}((S_1, \cdots, S_m))$ instead of $\text{sum}((X_1, \cdots, X_m))$. Similarly, for a union operation, the client computes $\text{acc}(U) = \text{union}((U_1, \cdots, U_m))$ instead of $\text{union}((X_1, \cdots, X_m))$.

Regarding the performance, since the number of subsets to compute the accumulative values from is reduced from $m$ to $\log(m)$, the SP’s computation, the client’s computation, and the communication complexity of computing the sum set are reduced from $O(m^2)$, $O(m)$ to $O(m)$, $O(\log(m))$, respectively. Similarly, the costs of computing the union set are reduced from $O(m \cdot |S| + |U|)$, $O(m)$ to $O(\log(m)) \cdot (|S| + |U|)$, $O(\log(m))$, respectively. As computing the $\text{sum}$ and $\text{union}$ sets dominates the total computation and communication complexity of their respective queries, we expect the overall performance gain is significant.

#### 7.2 Accumulating $\text{sum}(\cdot)$ by Linear Ordering

For the $\text{sum}(\cdot)$ operation, it has a nice additive property that $\text{sum}(\{X\}) + \text{sum}(\{Y\}) = \text{sum}(\{X \cup Y\})$. Therefore, we can ask the DO to pre-compute $\text{sum}$ accumulative values in an accumulative manner. Let us consider the 1D case for simplicity and sort the objects in ascending order. For each $i \in [1, n]$ ($n$ is the number of objects), the DO pre-computes, $\text{sum}_{1-i}$, the sum accumulative value for the objects from $o_1$ up to $o_i$. Then, for any range query $[\alpha, \beta]$ that selects the objects from $i$ to $j$, the sum value $\text{sum}_{i-j}$ can be computed as $\text{sum}_{i-j} = \text{sub}(\text{sum}_{1-(i-1)}, \text{sum}_{1-j})$. For example, in
Regarding the performance, similar to the optimized MG-tree, since the DO knows the secret value of $s$, the pre-computation is very efficient. As for the query cost, let $m$ denote the number of candidate objects; both the client’s computation and communication complexity will be reduced to $O(1)$. While the computation complexity for the SP is still $O(m)$, the constant ratio will be greatly reduced with fewer $sum(\cdot)$ operations. Note that for multidimensional space, we can still enforce linear ordering on the objects and adopt this optimization, by using space embedding techniques such as the Hilbert Curve. Nonetheless, the performance gain is less significant because $sum_{i-j}$ might not be the exact objects selected by a range and some further $subset$ operations would be needed.

### 7.3 Acceleration by Parallelism

Both the multiset authentication protocols and PA$^2$ algorithms are highly parallelizable. As such, we can accelerate the authentication performance by embracing parallel computing architectures, such as multithreading, GPU computing, and MapReduce. For example, in the $union(\cdot)$ protocol, we can map all $sub(\cdot, \cdot)$ jobs to the available computing units, whether they are CPU cores or worker nodes in MapReduce. Similarly, in the $empty(\cdot)$ protocol, the server can compute the accumulative values of polynomials $Q_i$ in a parallel manner. Note that, however, in the $sum(\cdot)$ and $times(\cdot, \cdot)$ protocols, the order of polynomials is accumulated and therefore the time cost of each parallel job may vary significantly. As such, instead of splitting all jobs equally at the beginning of the query, we schedule a small set of jobs at a time. With this graceful scheduling technique, we can minimize the waiting time of idle computing units while still reducing the job scheduling overhead.

### 8 Performance Evaluation

In this section, we evaluate the performance of PA$^2$ framework for aggregate queries over set-valued data, including the $max$, $top-k$, $sum$, and $FFQ$ queries. Three datasets are used in the experiments, namely Personal Genome Project at Harvard Medical School (PGP) [2], Foodmarket from Microsoft (FoodMarket) [41], and TPC Benchmark H (TPC-H) [42]. As introduced in Section 1, the PGP dataset contains personal genome data of 600 participants. The objects in this dataset are in the form of $<latitude, longitude, mut-genes>$, where mut-genes is a feature set of mutation genes of an individual, e.g., ‘A-C130R’, ‘P-I696M’). There are totally 395 unique mutation genes, and on average each individual has 35 of them. The FoodMarket is a larger dataset that contains 164,558 shopping transaction records from 8,842 users and 1,560 products, in the form of $<birthday, membershipAge, items>$. Finally, the TPC-H dataset contains 1,020, 116 transaction records from 255,000 orders and 1,700 suppliers, in the form of $<order_priority, date, suppliers>$. In all these three datasets, the first two attributes are non-sensitive, whereas the last attribute is sensitive. We use the FoodMarket as the default dataset unless otherwise stated. For the experiments requiring higher dimensionality, we also synthesize some attributes into this dataset.

Both the DO and the query client are set up on a commodity laptop computer with Intel Core i5 CPU and 4GB RAM, running on macOS Sierra. The SP is set up on an x64 blade server with dual Intel Xeon 2.2GHz E5-2630 CPU and 256GB RAM, running on CentOS 6. The experiments are written in C++ and use the following libraries: Pairing-based cryptography for bilinear pairing computations, Flint for modular arithmetic operations, Crypto++ for 160-bit SHA-1 hash operations, OpenMP for parallel computation.

To evaluate the performance of the authentication process, we mainly measure two costs: (i) the computational cost in terms of the SP and client CPU time, and (ii) the communication cost in terms of the size of data transmitted (a) between the DO and the SP (i.e., the index size) and (b) between the SP and the client (i.e., the VO size). The results are reported based on an average of 10 randomly generated operations/queries.

### 8.1 Performance of Privacy-Preserving Multiset Authentication Protocols

We first use two synthetic multisets to evaluate the performance of the five privacy-preserving authentication protocols, i.e., $sub(\cdot, \cdot)$, $sum(\cdot)$, $union(\cdot)$, $empty(\cdot)$, and $times(\cdot, \cdot)$ in terms of (i) the SP CPU time, (ii) the client CPU time, and (iii) the VO size. We fix the coefficient for $times(\cdot, \cdot)$ to 10, and vary the number of features from 100 to 10,000. The SP CPU time (single-threaded) is reported in Table 4. As coinciding with our cost model analysis, the SP CPU time is almost linear to the number of features. In contrast, the client CPU time and VO size are all constants, irrespective of the number of features.

We further investigate the SP CPU time of $sum(\cdot)$, $union(\cdot)$, $empty(\cdot)$ for more than two input multisets. We fix the number of features in each multiset to 20 and vary the number of multisets. Since the client time and VO size are both linear to the number of input multisets, Table 5 only shows the SP CPU time. For the $sum(\cdot)$ and $union(\cdot)$ operations, the cost is almost quadratic to the number of multisets, which coincides with our analysis in Table 3. For $union(\cdot)$, we also investigate whether the cardinality of feature universe affects its performance. Table 6 shows the
8.2 Query Authentication Performance

We turn on all the optimizations and evaluate the overall performance of privacy-preserving query authentication on all three datasets. First of all, Table 7 shows the DO setup cost for generating the acc values and constructing the MG-tree. As explained in the cost model analysis, since the DO knows the secret value of s, the setup process is relatively efficient.

To select the candidate objects, we choose the participants’ location attributes in PGP, the customers’ birthday and membership age in FoodMarket, and the order_priority and date in TPC-H as the input attributes for 2D range queries. The aggregate queries, max, top-k, sum, and FFQ, are then performed on the feature sets of the selected candidate objects. For top-k queries, we set k = 5; and for FFQ queries, we set the threshold to be 80% of the highest multiplicity of all features. The experiments are conducted under a multi-threaded setting (40 hyper-threads on 20 CPU cores). The results are shown in Figure 6, Figure 7, and Figure 8 respectively, where the query range varies from 1% to 20% of the data space. It is observed that all the costs are generally sublinear to the query range, thanks to the optimized MG-tree that significantly reduces the multiset operations and accumulative value computations. It is interesting to note that the cost of client CPU time is relatively higher for FFQ queries on TPC-H dataset. This is caused by the large multiplicity of the returned multiset result, which incurs more overhead when computing the result acc value on the client. Nevertheless, the consistent results on the three different datasets suggest that our algorithms are robust against various data distributions.

For FFQ queries, we also compare our proposed privacy-preserving authentication algorithm with the state-of-the-art nonprivacy-preserving counterpart proposed in [28], denoted by FFQ-NP. This comparison intends to show the overhead incurred by the confidentiality preservation requirement. We select 1% of the PGP data as input and set the maximum size of frequent feature sets to 3. As shown in Table 8, the overhead is moderate in that our FFQ is only tens of times worse than FFQ-NP in terms of the DO index size and SP CPU time. The VO size of our FFQ is even smaller than that of FFQ-NP. Another interesting observation is that the SP CPU time is positively correlated to the threshold δ in our algorithm, but negatively in FFQ-NP. The threshold times operation involved in FFQ, whose input is the infrequent feature sets, becomes more costly with increasing threshold. In comparison, the SP CPU time of FFQ-NP is mainly affected by the number of frequent feature sets, which decreases with a larger threshold. As FFQ queries are often used in the early stage of the whole data analytics pipeline, their threshold is usually set to a low or medium value, which falls in the comfort zone of our algorithm.

8.3 Impact of Optimizations

To evaluate the impact of the proposed optimizations, namely, the optimized MG-tree (OMG) and parallelizing techniques, we use the FoodMarket dataset and set the selectivity of candidate objects to 5% of the whole dataset. Their effects for different aggregate queries are shown in Figure 9. It can be seen that both the OMG-tree and multithreading techniques substantially reduce the SP CPU time by one order of magnitude. In Figure 10, we further vary the number of threads in parallelization. We observe that the SP can be effectively accelerated by up to 30–40 threads, only bounded by the number of logical CPUs of our server. As we show the computation of the SP is highly parallelizable, we believe its performance can be further accelerated by clusters or cloud IaaS widely available in today’s data centers.

We also evaluate the performance of the linear ordering technique in Figure 11. We compare it with the baseline sum under one-dimensional data. The results show that it dramatically improves the performance by more than four times. In addition, as the selectivity increases, the performance gain becomes even more eminent.
8.4 Scalability Test on Dimensionality and Object Cardinality

We investigate the impact of dimensionality on the authentication performance using the FoodMarket dataset. We fix the selectivity of candidate objects to 5% of the whole dataset and vary the dimensionality as 1, 3, and 5. For the latter two dimensionalities, we synthesize the non-sensitive attributes based on a uniform distribution and therefore the size of the range query changes accordingly. Figure 12 plots the costs of various aggregate queries. We observe that the performance of max, top-k, and FFQ queries deteriorates severely due to the curse of dimensionality, while the performance of sum queries does less. This is because union(·), the most affected operation by dimensionality, is not needed in sum queries.

Finally, Figure 13 shows our scalability test with respect to the cardinality of objects. The results show that the SP CPU time increases monotonically for sum, but is U-shape for other types of queries. This is because the other queries involve not only sum(·) operations but also union(·) operations, whose costs decrease with increasing object cardinality under a fixed number of unique features. On the other hand, the CPU time of the client is independent of total number of features, but is positively related to the multiplicity of the returned multiset result. This also causes a relative high cost in client CPU time for FFQ queries, similar to what we have observed in query performance on TPC-H dataset. As for the VO size, it is almost constant as it only depends on the number of OMG-tree nodes that cover the selected candidate objects.

9 Conclusions

In this paper, we have studied the problem of privacy-preserving authentication of aggregate queries over set-valued data. We presented a family of privacy-preserving protocols on multiset operations. Based on that, we proposed PA2, a framework that supports various privacy-preserving aggregate queries. We also designed the MG-tree to guarantee the correctness of the candidate objects selected by a range query and developed several advanced optimization techniques. Analytical models and empirical results validate the feasibility and robustness of our proposals. In particular, the overhead of authenticating aggregate queries at the client side is only sublinear to the number of candidate objects and is independent of the cardinality of multisets. The evaluation results also reveal that the authentication overhead incurred by the confidentiality preservation requirement is only moderate.

As for future work, we plan to extend the proposed privacy-preserving authentication techniques to more complex aggregate queries such as median and percentile. We also plan to leverage GPGPU technologies to accelerate the computation of accumulative values during query processing at the SP.

Acknowledgement

This work was supported by Research Grants Council (RGC) of Hong Kong under GRF Projects 12244916, 15238116, 12202414, 12200114, 12200914, CRF Project C1008-16G, and NSFC Grant 61370231.

References


Cheng Xu is a PhD student in the Department of Computer Science, Hong Kong Baptist University. He received his BEng degree in Computer Science from Huazhong University of Science and Technology in 2011. His research interests include database security and privacy-aware data management.

Haibo Hu is an Assistant Professor in the Department of Electronic and Information Engineering, Hong Kong Polytechnic University. Prior to this, he has held academic positions in HKUST and HKBU since he received his PhD degree from HKUST in 2005. His research interests include information security, privacy-aware computing, wireless data management, and cloud technology and engineering. He has published over 60 research papers in refereed journals, international conferences, and book chapters.

As pricipal investigator, he has received over 5 million HK dollars of external research grants from Hong Kong and mainland China. He is the Panel Co-chair of DASFAA 2011, and Program Co-chair of DaMEN 2011, 2013 and CloudDB 2011. He is also the recipient of a number of awards, including ACM-HK Best PhD Paper Award, Microsoft Imagine Cup, and GS1 Internet of Things Award.

Jianliang Xu is a Professor in the Department of Computer Science, Hong Kong Baptist University. He received the BEng degree from Zhejiang University and the PhD degree from Hong Kong University of Science and Technology. He held visiting positions at Pennsylvania State University and Fudan University. His current research interests include big data management, database security and privacy, and location-aware computing. He has published more than 150 technical papers in these areas.

He served as a program co-chair/vice chair for a number of major international conferences including IEEE ICCDCS 2012, WAIM 2016, and APWeb-WAIM 2018. He is an Associate Editor of IEEE Transactions on Knowledge and Data Engineering (TKDE) and PVldb 2018.

Xiaojun Hei received the B.Eng. degree in information engineering from the Huazhong University of Science and Technology, Wuhan, China, in 1998, the M.Phil. degree in electrical and electronic engineering from The Hong Kong University of Science and Technology, Hong Kong, in 2000, and the Ph.D. degree from the Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology in 2008. Since 2008, he has been with the Information Technology and Engineering Research and Development Center, School of Electronic Information and Communications, Huazhong University of Science and Technology. He is currently an Associate Professor with the School of Electronic Information and Communications, Huazhong University of Science and Technology.