VERID: Towards Verifiable IoT Data Management

Xin Li
University of California Santa Cruz
xinli@ucsc.edu

Minmei Wang
University of California Santa Cruz
mwang107@ucsc.edu

Shouqian Shi
University of California Santa Cruz
sshi27@ucsc.edu

Chen Qian
University of California Santa Cruz
cqian12@ucsc.edu

ABSTRACT

Ensuring the authenticity and integrity of the sensing data that are stored in a third-party cloud is a crucial task for the correctness and safety of many IoT applications. Although verifiable data outsourcing has been studied for over a decade, current solutions are not fully suitable for IoT systems, due to the hardware constraints, deployment features, and application requirements of IoT. This paper presents VERID, a verifiable data management system designed for IoT applications. VERID enables important ranged selection and aggregate queries of sensing data while imposing minimal overhead for resource-constraint IoT devices. Our important innovation is a computational and space-efficient authentication data structure called PrefixMHT which fits into resource-constrained IoT devices and supports both range and aggregate queries. We design a new signature aggregation scheme called Condensed Bilinear Pairing to further improve the efficiency. The experiments using real IoT datasets show that VERID is able to provide authenticity, integrity, and completeness of data queries while achieving substantial advantages in computation, memory, and communication efficiency than possible methods.

CCS CONCEPTS

• **Information systems → Data management systems**; • **Security and privacy → Security protocols**; • **Networks → Cloud computing**; **Cyber-physical networks**.

KEYWORDS

the Internet of Things; Verifiable Data Outsourcing

1 INTRODUCTION

The Internet of Things, or IoT, is gaining increasing public attentions and continuously reshaping the world in various aspects [20, 30, 34, 35, 57, 58]. Equipped with sensors, large groups of IoT devices perceive the physical environment or monitor conditions of target objects or human beings and therefore generate massive sensing data during their operation. Even though IoT devices differ vastly in external form, ranging from tiny wearables to larger industrial devices, most of them are embedded devices with limited cost, resources and size. Storing the sensing data at a cloud is the most popular paradigm for IoT data management as adopted by both recent proposals [33, 34, 41] and industrial practices [5]. The data consumers such as intelligent analytics programs retrieve data from the cloud to make decisions accordingly. They might also be IoT devices. With tampered or erroneous data, IoT applications may make wrong decisions and cause economic and even human-life losses [55]. The sensing data are stored in a third-party cloud, which may not be fully trusted. The query result could be corrupted by outside attackers, malicious cloud employees [42], transmission failures, or storage loss [3]. According to a survey of 1400 IT decision-makers conducted by McAfee [10], 25% cloud tenants have experienced data theft from the public cloud and 20% cloud tenants have experienced an advanced attack against their public cloud infrastructure.

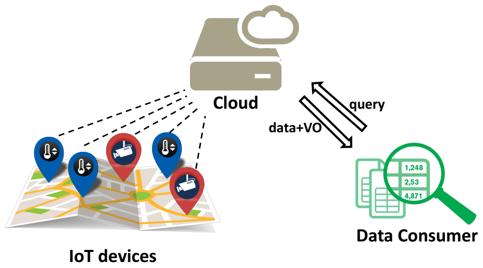


Figure 1: Overview of IoT data framework in VERID

The canonical IoT-cloud communication model is shown in Fig. 1. Hence a critical task of IoT data management is to allow the data consumers to verify the **correctness** of the sensor data retrieved from the cloud. The "correctness" here includes two requirements: 1) **Authenticity and integrity**: the data should be collected from the sensing devices and not be tampered by any third party; 2) **Completeness**: the data consumer should receive all and only the data satisfying the conditions in its query. An IoT data management method is **verifiable** if a data consumer can verify the correctness of the received data.

The general problem of verifiable database outsourcing has been studied for over a decade [15, 21, 36, 39, 48, 64, 65]. A common approach is that a data publisher also uploads an *Authentication Data Structures* (ADS) to the cloud and keeps updating it. An ADS is a data structure signed by the IoT device using its private key. When

IoTDI '19, April 15–18, 2019, Montreal, QC, Canada
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6283-2/19/04…$15.00
https://doi.org/10.1145/3302505.3310074

a *Data Consumer* (DC) receives the data from the cloud, it also gets a number of *Verification Objects* (VOs) which are constructed by the cloud from the ADS and can be used to verify the data correctness. However, existing solutions are not fully suitable or optimized for the IoT. Existing work mainly focuses on general-purpose resource-rich platforms such as servers with infrequent updates.

We identify the communication pattern, the hardware constraints, deployment features and application requirements of IoT that demand a new design of verifiable data management.

Append-only updates. IoT devices send sensing data to the cloud for storage. Therefore, appending is the only operation needed for updating. Without considering modifying or deleting history data, the design of ADS could be optimized towards append-only updates.

Computation efficiency. IoT devices are usually limited in computation power. In addition energy efficiency may be another top concern for them. Hence solutions based on computation-intensive cryptographic operations [8, 64, 65] are not appropriate for IoT applications. Even though the performance of IoT CPUs increases significantly over the years, public crypto operations are still slow from one recent field test [41] which showed that it takes the IoT device, which is built atop one popular IoT platform with ARM Cortex@72MHz, 0.1 second on average to finish one 1024-bit RSA encryption operation.

Memory efficiency. IoT devices are constrained by memory capacity as well. Even crypto operations themselves do not consumer too much memory space, the data structure to support verifiable queries do! Solutions based on multi-way trees [21, 39] are not efficient on disk-less IoT devices with limited available memory. For dynamic database outsourcing schemes using memory-friendly ADSes [48, 65], these ADSes however grow fast with #update. When the ADS exceeds memory limit, it will be signed and then flushed to the cloud. In this case a query may result in a jumbo VO constructed from an overwhelming number of ADSes.

Communication efficiency. Communication of IoT devices is often more power-consuming than computation by orders of magnitude [27, 28]. Therefore, the sizes of updates from IoT devices to the cloud is an important metric. Multiway-tree based schemes such as AAR-tree [39] configure the ADS node size to that of a page. When the IoT device updates its ADS, entire stale nodes (nodes different from the previous version) are transmitted to the cloud even with a small modification, which are in large size.

Multiple data publishers. Multiple homogeneous IoT devices being deployed to collectively monitor the physical environment is an unique feature of IoT applications. These homogeneous IoT devices form a *task group*. For example, one dataset [13] provides the outdoor temperature of areas in Rome collected by 289 taxicabs over 4 days.

We summarize our contributions in this paper as follows.

1) We design and implement a verifiable IoT data management system, VERID, considering all requirements of IoT discussed above. **It is a holistic design taking into account verifiable data communication, storage and verification altogether.**

2) Our important innovation is a computational and space-efficient ADS called PrefixMHT which fits into resource-constrained IoT devices and supports both range and aggregate queries. PrefixMHT is optimized for append-only updates.

3) We build an efficient data management system in the cloud. It novelly exploits spatial locality among IoT devices to reduce disk I/O.

4) We design a signature scheme, CBP-VERID, to significantly reduce both communication and computation costs especially in the sparse setting. Even though CBP-VERID are not fundamentally different from other pairing-based signature schemes in terms of cryptography, CBP-VERID are specially designed to fit the needs of sparse IoT applications. To the best of our knowledge, CBP-VERID is the first signature scheme that enables signature aggregation in both spatial and temporal dimensions.

5) We also investigate the problem of shared budget constraint for a group of IoT devices and extend VERID to resolve this problem. It is a novel and important problem that no existing data outsourcing solutions have consider it.

The rest of this paper is organized as follows. The related work is discussed in Sec. 2. We present the problem statement in Sec. 3. We describe the system design details in Sec. 4. We present the digital signature scheme and its security and cost analysis in Sec. 5. The experimental results are presented in Sec. 6. Further discussions are presented in Sec. 7. Sec. 8 concludes this paper.

2 RELATED WORK

Verifiable database outsourcing have been studied for over a decade and two broad categories of approaches are exerted towards this goal. General verifiable delegation of computation can handle any query on outsourced data. Circuit-based solutions [8, 17, 23, 26, 29, 53] require the data owner to compile the entire dataset into an arithmetic circuit. Circuit-based systems incur excessive proof construction overhead. A very recent work vSQL [64] improves the performance of this approach by combining an information-theoretic interactive proof system [25] and a polynomial-delegation protocol [50]. However, the overhead of vSQL is still high for practical uses. Another line of research efforts [12, 52] to realize general verifiable computation over authenticated data are based on homomorphic signature, which are also of theoretical interests only.

On the other side, numerous prior works aim at verifying one or multiple specific data query types, including range query [15, 21, 48, 49, 65], data aggregation [36, 39], join [48, 62, 66], search over encrypted data [61], *etc.*. VERID falls into this category. We conduct the following literature review of the methods in the series of work and use Table 1 to show an qualitative comparison between representative database outsourcing schemes.

Chained Signature Approach [48]. The ADS is an authenticated and unforgeable linked list ordered by one dimension (such as time, temperature) over the dataset where each node contains the cryptographic hashes of its predecessor and successor. At query execution, all nodes falling in the query range are lined up to form the VO. The data consumer verifies authenticity and completeness of the results by sequentially checking the signatures of the nodes in the VO. For data publishing, the newly inserted node along with its two neighbors are updated, re-signed and then uploaded to the cloud by the data publisher. Chained signature approaches perform three signing operations per data insertion which would be inefficient on IoT devices. This approach does not support aggregate queries such as SUM or multi-dimensional queries.

Table 1: Qualitative comparison of representative database outsourcing schemes. ○: efficient; ●: inefficient; ◐: inefficient in some situations/metics.

Scheme	Category	Memory	Computation	Log ADS update	Multi-dimension	Comments
BAS [48]	Signature-based	◐	○	✓	✗	inefficient for aggregate
APS-tree [36]	Prefix Sum	○	◐	✓	✓	coarse granularity
IntegriDB [65]	Set operation	●	●	✓	✓	crypto heavy
vSQL [64]	Circuit-based	○	●	✓	✓	computational intensive
AAR-tree [39]	Multiway-tree	◐	○	✓	✓	large VO size
VKD-tree [21]	Binary-tree	◐	○	✗	✓	unbalanced tree
CorrectDB [15]	Hardware-aid	○	○	✗	✗	trusted hardware
VERID (this work)	Binary-tree	○	○	✓	✓	holistic design for IoT

Prefix Sum. APS-tree [36] uses Prefix Sum for efficient validation of aggregate operations, specially on SUM. The basic idea is to pre-process the data at the data publisher’s side such that the aggregated value could be easily assembled by those pre-processed values. Compared to signature-based approaches, Prefix Sum reduces the communication cost of aggregate queries to $O(1)$. Prefix Sum however suffers from inefficient update: A single update may trigger conducting pre-processing over the whole dataset in the worst case.

Authenticated Set Operations. Some prior works [49, 65] achieve authentication of set operations including *union*, *intersection*, and *set-difference*, which are powerful building blocks to compute multi-dimensional range queries. However they are very inefficient in computation. Each data publisher needs 1) computing q exponents with up to the s^qth powers where s is secret value and q is a big integer, and future ranged queries are limited to those whose results have cardinality less than q; 2) $O(\log |r|)$ encryptions per insertion ($|r|$ denotes #rows of the relation).

Tree-based Approaches. Tree-based approaches employ Merkle Hash Tree (MHT) [43] or its variants (*e.g.* Merle B+-tree [32, 38], Merkle R*-tree [39, 62]) as the core ADSes. During a query phase, the cloud traverses the tree to identify the query results and construct the tree traversal path as the proof to the data consumer. Based on the received proof information, the data consumer reconstructs and replays the path to verify the correctness of the query results. Most tree-based approaches employ disk-based multi-way search tree such as B+ tree and R* tree as the indexing structure. The node size of one Multiway-tree is configured to that of a page whose minimum size is 4KB on most architectures. When the IoT device updates the multi-way tree, the entire stale nodes are transmitted to the cloud even only a small fraction inside a stale node is modified.

Trusted Hardware Aided Approaches. CorrectDB [15] and EnclaveDB [22] rely on the specialized trusted hardware Intel SGX [6] inside the cloud to conduct the heavy-lifting tasks in database outsourcing. VERID does not depend on any special hardware.

3 PROBLEM STATEMENT

3.1 Data Communication Model

The life cycle of IoT sensing data is demonstrated in Figure 1. The communication model consists of three different kinds of entities:

IoT devices are resource-constraint devices that generate sensing data. Programs running on the IoT devices should abide stringent resource limits in computation, memory, and power resources. IoT devices do not require perfect synchronization, but we do assume one synchronization protocol available to loosely synchronize clocks on different IoT devices with bounded drift. At the end of each interval, called an *epoch*, devices send all generated data within the epoch to the cloud. We define all IoT devices collaboratively perform a monitoring task as a *task group*. We assume that all data from one single group follow a common schema.

Cloud is a third-party storage provider who has rich resources. It stores the sensing data from IoT devices and exposes a SQL-like interface for Data Consumers (DCs) to make queries.

Data Consumers (DCs) are a vast variety of software systems, devices and human clients that retrieve the IoT sensing data for analysis purposes. DCs may also be IoT devices.

IoT sensing data can be classified into two types: time series data and event data [63]. Time series data, generated periodically, are used to describe continuously changing environment parameters such as temperature. Event data are generated whenever a certain type of events occurs, such as a door with one smart lock being opened. Note time series data can be viewed as a special case of event-based data driven by clocks. Hence this paper uses event-based data in the model for generalizability.

3.2 Supported Operations

VERID supports many SQL-style range selection and aggregation operations including AVG, MAX, MIN, COUNT, SUM and MEDIAN. VERID does not support JOIN at this point and would be our future work. VERID follows the relational data model, in which the query/operation can be expressed by the relational algebra [24]. For a relation r, VERID supports:

1) Selection (σ). $\sigma_C(r) = \{t \in r | C(t)\}$. A selection operation $\sigma_C(r)$ over relation r returns all the tuples in the relation r meeting the condition C. C can be used to specify the range of a range query on indexed attributes (dimensions).

2) Aggregate ($\mathcal{G}$). Aggregate function f maps a set of values into a single value. Common aggregate functions include AVG, MAX, MIN, COUNT, SUM. In addition, VERID supports MEDIAN and its generalization p-th percentile.

3.3 Threat Model

There is an increasing concern about the security of outsourced data in the cloud [41, 55, 59]. The query result could be corrupted by outside attackers, malicious cloud employees [42], transmission failures, or storage loss [3].

We assume only IoT sensing devices and data consumers are trustworthy and any entities in between including the cloud are subject to attack or may perform functionalities in a dishonest way. The correctness of range selection are two-folded:

(1) Each data item in the query result should be from the intended database and not tampered by any third party. This property is called as **authenticity** and **integrity**.
(2) Each data item in the result must satisfy the query predicates and all data items satisfying the query predicates must be included in the result. This second requirement is denoted as **completeness**.

Likewise, the aggregate query correctness means the aggregated value is computed from the data items satisfying the above properties.

VERID does not address the issue of privacy and confidentiality in this paper. They are orthogonal to the database query correctness and there are exiting works [40, 51, 56] on solving these two aspects of data outsourcing.

The cloud can claim that it stores no data to a data consumer. However this cheating is relatively easy to audit and detect. As long as the cloud claims it stores sensing data, the returned data to a DC must be correct and complete.

Even if IoT devices are likely to be compromised, different IoT devices are isolated in terms of trust. The compromising of one IoT device does not propagate to others. There is some literature [14, 60] to detected compromised IoT devices, which is complimentary to the content of this paper. On the other hand, the cloud impacts the whole systems and it is out of the control of IoT applications. To make the system robust against device compromise, each IoT device hosts and uses its own private key. The paper assumes the existence of a well-functioning PKI which manages the distribution of the public keys. The asynchronous IoT-cloud communication is protected by the digital signature and man-in-the-middle attack is hard if digital signatures are certified by PKI.

There is also an external mechanism for the data consumer to get the relation schema and the names of all collaborative IoT devices in the task groups of interest.

4 SYSTEM DESIGN

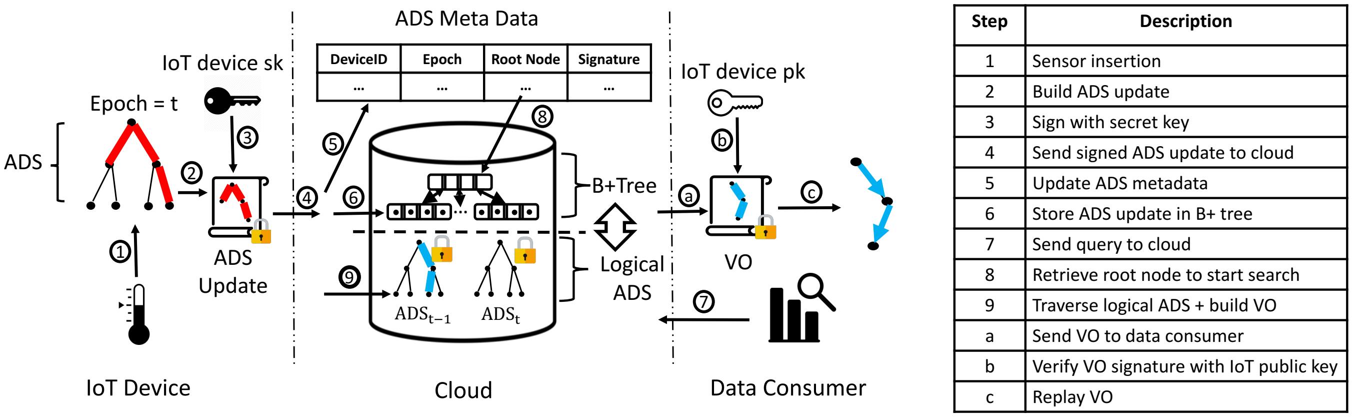
4.1 Overview

We dissect VERID into steps and depict the design overview in Figure 2. We will explains the procedures in this subsection about how different entities interact to achieve verifiable data query in the IoT scenario. **Multiple IoT devices** are sending data to the cloud simultaneously, but we only demonstrate one IoT device in the picture for ease of presentation.

Authentication Data Structure (ADS) is an indexing data structure whose operations can be carried out by an untrusted cloud and the result could be verified by data consumers. Each IoT device maintains and updates one ADS in accordance with the sensor readings during its entire life cycle. In VERID, the ADS is a new data structure PrefixMHT, which is essentially a binary search tree (BST) and can be authenticated similar to the Merkle Hash Tree (MHT) [43] (see details in Section 4.3). Therefore, VERID falls in the broad category of tree-based method. Upon new sensor readings, the IoT device inserts the value into PrefixMHT as a normal BST insertion except that all visited nodes are marked as *stale*. (**Step 1**). VERID updates the latest ADS to the cloud at a fixed time interval called *epoch*. Therefore, each PrefixMHT node includes an epoch attribute to indicate the epoch when the node is updated. The epoch attribute together with the node value can uniquely identify a PrefixMHT node over time and thus are collectively defined as the *NodeID*. The epoch attribute is the key enabler for VERID to perform queries on historical data. At the end of each epoch, the IoT device updates the digests of the PrefixMHT by recomputing the hash of stale nodes from bottom up like all other MHT variants. As such, the hash of the root summarizes the whole PrefixMHT and therefore is also referred as the digest of the PrefixMHT. (**Step 2**). Afterwards, the IoT device signs the root node using its own private key (**Step 3**). Thanks to the tree structure of the PrefixMHT, the IoT device sends to the cloud only the stale nodes along with the new signature instead of the entire PrefixMHT (**Step 4**). Since BST insertion starts from the root, the root node for every epoch is always stale and included in every PrefixMHT update. To maintain the tree structure when serializing PrefixMHT nodes, the NodeIDs of both left and right children are explicitly included in each node.

When the PrefixMHT update is received, the cloud first stores the root NodeID as well as the signature in one structure for maintaining ADS meta-data. ADS meta-data are used at the data query procedure described later (**Step 5**). All other parts of the ADS update, *i.e.* the stale nodes, are stored at the leaf nodes of one B+ Tree, which indexes PrefixMHT nodes by their NodeIDs. Since the cloud has all the incremental updates history (*i.e.* PrefixMHT updates from all proceeding epochs), it is able to reconstruct the complete PrefixMHT of any epochs hitherto from PrefixMHT nodes stored in the B+ tree. As a result, multi-version *logical PrefixMHTs* embed in the B+ tree to answer data queries.

Steps 1-6 run repetitively during the whole lifecycle of the IoT device. On the other hand, one round of Steps 7-c are stimulated when the data consumer issues an data query to the cloud through the query interface described in this paper. The query may span multiple epochs across the IoT sensing devices. The same search criteria will be applied to all IoT devices in the same *task group* (**Step 7**). Upon receiving and parsing the data query request, the cloud uses (*DeviceID*, *epoch*) as the key to retrieve from ADS metadata the root NodeIDs and the signatures of interest (**Step 8**). Given the root node, the logical PrefixMHT, basically a binary search tree, is traversed according to the search criteria specified by the data consumer. The PrefixMHT nodes on the search path in the logical PrefixMHTs are assembled as an unforgeable *Verification Object* (VO), which will be used by the data consumer later to verify the query result. The VO consists of one or multiple partial PrefixMHTs which have already signed by the IoT device (**Step 9**). The query result, together with the VO and associated signatures, are returned to the data consumer (**Step a**). Following the signatures verification with the public key of the IoT device (**Step b**), the data consumer



Step	Description
1	Sensor insertion
2	Build ADS update
3	Sign with secret key
4	Send signed ADS update to cloud
5	Update ADS metadata
6	Store ADS update in B+ tree
7	Send query to cloud
8	Retrieve root node to start search
9	Traverse logical ADS + build VO
a	Send VO to data consumer
b	Verify VO signature with IoT public key
c	Replay VO

Figure 2: VERID design overview

replays the searching path embedded in the VO to verify the query result (**Step c**).

4.2 Design of Prefix Tree

We first introduce a new design called *Prefix Tree* which enables efficient aggregation queries. Operations in a Prefix Tree are motivated by the idea from Prefix Sum [36] but are further extended to handle dynamic updates. We then design an ADS which embeds Prefix Tree in Merkle Hash Tree (MHT) called *PrefixMHT*.

Prefix Sum [36] performs the SUM operation over static dataset with low overhead. Given an integer array nums, Prefix Sum can efficiently find the sum of the elements between any two indices. The basic idea is to pre-process the array such that range sum query could be easily assembled by pre-processed values. The prefix sum PS of array M is an array and each element is: $PS[i] = \sum_{j=0}^{i} M[j]$. Given the prefix sum, the range sum can be computed as: $Sum([l, h]) = PS[h] - PS[l]$, where l and h denote the exclusive lower bound and inclusive upper bound of the range query respectively. To apply Prefix Sum in IoT applications to answer the question like "show the number of temperature readings between 5 and 13 degree", the sensor readings are bucketized and an array M is initiated to maintain the number of readings in each bucket. Obviously the number of buckets hence space complexity determines the query precision. More significantly, Prefix Sum suffers from inefficient update: A single update may trigger conducting pre-processing over the whole dataset in the worst case. The situation is even worse for IoT applications which usually feature high write/read ratios. Additionally, Prefix Sum wastes considerable space when the array itself is sparse, which is a common situation in IoT applications if the distribution of some sensor readings are highly skewed.

Prefix Tree thus is proposed to handle the dynamic updates while achieving the efficiency on aggregation queries. Prefix Tree does not suffer from space-precision dilemma. Each interval node of Prefix Tree maintains four attributes: *key*, *cardinality*, *sub-tree count* and *sub-tree sum*. The key attribute stores the searchable sensor reading value (*e.g.* temperature) and is used as the search key. All data items are sorted along the tree based on the key attribute. All future range and aggregate queries should be on the key attribute dimension. We will show how this method can be extended to support multi-dimensional range queries in Sec. 4.6. Cardinality refers to #elements having the associated value. Sub-tree counts and sums summarize the corresponding sub-trees (including the node itself) which are analogous to the element of Prefix Sum array *PS*. Since the operations of the both aggregated values are similar, we concentrate on the sub-tree count only for the following discussion. Prefix Tree enables efficient *prefix count* queries, like "show the number of temperature readings below 13 degree". Figure 3 gives an illustrative example to accomplish the query "SELECT COUNT(*) FROM value ≤ 13". In Figure 3, the two numbers inside each node attribute to the value and its cardinality respectively. For instance, 10(3) inside N_{01} indicates three instances of value 10. The numeric aside the link summarizes the total number of instances under the subtree: the *sub-tree count* attribute of lower node of this link. The prefix-count query starts from the root and traverses the Prefix Tree like a normal BST. *Count* is initialized to 0 in the beginning. On delving into the right child, count is increased by the difference of the sub-tree count of the parent node and that of the right-child node. Take Figure 3 for instance. On traversing from N_{root} to N_1 the count is increased from 0 to $14 - 7 = 7$, meaning that number of instances equal or smaller than 11 is exactly 7. On traversing from N_1 to N_{10}, the count does not change. When the leaf node N_{10} is reached, its cardinality is then added to the count if its value equals to the higher searching bound. In particular, the query result of the aforementioned example is $7 + 2 = 9$. Range sum query can be conducted similarly (*sub-tree sum* is not shown in Figure 3).

Upon insertion, only nodes on the path from the newly inserted node to the root are updated. Therefore, the insertion complexity is $O(\log n)$, where n is #nodes in the Prefix Tree. Prefix Tree is far more efficient than Sum Prefix in dynamic settings.

4.3 Design of PrefixMHT

We design the ADS based on Prefix Tree called PrefixMHT which allows a Prefix Tree to be authenticated in a fashion similar to a Merkle Hash Tree (MHT). **PrefixMHT ensures query correctness of both selection and aggregation queries.**

Each PrefixMHT node is composed of the following attributes: *key*, *cardinality*, *sub-tree count*, *sub-tree sum*, *epoch*, *lchild NodeID*,

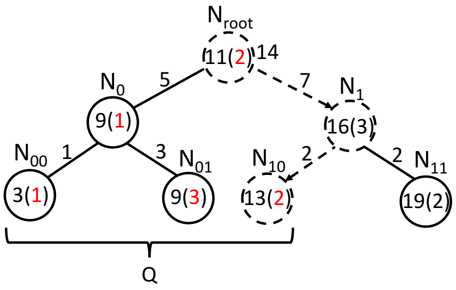


Figure 3: Intuition on PrefixTree

rchild NodeID, *lchild hash* and *rchild hash*. The first four attributes are consistent to those of the Prefix Tree. The epoch attribute indicating when the node is updated which is indispensable to reconstructing multi-version ADSes in the cloud. In addition, PrefixMHT node stores the NodeIDs of its two children to enable logical tree traversal in the cloud. Note again that one NodeID is a tuple of the node value and the epoch.

In order to explain *lchild* (*rchild*) hash, we first introduce the concept of *node hash*. Node hash encompasses all aforementioned attributes in the PrefixMHT node and is used for authentication at node level. The attribute *lchild* (*rchild*) hash is simply the node hash of its left (right) child. Therefore the node hash recursively summarizes the whole subtree. The root node hash is referred as the *digest* of the whole PrefixMHT and is signed by the private key to create the *digital signature*. The authentication of PrefixMHT can therefore be conducted in a top-down approach starting from the root node.

The IoT device does not create a PrefixMHT from scratch for every individual epoch. PrefixMHT incrementally accumulates cardinality, sub-tree count and sum over time. The directly impacted nodes due to insertion and nodes on their paths to the root are collectively defined as *stale* nodes because they all contribute to the incremental update. The prefix count $PC[k, t]$, for instance, can be interpreted as accumulated #insertions whose key values are smaller than or equal to k from epoch 0 to epoch t. As such, the answer for range count across multiple epochs are modified to:

$$\begin{aligned} &Count([l, h], [t_b, t_e]) \\ =&PC[h, t_e] - PC[l^-, t_e] - PC[h, t_b - 1] + PC[l^-, t_b - 1] \end{aligned} \tag{1}$$

The two inclusive key search bounds are denoted as l and h. Let t_b and t_e be the inclusive start and end epochs.

4.4 Efficient PrefixMHT Update

Upon the end of an epoch, the IoT device updates PrefixMHT according to the sensing data generated at that epoch. The root node is always marked with the new epoch even if no insertion occurs in that epoch; otherwise the cloud may discard data without being detected. The hash of the root node summaries a snapshot of the whole PrefixMHT, which is signed at every epoch by the IoT device using its own private key. We will discuss in detail the signing procedure in Section 5. Stale nodes and the PrefixMHT signature are transmitted in JSON format [7] to the cloud for storage.

4.5 Storage in the Cloud

As shown in Figure 2 PrefixMHT updates are stored at the leaves of a B+ tree in the cloud. Most prior tree-based methods let the cloud create individual B+ trees, one for each IoT device, since different IoT devices host their own private keys. VERID however takes the opposite way: PrefixMHT updates from the a same task group are stored one per-group B+ tree. In the per-group B+ tree, a PrefixMHT node in an update is uniquely identified by a tuple of ($DeviceID$, key, $Epoch$), where $DeviceID$ is augmented upon the node arriving at the cloud and key is the key field of the PrefixMHT node. How the updates are sorted in the B+ tree have a profound impact on the I/O cost. We discover through excessive experiments that the three attributes prioritized as $Epoch > key > DeviceID$ exhibits the best I/O performance. Basically, $Epoch$ preserves least locality because the query can specify arbitrary starting and ending epochs. On the other hand, since the same query range is applied to all IoT devices in the same task group, clustering together all PrefixMHT nodes associated with the same key values exploits the spatial locality. Thus, $DeviceID$ possesses the lowest priority among the three constitutional attributes of the node identification.

Buffer management is an integral part of VERID, which caches the content of disk reads in the memory to reduce I/O cost. The basic management unit is one *page*, a fixed-length contiguous disk block (*e.g.* 4KB). In VERID, its buffer management uses Clock replacement algorithm [54] for its simplicity.

4.6 Extending to Multi-dimensional Data

The previously discussed PrefixMHT is a binary search tree which can only handle one-dimensional data. However, most IoT devices carry multiple sensors and as a result, IoT data are mainly high-dimensional. VERID rests on Space-filling Curves (SFCs) to map a high dimensional data point to a one 1-D value. Notable examples of SFC include C-curve, Z-curve (a.k.a. Morton order) [45] and Hilbert curve [44]. Therefore, by utilizing SFC, one high-dimensional range is transformed to one or multiple 1-D segments which can be computed efficiently using the divide & conquer strategy [47]. With the help of SFC, validation of one range query is converted to authenticating its corresponding end points in the SFC space.

Clustering number [44] is proposed to capture #segments during range query processing. From the viewpoint of VERID, smaller clustering number leads to shorter VO size. For IoT applications, the range query usually exhibits some constraints or patterns. For example, in geographical IoT applications, each query zone (*e.g.* from citywide to street block) is bounded to one 2-D query range with fixed length and width. As a result, we leverage the query-aware QUILTS [46] as the space-filling curve on VERID. QUILTS is a framework optimizing clustering number by choosing one SFC from a family of *Bit-Merging Curves* [46] based on the query pattern. IoT devices are also in favor of QUILTS over other SFC such as Hilbert curve [44] due to the low computational overhead to map the multi-dimensional data point to 1-D QUILT point.

Moreover, VERID is also able to amend multi-modal dataset, as long as these data are associated with some searchable meta data. For instance, one dimension to describe sound is its volume. VERID supports search the sound based on its volume. In this way, multi-modal data could be viewed as ordinary multi-dimensional data.

5 SIGNATURE SCHEME FOR VERID

The ultimate goal of this section is the introduction of CBP-VERID which significantly reduces communication cost in sparse settings. In this section, we first give the preliminaries to understand the signature scheme. After that, *Hash Fusion Signature* (HFS) generated at the IoT device side is described in Section 5.2. The signature aggregation scheme, *Condensed Bilinear Pairing* (CBP) and its verification are discussed in Section 5.3. Even though the working scenarios of CBP is limited, the introduction of CBP makes audiences understand CBP-VERID more easily. CBP-VERID is illustrated in Section 5.4.

5.1 Preliminaries

5.1.1 Notations. Let λ denote the security parameter. $\upsilon(\cdot)$ represents a negligible function, which means $\exists L \in \mathbb{N}$, such that $\upsilon(x) < 1/f(x)$ for any $x > L$ and any polynomial function $f(\cdot)$. $x \leftarrow_R S$ means assigning x a uniformly dawn value from set S. If $A(\cdot)$ is a probabilistic algorithm, $y \leftarrow A(x)$ means A return its output to x. PPT is a shorthand for Probabilistic Polynomial-Time. $\{0,1\}^*$ represents the set of string of any length. Concatenation is written as $||$. $Pr[E]$ means the probability of the occurrence of event E.

5.1.2 Bilinear Pairing. CBP is based on an algebraic structure, namely *bilinear pairing*. Suppose $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic multiplicative groups of prime order p with the cyclic generators g_1 and g_2 respectively. $\mathbb{G}_T$ is another cyclic multiplicative group of the prime order p. A bilinear pairing is a map e: $\mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$ satisfying 1) Bilinearity: $\forall u \in \mathbb{G}_1, \forall v \in \mathbb{G}_2, \forall a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u^a, v)^b = e(u, v^b)^a = e(u, v)^{ab}$. 2) Non-degeneracy: $e(g_1, g_2) \neq 1$. 3) Computability: An algorithm exists to compute the mapping efficiently. Bilinearity and non-degeneracy imply another two important property:

$$e(u_1u_2, v) = e(u_1, v)e(u_2, v) \qquad \forall u_1, u_2 \in \mathbb{G}_1, \forall v \in \mathbb{G}_2 \tag{2}$$

$$e(\psi(u), v) = e(\psi(v), u) \qquad \forall u, v \in \mathbb{G}_2 \tag{3}$$

Here ψ is an efficient computable isomorphism function $\psi : \mathbb{G}_2 \mapsto \mathbb{G}_1$ such that $\psi(g_2) = g_1$. The 7-tuple $bp := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ defines a bilinear pairing.

5.2 Hash Fusion Signature

We start with the building block of our signature scheme, called *Hash Fusion Signature* (HFS) which is used by each IoT device to sign data. HFS is a variant of BGLS signature scheme [19] which is constructed based on bilinear pairing. **Even though HFS could be easily derived from BGLS, their definitions of unforgeability are different.** HFS is defined as a tuple $(KeyGen, Sig, Ver)$ consisting of three algorithms. Assume an algorithm $BilGen(1^\lambda)$ is available to setup the public parameters of bilinear paring, which is used as a subroutine in *KeyGen*. $bp := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow BilGen(1^\lambda)$, where λ is the security parameter and p is a λ-bit prime. Let H: $\{0,1\}^* \rightarrow \mathbb{G}_1$ be a full domain hash function modeled as a random oracle [16]. The message to be signed are composed of two sub-strings, *i.e.* $\overrightarrow{m} = m_1||m_2$. $\overrightarrow{m}[i]$ represents the ith component of $\overrightarrow{m}$, $i \geq 1$.

Definition 5.1 (HFS). *Hash Fusion Signature (HFS) is a tuple of three algorithm $(KeyGen, Sig, Ver)$ as described below.*

***KeyGen*(1^λ):** *$bp \leftarrow BilGen(1^\lambda)$, pick a random secrete $s \leftarrow_R \mathbb{Z}_p^*$ and compute g_2^s. Set the public key $pk \leftarrow (g_2^s)$ and the private key $sk \leftarrow s$. The public parameter is $pp \leftarrow (bp, pk)$.*

***Sig*$(\overrightarrow{m}, sk)$:** *for message $\overrightarrow{m} = m_1||m_2$, let $h \leftarrow H(m_1) * H(m_2)$. The signature is $\sigma \leftarrow h^s$. Return $\alpha \leftarrow (\overrightarrow{m}, \sigma, pk)$.*

***Ver*$(\overrightarrow{m}, \sigma, pk)$:** *for message $\overrightarrow{m} = m_1||m_2$, let $h \leftarrow H(m_1) * H(m_2)$. Check $e(g_1^h, g_2^s) \overset{?}{=} e(\sigma, g_2)$. If the two terms are equal, return 1; otherwise return $\perp$.*

In the case of VERID, m_2 is the epoch and m_1 represents other content of PrefixMHT root node. The purpose to isolate epoch is to align the signature scheme with the highly structured communication pattern of IoT applications where only the epoch increases and other part stays unchanged most of the time. This design is the key enabler to reduce communication and computation complexity of sparse settings described in Section 5.4.

The security of HFS is captured by a standard notation, *Existential Unforgeability under Chosen Message Attack* (EU-CMA) [31]. EU-CMA of HFS is defined by an experiment where the advantage of any PPT adversary $\mathcal{A}$ is negligible. In this experiment, $\mathcal{A}$ is provided with a signing oracles $HFS.Sig_{sk}(\cdot)$. The signing oracle $HFS.Sig_{sk}(\cdot)$ returns the signature of the input under private key sk. $\mathcal{A}$ can adaptively choose the messages as the input to the signing oracle, but it can only query the signing oracles for up to $poly(\lambda)$ times, where $poly(\cdot)$ can be any polynomial function and λ denotes the security parameter. $\mathcal{A}$ finally returns a forgery $(\overrightarrow{m}^*, \sigma^*)$ under pk, where $\mathcal{A}$ did not query the signing oracle on $\overrightarrow{m}^*$ before. $\mathcal{A}$ wins iff $Ver(\sigma^*, \overrightarrow{m}^*, pk) = 1$. EU-CMA of HFS can be expressed formally as:

Definition 5.2 (EU-CMA of HFS). *HFS is Existential Unforgeable Secure under Chosen Message Attack if the following formula holds.*

$$\Pr\left[\begin{array}{l}(sk, pk) \leftarrow HFS.KeyGen(1^\lambda)\\(m^*, \sigma^*) \leftarrow \mathcal{A}^{HFS.Sig_{sk}(\cdot)}(pk)\\1 \leftarrow HFS.Ver(\sigma^*, m^*, pk)\end{array}\right] < \upsilon(\lambda)$$

At first glance, EU-CMA of HFS is directly implied by the BGLS which has shown its provable unforgeability in [19], because a HFS signature can be viewed as the aggregation of two BGLS signatures from a same user. However, the definition of unforgeability of BGLS is slightly different from that of HFS. EU-CMA of HFS focuses on $m_1||m_2$ as a whole whereas the notion of unforgeability in BGLS captures individual parts.

5.3 Condensed Bilinear Pairing

We propose to aggregate multi-user HFS signatures to save the bandwidth consumption as well as computational-intensive bilinear pairings. We strive to reduce the bandwidth consumption by exploiting the fact that some IoT devices only send default messages in some extreme cases. Imagine a fire alarm system with thousands of sensors deployed in the forest. Individual sensor sends an alarm only when detecting hazardous situations. If high temperature never occur in the forest, the sensor periodically generates *default* *messages* indicating safety, *i.e.* a PrefixMHT node with sub-tree count = 0. Since the content of the default message has been known priori, sending default messages on the network is not necessary. The epoch attribute is the same across sensors. In the case of most sensors sending default messages, Condensed Bilinear Pairing (CBP) can accelerate the verifying speed and save communication cost.

Definition 5.3 (CBP). *Condensed Bilinear Pairing (CBP) is a tuple of six algorithms* $(KeyGen, Sig, Ver, PkAgg, SigAgg, VerAgg)$*. The default message is denoted as* $M||t$ *where* M *is the static content and* t *is the epoch.* $SigAgg$ *and* $VerAgg$ *are only applicable to messages from the same epoch, and hence identical epoch value.* $KeyGen$*,* Sig *and* Ver *have already been formalized in Definition 5.1.* $SigAgg$ *and* $VerAgg$ *are described below.*

$\boldsymbol{PkAgg(pk_1, pk_2, \cdots, pk_n)}$*: given public keys for every IoT device in a task group, the aggregated public key is* $apk = \prod_{i=1}^{n} pk_i$

$\boldsymbol{SigAgg(\alpha_i, \alpha_j)}$*: for any two HFS signatures,* $\alpha_i = (\overrightarrow{m}_i, \sigma_i, pk_i)$ *and* $\alpha_j = (\overrightarrow{m}_j, \sigma_j, pk_j)$*, if* $\overrightarrow{m}_i[2] \neq \overrightarrow{m}_j[2]$*, return* $\perp$*. We define* $\widetilde{\Gamma_i} = (\overrightarrow{m}_i[1], pk_i)$ *if* $\overrightarrow{m}[1] \neq M$*; otherwise* $\widetilde{\Gamma_i} = \emptyset$*.* $\widetilde{\Gamma_j}$ *is defined similarly. The aggregated signature is* $\alpha = (\widetilde{\Gamma_i} \sqcup \widetilde{\Gamma_j}, \sigma_i * \sigma_j, \overrightarrow{m}_i[2])$*, where* $\sqcup$ *represents the merging operation.*

$\boldsymbol{VerAgg(\alpha, apk)}$*: let the aggregated signature be* $\alpha = (\Gamma, \sigma, t)$ *where* $\Gamma = \{(\overrightarrow{m}_1[1], pk_1), \cdots, (\overrightarrow{m}_\kappa[1], pk_\kappa)\}$*. Define* $\widetilde{apk} = \frac{apk}{\prod_{i=1}^{\kappa} pk_i}$*. Check* $e(\sigma, g_2) \stackrel{?}{=} e\left(H(M), \widetilde{apk}\right) * \prod_{i=1}^{\kappa} e\left(H(m_i), pk_i\right) * e\left(H(t), apk\right)$*. If the equation holds, return* 1*; otherwise return* $\perp$*.*

The following derivation provides the intuition on the correctness of CBP.

$$
\begin{aligned}
& e\left(H(M), \widetilde{apk}\right) * \prod_{(m_i, pk_i) \in \Gamma} e\left(H(m_i), pk_i\right) * e\left(H(t), apk\right) \\
=& e\left((H(M) * H(t)), \widetilde{apk}\right) * \prod_{(m_i, pk_i) \in \Gamma} e\left((H(m_i) * H(t)), pk_i\right) \\
=& e(\sigma, g_2)
\end{aligned}
$$

The unforgeability of aggregated signature is different from that of a single message, which requires that an adversary cannot generate a signature indicating the authenticity of an unsigned message, even if all other signers are dishonest.

Compared to the naïve method where the receiver verifies every single timestamped default message separately, our newly proposed scheme reduces both the communication cost and computation cost from $O(k)$ to $O(1)$, where k is #sensors. **Even if the assumption that most sensors have not detected any events does not hold, CBP can still work but is degenerated into BGLS signature scheme.**

5.4 CBP-VERID

We identify that in the sparse setting, most of roots of PrefixMHT remains untouched except for their monotonically increased epochs. However, applying CBP directly does not work in general because no default message is shared by most sensors in VERID. We make minor changes to Condensed Bilinear Pairing in order to adapt to VERID's needs by leveraging the special structure of Bilinear pairing.

If no event is detected between two epochs t^b (begin epoch) and t^e (end epoch), we can reach to:

$$
e(\sigma_b / \sigma_e, g_2) = e\left(\frac{(H(M) * H(t_b))^s}{(H(M) * H(t_e))^s}, g_2\right) = e\left(\frac{H(t_b)}{H(t_e)}, g_2^s\right) \quad (4)
$$

By verifying the above equality, the data consumer is able to ensure no new event being detected without knowing concrete M.

THEOREM 5.1. *Given two HFS signatures* (σ_b, σ_e) *and two epochs* (t_b, t_e)*,* $e(\sigma_b/\sigma_e, g_2) = e\left(\frac{H(t_b)}{H(t_e)}, g_2^s\right)$ *indicates identical content at* t_b *and* t_e *except with negligible probability.*

We could further aggregate the roots' signatures even with different M value, because the two instances of $H(M)$ at the denominator and the numerator in Eq. (4) are canceled out. To this end, we propose Condensed Bilinear Pairing-VERID (CBP-VERID) to opportunistically aggregate HFS signatures.

Definition 5.4 (CBP-VERID). *Condensed Bilinear Pairing-VERID is a tuple of six algorithm* $(KeyGen, Sig, Ver, PkAgg, SigAgg, VerAgg)$*.* $KeyGen$*,* Sig*,* Ver *and* $SigAgg$ *are exactly the same as those of CBP presented in Definition 5.3. Inherent from CBP,* $SigAgg$ *and* $VerAgg$ *in CBP-VERID also require that the operated messages are from two epoches only: the begin and the end. For ease of representation, we define* augmented signature $\hat{\alpha} \triangleq (m_b, t_b, \sigma_b, m_e, t_e, \sigma_e, pk)$ *from a single device, where the subscription* b *and* e *stands for begin epoch and end epoch respectively.* m, t *are the two constructional components of the PrefixMHT root: content and epoch value. Aggregated signature is* $\Sigma = (\Gamma, \sigma, t)$*, where* Γ *concatenates changing content,* σ *is the product of individual signatures and* t *represents the epoch.*

$\boldsymbol{SigAgg(\hat{\alpha}_i, \Sigma^b, \Sigma^e)}$*: for the input,* $\hat{\alpha}_i = (m_i^b, t_i^b, \sigma_i^b, m_i^e, t_i^e, \sigma_i^e, pk_i)$*, If* $(t_i^b \neq \Sigma^b.t) \vee (t_i^e \neq \Sigma^e.t)$*, return* $\perp$*. We define* $\hat{\Gamma}_i^b = (m_i^b, pk_i)$*,* $\hat{\Gamma}_i^e = (m_i^e, pk_i)$ *if* $m_i^b \neq m_i^e$*; otherwise* $\Gamma_i^{\hat{b(e)}} = \emptyset$*. The aggregated signature is updated as follows.* $\Sigma^{b(e)}.\Gamma = \Sigma^{b(e)}.\Gamma \sqcup \Gamma_i^{\hat{b(e)}}$*,* $\Sigma^{b(e)}.\sigma = \Sigma^{b(e)}.\sigma * \sigma_i^{b(e)}$*.*

$\boldsymbol{VerAgg(\Sigma^b, \Sigma^e, apk)}$ *: if* $(|\Sigma^b.\Gamma| \neq |\Sigma^e.\Gamma|)$*, return* $\perp$*. Let* $\Sigma^{b(e)}.\Gamma = \{(m_1^{b(e)}, pk_1), \cdots, (m_\kappa^{b(e)}, pk_\kappa)\}$ *Define* $\widetilde{apk} = \frac{apk}{\prod_{i=1}^{\kappa} pk_i}$*. Then check* $e\left(\sigma^b/\sigma^e, g_2\right) \stackrel{?}{=} e\left(H(t^b)/H(t^e), \widetilde{apk}\right) * \prod_{i=1}^{\kappa} e\left(H(m_i^b)/H(m_i^e), pk_i\right)$*. If the equation holds, return* 1*; otherwise return* $\perp$*.*

To speed up the computation of $\widetilde{apk}$ when more than half devices contribute Σ, $\widetilde{apk}$ can alternatively calculated as the product of their signatures.

THEOREM 5.2. *Condensed Bilinear Pairing-VERID is EU-CMA secure with the assumption of Computational Co-CDH hardness under random oracle model.*

The experiment to define EU-CMA and the corresponding proof are omitted for brevity.

6 EVALUATION

We implement a prototype of VERID including all parts: the working programs on sensing devices, cloud, DCs and coordinators. The cloud and DC instances run on one quadcore@3.40GHz Linux desktop with 32GB memory and each sensing device instance runs on a M3 Open Node [9] equipped with a 32-bit ARM Cortex M3@72MHz

Table 2: Summary of the Two Datasets

	IntelLab	Rome
#Devices	54	289
#Original readings	370667	4992
#Injections	387	110719
#Total readings	371054	115711

as well as one 16-MByte external Nor flash. We conduct extensive experiments driven by real datasets.

6.1 Evaluation Methodology

We leverage two publicly available IoT datasets:

(1) **IntelLab [1]**: 54 Mica2Dot sensors with weather board deployed in the Intel Berkeley Research Lab collecting timestamped temperature, light, *etc.* once every 31 seconds. In the experiments, timestamp and temperature (in Fahrenheit) information is extracted to represent 1-D dataset. The queries for experiments are synthetic. We use 100 randomly generated aggregation count queries. For the synthetic queries, the lower temperature bound is a float number uniformly drawn from $[0, 200]$. The upper bound is set to always 20 degrees higher than the lower bound.

(2) **Rome [13]**: 289 taxicabs in Rome occasionally report outdoor temperature readings together with the GPS coordinates (*i.e.* longitude and *i.e.* latitude). The dataset spans 4 days and the tuple $(longitude, latidues, temperature)$ forms **multi-dimensional** sensing readings. VERID utilizes QUILT to map individual multi-dimensional readings into 1-D points. The temperature is represented by one 16-bit integer. Synthetic count queries are generated to simulate those from geographic information system applications, where the spatial range is displayed in different hierarchical levels: from city-wide to street block level. Our experiments have 24 levels of hierarchical spatial query ranges. Given level l, the entire region in a $2^{24} \times 2^{24}$ grid, is partitioned into l^2 square areas with each being $2^{24-l} \times 2^{24-l}$. To synthesize one spatial range query, we first uniformly generate one hierarchical level l and then choose a spatial range from the l^2 square areas at random. The temperature range $[0, 2^{16} - 1]$ is evenly partitioned into 16 sub-ranges and each synthetic query specified one sub-range to explore.

For both datasets, one epoch is set to 15 minutes and all experiments driven by the data are from the first 400 epochs, which is approximately 4 days. To make sure every epoch is signed, we artificially inject one *dummy message* indicating an empty epoch into the epochs that do not possess any sensor reading. For the two sets of synthetic queries, the start and end epochs are randomly drawn from $[1, 400]$. We summarize the datasets details in TABLE 2. Rome represents one sparse setting as can be inferred from the large injection to reading ratio. IntelLab obviously posits at the opposite side. The purpose to set the epoch to 15 minutes is to create two data traces representing dense setting and sparse setting respectively. If the epoch during is much longer than 15 minutes, say 4 days, both IntelLab and Rome would representing the dense setting because every epoch from every IoT device encompasses data.

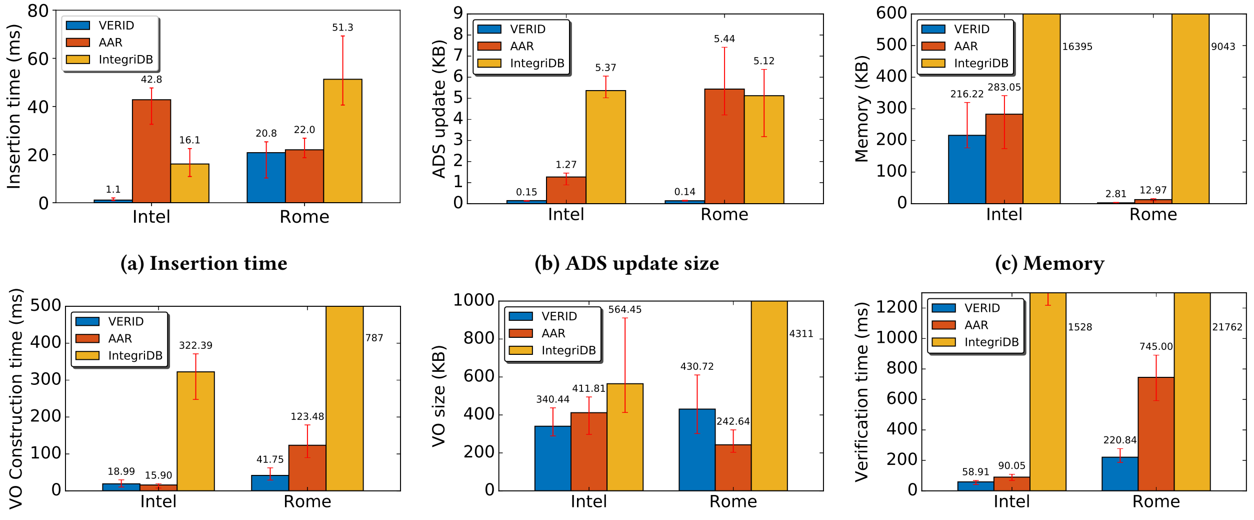
Methods to compare with. We compare VERID with two other state-of-art works, Authenticated Aggregation R-tree (AAR-tree) [39] and IntegriDB [65]. We select these two works among a large number of existing methods because 1) they support both aggregation and selection queries; and 2) they are relatively recent and demonstrate good performance compared to other methods. Infact, IntegriDB supports more operations (*e.g.* JOIN) than VERID does at extra cost. To the best of our knowledge, our paper first discusses and measures its performance in IoT scenarios for reference. For VERID and AAR, we set the capacity of the buffer pool to host 1000 frames of size 4KB. IntegriDB has a parameter q which determines the largest possible cardinality of verifiable query result. In our experiment, q is set to 1000.

More recent papers [15, 22] on this topic leverage special hardware such as Intel SGX[6]. VERID, AAR and IntegriDB all do not assume special hardware.

Cryptographic algorithms to use. SHA-256 [4] in OpenSSL [11] are used as the cryptographic hash function for all the three works in our experiments. For both VERID and IntegriDB, the bilinear pairing for signature is Ate-paring [2, 18] on a 254-bit elliptic curve which is estimated to offer 128-bit security. Since the encryption scheme to generate signatures is not specified in the original paper of AAR, in our experiments we specify it to be BGLS [19] on a 254-bit elliptic curve rather than the classic RSA [37] for fair comparison. From our own measurement in generating signatures, BGLS is nearly $30X$ faster than RSA with 3072-bit keys which also approximately offers 128-bit security.

Methodology and Metrics. The prototype experiments on IoT devices are conducted on one M3 open node board. The data trace contributed by each individual device is feed into the VERID prototype program at full speed. The timestamp from the trace drives the program to update the PrefixMHT digest and generate a signature when one epoch ends as indicated by the timestamp. The output of the program, *i.e.* what should be transmitted to the cloud, is stored locally at the flash drive. We measure the following metrics: **1)** Average time to process one insertion captures the computation costs at the IoT device side, which is denoted as the *insertion time*. **2)** The *ADS update cost* reflects the amortized communication cost which is computed by the size of all updates (excluding original data and signatures) divided by #insertions. **3)** The *memory* is an average memory usage of IoT devices. These three metrics for AAR and IntegriDB are measured on the M3 open node in the same way.

After the M3 open node experiments, the generated ADS updates and signatures along with original data are directly restored in the Linux desktop to study all other parts of VERID. Therefore, we avoid the impact of varying networking environments in our experiments. The synthetic queries are applied to all devices. VERID (or AAR/IntegriDB) constructs the VO when receiving the query request. We measure both **4)** the *VO construction time* and **5)** the *VO size*. The verification procedure starts in the same program immediately after the query result and its VO are produced. **6)** The *verification time* captures the computation efficiency at the data consumer's side.



(a) Insertion time

(b) ADS update size

(c) Memory

(d) VO construction time

(e) VO size

(f) Verification time

Figure 4: Aggregation query performance results

6.2 Aggregation Queries

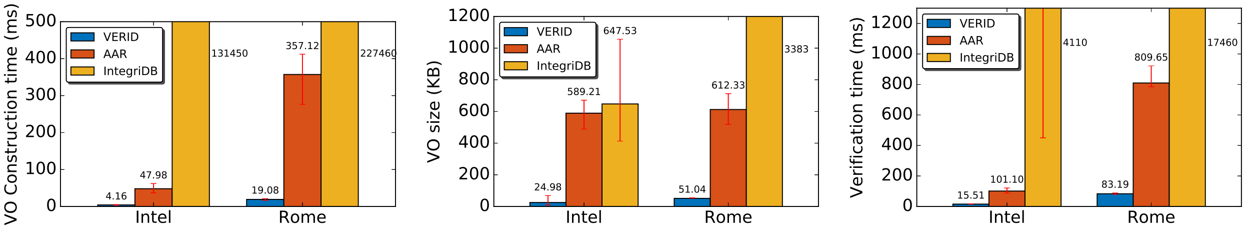
We demonstrate the aggregation queries performance results for IntelLab and Rome in Figure 4. Particularly, the aggregation queries are requesting for SUMs. Since COUNT is just a special case of SUM and other aggregate functions (*i.e.* AVG, MAX) can be derived from COUNT and SUM, we do not evaluate aggregate functions other than COUNT for brevity.

From the IoT devices' perspective, VERID outperforms AAR and IntegriDB in terms of computation (insertion time), communication (ADS update) and memory consumption for both datasets. Particularly, for VERID the insertion time in IntelLab experiments is smaller than that of Rome (Fig. 4a) because more insertions amortize the signature generation time at each epoch. When the insertion is rare as in Rome dataset, the time to generate signatures dominates the computation cost and this explains the comparable insertion time for VERID and AAR in the experiment using Rome dataset. For AAR, the amortized insertion time increases in the IntelLab experiments due to excessive time spent on computing hashes of 4-KB tree nodes. The long insertion time for IntegriDB is mainly due to excessive cryptographic operations. Since PrefixMHT of VERID is operated at fine grain, the ADS update size is much smaller than that of AAR which needs to update the whole stale 4-KB nodes (Fig. 4b). The update size of AAR is worse when rare insertions amortize the update cost, as can be validated from the results of Rome. The average ADS size for VERID in the Rome experiments is slightly smaller than in the IntelLab experiments due to the lower PrefixMHT height. VERID is memory efficient and acquires additional memory space when a new PrefixMHT node is inserted (Fig. 4c). On the other hand, AAR allocates memory at per-page basis. If the dataset is small, no enough readings are available to fill up the allocated memory space. Therefore, the memory usage gap between VERID and AAR is huge for Rome. Even if IntegriDB updates ADS at fine grain like VERID, its ADS node size is much larger hence ADS update size and memory footprint.

The mechanisms to construct VO for VERID and AAR are similar: recording the searching path on the ADS. The experiments conducted at the Linux desktop involves all devices collectively. For example, the VO construction time is the total time to find ADS paths for all 54 (387) devices in the IntelLab (Rome) experiments. VERID spends slightly more time to construct the VO than AAR does in the IntelLab experiments but the situation reverses in the Rome experiments where the data exhibit higher locality hence faster searching speed (Fig. 4d). The VO size of AAR is smaller than that of VERID for Rome (Fig. 4e), because the ADS of AAR is extended from R* tree, which is originally designed for GIS applications. VERID outperforms AAR regarding verification time for both datasets (Fig. 4f) because the AAR data consumers need to compute excessive hashes to verify the query results. Another contributing factor is our signature scheme which avoids some bilinear pairing evaluations. IntegriDB constructs and verifies VO by doing complicated cryptographic operations, which consume considerable time.

6.3 Selection Queries

We also conduct experiments to evaluate the performance of different works under selection queries. Since AAR and IntegriDB are not aware of raw signal data, we only consider readings consisting of indexable values for fair comparison. Each selection query requests for data from a specific epoch, which represents the scenario where the data consumer retrieves data from latest epoch for further analysis. We generate the selection queries by modifying the range queries described in the experiment setup: The range exception for the epoch attribute stays the same and the epoch of interest is set to the upper epoch bound from the corresponding aggregation query.



(a) VO construction time

(b) VO size

(c) Verification time

Figure 5: Selection query performance results

The VO construction time, VO size and verification time are reduced compared to the results from aggregation experiments as shown in Fig. 5. VO only contains one ADS path according to Proposition 4.1. Each range query requires 4 ADS paths as indicated by Eq.(1). On the other hand, the performance of AAR and IntegriDB downgrades significantly. The ADS of AAR relies on the R* tree. The range query within a single epoch can be geometrically viewed as a “long strip”. R* tree is not good at processing “long strip” range queries. For IntegriDB, the mechanism of selection query is different from that of count queries.

6.4 Comparison of Signature Schemes

For VERID, validating signatures dominates the verification time at the data consumer’s side: 99% and 98% verification time are spent on signature validation for IntelLab and Rome datasets respectively. Our signature scheme noticeably reduces the signature validation time for sparse settings. For Rome dataset, our signature scheme totally avoids 8143 out of 38700 PrefixMHT traversals in the cloud and bilinear pairing evaluations at the data consumer’s side. We evaluate the performance gain from our signature scheme by comparing it with BGLS [19]. The verification time is 307.19ms for BGLS and 220.84ms for VERID, an approximately 28% **reduction**. The VO construction time is reduced from 48.10ms (BGLS) to 41.75ms (VERID). Similarly, the VO size experiences a 19.8% **decrease**. In IntelLab dataset, nearly all devices have insertions for all epochs. Only 8/5400 PrefixMHT traversals thus are avoided in this case. As a result, there are no apparent changes on construction and verification time as well as VO size. Note that the proposed signature scheme is a variant of BGLS and thus the comparison highlights our contribution; otherwise the comparison is not fair because the performance advantage may come from BGLS.

6.5 Disk I/O at Cloud Storage

VERID leverages the special query pattern of IoT application to build per-group B+ tree to store PrefixMHT nodes instead of using individual per-device B+ trees. We measure the I/O cost, *i.e.* #disk reads for both IntelLab and Rome datasets. Per-group B+ tree greatly reduce the I/O cost as as illustrated in Table 3. Hence we use per-group B+ tree in VERID.

Table 3: I/O Cost Comparison

#disk I/O times	**IntelLab**	**Rome**
Per-device B+ tree	20051	10125
Per-group B+ tree	5610	1209

7 DISCUSSIONS

There are two ways to deploy VERID in the cloud. The application could rent raw storage and virtual machines from the cloud to run the VERID cloud-side protocol. In this case, the cloud is Infrastructure as a Service (IaaS). The other way is to build the cloud-side protocol atop existing platforms. The incremental updates could be installed in the database and then lambda function computing components are leveraged to traverse the database to compute the query result as well as VO. In this case, the cloud is leveraged as Platform as a Service (PaaS).

This paper mainly focus on verifiable orthogonal range queries (*i.e.* the searching boundaries are aligned with coordinate axes) do not directly support the k-nearest neighbors (k-NN) algorithm which has a wide range of applications in machine learning and spatial database. VERID can be easily extended to support verifiable k-NN query. Suppose a user search on a remote spatial database for the k most nearest restaurants from his (her) location. VERID can first invoke an external method to get the coordinates of kth nearest restaurant, all QUILT segments encompassed by the searching circle then are sent the user to prove the query result.

This paper assumes that 1) There is an external mechanism for the data consumers to get the names and public keys. 2) All IoT devices in a task group are homogeneous. However, these assumptions may not be valid all the time. For instance, in the case where IoT devices are statically deployed in different geographic locations, the data consumer may only request for data from the IoT devices in a specific region. In our future work, we plan to build a full-fledged verifiable rational database above VERID for the data consumer to query for the names and public keys of the IoT devices of interest. The task group owner is responsible for uploading the correct IoT device meta-data, including various attributes and public keys.

8 CONCLUSION

Due to the lack of studies of verifiable data management for IoT applications, VERID is designed to satisfy the unique properties and requirements. VERID is designed to resolve the unique requirements of IoT systems. Our innovations include a new authentication data structure PrefixMHT and a novel signature aggregation scheme Condensed Bilinear Pairing. Experimental results show that VERID is much more efficient in memory, update, and time cost than prior works on both sensing devices and data consumers.

ACKNOWLEDGMENTS

The authors are supported by National Science Foundation Grant CNS-1743064, CNS-1717948, CNS-1750704. The authors also thank anonymous IoTDI reviews for their constructive comments and suggestions.

REFERENCES

[1] 2004. Intel Lab Data. http://db.csail.mit.edu/labdata/labdata.html.

[2] 2015. https://github.com/herumi/ate-pairing.

[3] 2015. The 10 Biggest Cloud Outages Of 2015. http://www.crn.com/slide-shows/cloud/300077635/the-10-biggest-cloud-outages-of-2015-so-far.htm/pgno/0/2.

[4] 2015. Secure Hash Standard. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf.

[5] 2018. https://www.opensensors.io/.

[6] 2018. Intel SGX. https://software.intel.com/en-us/sgx.

[7] 2018. Introducing JSON. https://www.json.org/.

[8] 2018. libsnark. https://github.com/scipr-lab/libsnark.

[9] 2018. M3 Open Node. https://www.iot-lab.info/hardware/m3/.

[10] 2018. Navigating a Cloudy Sky: Practical Guidance and the State of Cloud Security. https://www.mcafee.com/enterprise/en-us/assets/executive-summaries/es-navigating-cloudy-sky.pdf.

[11] 2018. OpenSSL. https://www.openssl.org/.

[12] J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, A. Shelat, and B. Waters. 2015. Computing on authenticated data. *Journal of Cryptology* 28, 2 (2015).

[13] Mohannad A. Alswailim, Hossam S. Hassanein, and Mohammad Zulkernine. 2015. CRAWDAD dataset queensu/crowd_temperature (v. 2015-11-20): derived from roma/taxi (v. 2014-07-17). Downloaded from https://crawdad.org/queensu/crowd_temperature/20151120. https://doi.org/10.15783/C7CG65

[14] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A. Sadeghi, and M. Schunter. 2016. SANA: secure and scalable aggregate network attestation. In *Proc. of ACM CCS*.

[15] S. Bajaj and R. Sion. 2013. CorrectDB: SQL engine with practical query authentication. *Proc. of the VLDB Endowment*.

[16] M. Bellare and P. Rogaway. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In *Proc. of ACM CCS*.

[17] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. 2013. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *CRYPTO*.

[18] J. Beuchat, J. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. 2010. High-speed software implementation of the optimal ate pairing over Barreto–Naehrig curves. In *International Conference on Pairing-Based Cryptography*.

[19] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. 2003. Aggregate and verifiably encrypted signatures from bilinear maps. In *In Proc. of EUROCRYPT*.

[20] A. J. Brush, J. Jung, R. Mahajan, and F. Martinez. 2013. Digital neighborhood watch: Investigating the sharing of camera data amongst neighbors. In *Proc. of ACM CSCW*.

[21] W. Cheng, H. Pang, and K. Tan. 2006. Authenticating multi-dimensional query results in data publishing. In *IFIP Annual Conference on Data and Applications Security and Privacy*.

[22] Manuel C. Christian P., Kapil V. 2018. EnclaveDB: A Secure Database using SGX. In *Proc. of IEEE S&P*.

[23] K. Chung, Y. T. Kalai, F. Liu, and R. Raz. 2011. Memory delegation. In *Crypto*.

[24] E. F. Codd. 1970. A relational model of data for large shared data banks. *CACM* 13, 6 (1970).

[25] G. Cormode, M. Mitzenmacher, and J. Thaler. 2012. Practical verified computation with streaming interactive proofs. In *Proc. of ACM ITCS*.

[26] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur. 2015. Geppetto: Versatile verifiable computation. In *Proc. of IEEE S&P*.

[27] K. Fan, S. Liu, and P. Sinha. 2006. Scalable data aggregation for dynamic events in sensor networks. In *Proc. of the ACM SenSys*.

[28] J. Gao, L. Guibas, N. Milosavljevic, and J. Hershberger. 2007. Sparse data aggregation in sensor networks. In *Proc. of ACM IPSN*.

[29] R. Gennaro, C. Gentry, and B. Parno. 2010. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Cryptology*.

[30] M. Gerla, E. Lee, G. Pau, and U. Lee. 2014. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *Proc. of IEEE WF-IoT*.

[31] S. Goldwasser, S. Micali, and R. L. Rivest. 1988. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.* 17, 2 (1988).

[32] M. T. Goodrich, R. Tamassia, and N. Triandopoulos. 2008. Super-efficient verification of dynamic outsourced databases. In *CT-RSA*.

[33] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems* 29, 7 (2013).

[34] T. Gupta, R. P. Singh, A. Phanishayee, J. Jung, and R. Mahajan. 2014. Bolt: Data management for connected homes. In *Proc. of USEIX NSDI*.

[35] J. Han, C. Qian, Y. Yang, G. Wang, H. Ding, X. Li, and K. Ren. 2018. Butterfly: Environment-Independent Physical-Layer Authentication for Passive RFID. In *Proc. of ACM UbiCom*.

[36] C. Ho, R. Agrawal, N. Megiddo, and R. Srikant. 1997. Range queries in OLAP data cubes. In *Proc. of ACM SIGMOD*.

[37] J. Jonsson, K. Moriarty, B. Kaliski, and A. Rusch. 2016. PKCS# 1: RSA Cryptography Specifications Version 2.2. https://tools.ietf.org/html/rfc8017. (2016).

[38] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. 2006. Dynamic authenticated index structures for outsourced databases. In *ProcACM SIGMOD*.

[39] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. 2010. Authenticated index structures for aggregation queries. *ACM TISSEC* 13, 4 (2010), 32.

[40] J. Li, L. Zhang, J. K. Liu, H. Qian, and Z. Dong. 2016. Privacy-Preserving Public Auditing Protocol for Low-Performance End Devices in Cloud. *IEEE Transactions on Information Forensics and Security* 11, 11 (2016).

[41] X. Li, M. Wang, H. Wang, Y. Yu, and C. Qian. 2019. Toward Secure and Efficient Communication for the Internet of Things. *IEEE/ACM ToN* (2019).

[42] Prince Mahajan, Srinath Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Mike Dahlin, and Michael Walfish. 2011. Depot: Cloud Storage with Minimal Trust. *ACM Trans. Comput. Syst.* 29, 4, Article 12 (Dec. 2011), 38 pages. https://doi.org/10.1145/2063509.2063512

[43] R. C. Merkle. 1987. A digital signature based on a conventional encryption function. In *Proc. of CRYPTO*.

[44] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. 2001. Analysis of the clustering properties of the Hilbert space-filling curve. *IEEE TKDE* 13, 1 (2001).

[45] G. M. Morton. 1966. A computer oriented geodetic data base and a new technique in file sequencing. (1966).

[46] S. Nishimura and H. Yokota. 2017. QUILTS: Multidimensional Data Partitioning Framework Based on Query-Aware and Skew-Tolerant Space-Filling Curves. In *Proc. of ACM SIGMOD*.

[47] J. A. Orenstein and T. H. Merrett. 1984. A class of data structures for associative searching. In *Proc. of ACM PODS*.

[48] H. Pang, J. Zhang, and K. Mouratidis. 2009. Scalable verification for outsourced dynamic databases. In *Proc. of the VLDB Endowment*.

[49] D. Papadopoulos, S. Papadopoulos, and N. Triandopoulos. 2014. Taking authenticated range queries to arbitrary dimensions. In *Proc. of ACM CCS*.

[50] C. Papamanthou, E. Shi, and R. Tamassia. 2013. Signatures of correct computation. In *Theory of Cryptography*.

[51] R. Ada Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In *Proc. of ACM SOSP*.

[52] L. Schabhüser, J. Buchmann, and P. Struck. 2017. A Linearly Homomorphic Signature Scheme from Weaker Assumptions. In *IMA International Conference on Cryptography and Coding*.

[53] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish. 2013. Resolving the conflict between generality and plausibility in verified computation. In *Proc. of ACM Eurosys*.

[54] A. J. Smith. 1978. Sequentiality and prefetching in database systems. *ACM TODS* 3, 3 (1978).

[55] C. Wang, Q. Wang, K. Ren, and W. Lou. 2009. Ensuring Data Storage Security in Cloud Computing. In *Proc. of IEEE IWQoS*.

[56] C. Wang, Q. Wang, K. Ren, and W. Lou. 2010. Privacy-preserving public auditing for data storage security in cloud computing. In *Proc. of IEEE INFOCOM*.

[57] G. Wang, H. Cai, C. Qian, J. Han, X. Li, H. Ding, and J. Zhao. 2018. Towards Replay-resilient RFID Authentication. In *Proc. of ACM Mobicom*.

[58] G. Wang, J. Han, C. Qian, W. Xi, H. Ding, Z. Jiang, and J. Zhao. 2018. Verifiable smart packaging with passive RFID. *IEEE TMC* (2018).

[59] H. Wang, X. Li, Y. Zhao, Y. Yu, H. Yang, and C. Qian. 2016. SICS: Secure In-Cloud Service Function Chaining. *arXiv preprint arXiv:1606.07079* (2016).

[60] M Wang, X. Li, S. Shi, and C. Qian. 2019. Collaborative Validation of Public-Key Certificates for IoT by Distributed Caching. In *Proc. of IEEE INFOCOM*.

[61] Z. Xia, X. Wang, X. Sun, and Q. Wang. 2016. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *IEEE TPDS* 27, 2 (2016).

[62] Y. Yang, D. Papadias, S. Papadopoulos, and P. Kalnis. 2009. Authenticated join processing in outsourced databases. In *Proc. of ACM SIGMOD*.

[63] Y. Zhang, L. Duan, and J. L. Chen. 2014. Event-driven soa for iot services. In *Proc. of IEEE SCC*.

[64] Y. Zhang, J. Katz, D. Genkin, D. Papadopoulos, and C. Papamanthou. 2017. vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases. In *Proc. of IEEE S&P*.

[65] Y. Zhang, J. Katz, and C. Papamanthou. 2015. IntegriDB: Verifiable SQL for outsourced databases. In *ACM CCS*.

[66] Q. Zheng, S. Xu, and G. Ateniese. 2012. Efficient query integrity for outsourced dynamic databases. In *Proc. of ACM CCSW*.