An IoT Data Communication Framework for Authenticity and Integrity

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

Huazhe Wang
University of California Santa Cruz
hwang137@ucsc.edu

Ye Yu
University of Kentucky
ye.yu@uky.edu

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

ABSTRACT

Internet of Things has been widely applied in everyday life, ranging from transportation, healthcare, to smart homes. As most IoT devices carry constrained resources and limited storage capacity, sensing data need to be transmitted to and stored at resource-rich platforms, such as a cloud. IoT applications retrieve sensing data from the cloud for analysis and decision-making purposes. Ensuring the authenticity and integrity of the sensing data is essential for the correctness and safety of IoT applications. We summarize the new challenges of the IoT data communication framework with authenticity and integrity and argue that existing solutions cannot be easily adopted. We present two solutions, called Dynamic Tree Chaining (DTC) and Geometric Star Chaining (GSC) that provide authenticity, integrity, sampling uniformity, system efficiency, and application flexibility to IoT data communication. Extensive simulations and prototype emulation experiments driven by real IoT data show that the proposed system is more efficient than alternative solutions in terms of time and space.

CCS CONCEPTS

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

KEYWORDS

IoT, Cloud, Authentication, Partial Data Retrieval, Sampling

ACM Reference format:
Xin Li, Huazhe Wang, Ye Yu, and Chen Qian. 2017. An IoT Data Communication Framework for Authenticity and Integrity. In *Proceedings of The 2nd ACM/IEEE International Conference on Internet-of-Things Design and Implementation, Pittsburgh, PA USA, April 2017 (IoTDI 2017),* 12 pages.
DOI: 10.1145/3054977.3054982

IoTDI 2017, Pittsburgh, PA USA

DOI: 10.1145/3054977.3054982

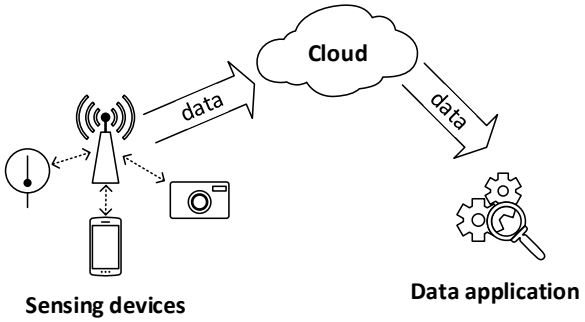


Figure 1: IoT data communication framework

1 INTRODUCTION

Internet of Things (IoT) is being widely applied in a great number of everyday applications such as healthcare [2, 13], transportation [22], smart home [9, 14, 24], and surveillance systems [17, 25]. Internet of Things (IoT) is fast growing at an unprecedented rate: the number of connected IoT sensing devices is expected to reach 8 billion by 2018, predicted by Cisco [29]. IoT devices generate a large amount of sensing data to reflect physical environments or conditions of objects and human beings. As most IoT devices carry constrained resource and limited storage capacity, sensing data need to be transmitted to and stored at resource-rich platforms, such as a cloud. On the other hand, analyzing historical sensing data is essential for decision-making in various IoT applications [14] [32]. For example, Nest Learning Thermostat [14], a system that controls the temperature of a smart home automatically and intellectually, learns a user's preference by analyzing history data of the home. Hence IoT applications retrieve sensing data from the cloud for analysis and decision making purposes. To this end, both state-of-art IoT proposals [23, 24] and industrial practices [11] adopt the centralized data store residing in the cloud, as depicted in Fig. 1.

In this paper we present the design of an IoT data communication framework involving the three key entities: *sensing devices*, *cloud*, and *data applications*. We summarize the following key requirements or challenges of the IoT data communication framework, which distinguish it from traditional data collection and management methods.

1) **Time series data and event data.** IoT sensing data can be classified into two types: time series data and event data [39]. Time

Table 1: Overall comparison of signature schemes.

Signature Scheme	Computational efficiency	Constant space	Partial data retrieval	Sampling uniformity
Sign-each	X	✓	✓	X
Concatenate	✓	X	X	X
Hash chaining	✓	✓	X	X
DTC	✓	X	✓	X
GSC	✓	✓	✓	✓

series data are generated by each device for every fixed time period, such as 1 second. They are used to conduct continuous monitoring tasks such as temperature reports. Event data are generated whenever certain types of events occur, such as a vehicle appearing in a smart camera. They are used to monitor discrete events. Note event-based data are more difficult to manage than time series. *Hence this paper focuses on finding a solution for event-based data. The proposed methods can be easily adjusted for time series data.*

2) **Data sampling.** A common but critical problem shared by state-of-art IoT designs is that the resources for transmitting and storing data (e.g. network bandwidth, storage quota) are limited in the presence of massive IoT data. By 2022, IoT data are expected to constitute 45% traffic in the Internet [20]. Cloud providers charge users for storage, retrieval and transferring of data [12]. It is desired to have *predictable cost* for both users (in finance) and the cloud (in resource) [18]. Hence only a fixed resource budget can be allocated to the sensing data over a time period, called an *epoch*. For example, the cloud can only keep up to 100 data records from any device during any minute. Data sampling is the approach to make sensing data within a fixed budget. To guarantee the sampling quality, every event should have an equal probability to be sampled among all events in the epoch, which is called the *uniformity* property [18].

3) **Authenticity and integrity.** Since the sensing data are stored in a third-party cloud, they could be corrupted by outside attackers, malicious cloud employees [27], transmission failures, or storage loss [8]. Therefore, data authenticity and integrity, which guarantee that data are from these sensing devices and has not been modified or partially dropped, are important for trustworthy IoT applications [34]. Without data authenticity and integrity, IoT applications may make wrong decisions and cause economic and human-life losses. Authenticity and integrity should be *verifiable* by data applications.

4) **Flexible application requirements.** Different applications may have different requirements on sensing data granularity. For example, applications like self-driving cars need fine-grained road information, while other applications like road-traffic estimation only need a few sampled data. Even if the cloud can store up to 100 records, some applications only retrieve part of them, e.g., 10 records, due to bandwidth/memory limit, or application requirements. A possible attack is that a malicious cloud operator may selectively send partial data to a user, e.g., those outlier data, and lead the user to a wrong decision. Hence we require the partial data should have verifiable authenticity, integrity, and uniformity. We call this feature as *partial data retrieval*.

There is no existing solution that can address all these challenges. Since sensing devices and data applications do not communicate directly in sessions, existing secure transport protocols cannot be used. *Digital signature* is a widely used method to protect data authenticity and integrity: The sender first computes a message digest D by hashing its original message m using a cryptographic hash function H, $D = H(m)$. H is also called message digest function. Note the length of D is significantly shorter than that of m. Then the sender uses its private key pk^{-1} to encrypt D and attach the *signature* $E_{pk^{-1}}[D]$ to the original message. When the receiver gets m' and $E_{pk^{-1}}[D]$, it decrypts $E_{pk^{-1}}[D]$ using the public key of the sender and verifies whether $D = H(m')$. However, applying digital signature to every sensing record, called the *Sign-each* method, is not practical, because public-key encryptions/decryptions are considered slow and expensive, especially for sensing devices with limited resources. A more efficient method, *concatenate*, is to compute the message digest D for a large number of records and sign once on D. This approach requires each sensing device to cache all records and has the all-or-nothing feature: If some applications only require part of the records, the signature cannot be verified. A well-known method to sign a data stream is hash chaining [21]. However it does not fit the IoT framework either, because event-based sampling and partial data retrieval will break the chain and make the signature unverifiable.

We summarize our contribution in this paper as follows.

1) Our first effort is to extend a well-known digital signature, Merkle tree [28, 37], to a method Dynamic Tree Chaining (DTC) that can be used in the IoT data communication framework. DTC supports verifiable authenticity and integrity for event-based sampling and partial data retrieval. However, it does not guarantee sampling uniformity.

2) Our main contribution is an efficient digital signature method, Geometric Star Chaining (GSC), a specific design for the IoT data communication framework. GSC allows each sensing device to sign only once for all data records in an epoch and provides verifiable authenticity, integrity, and uniformity for partial data retrieval. The signing and verifying throughput of GSC are significantly higher than that of DTC. We compare GSC, DTC, and other possible methods in TABLE 1.

3) We also investigate the problem of shared budget constraint for a group of sensing devices and extend GSC to resolve this problem

4) We conduct excessive simulations and prototype emulation experiments. The results show that the proposed system is efficient and practical.

The rest of this paper is organized as follows. We present the problem statement in Sec. 2. We describe the system design details and extend it to incorporate budget limit in Sec. 3 and Sec. 4 respectively. Security analysis is presented in Sec. 5. We conduct extensive simulations and prototype emulations in Sec. 6. Sec. 7

presents related work. Some practical issues are discussed in Sec. 8. Finally, we conclude this work in Sec. 9.

2 PROBLEM STATEMENT

2.1 Network Model

We demonstrate the life cycle of IoT sensing data in Fig. 1. Three different kinds of entities are identified as follows.

- **Sensing devices** are distributed electronic equipments that generate IoT sensing data. They usually have limited computation, memory, and power resources.
- **Cloud** is an ISP or a third-party cloud provider who has rich resources and expertise in operating cloud computing services. It charges clients for data storage and data access.
- **Data applications** are software systems or devices that may request to retrieve the sensing data for analysis purposes. Different data applications may have different data requirements.

Sensing devices generate data reports when events of interests happen. A data source is modeled as an unpredictable sequence of events. Due to resource budget constraints (e.g. network bandwidth and storage quota at the cloud), not all events are actually reported or stored in a given period of time (i.e. an epoch). However, all events should have a uniform probability to be sampled and stored. An application may fetch all or a fraction of data from the cloud of an epoch to conduct postprocessing based on their requirements.

A formal description of the network model is presented as below. For device i, its source event data are represented as $E_i = (e_i^1, e_i^2, ..., e_i^n, ...)$. Event e_i^j is generated by device d_i at time t_i^j in an online fashion. That is, $t_i^j < t_i^{j+1}$ and the value of time stamp t_i^j is unpredictable until event e_i^j occurs. Suppose the k-th epoch starts at time T_s^k and ends at T_e^k. We define $E_i^k \triangleq \{e_i^j \in \|E_i\| : T_s^k \leq t_i^j < T_e^k\}$ as the set of events monitored by device d_i during the k-th epoch [1]. Let B be the budget limit for each epoch. Due to the budget constraint, only n_i events are sampled from device i during every epoch such that $\sum n_i \leq B$.

2.2 Threat Model

We assume only IoT sensing devices and data applications are trustworthy and any entities in between are subject to attack or may perform functionalities in a dishonest way.

While clients trust cloud providers to perform their services correctly, there are increasing concerns about the security of outsourced data. The security threats may be attributed to management errors or software/hardware bugs which lead to Byzantine failures. A recent report [8] describes massive cloud service outages that affect many companies and result in data corruption. Even worse, there may exist adversaries and hackers who have gained accesses to data on clouds and are able to manipulate or delete clients' data without being detected by cloud providers.

The goal of this paper is to allow IoT applications to have the capability to verify the authentication and integration of the stored sensing data and support partial data retrieval. *We do not address the issue of data confidentiality or privacy in this paper.* They are orthogonal to the problem we study in this paper and there are many existing works focusing on these issues in the cloud [35] [26].

[1] $\|E_i\|$ denotes the set interpretation of sequence E_i.

Table 2: Important Notations.

Notation	Definition
D_i	Message digest of sample block i or event i
D_{ij}	Message digest summarizing ith till jth events
$H(\cdot)$	Message digest function
pk	Public key
pk^{-1}	Private key
$E_{pk^{-1}}[\cdot]$	Encrypt using private key pk^{-1}
$E_{pk}[\cdot]$	Decrypt using public key pk
n	Number of events monitored
K	Number of sensing devices
S_i	Numerical interval between 2^{-i-1} and 2^i
$h(\cdot)$	Hashing function whose range is between 0 and 1
B	Budget limit

3 SYSTEM DESIGN

In this section, we discuss how the IoT data communication framework should be designed to address all challenges presented in Sec. 1.

3.1 Existing Signature Schemes

Digital signature is widely used to ensure data authenticity and integrity. However, none of existing signature schemes are appropriate for the IoT scenario described in Sec. 1.

First, the Sign-each method causes expensive computational cost on both signer/verifier sides owe to excessive public-key encryption and decryption operations. It is known that public-key encryption and decryption is much slower and more energy-consuming than symmetric-key encryption/decryption and cryptographic hashing. For example signing one short message using RSA with a 1024-bit key consumes approximately 360mWs and takes about 12 seconds on one popular wireless sensor network platform, while computing SHA-1 of the same message consumes less than 1mWs [30]. Furthermore, the Sign-each method may not be able to detect data loss.

The *concatenate* signature scheme can amortize the signing and verification cost to multiple messages, but it is not suitable for sensing devices which may be lack of buffer space to accommodate all messages. In addition, it does not support partial data retrieval.

Hash chaining [21] reduces the buffer space complexity from $O(m)$ to $O(1)$ for both the signer and verifier, where m is the number of messages buffered in the sensing device to be jointly signed. In hash chaining signature scheme, only the first message is signed and each message carries the one-time signature for the succeeding message. However, hash chaining fails when some events are dropped due to sampling or partial data retrieval.

To address the aforementioned problems when applying digital signature in the IoT scenario, we propose two signature schemes. 1) the Dynamic Tree Chaining (DTC) that is developed based on Merkle tree [28]. DTC serves as the baseline in this paper. 2) a novel

signature scheme specifically designed for the IoT data communication framework, called Geometric Star Chaining (GSC). GSC outperforms DTC in terms of throughput and memory consumption. GSC provides verifiable uniformity while DTC does not.

3.2 Dynamic Tree Chaining (DTC)

We start from the Tree chaining designed by Wong and Lam [37], one variation of Merkle tree [28]. The digest of each event report is one leaf node in the binary *authentication tree* presented in Fig. 2. The value of any internal node is computed as the hashing of the concatenation of its two children. Take the authentication tree in Fig. 2 as an example. D_{12} is the parent of D_1 and D_2 and $D_{12} = H(D_1||D_2)$, where $H(\cdot)$ is the message digest function, such as SHA-1 [10] or MD5 [1]. Likewise, $D_{14} = H(D_{12}||D_{34})$ and $D_{18} = H(D_{14}||D_{58})$. As a result, the root summarizes all the leaf nodes. The root node is regarded as the block digest and is signed by the private key to create the block signature.

The verification process is on a per-event basis. In order to verify the authenticity/integrity of an event e, the verifier requires the block signature, the position of event e in the authentication tree and the sibling nodes in the path to the root, which are all appended to event e. Basically, the verification algorithm is to replay the process to build the authentication tree and to verify the nodes in the path to the root. Image the receiver begins to verify event e_3 which is represented as the dashed circle in Fig. 2. First, the receive computes $D'_3 = H(e_3)$ and then its ancestors in order: $D'_{34} = H(D'_3||D_4)$, $D'_{14} = H(D_{12}||D'_{34})$, $D'_{18} = H(D'_{14}||H_{48})$. Event e_3 is verified if the decrypted block signature equal D'_{18}, that is to say $E_{pk}\left[E_{pk^{-1}}[D_{18}]\right] = D'_{18}$, where $E_{pk^{-1}}[\cdot]$ denotes singing using private key and $E_{pk}[\cdot]$ is the function to decrypt signature with public key. In this case, all the nodes in the path as well as their siblings are verified and they could be cached to accelerate the verification process. Suppose the event e_4 arrives after e_3 have been verified. Event e_4 is verified directly if $H(e_4) = D_4$.

Note that the expensive encryption operation is amortized to all events in one authentication tree and thus tree chaining is computational efficient. More importantly, since every single event is verifiable in tree chaining, it is fully compatible with partial data retrieval. The most severe issue that impedes the adoption of the original tree chaining in IoT environment is that all events should be buffered in the sensing device before the building of the authentication tree, since each event ought to be appended auxiliary authentication information from the authentication tree.

We observe that introducing the cloud can greatly reduce the memory footprint at sensing devices. The sensing device only maintains the message digest of each event and stores all events to the cloud directly without caching. At the end of each epoch, with all leaf nodes available, the sensing device build the authentication tree, which is then sent to the cloud immediately. The cloud in turn attaches essential authentication information to each event just received. The memory footprint can be further optimized if the authentication tree grows in an online fashion and the sensing device transmits to the cloud the nodes that are no longer needed for calculating the remaining authentication tree. We reuse Fig. 2 to illustrate the online authentication tree building process. $D_1 - D_8$ represent the message digests of the events in timely order. D_{12} is calculated immediately when D_2 comes into play. In the meantime, D_1 and D_2 cached in the sensing device are transmitted to the cloud. Likewise, when D_4 is available, D_{34} is computed, which in turn immediately contributes the calculation of D_{14}. As a result, at that time D_3, D_4, D_{12} and D_{34} are dismissed from the sensing device. It is not hard to imply that this optimization reduces the space complexity in the sensing device to host nodes of authentication tree from $O(n)$ to $O(\log n)$, where n denotes the number of events monitored in one epoch.

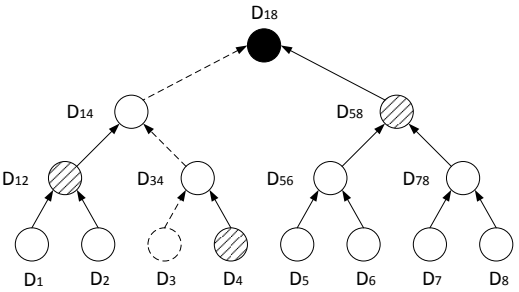


Figure 2: Illustration of tree chaining. Verifying D_3 requires sibling nodes in the path to the root (D_4, D_{34}, D_{58}), signature of the root ($E_{pk^{-1}}[D_{18}]$) and the position of e in the tree.

Nevertheless, the number of generated events is unpredictable and may be unbounded. Once the buffer in the sensing device is full, the root node in the authentication tree is signed and the remaining nodes are flushed to the cloud to spare space for upcoming events. Therefore, one sensing device may apply digital signature more than once in one single epoch. The verifier also requires additional space to cache the verified nodes. The verifier stops caching new verified nodes when the buffer is full.

As a result, the buffer space constrains the performance of DTC, which is a particularly severe problem in IoT environment where most devices possess little buffer space. More importantly, *DTC provides no verifiable uniformity for event sampling and partial data retrieval*, because data selection is completely executed in the cloud. If the cloud selects event samples or the partial data with bias, data applications are unaware of it.

3.3 Geometric Star Chaining (GSC)

We propose a more efficient and secure data communication framework in this paper, called Geometric Star Chaining (GSC). The basic idea of GSC is inspired by one observation that any arbitrary fraction value can be represented or closely approximated by a few number of binary digits. For instance, $5/8 = (0.101)_2$. Thus, partial data with sample rate p, where $p = \sum 2^{-b_i}$, is equivalent to the union of multiple data blocks each corresponds to one set bit in the binary representation. One data block is called a *sample blocks* in this paper. For instance, to retrieve a sampled data from all sensing data from a device within an epoch with sampling rate 5/8, the cloud can send the data application two blocks containing (approximately) 1/2 and 1/8 of the data respectively.

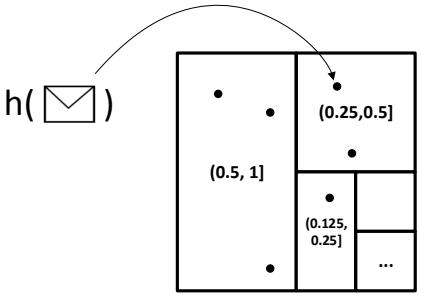


Figure 3: Visualization of numerical intervals.

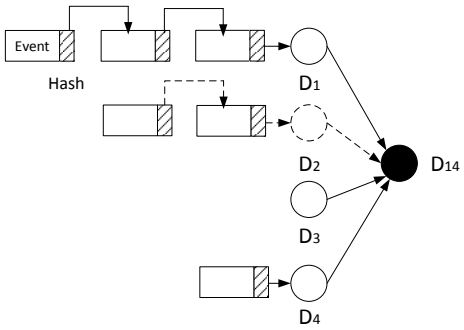


Figure 4: Illustration of GSC. Verifying the second sample block requires all events within it as well as D_1, D_3, D_4 and signature of the root ($E_{pk^{-1}}[D_{14}]$).

The events included in the sample blocks are in *geometric distribution*. Each sample block should draw events uniformly from the IoT data stream. In order to ease the presentation of how sample blocks form, we define a set of successive numerical intervals $\{S_i\}$ where $S_i \triangleq \{x \in \mathbb{R} : 2^{-i-1} < x \leq 2^{-i}, i \in \mathbb{N}\}$, which are visually represented as rectangles in Fig. 3. On receiving a new event e, the sensing device computes which numeric interval in $\{S_i\}$ that $h(e)$ falls in and event e is inserted into the corresponding sample block, where $h(\cdot)$ is a non-cryptographic uniform random hashing function and $\forall x : 0 \leq h(x) \leq 1$.

Events within a same data block are either completely retrieved or not retrieved at all. Thus we can view each data block as an atomic "giant event". GSC computes one message digest for every block and concatenates these digests to a single digest for digital signature, as is depicted in Fig. 4. The digest of each sample block is computed in an online fashion. One variable D_i is allocated to each sample block to capture the newest value of the message digest. Suppose a new event e is observed at the device which belongs to the ith sample block. The message digest is updated as $D_i = h\,(h(e)||D_i)$. This online updating proceeds until the end of the epoch. At last, the concatenate approach is applied to all the message digests $\{D_i\}$. Note the value i, which indicates the sampling rate of each block, should also be stored and hashed with the block. In this way, the application that receives the block can verify the sampling rate.

In fact, any random function can be used to implement the geometric distribution for GSC, such as continuous coin-tossing, but using a uniform random hash is convenient. One practical issue about hashing is that the raw output of hashing functions is one finite-length bit sequence. Computing which numerical interval in $\{S_i\}$ that $h(e)$ fall in is equivalent to counting leading zeros (CLZ) in that bit sequence, which is intrinsically supported in many hardware platforms including X86 and ARM. Therefore, $|\{S_i\}|$ and hence $|\{D_i\}|$ are bounded by the size of the bit sequence. For the case of xxHash64 [16], this function produces 64-bit hash values and thus $|\{S_i\}_{0\leq i\leq 64}| = 65$ and $|\{D_i\}_{0\leq i\leq 64}| = 65$. It is evident that space cost for this signature scheme at the sensing device is constant.

If the number of events exceeds the budget for a sensing device, the cloud may drop sample blocks starting from $i = 0$ until the remaining blocks fit the storage budget. Note uniformity is still preserved because all events are equally likely to fall into any sample block.

3.4 Data Retrieval and Verification

A sampled fraction of sensing data is usually sufficient for most IoT applications [6]. In the network model presented in Sec. 2.1, an application requests for a certain fraction of events observed at a particular sensing device from the cloud. GSC provides verifiable authenticity, integrity, and uniformity for partial data retrieval with an arbitrary sampling rate.

Based on the application requirement, a data application first determines the maximum number of events of each sensing device for an epoch it wants to receive, called a portion number. It then sends all portion numbers to the cloud. For each portion number, the cloud converts it to a sampling rate p and constructs the binary expression of p, such that $p = \sum 2^{-b_i}$ where b_i is the position of the i-th 1 in the binary expression of p. Then the cloud sends the corresponding sample blocks to the application. For example, if the application requests for data with a sample rate of $5/8 = (0.101)_2$, it should fetch two sample blocks correspond to sample rate of 2^{-1} and 2^{-3} respectively. The message digests associated with all sample blocks from one epoch are stored in a single file, such that it is convenient for the application to access the necessary information to verify the data. This step verifies the following properties. 1) The received blocks were not modified or partially dropped and 2) The data were indeed uniformly sampled based on the given sampling rates.

Compared to DTC, GSC requires smaller buffer size on each sensing device and the data application. It also provides verifiable uniformity which DTC does not. In addition, retrieving GSC-signed IoT data from the cloud can be achieved by sequential reads which are much faster than random reads [5] [7].

We dot not discuss how the data is stored in the cloud internally, as it is orthogonal to our framework design.

4 INCORPORATING BUDGET LIMIT

IoT data volume is growing in an unprecedent speed over the years. With ever-increasing volume of IoT data, storing all raw IoT data in the cloud poses a heavy monetary burden on the users. Since a small fraction of uniformly sampled IoT data satisfy most IoT

applications, we identify the necessity to uniformly sample IoT data before storing them in the cloud.

4.1 Sampling Protocol Design

In this section, we describe a sampling protocol taking the budget limit into consideration, which respects the network model described in Sec. 2.1. This sampling protocol introduces a new entity, called a *coordinator* (sometimes it is also called a *hub*), in the network model. One coordinator is a software working as a sampler which sits between the sensing devices and the cloud. A coordinator can be installed on an access point or a server at the edge of Internet. It maintains communications with all sensing devices on behalf of the cloud and temporarily buffers IoT data samples.

We focus on one single epoch in this discussion since at the beginning of each epoch, the *sampling protocol* (SP) is reset to the initial state. At the end of each epoch, the coordinator signal all sensing devices to advance to the next epoch. The straightforward solution is to buffer all the events in the coordinator and uniformly sample them based on the budget limit. However, the number of these events could possibly be very large, and therefore the storage capacity of the coordinator may be not enough to accommodate them all. Thus, a sampling protocol with space bounds for both the sensing device and the coordinator is desired. The challenge of such sampling protocol design derives from the combination of the distributed setting and the unpredictability of streams. If only one stream of data is considered, the problem is regressed to classic *reservoir sampling* [33], which has been studied extensively in the literature. Also, as long as the number of elements in each stream of data is known in advance, the central coordinator can decide how many samples are allocated to different sensing devices, each of which runs an instance of reservoir sampling.

To this end, we utilize the recent study in distributed streams [18] and design an efficient sampling protocol based on it. The basic idea of this sampling protocol is to dynamically maintain events with the smallest hash values on the coordinator. As long as the hashing is uniform and random but not necessary to be cryptographic, the events maintained in the coordinator are drawn uniformly from all events already observed hitherto. In order to obviate unnecessary network bandwidth consumption for the events which should be discarded at the sensing devices locally, the coordinator broadcasts to all sensing devices current global B-th smallest hash value τ, where B is the sampling budget. One strawman sampling protocol is that the coordinator broadcasts the new value of τ every time it changes. Since τ changes $O(B \log \sigma)$ times, the communication cost between the coordinator and the sensing devices is $O(KB \log \sigma)$, where σ denotes the total number of events sent to the coordinator, K is the number of sensing devices. Cormode et al. proposed a distributed sampling algorithm [18], which is proved to be optimal in terms of communication cost, which is $O(K \log_{K/B} \sigma + B \log \sigma)$ with high probability. We tailor it to fit the signature scheme in Sec. 3.3 in this paper as demonstrated follows.

The coordinator as well as the sensing devices maintain a variable which represents which round the sampling protocol is in, and the coordinator ensures that all devices are kept up to date with this information. Initially, the sampling protocol begins at round 0. Suppose the sampling protocol is at round j. As we will see, round j indicates a sample rate of 2^{-j}. This protocol involves two algorithms at the sensing device and the coordinator respectively.

Algorithm 1: SP at sensing device k in round j

```
foreach event e do
    i ← argmin_{x∈ℕ} {h(e) ≥ 2^{-x-1}};
    l_i^k ← l_i^k + 1;
    if i ≥ j then
        Forward e to the coordinator;
    else
        Discard e;
    end
end
```

Sensing device: On receiving a new event e, the sensing device first computes which numeric interval in $\{S_i\}$ that $h(e)$ falls in, and updates the *local counter* associated with this set, where $h(\cdot)$ is a uniform random hashing function shared by all the sensing devices and the coordinator and. Let l_i^k be the local counter for S_i at device k. Each sensing device and the coordinator maintain their own local counters. The local counters at the sensing devices are for the security issues only, which will be discussed in Sec. 5 and the sampling protocol still works correctly without these counters. It is worthy mention that all sensing devices and the coordinator use the same hashing function. Suppose $h(e) \in S_i$. If $i \geq j$, the device instantly forward event e to the coordinator; otherwise, the event is discarded locally. Note that no events are buffered at the device in any cases. The pseudo-code is shown in Algorithm 1.

Coordinator: The coordinator maintains queues $\{Q_i^k\}$, each of which corresponds to one numerical interval in $\{S_i\}$ of each sensing device. Upon receiving an event e, the coordinator first computes i, such that $h(e) \in S_i$, followed by comparing the value of i and j. In the case of $i < j$, event e is discarded; otherwise, it is buffered at queue Q_i^k (suppose the event is from kth sensing device) followed by updating both the counter associated with numerical interval S_i and the global counter g, which records the total number of events buffered at the coordinator. At this moment, as long as the value of the global counter g exceeds the budget limit B, all event queues associated with S_i are discarded, the global counter updates accordingly and the sampling protocol advances to the next round (i.e. $j \leftarrow j + 1$). The coordinator then signals all sensing devices to promote to the newest round j. It is evident that coordinator buffers at most $B + 1$ events all the time. Algorithm 2 is the pseduo-code for the coordinator part of this sampling protocol.

4.2 Copping with Network Latency

In the last subsection, it is assumed that all communications between the coordinator and the sensing device are instantaneous. However, it is not the case in real networks and the network latency attributed to propagation and processing is inevitable. Consequently, different sensing devices and the coordinator miss synchronization when the coordinator signal all sensing devices to advance to the next epoch. To be more specific, at one single time point, different sensing devices may not stay at a same epoch. We propose that one epoch identifier is appended to each event sent

Algorithm 2: SP at the coordinator in round j

```
1  foreach event e do
2      i ← argmin_{x∈ℕ} {h(e) ≥ 2^{-x-1}};
3      k ← e.source;
4      if i ≥ j then
5          Q_i^k.add(e);
6          l'_i ← l'_i + 1;
7          g ← g + 1;
8          while g > B do
9              Discard queues {∀k̂, Q_j^k̂};
10             g ← g − l'_j;
11             j ← j + 1;
12             Broadcast j to all sensing devices;
13         end
14     else
15         Discard e;
16     end
17 end
```

to the coordinator to resolve the confusion. The coordinator stores the data of a particular epoch to the cloud only after the successful reception of acknowledgments from all sensing devices to advance to the next round. The coordinator should be able to buffer events from more than one epoch simultaneously.

Note that the coordinator also signals all sensing devices to promote to the newest round (line 12, Algorithm 2). Lagged round promotion does not impact the eventual correctness of the sampling protocol, even though a lot of network bandwidth is wasted owe to the transmission of events that should be discarded at devices locally.

4.3 Signature Scheme

Besides the basic requirement to support partial data retrieval, the choice of signature scheme is constrained by the fact that the coordinator may discard events as shown in Algorithm 2. Hash chaining can not coexist with the sampling protocol, because the coordinator is allowed to discard events that are essential for the verifier to validate the received data.

Both the sensing device and the coordinator are sensitive to space consumption. Since the space consumption for the events themselves and the sampling protocol is the same, we concentrate on the space cost for different signature schemes. Moreover, at the coordinator we assume that events are stored in the disk and not consume memory space. We go into details on the space complexity of different signature schemes that are compatible with the sampling protocol, as shown in TABLE. 3, where n is the number of events monitored at one device and n' denotes the maximum value of n among all the sensing devices. The space complexity of DTC at the coordinator is $O(B \log n')$ because there are $O(B)$ events buffered at the coordinator and in the worse case each event is appended $O(\log n')$ with hash values for verification. Following the same line of reasoning in Sec. 3.2, the lack of buffer space in the coordinator may significantly degrade the performance of DTC. GSC only requires $O(1)$ space at the coordinator because no message digest is maintained at the coordinator.

Table 3: Space Complexity of signature schemes.

Signature Scheme	Device	Coordinator
Sign-each	O(1)	O(B)
DTC	O(log n)	O($B \log n'$)
GSC	O(1)	O(1)

4.4 Data Retrieval

The process for the data application to fetch data from the cloud is very similar to the procedures described in Sec. 3.4, except for that there may be not enough data in the cloud to satisfy the data application. Nonetheless, the sampling protocol results in the cloud containing sample blocks that correspond to successive numerical intervals. Therefore, the data application can fetch any fraction of data that is stored in the cloud.

5 SECURITY GUARANTEE

We use digital signatures to verify data integration and authentication. Any inconsistency in the verification procedure indicates data in the cloud untrusted. In the sampling protocol, each sensing device maintains a counter to record the number of events that fall in a certain sample block. At the end of each epoch, the sensing device signatures both sampled events and all counters it maintains. Meanwhile, each sensing device is required to sign its counters even if it has not generated any event during an epoch. With the signature, an attacker cannot manipulate, delete or produce fake samples by modifying contents of events and counters without being detected.

The sampling protocol also defends against dishonest coordinators which execute protocols incompletely. A compromised coordinator may stop monitoring sensing devices by intentionally setting a negligible sample rate. In the verification of this sampling protocol, this kind of violation will be easily detected since the supposed final round is uniquely determined by the values of counters from all sensing devices.

Theorem 5.1. *The final round is uniquely determined by the values of counters from all sensing devices.*

Proof. Suppose the sampling protocol stops at round j at the end of the epoch, which means all events associated with sample block i are finally stored in the cloud, where $i \geq j$. Therefore, $\sum_k l_i^k = l'_i$ $(i \geq j)$ and hence

$$g = \sum_i^{i \geq j} l'_i = \sum_{k,i}^{i \geq j} l_i^k \leq B \tag{1}$$

On the other hand, the necessary condition for sampling algorithm to promote from round $j-1$ to j is that $\sum_i^{i \geq j-1} l'_i > B$. Since $\forall i : \sum_k l_i^k \geq l'_i$,

$$\sum_{k,i}^{i \geq j-1} l_i^k \geq \sum_i^{i \geq j-1} l'_i > B \tag{2}$$

Combining Eq. (1) and Eq. (2), it is not hard to reach

$$j = \underset{\hat{j}}{\operatorname{argmin}} \left\{ \sum_{k,i}^{i \geq \hat{j}} l_i^k \leq B \right\} \tag{3}$$

As a result, the final round and therefore the sample blocks should be stored in the cloud can be computed from the counters. □

Inconsistency between the calculated stop round and the actual stored sample blocks will lead to an alert. This operation prevents the coordinator from neglecting events from a sensing device on purpose since counters from the device indicate the number of events that should be sampled by the coordinator.

Theorem 5.2. *The counters determine the events that should be stored in the cloud.*

Proof. By Theorem 5.1, the counters can be leveraged to compute the final round, which in turn determines the range of $h(\cdot)$ that is finally accepted in the cloud. Suppose the final round is j, any event e satisfying $h(e) \leq 2^{-j-1}$ should be stored in the cloud. Consequently, the events that should be stored in the cloud are deterministic. □

Theorem 5.3. *Uniformity is auditable.*

Proof. The sampling protocol picks events by random and uniform consistent hashing function which ensures uniformity. The receiver can check the hash value of the retrieved event to testify whether the sensing devices honestly insert event data to corresponding sample blocks according to their hash value. A sensing device increasing the sample rate by cheating on the hash value can be easily detected. □

Note that the aforementioned theorems hold whether DTC or GSC is leveraged.

6 EVALUATION

We conduct extensive trace-driven simulations and prototype experiments using real dataset in this section.

6.1 Experiment Setup and Methodology

The dataset [3] used in this section includes a wide variety of data collected from the sensing devices at three real homes from May 1st, 2012 through July 31st, 2012 (the data for May 31st, 2012 and June 26th, 2012 are missing). We select 7 sources of data comprising of time series data and event-based data generated at sensing devices: environmental information (including temperature and humidity, etc.) about homeA, homeB and homeC respectively; electrical data from dimmable and non-dimmable switches for homeA; two sets of operational data on door and furnace on/off for homeA; the data from the motion detector located at homeA. We ignore the other 5 sources of data, which are from energy meter readings because energy meters in smart grid usually utilize dedicated communication infrastructure to deliver meter readings [40]. Moreover, sizes of these data are much larger than the 7 data sources aforementioned and thus these data would overwhelm the whole system and few events from the 7 smaller-sized data sources would be sampled if the other 5 sets of data are included in the evaluation trace. We left setting the weight for each sensing device as our future work.

In both the simulation and prototype emulation experiments, non-cryptographic 64-bit xxHash [16] is leveraged as the hashing function, which performs on the entire event report (which is one line of record). We simulate the sampling protocol driven by the 7 sets of data listed above. In the simulation, each day is one epoch and the 7 sources of data from the same day are replayed simultaneously. We vary the budget limit and evaluate its impact on the simulation results. We implement the prototypes of DTC and GSC for performance comparison. In the prototype experiment, we first test the signing and verifying performance without sampling protocol involved. We next conduct prototype experiment in a setting where sampling protocol is involved. The second prototype experiment focuses more on the potential maximal throughput of tested signature schemes, since other impacting factors (e.g. space) are explored in the prior prototype experiment. In the second prototype experiment, the events sent to the coordinator (which may be discarded later at the coordinator) are used for signing performance evaluation whereas the verification algorithm is fed by the events saved at the cloud.

We utilize OpenSSL [15] to implement two widely-used asymmetric encryption algorithms, RSA [31] and DSA [4]. MD5 [1], SHA-1 or SHA-256 [10] are leveraged as the message digest function. The prototype emulation experiments are conducted on a quadcore@3.40G Linux desktop with 32GB memory and only one core is used.

6.2 Simulation Result

We first conduct one micro-scale experiment to illustrate how the sampling protocol proceeds when new events arrival, as depicted in Fig. 5. We fix the budget limit to 500 events in this micro-scale experiment. The three lines in Fig. 5 represent the number of events buffered at the coordinator, sent to the coordinator by all the 7 sensing devices and monitored at all sensing devices, respectively. The three lines vary against time in one day (May 1st, 2012). Initially, the number of events are the same for the three lines until the number of buffered events at the coordinator reaches the budget limit. At this time, approximately half buffered events are discarded, illustrated as the first vertical drop in Fig. 5. The events at coordinator then are accumulated over time until the next sharp decrease. This process repeats down to the end of this epoch. It is evident that the space used at the coordinator never exceeds the budget limit. It is worthy mention that the total number of events sent to the coordinator grows slower as time proceeds, which is a desirable property since the communication cost stays low even if much more events are monitored. Moreover, it seems that the number of events sent to the coordinator is more related to the budget than the total number of monitored events. The formal proof is left as future work. Lastly, it can be inferred from Fig. 5 is that the event occurrence rate is not constant; otherwise, more trivial solutions such as fixing sampling rate at each device are enough for the problem we are addressing in this paper.

Next, we investigate how different values of budget limit impact the number of events eventually saved to the cloud. We present the number of events saved at the cloud each day from May 1st, 2012 till

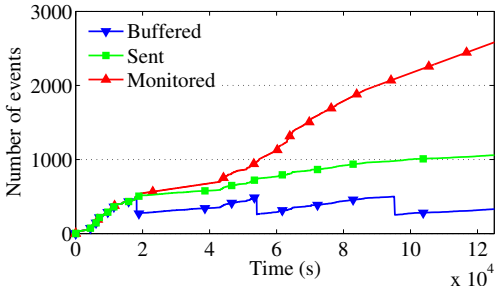


Figure 5: One-day micro-scale experiment unveiling sampling protocol.

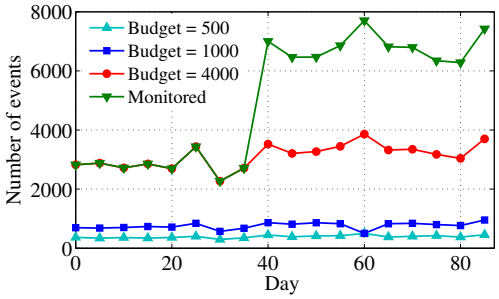
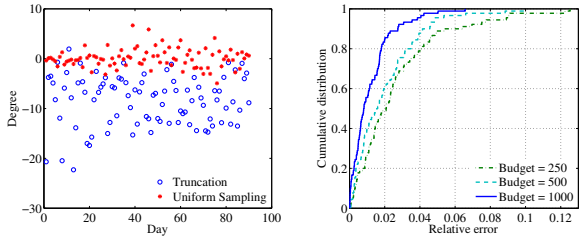


Figure 6: Number of events saved in the cloud.

July 31st, 2012. with different values of budget limit in Fig. 6. From the description of the sampling protocol, we can infer that the final number of saved events is not necessarily equal to the budget limit. Fig. 6 shows that this sampling protocol utilizes approximately 75% of the budget on average for different budget values. In Fig. 6, we also demonstrate that this sampling protocol works correctly in the presence of drastic changes, as the number of events monitored soars on the 40th day. In this case, the sampling protocol does not violate the budget constraints.

The underlying foundation of our sampling protocol is that uniformly sampling is ensured. We will see the importance of uniformity in one real application. The temperature sensor periodically measures the environmental temperature and sends the sensing data to the cloud for archiving purpose. We calculate the average temperature outside homeA each day based on the sampled data saved at the cloud. The ground truth is the mean of all temperate sensing data from the temperature sensor. In order to illustrate the need for uniformity, we calculate the average temperature by the first 40 truncated sensing data (which is greater than the number of saved data in the cloud on most days in this simulation experiment). Fig. 7(a) demonstrates how estimated average temperature deviates from the ground truth in respect to using our proposed sampling protocol and using naive truncation. The budget limit is first fixed to 500. It is obvious that the average temperatures calculated by uniformly sampled data are much more accurate in reflecting the real data. In this example, the truncated data is measured in the morning. Thus, the average temperatures calculated by truncated data is smaller than real average temperatures on nearly all days



(a) Deviation from the ground truth (budget = 500)

(b) Impact of budget limit

Figure 7: Computing average temperature from data saved in the cloud.

(only one day is an exception). We then evaluate how the budget limit affects the accuracy of estimated average temperature. As expected, a greater value of budget limit yields more accurate results, as illustrated in Fig. 7(b).

6.3 Emulation Experiments Without Budget Limit

We conduct extensive prototype emulation experiments in this subsection. The efficiency of the signature scheme greatly impacts the adoption of sensing devices, since most sensing devices are resource-constraint. As an indirect measurement of power consumption, we evaluation the speed of signing under different parameter settings. We also evaluate the performance at the verifying phase. The data applications may fetch data of hundreds or thousands of devices, therefore the verifying speed is also critical for a scalable application.

The 7 data sources each divided into 90 epochs are the input to the singing phase of the signature scheme, whose output feeds the verifying phase afterwards. In the first prototype experiment, no budget constraints are involved. The parameter space consists of the space available at the signer/verifier as well as the sampling rate of the data application. The space cost at both the signer and the verifier to host the events themselves is orthogonal to the choice of signature scheme. Thus, the space cost in this subsection is in particular referred to the memory footprint of the signature scheme. It can be implied from the algorithm descriptions in Sec. 3 that the memory usage for all signature schemes mentioned in this paper is a multiple of the message digest. In order to simplify the presentation, we refer one unit of space cost as the memory space used for storing one message digest. DSA is applied as the public encryption/decryption algorithm and MD5 is leveraged to compute the message digest in this subsection.

We measure the performance of DTC and GSC with given space in the signer. To focus on the impact of the space issues at the signer side, at first we allocate enough free space to the verification process. If DTC is used, once the buffer in the signer is full, the root node in the authentication tree is signed and the remaining nodes are flushed to the cloud to spare space for upcoming events. Thus, lacking space in the signer may lead to multiple expensive encryption operations in one epoch. Furthermore, the same number

of decryption operations are also needed at the verifier side. On the other hand, for GSC the available space affects the resolution of the sample blocks, rather than signing speed. Only one encryption operation is performed in a single epoch. Fig. 8 illustrates the signing/verifying performance comparison between GSC and DTC under varied space available at the signer. It is obvious that both signing and verifying performance of DTC are capped by available memory at the signer, whereas GSC runs at full speed all the time.

The sampling rate at the receiver side also affects the verifying performance for both GSC and DTC, because it directly determines the number of events to share the cost of encryption/decryption, as depicted in Fig. 10, where higher sampling rate yields better verifying throughput. Another observation from Fig. 10 is that sampling rate also impacts GSC in terms of the space needed at the signer to achieve maximal verifying throughput. Recall that the available space in the signer defines the resolution of sample blocks. The number of unused but verified events decreases as the resolution of the sample block improves. Suppose the receiver asks for 10% data in the cloud. In the case where there are 2 units of space in the signer, the receiver fetches and verifies 20% unused data because the finest sample block contains 50% data. If the available space increases to 3 units, the smallest sample block consists 25% data and thus the unused data shrinks to 15%. The time wasted owe to unused data becomes increasing prominent when the sampling rate decreases. Therefore, smaller the sampling rate, more space required at the signer. The good news is that as small as 7 units of space are enough to support maximal verifying throughput when the sampling rate is 1%.

Moreover, the verifying throughput varies with the space allocated to cache verified nodes in the authentication tree for DSC. In our current prototype implementation, the verifier stops caching new verified nodes if the buffer is full. As expected, the performance acceleration is more evident with more cached verified nodes, as illustrated in Fig. 9. It is interesting to note that before any of the three lines in Fig. 9 reaches full speed, for a given fixed space at the verifier, the verifying throughput is higher when there is less space available in the signer. This is because the locality of the cache nodes favors higher refreshing frequency. When smaller space is available at the signer, the number of jointly signed events is less and thus the cached nodes are refreshed more frequently.

6.4 Emulation Experiments with Sampling Protocol

From the the prototype experiment without budget limit, it seems that the space requirement ($\log n$ units) at the signer is trivial, where n is the number of event reports generated in one sensing device. If the sampling protocol is utilized, the signing/verifying performance is likely to be limited by the space available at the coordinator. The spacial cost to host auxiliary authentication information is $B \log n$, where B could be very large. Suppose the space available at the coordinator is C. It is equivalent to the situation where there are $\frac{C}{B}$ units of space in the signer. Since we have already illustrated the impact of space in one single signer in Fig. 8, how signing/verifying throughput changes with varied available space in the coordinator is not shown for brevity.

Budget	500	1000
Signature	115821	186619
Verify	35038	70296

Table 4: Simulation results on #events.

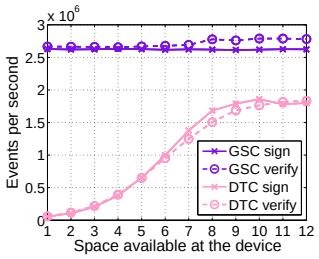


Figure 8: Thrpt. comp.

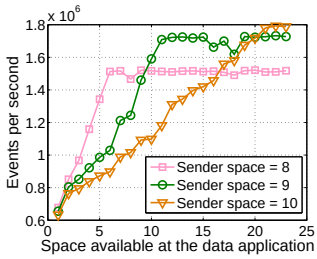


Figure 9: DTC rev. thrpt.

We focus more on potential maximal throughput of tested signature schemes. We suppose there is enough space at both the signer and verifier sides. The events sent to the coordinator are used for signing performance evaluation whereas the verification algorithm is fed by the events saved at the cloud. The number of events involved is listed in TABLE. 6.4.

Fig. 11 shows the throughput comparison between GSC and DTC. We do not put results for the sign-each approach in this figure because its throughput is much slower than the other two and we focus more on the visualization of more comparable results. From all performance evaluation experiments, sign-each approach is more than 50X slower than the other two. Since each day is one epoch and there are only 7 sensing devices, there are only $90 \times 7 = 630$ encryption/decryption operations for both GSC and DTC. For all the experiments conducted in this subsection, GSC is faster than DTC in terms of both signing and verifying. This is because DTC needs to perform the message digest function possibly more than once per event, whereas GSC calculates the message digest exactly once per event. We can verify this conjecture by analyzing the performance comparison in Fig. 11. We can see that performance gap between GSC and DTC becomes more prominent when faster message digest function is applied. For example, the throughput different increases from 0.35M events per second to 0.7M events per second if MD5 replaces SHA256 when DSA is used and the budget limit is 500. The throughput decreases when the value of budget limit is reduced as implicated in Fig. 11, because the same number of encryption/decryption operations are amortized to fewer events.

7 RELATED WORK

Wang et al. [34] rely on erasure-correcting code to ensure the security of cloud data storage. They compute homomorphic tokens for data blocks dispersed across distributed servers and use a challenge-response protocol between users and the cloud to verify cloud data correctness. However, their scheme is not practical for signing sensing data samples. The set of data blocks that can be verified is pre-decided before data distribution while data applications determine which part of data to use in real time. The other reason is that

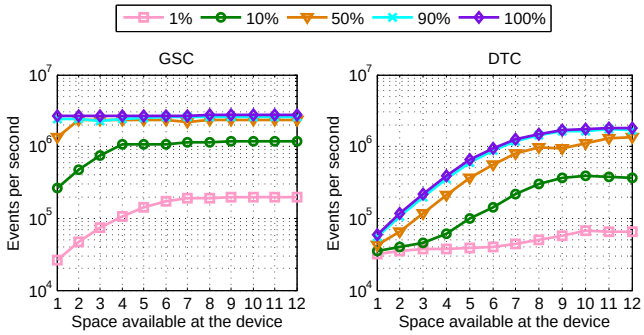


Figure 10: Verifying throughput comparison with different sample rate.

the verification process can only be conducted by data owners instead of people who use the data. Their follow-up work [36] enables a third-party auditor to be a delegate for users to conduct cloud data verification. Their solution bases on Merkle Hash Tree. Our proposed signature scheme, GSC, achieves higher throughput with less overhead. GSC can be applied to a public audit to periodically check integration and authentication of cloud data storage.

8 DISCUSSION AND FUTURE WORK

Different sensing devices may send generated data to the coordinator at vastly different speed. The video surveillance system [17] continuously generates tons of data whereas human-motion detector [38] sends much less data occasionally. In order to avoid starvation of devices, the sampling protocol discussed in this paper can be easily generalized to allow weighted items. How to automatically set weights for different devices with little human intervention would be our future work.

Our sampling and signature scheme can be also applied to other areas besides IoT data storage. It is increasing important to monitor networks at different geographic locations in a scalable way [19]. Our sampling and signature scheme provides the opportunity to relieve the burden of the network, where the local collector acts as the coordinator and periodically transmits the sampled packets to the global traffic analytic. Since the sampled packets may be transmitted over the Internet [19], DTC is no longer applicable in this scenario where network switches send data at 10/40 Gbps. The memory of switches cannot sustain to store such tremendous amount of internal nodes of authentication tree in DTC.

In this paper, we do not discuss the network issues associated with the sampling protocol, in which it is assumed that the communication between the coordinator and the device is instantaneous. Even though the network latency does not impact the eventual correctness of the sampling protocol, a lot of network bandwidth is wasted owe to the transmission of events that should be discarded at devices locally. In our future work, we plan to design a queuing principle which prioritizes the coordinating messages to favor the devices sending more events thus to reduce the network bandwidth waste.

Fog computing is a new paradigm where small-scale cloud datacenters are deployed at ISP network edge. Because their proximity to end-users, VMs of fog computing can be leveraged as the coordinator described in the sampling protocol. In this way, the end users are not bothered to update their access point to support the sampling protocol.

9 CONCLUSION

We summarize the new challenges of the IoT data communication framework with authenticity and integrity and argue that existing solutions cannot be easily adopted. We design a system aimed to address these challenges. This system is able to uniformly sample data from sensing devices and then securely store the data in the cloud while respecting resource budget constraint. The sub-systems in our paper symbiotically operate together and this system is efficient in terms of space and time, as is validated by extensive simulation and prototype emulation experiments.

ACKNOWLEDGMENTS

The authors are supported by University of California Santa Cruz startup funding and National Science Foundation Grant CNS-1701681. The authors also thank anonymous IoTDI reviews for their constructive comments and suggestions.

REFERENCES

[1] 1992. The MD5 Message-Digest Algorithm. https://tools.ietf.org/html/rfc1321. (1992).
[2] 2011. mHealth: New horizons for health through mobile technologies. http://www.who.int/goe/publications/goe_mhealth_web.pdf. (2011).
[3] 2013. http://traces.cs.umass.edu/index.php/Smart/Smart. (2013).
[4] 2013. DSA. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf. (2013).
[5] 2013. The Seagate 600 & 600 Pro SSD Review. http://www.anandtech.com/show/6935/seagate-600-ssd-review/5. (2013).
[6] 2014. Sampling for Big Data. www.kdd.org/kdd2014/tutorials/t10_part1.pptx. (2014).
[7] 2014. Samsung SSD 850 EVO 120GB, 250GB, 500GB & 1TB Review. http://www.anandtech.com/show/8747/samsung-ssd-850-evo-review/8. (2014).
[8] 2015. http://www.crn.com/slide-shows/cloud/300077635/the-10-biggest-cloud-outages-of-2015-so-far.htm/pgno/0/2. (2015).
[9] 2015. HVAC Monitoring, Energy Monitoring and Control. http://goo.gl/Ybq8Ai. (2015).
[10] 2015. SHA-1. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf. (2015).
[11] n.d.. https://www.opensensors.io/. (n.d.).
[12] n.d.. Amazon S3 Pricing. https://aws.amazon.com/s3/pricing/. (n.d.).
[13] n.d.. eHealth. http://www.who.int/topics/ehealth/en/. (n.d.).
[14] n.d.. Nest. https://goo.gl/7uMdA1. (n.d.).
[15] n.d.. OpenSSL. https://www.openssl.org/. (n.d.).
[16] n.d.. xxHash. http://www.xxhash.com/. (n.d.).
[17] 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*.
[18] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. 2012. Continuous sampling from distributed streams. *JACM* 59, 2 (2012).
[19] L. Elsen, F. Kohn, C. Decker, and R. Wattenhofer. 2015. goProbe: a scalable distributed network monitoring solution. In *Proc. of IEEE P2P*.
[20] D. Evan. 2011. The Internet of Things, Cisco White Paper. https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf. (2011).
[21] R. Gennaro and P. Rohatgi. 1997. How to sign digital streams. In *Crypto*.
[22] 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*.
[23] 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).
[24] T. Gupta, R. P. Singh, A. Phanishayee, J. Jung, and R. Mahajan. 2014. Bolt: Data management for connected homes. In *Proc. of USEIX NSDI*.
[25] Y. Kim, J. Kang, D. Kim, E. Kim, P. K. Chong, and S. Seo. 2008. Design of a fence surveillance system based on wireless sensor networks. In *Proc. of the Autonomics*.

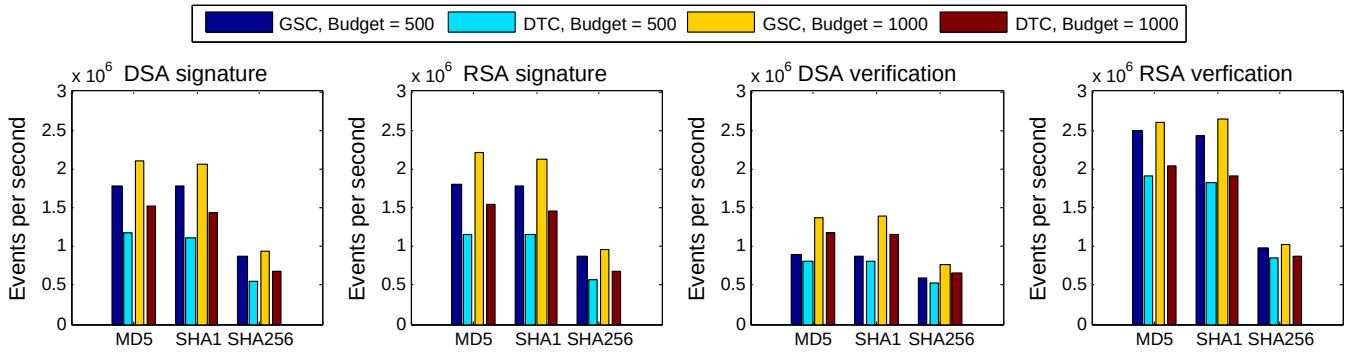


Figure 11: Throughput comparison with different parameter settings.

[26] 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).

[27] 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. DOI: http://dx.doi.org/10.1145/2063509.2063512

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

[29] S. Monterde. 2014. Cisco Technology Radar. (2014).

[30] K. Piotrowski, P. Langendoerfer, and S. Peter. 2006. How public key cryptography influences wireless sensor node lifetime. In *Proc of ACM SASN*.

[31] R. L. Rivest, A. Shamir, and L. Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *CACM* 21, 2 (1978).

[32] J. Scott, Bernheim B., J. Krumm, B. Meyers, M. Hazas, S. Hodges, and N. Villar. 2011. PreHeat: controlling home heating using occupancy prediction. In *Proc. of ACM Ubicomp*.

[33] J. S. Vitter. 1985. Random sampling with a reservoir. *ACM Trans. Math. Software* 11, 1 (1985).

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

[35] 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*.

[36] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou, and Jin Li. 2011. Enabling public auditability and data dynamics for storage security in cloud computing. *IEEE transactions on parallel and distributed systems* 22, 5 (2011), 847–859.

[37] C. K. Wong and S. S. Lam. 1998. Digital signatures for flows and multicasts. In *Proc. of IEEE ICNP*.

[38] T. Yamada, Y. Hayamizu, Y. Yamamoto, Y. Yomogida, A. Izadi-Najafabadi, D. N. Futaba, and K. Hata. 2011. A stretchable carbon nanotube strain sensor for human-motion detection. *Nature nanotechnology* 6, 5 (2011).

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

[40] J. Zheng, D. W. Gao, and L. Lin. 2013. Smart meters in smart grid: An overview. In *Proc. of IEEE GT*.