Fine-grained sharing of encrypted sensor data over cloud storage with key aggregation

Hung Dang
School of Computing
National University of Singapore

Francois Brun
School of Computing
National University of Singapore

Ee-Chien Chang
School of Computing
National University of Singapore

Yun Long Chong
School of Computing
National University of Singapore

Abstract

We consider scenarios in sensor network where the sensed samples are each encrypted with a different key and streamed to a cloud storage. The large number of samples poses technical challenge in fine-grained sharing. For instance, if the data owner wants to grant a user access to a large subset of the samples, the straightforward solution of sending all corresponding keys to the user would overwhelm the data owner's network resources. Although existing solution such as Attribute-Based Encryption (ABE) and Key Aggregation Cryptosystem (KAC) can aggregate a number of keys into a single key of small size, each of the techniques has limitations in certain aspects, which render them impractical in our applications. In particular, ABE generally incurs large overhead in ciphertext size, while KAC, though attaining constant ciphertext size and aggregated key size, requires quadratic reconstruction time with respect to the number of keys to be reconstructed. In this paper, we made an observation that for a large class of queries, specifically the combination of range and down-sampling queries, there is a algorithmic enhancement for KAC that reduces its reconstruction time from quadratic to linear. Such improvement addresses the main hurdle in adopting KAC for large datasets. Experimental studies show that on those class of queries, the proposed algorithm outperforms the original KAC by at least 90 times when reconstructing 2^{15} keys. We also give a Minimum Spanning Tree (MST)-based algorithm for general queries and a clustering algorithm to trade-off the reconstruction time with the size of aggregated key. Experimental studies show that these algorithms can reduce the reconstruction time for keys that are dense in small range.

1. INTRODUCTION

There is a growing interest of incorporating cloud resources into wide-area sensor network [25, 15, 13]. In such solutions, the sensors continuously sense and stream samples to the cloud, wherein various users can retrieve and process the data. Nevertheless, storing sensitive data in public cloud storage faces a higher risk of information leakage as demonstrated by many well-known incidents [23]. Hence, it is desired to protect the sensitive data from potentially curious servers using strong cryptographic means. This, in turn, poses challenges in fine-grained sharing of the data with multiple users. Although there are generic techniques such as Attributed-Based-Encryption (ABE) to facilitate fine-grained access control of encrypted data, adopting these techniques in large-scale systems remains challenging.

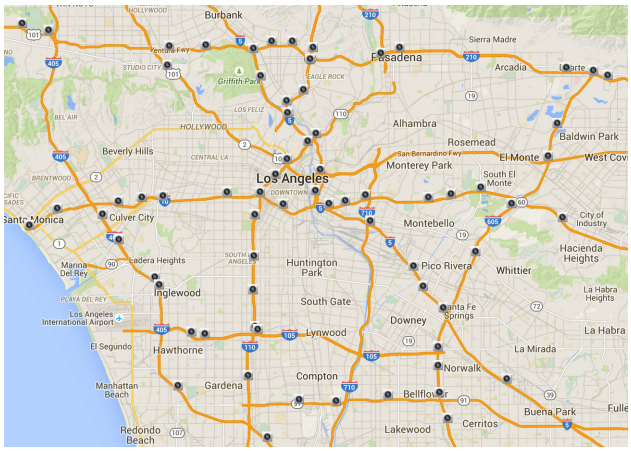


Figure 1: CCTV network in the City of Pasadena under the Real-Time Data Capture and Management Program [1]. Each icon indicate location of a camera.

To illustrate the challenge, let us consider the following scenario. A *data owner* has a collection of sensors deployed along roads in a city. The sensors continuously capture, encrypt and stream samples to the storage servers. The size of each sample can be large (e.g hundreds of Kbytes for images) or small (e.g a few bytes of temperature reading), with different sampling rate (e.g. ranging from 24 frames per second for video to single sample per second for temperature reading). In addition, each sample can consist of multiple components; for example, scalable coding that consists of different layers of resolution. The data owner wants to share selected samples with other users, for instance, sharing images captured by 100 cameras along a particular road segment, during every weekday from 6 am to 10 am at a reduced rate of 1 frame per second, and at a low image resolution. The users can be third party cloud-service providers who are engaged by the data owner to perform certain processing, or personnels who are authorised to access certain cameras, etc. To handle multiple users, a fine-grained sharing mechanism is required. Furthermore, due to privacy concerns, it is desired that the samples remain encrypted at rest in the storage servers, with the encrypted keys kept secret from the potentially curious storage servers.

In a straightforward download-and-share method, the data owner simply retrieves the encrypted video, decrypts

and sends them to the users in real-time. Clearly, such solution consumes significant computation and networking resources. Another method is to encrypt each sample (or each component of the sample) using a different key. To grant a user access to a set of samples, all keys corresponding to those samples are sent to the users, either by the data owner or a proxy. Using those keys to decrypt encrypted samples which could be downloaded from the storage servers, the user can obtain the requested samples. However, the number of keys in consideration can be very large. In our example of sharing images extracted from 100 cameras for four hours at the sampling rate of 1 frame per second, the number of keys required per day is more than 1.4×10^6. In order to address this issue, there exists known techniques that "aggregate" all the keys into a single key of small size [11, 12, 10, 24, 5]. Hence, instead of sending a large number of keys to the user, the data owner only needs to have one small aggregated key delivered. Unfortunately, each of the known techniques has limitations in certain aspects, which renders them impractical in our context. In particular, key-policy Attributed-Based Encryption (KP-ABE)[12, 10] would lead to large overhead on the ciphertext size, while Key-Aggregation Cryptosystem (KAC) [11] incurs quadratic key reconstruction time with respect to the number of keys to be reconstructed.

In this work, we place our focus on time-series data which can be found in a wide range of applications. Indeed, many interesting sensor data are inherently time-series in nature, such as CCTV's images or environmental readings. Moreover, the sensors are typically spatially arranged. For example, the public dataset made available by the US Department of Transportations's (US DOT) Real-Time Data Capture and Management Program [1] contains images captured by 103 cameras deployed along roadway network in the City of Pasadena, California (Figure 1). These images are indexed by the their timestamps and cameras' locations. Major metropolitan cities such as Beijing and London are known to have hundreds of thousands of surveillance cameras and sensors installed; and the numbers are rapidly growing [2]. For these sensor data, we treat the spatial, temporal and other meta information as non-sensitive, whereas the confidentiality of the actual sensed sample are to be protected. Such assumption is reasonable, since after all, the storage server is likely able to derive the source and timing of the sensed data from the received network packets.

Our solution adopts Key-Aggregation Cryptosystem (KAC) [11] as the underlying cryptographic scheme. KAC enables aggregation of decryption keys for arbitrary set S of samples into a constant size key, but incurs high cost in reconstruction, requiring $O(|S|^2)$ group multiplications to reconstruct all keys for S. We made an observation that, for many types of queries, there are computation redundancies in the reconstruction process that can be eliminated. For combinations of multidimensional range (e.g. asking for samples from cameras along a specific road segment during a specific time period) and down-sampling (e.g. asking for 1 sample per second instead of the original 24 samples per second) queries, we derive a "computation plan" which achieves optimal linear time reconstruction; i.e. $(O|S|)$. For general query, speeding up reconstruction time is also possible. We give an heuristic to derive the computation plan for those queries, and another algorithm to trade-off the size of the aggregated key with the reconstruction time.

Our enhancement addresses computational aspects of KAC's algorithms while preserves other characteristics such as constant size ciphertext and constant size aggregated key as well as its semantic security and collusion resistance. Experimental studies show that the proposed method is efficient, which outperforms relevant alternatives by significant factors. To reconstruct 2^{15} keys of down-sampled data within a two-dimensional range, the reconstruction time taken by our method is at least 90 times faster than the original KAC (see Figure 7). For general query asking for 2^{15} samples, our approach can achieve 24 times improvement in reconstruction cost.

2. BACKGROUND ON KEY-AGGREGATE ENCRYPTION

Key-Aggregate Encryption (KAC) [11] is a public key cryptosystem that can aggregate any set of decryption keys to derive a constant size decryption key. With a public key, given a plaintext x and an index $i \in [1, n]$, one can encrypt x to get a ciphertext associated to the index i. Hence, if the plaintexts are a sequence $\langle x_1, x_2, \ldots, x_n \rangle$, the ciphertexts $\langle c_1, c_2, \ldots, c_n \rangle$ form the corresponding sequence.

Same as typical public key cryptosystem, each ciphertext c_i can be decrypted using the private key. In addition, KAC supports key aggregation. That is, for any set of indices $S \subseteq \{1, 2, \ldots, n\}$, the secret key holder can generate a small aggregated key K_S for another user. With only the aggregated key K_S and the public key, any c_i where $i \in S$ can be decrypted. However, the aggregated key K_S is unable to obtain information from c_i for any $i \notin S$. KAC's security relies on decisional Bilinear Diffie-Hellman Exponent (BDHE)[7].

The KAC comprises of five basic functions, *Setup*, *KeyGen*, *Aggregate*[1] and *Decrypt*.

- $param \leftarrow$**Setup**$(1^\lambda, n)$: Given the security parameter λ and n, output a bilinear group $\mathbb{G}$ of prime order p where $2^\lambda \leq p \leq 2^{\lambda+1}$, a generator $g \in \mathbb{G}$ and a random number $\alpha \in_R \mathbb{Z}_p$, output the system parameter $param = \langle g, g_1, g_2, ..., g_n, g_{n+2}, ..., g_{2n} \rangle$ where $g_i = g^{\alpha^i}$.
- $(PK, SK) \leftarrow$**KeyGen**(): Pick a value $\gamma \in_R \mathbb{Z}_p$, output the public and master-secret key pair: $(PK = v = g^\gamma, SK = \gamma)$.
- $\zeta \leftarrow$**Encrypt**(PK, i, x): Given a public key PK, an index $i \in \{1, 2, ..., n\}$ and a message $x \in \mathbb{G}_\mathbb{T}$, randomly pick $t \in_R \mathbb{Z}_p$ and output $\zeta = \langle g^t, (vg_i)^t, x \cdot e(g_1, g_n)^t \rangle$.
- $K_S \leftarrow$**Aggregate**(SK, S): Given a set S of indices j's, output the aggregated decryption key $K_S = \prod_{j \in S} g^\gamma_{n+1-j}$.
- $\{x, \perp\} \leftarrow$**Decrypt**$(K_S, S, i, \zeta = \langle c_1, c_2, c_3 \rangle)$: If $i \notin S$, output $\perp$, else output $x = c_3 \cdot e(K_S \cdot \rho, c_1)/e(\hat{\rho}, c_2)$ where $\rho = \prod_{j \in S, j \neq i} g_{n+1+i-j}$ and $\hat{\rho} = \prod_{j \in S} g_{n+1-j}$

The aggregated key K_S only consists of a single group element and thus its size is $O(\lambda)$ where λ is the security parameter. However, decrypting cost for each ciphertext increases

[1]The function *Aggregate* is also known as *Extract* in the literature [11].

proportionally to the size of the set. Specifically, given the aggregated key K_S corresponding to a set of ciphertext C whose indices are in S, it takes $O(|S|)$ group operations to decrypt a single ciphertext in C, and thus $O(|S|^2)$ group operations to fully reconstruct the ciphertext set. The high reconstruction cost renders the scheme impractical for our application.

3. PROBLEM DEFINITION

3.1 Sensor Data

We adopt a convention that $[a, b]$ represents an interval of integers from a to b, inclusively. We call $\mathbb{L}_\Delta = [1, T_1] \times [1, T_2] \times \ldots \times [1, T_d]$ a d-dimensional lattice with the bounds $T_1, T_2, \ldots, T_d$. A *hyper-rectangle* S in $\mathbb{L}_\Delta$ is the subset $R_1 \times \ldots \times R_d$ of $\mathbb{L}_\Delta$ where each R_i is an interval in the i-th dimension.

A sensor continuously senses and generates a sequence of *samples*. A sample is represented by a tuple (i, x) where i and x are its index and sample value respectively. The sample value is the data captured by the sensor at a particular instance. It's size can be large, such as an image, or just a few bytes, such as temperature reading. The index i is a multidimensional point, representing the sample's temporal, spatial and other meta information such as resolution level. We assume that some normalisations have been applied such that the indices are mapped to points in $\mathbb{L}_\Delta$. Note that the temporal information is not restricted to be one-dimension. For example, temporal information can be represented as a multidimensional point with day, month, year, etc as its dimensions. The indices are considered as non-sensitive. As such, they can be stored in plaintext in the storage server to facilitate efficient searching.

3.2 System Model

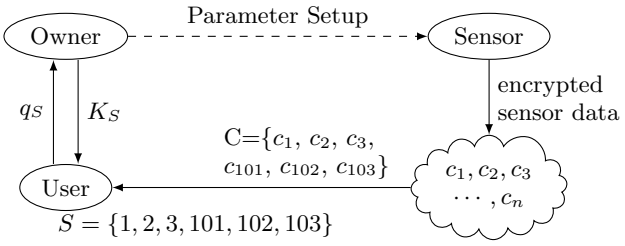


Figure 2: System model supporting fine-grained sharing of encrypted sensor data

Figure 2 illustrates our system model. To protect the confidentiality of sensor data, samples are encrypted before being streamed to the cloud. When an user wants to gain access to a subset C of encrypted sensor data whose indices are in the set S, the user requests a decrypting capability of C by sending a query q_S to the owner. Upon approval, the owner issues an aggregated key K_S to the user. She can then use K_S to reconstruct (decrypt) the encrypted samples in C, which can be directly retrieved from the storage server. However, it is impossible for her to use such K_S to decrypt any sample which does not belong to C. An additional layer of protection can also be implemented to guarantee that only authorized users can download the relevant encrypted samples. We discuss this in more details in section 7.

3.2.1 Security requirements

For security analysis, we consider a worst case scenario in which the storage server is completely under the user's control; i.e. she has full access to all encrypted samples stored in the cloud storage.

The key aggregation must be *collusion resistance*. A collusion attack is carried out by combining multiple aggregated keys, with the goal of deriving more information than each aggregated key can individually derive. For example, if an user has the aggregated key to decrypt images of road segment A on Jan 1^{st}, and another aggregated key for road segment B on Feb 2^{nd}, then he must not able to obtain other images, including images captured on A during Feb 2^{nd}. We follow the model by Boneh et. al [8] on collusion resistance.

We assume that sensors are trusted. Nevertheless, in case a sensor is compromised and the secrets it holds are revealed to an adversary, confidentiality of data generated by other sensors must not be compromised. This implies that it is impossible to derive the master key from the secrets known to the sensors.

3.2.2 Efficiency requirements

As the sensors and the users can be operating on low-powered devices, it is crucial to keep computation load low. Furthermore, although cloud storage is relatively low in cost, the communication and storage overhead incurred by the security mechanisms has to be sufficiently reasonable so as to keep the cloud solution economically attractive. In view of the above considerations, we focus on the following three measures of performance:

Reconstruction time..

Clearly, computation load of reconstructing the keys from the aggregated key K_S has to be low. In some applications (e.g. viewing of video stream), the reconstruction time also has to meet the real-time requirement. As mentioned in the introduction, the known KAC scheme take quadratic time with respect to $|S|$ and thus is not acceptable.

Size of aggregated key..

To reduce the communication between the owner and users, the size of the aggregated key K_S has to be small.

Overhead of ciphertext size..

The overhead of ciphertext size directly increases the storage and communication cost of the storage server. Since the number of ciphertexts is large, the actual multiplicative overhead on the ciphertext size is a practical concern, especially for sample value that are relatively small.

3.3 Query Types

We classify queries for sensor data into three types:

Q1 - d-dimensional range query. .

Such a query asks for all samples whose indices reside in a d-dimensional hyper-rectangle. For example, images from cameras along a road segment during a certain period corresponds to a 2-dimensional range query. A d-dimensional range can be compactly represented by $2d$ integers, which indicates the corners of the hyper-rectangle.

In some cases, it is possible to represent multiple range

queries as a single range query. For example, the images from 6 am to 12 pm of every weekday is an union of a series of queries. We can represent this by a single query by re-arranging and "lifting" the one-dimensional time component to multi-dimensions. Specifically, by decomposing the single time dimension into four dimensions which are (1) time in a day, (2) day in a week, (3) week number and (4) year, the aforementioned image subset can be represented by a single query.

Q2 - Down-sampling query. .

Such a query asks for *a down-sampled lattice*. In one-dimension, if one sample is extracted for every p samples, we say that the down-sampling rate is $1/p$. In higher dimension, a t-dimensional down-sampled lattice is the subset $\mathcal{L} = \{\sum_{i=1}^{t} a_i v_i | a_i \in \mathbb{Z}\} \cap \mathcal{L}_\Delta$ where each of the v_i is a d-dimensional vector and the basis $\{v_1, v_2, .., v_t\}$ is independent. A down-sampling query is represented by its basis. A query can also be an intersection of range and down-sampling queries. For example, the query for a few images per each hour captured along a road segment on a certain day is a down-sampling range query.

Q3 - General query. .

A general query asks for an arbitrary set of samples which are not necessary a combination of range and down-sampling. If the query is sent from the user to the owner by listing down all the indices, then the straightforward solution of sending individual decryption key to the user would incur similar cost in both directions of communication, and thus arguably acceptable. However, a query could be combination of an arbitrary set in some dimensions, with range and down-sampling in the other dimensions. For example, an query that asks for samples from an arbitrary set of sensors during all weekend's morning. Hence, it is meaningful to aggregate keys in general query.

As the distribution of the queries is applications dependent, in this paper, we use the following simple distribution model for general query in evaluating our algorithms: the set S contains a set of $r\beta$ indices that are randomly selected from the interval $[1, \beta]$ where $r < 1$ and β are some parameters.

4. ALTERNATIVE CONSTRUCTIONS

Before presenting the proposed solution, we briefly discuss a few alternative cryptographic solutions and their limitations.

Top-down Hash-tree..

One possible approach is to use a binary tree to maintain symmetric encryption keys (Figure 3) for sensor data. The root is the master key, while the intermediate sub-keys are generated in a top-down manner. The actual keys for encryption/decryption are located at the leaves. Each sample is associated with one external leaf, and is encrypted by the corresponding key. In this construction, keys for m samples in a range can be reconstructed using only $O(\log(m))$ aggregated keys. These aggregated keys are essentially intermediate sub-keys whose descendants are the m encryption keys under consideration. For instance, in Figure 3, sub-keys 19,5 and 24 are aggregated keys from which encryption keys in $\{4, 5, 6, 7, 8, 9\}$ can be "reconstructed".

However, it is not straightforward to extend this method to support d-dimensions, where $d > 1$. A trivial method of using multiple trees, one for each dimension, to generate d keys for each sample is not secure against collusion attack [21]. Furthermore, this method fails to aggregate keys for down-sampling and general queries, such as ones asking for encryption keys $\{1, 3, 5, 7, 9\}$ or $\{1, 4, 5, 7, 10\}$.

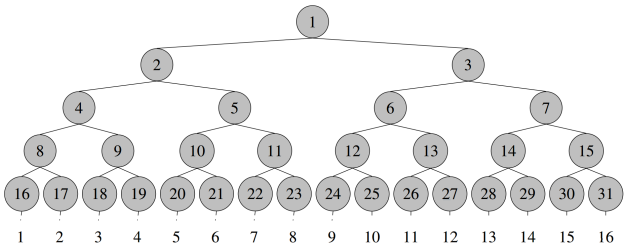


Figure 3: Tree based construction for one-dimensional data.

ABE-based construction. .

There are a few ways to employ Attribute-Based Encryption (ABE) to aggregate decryption keys for multidimensional range query. The most intuitive approach is to adopt Key-Policy ABE (KP-ABE)[16] in the following way: An index is represented by a set of attributes, each of which corresponds to the location of an 1 in the index's binary representation. For instance, the index $9 = 1001_2$ is represented by 2 attributes A_0 and A_3. In delegating decryption ability of ciphertexts in a range of S, the data owner first determines the "policy" $\mathcal{A}$, which is a logical expression on the attributes for indices in S. The aggregated key is then determined from the policy. The size of the aggregated key is often proportional to the number of logical operations in the logical expression, and thus incurs a $\log(n)$ factor overhead in specifying a range, where n is the total number of samples encrypted under the same security setting. For example, if $n = 2^{10}$ and an index set in question is $S = [1019, 1023]$, then the policy $\mathcal{A} = \{A_9 \wedge A_8 \wedge A_7 \wedge A_6 \wedge A_5 \wedge A_4 \wedge A_3\}$. Furthermore, the ciphertext size of each index is proportional to the number of attributes associated to it, which implies a multiplicative $\log(n)$ factor overhead. Experimental studies also show that the reconstruction time of this approach is slower than our proposed method, probably due to the larger number bilinear map operations required. Finally, while it is easy to express down-sampling of rate $1/p$ using short expression, where p is a power of 2, it is not clear how to efficiently express other down-sampling rates. Hence, it is not trivial to obtain short aggregated key for other rates.

Multi-dimensional Range Query over Encrypted Data..

Shi et al. address Multi-dimensional Range Query over Encrypted Data (MRQED) problem [21]. The work can be viewed as an enhancement of the ABE-based construction. They aim to protect confidentiality of both query and the indices. Specifically, if an index of a sample under consideration is outside the queried range, one would learn no information beyond the fact that an aggregated key fails to recover the sample from its ciphertext. Note that in our

application, the indices are not considered secret and made publicly available. Thus, we do not have to enforce this security requirement. Similar to the ABE-based construction, the method incurs an overhead of at least $\log(n)$ multiplicative factor in ciphertext size and aggregated key size.

5. PROPOSED FAST RECONSTRUCTION

5.1 Main observation

To decrypt a single ciphertext with index i using an aggregated keys of a set S ($i \in S$), the following two values have to be computed (see Section 2):

$$\rho_i = \prod_{j \in S, j \neq i} g_{n+1+i-j} \ , \ \hat{\rho} = \prod_{j \in S} g_{n+1-j}$$

Since $\hat{\rho}$ is independent of i, it can be computed once for all i's. Hence, let us focus on the computations of ρ_i, which requires $|S|-2$ group multiplications. Therefore, $|S|(|S|-2)$ group multiplications are needed for all i, which is approximately $|S|^2$.

Our main observation is that, for range and down-sampling query, the indices in S follows some patterns that permit fast computations. Let us first consider a 1-dimensional range $S = [1, m]$ for some m. Note that the term $(n+1)$ in the subscript is a common offset of the indices. For clarity in exposition, let us define $\hat{g}_t = g_{n+1+t}$, and

$$R_i = \prod_{j \in S} \hat{g}_{i-j}$$

for all $i \in S$. For each i, we have $\rho_i = \hat{g}_i^{-1} R_i$, and thus it can be easily computed from R_i. Now, we explore how to compute all R_i efficiently. Under the straightforward method, to compute R_i requires $|S|-1$ multiplications for each i, and a total of $|S|(|S|-1)$ multiplications are required to compute R_i for all $i \in S$. However, by exploiting the recurrence relation

$$R_{i+1} = (\hat{g}_{i-m})^{-1} \cdot R_i \cdot \hat{g}_i$$

we can obtain R_{i+1} from R_i using only 2 multiplications. This leads to a fast linear time algorithm that computes all R_i's recursively, which improves the original quadratic time algorithm to linear time.

The next two sections show how to extend the observation to multidimensional range and down-sampling queries.

5.2 Extension to multidimensional range queries (Q1)

Let us first consider two dimensional lattice. Let $S = [1, m] \times [1, m]$ be a rectangular range within the 2-dimensional lattice with bound n in both dimension. Here, the indices are two dimensional vectors. Let $\sigma(x_1, x_2) = x_1(n-1) + x_2$ be the mapping that maps the two dimensional lattice to the one dimensional lattice. Similarly, to decrypt the ciphertext with the index (i_1, i_2), the following value has to be computed:

$$\rho_{(i_1,i_2)} = \prod_{(j_1,j_2) \in S, (j_1,j_2) \neq (i_1,i_2)} g_{n^2+1+\sigma(i_1,i_2)-\sigma(j_1,j_2)}$$

Likewise, the term n^2+1 in the subscript is simply some fixed offset. For clarity, we can rewrite the coordinate, and define $\hat{g}_{(i_1,i_2)} = g_{n^2+1+\sigma(i_1,i_2)}$ and $R_{(i_1,i_2)}$ as follow:

$$R_{(i_1,i_2)} = \prod_{(x,y) \in S} \hat{g}_{(i_1,i_2)-(x,y)} = \prod_{x=1}^{m} \prod_{y=1}^{m} \hat{g}_{(i_1,i_2)-(x,y)}$$

Note that the required $\rho_{(i_i,i_2)}$ can be easily obtained from $R_{(i_1,i_2)}$. Computing $R_{(i_1,i_2)}$ naively requires $|S|-1$ group multiplications. However, we can rewrite the above into a recurrence relation:

$$R_{(i_1+1,i_2)} = R_{(i_1,i_2)} \prod_{y=1}^{m} \hat{g}^{-1}_{(i_1,i_2)-(i_1-m,y)} \prod_{\widetilde{y}=1}^{m} \hat{g}_{(i_1,i_2)-(i_1,\widetilde{y})}$$

Let us define

$$T_{(i_1,i_2)} = \textstyle\prod_{y=1}^{m} \hat{g}^{-1}_{(i_1,i_2)-(i_1-m,y)}, \quad \text{and}$$
$$\widetilde{T}_{(i_1,i_2)} = \textstyle\prod_{\widetilde{y}=1}^{m} \hat{g}_{(i_1,i_2)-(i_1,\widetilde{y})}.$$

By substituting the above definitions into the recurrence relation, we have

$$R_{(i_1,i_2)} = R_{(i_1-1,i_2)} T_{(i_1-1,i_2)} \widetilde{T}_{(i_1-1,i_2)}.$$

Now, observe that $T_{(i_1,i_2)}$ can also be expressed as a recurrence relation and all of them can be computed in linear time (with respect to $|S|$). Similarly for $\widetilde{T}_{(i_1,i_2)}$. Putting all together, we have a linear time algorithm to compute all $R_{(i_1,i_2)}$'s.

In general, for a d-dimensional range, the number of group multiplications required is in $O(d|S|)$. Since in our application the dimension d is small, by treating it as a constant, we have a linear time algorithm.

5.3 Extension to down-sampling queries (Q2)

Let us consider an example in 2-dimension. Given an independent basis $\{(3,0),(0,2)\}$ of down-sampling, we can transform the co-ordinate (x, y) to $(x/3, y/2)$ such that the required samples correspond to samples with integer coordinate. Hence for an intersection of a down-sampled range, the above linear time algorithm can be applied under the transformed co-ordinate. In general, for a down-sampled d-dimensional range, the number of group multiplications required is also in $O(d|S|)$. Although additional computations are required to transform the coordinate, they are significantly less expensive compare to the group multiplications.

5.4 MST for General queries (Q3)

The algorithm we discuss above reuses common terms among different ρ_i's to save computations. Such common terms are apparent in range and down-sampling queries, but not so for general queries. An interesting question to consider is how to find a computation plan that computes all ρ_i's with the least number of group multiplications for a given arbitrary S.

Given the set of indices S, recall that for each $i \in S$, ρ_i is a product of selected elements from $\langle g_0, g_1, g_2, \ldots \rangle$. Let s_i be the set of indices of these terms. For any $i, j \in S$,

$$\rho_i = \rho_j \cdot \prod_{a \in (s_i \setminus s_j)} g_a \cdot \prod_{b \in (s_j \setminus s_i)} g_b^{-1}.$$

Hence, to compute ρ_i, if the value of ρ_j is already known, we can derive ρ_i from ρ_j with $|s_i \setminus s_j| + |s_j \setminus s_i|$ group multiplications. Alternatively, we can compute ρ_i by directly multiplying the terms with indices in s_i using $|s_i| - 1 = |S| - 2$ multiplications. Let $M(i, j) = \min(|S|-2, |s_i \setminus s_j| + |s_j \setminus s_i|)$, which is the number of multiplications required to obtain ρ_i if value of ρ_j is known.

The above definitions lead to a strategy – we can compute the ρ_i's one by one by reusing common terms that have been previously computed. To compute ρ_i, we derive it from ρ_j, where $M(i,j)$ is the smallest and ρ_j has already been computed. Now, the question is on how to determine the optimal order in computing the ρ_i's. It is easy to see that we can formulate this as the Minimum-Spanning-Tree problem where $M(i,j)$ are the edges' weights. This leads to our algorithm for arbitrary set S.

Remarks. .

We stress that though the above MST algorithm is optimal in determining an order and a strategy with respect to such an order in which ρ_i's are to be computed, it may not be optimal in minimising the number of multiplications. Minimising the number of multiplications turns out to be difficult, as one may introduce intermediate values to reduce the number of operations. For instance, observe that in section 5.2, the intermediate values $T_{(i_1,i_2)}$ help to significantly reducing the number of multiplications.

We assume that the set S is pre-determined and thus the computation plan can be pre-computed. Although restricted, such scenarios are realistic, for example, in cases where the same set S is to be repeated for multiple users and thus the computation plan can be computed once for all users. In such scenarios, the time taken by the greedy algorithm is not a main concern in our applications. Also note that since its computation only involves public data, the MST algorithm can be performed by any entity, including the user, owner or proxy in the cloud. Section 6 reports empirical studies on the effectiveness of our algorithm.

5.5 Trade-off between number of aggregated keys and reconstruction time

As the reconstruction time is superlinear, we can reduce the time at the expense of using more aggregated keys by partitioning the set S into a collection of clusters, where each cluster corresponds to one aggregated key. To find a good partition of k clusters, we employ the single-linkage clustering approach. That is, initially, each element in S is a cluster by itself, and the clusters are then sequentially merged until only k clusters are left. The clustering relying on a distance function on the clusters. At each step, the two clusters with the shortest distance are selected to be merged. The effectiveness of the partition, i.e. the reduction of reconstruction time, depends on the definition of this distance function.

We adopt the following distance function. Let $C(S)$ be the number of group multiplications required to reconstruct all keys in S. This value, in turn, relies on the computation plan according to which all the required ρ_i are evaluated. Such a computation plan can be determined using the MST method described in the previous section. For two clusters S_1 and S_2, their distance is simply $C(S_1 \cup S_2)$.

Although there are efficient algorithms for MST, it is still too expensive as the distance function has to be evaluated large number of times (quadratic with respect to $|S|$). In our experiment, we adopt a simplified function. Given S, instead of using MST to determine the order whereby the ρ_i's are to be computed, we only consider the sorted order starting from the smallest index up to the largest index, and determine the number of multiplications required by this order. Section 6 gives empirical result on the effectiveness of this algorithm on randomly chosen S.

		KP-ABE	KAC	Ours
Encrypt	Mult.	$O(\log n)$	2	2
	Exp.	$O(\log n)$	3	3
	Pairing	1	1	1
Aggregate	Mult.	$O(\log n)$	m	m
	Exp.	$O(\log n)$	1	1
	Pairing	0	0	0
Reconstruct	Mult.	$O(m \log n)$	$O(m^2)$	$O(m)$
	Exp.	$O(m \log n)$	0	0
	Pairing	$O(m \log n)$	$m+1$	$m+1$
Ciphertext size		$O(\log n)$	3	3

Table 1: Costs of encrypting one record, extracting aggregated key and reconstructing (decrypt) a range query of size m, where n is total number of samples to be encrypted under the same public key. The ciphertext size is measured by the number of group elements.

6. PERFORMANCE

This section compares performance of our proposed method with KP-ABE [16] and KAC [11] without the proposed fast reconstruction.

6.1 Performance Analysis

Table 1 summarises the numbers of group operations, i.e. multiplication, exponential and pairing, required by the three procedures: (a) encryption of the samples, (b) aggregation of the keys, and (c) reconstruction of the keys, and (d) the size of the ciphertext in term of number of group elements. Observe that KP-ABE suffers from a $O(\log n)$ factor, which is arguably inevitable since $\log n$ attributes are required to represent n indices. Since the total number of samples can be very large (e.g. at 25 samples per second, more than 2 millions samples will be generated every day), the large overhead is not acceptable, especially for ciphertext size which affects the storage and communication cost. Although KAC outperforms KP-ABE in almost all aspects, its reconstruction cost is quadratic, which renders the scheme impractical in our application. The proposed method reduces the number of multiplications to linear on range and down-sampled queries. Although KAC and our method require more group multiplications than KP-ABS during key aggregation, they require much fewer number of expensive exponentiations, and thus it is not clear which is more efficient in practice.

6.2 Experimental Setup.

In the experiments, we use randomly selected AES keys as the sample values. Hence, a sample can be represented by a single group element. Such assumption is appropriate since in applications, when the size of a single sample is large, it is more efficient to encrypt the sample using symmetric encryption such as AES with a randomly chosen key, and apply the key aggregation on the symmetric keys.

To measure the performance of aggregation and reconstruction on range (i.e. type Q1) and downsampling (i.e. type Q2) queries, the total number of samples is fixed at 2^{18}, while the size of the query $m = |S|$ varies from 2^5 to 2^{15}. The query is a two-dimensional range (with the same width

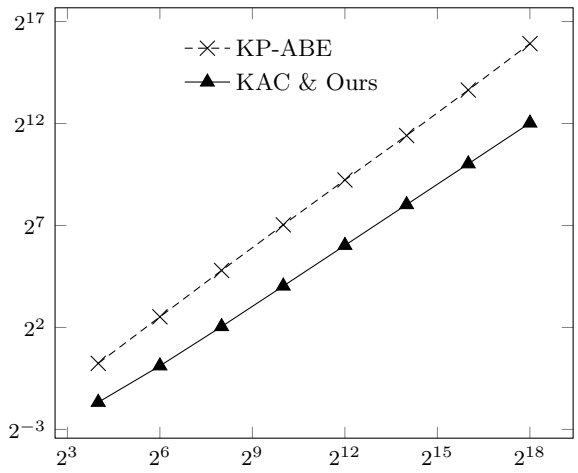


Figure 4: Encryption time (seconds) vs n, the total number of samples.

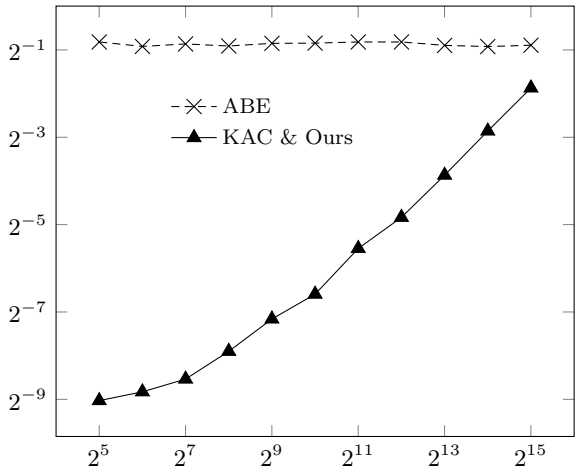


Figure 6: Aggregation time (seconds) vs m, the number of keys.

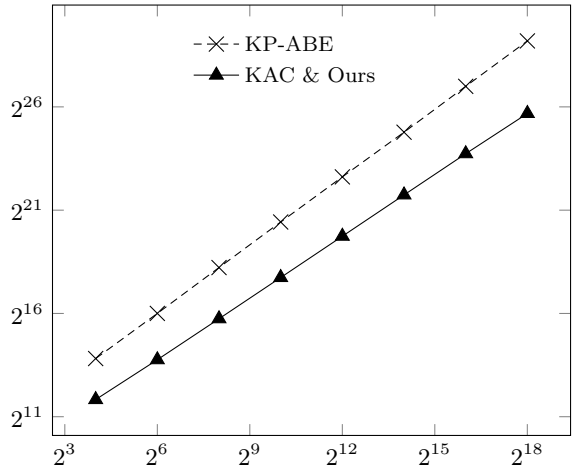


Figure 5: Total ciphertext size (bytes) vs n, the total number of samples

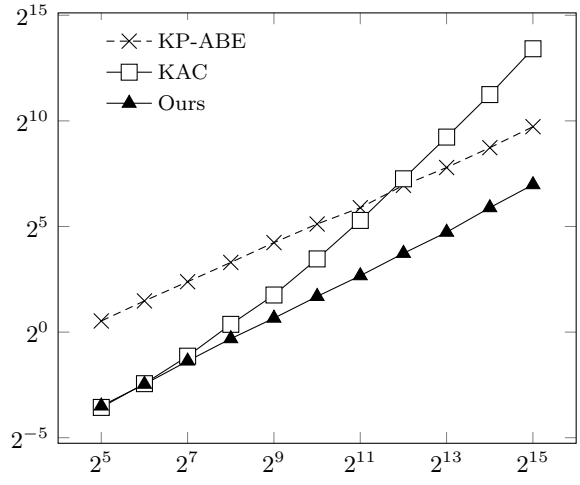


Figure 7: Reconstruction time (seconds) vs m, the number of keys, for a range with down-sampling query (i.e. Q1 & Q2).

along both dimensions) with downsampling when testing on KAC and the proposed method. Since KP-ABE can only support downsampling of limited rate, we only conduct experiment on the two-dimensional range query for KP-ABE.

For general queries (i.e. type Q3), the queries are generated by selecting m indices randomly from the range $[1, n]$. Let us call the ratio $r = m/n$ the *query density*. Various query densities are investigated. We do not study KP-ABE's performance on general queries, since it requires another algorithm to find a compact logical expression for the arbitrary given query, which could be a separate topic of interest.

To measure the trade-off achieved by the clustering algorithm described in Section 5.5, we experiment with two different numbers of aggregated keys $k = \sqrt[4]{m}$ and $k = \sqrt{m}$ for various m.

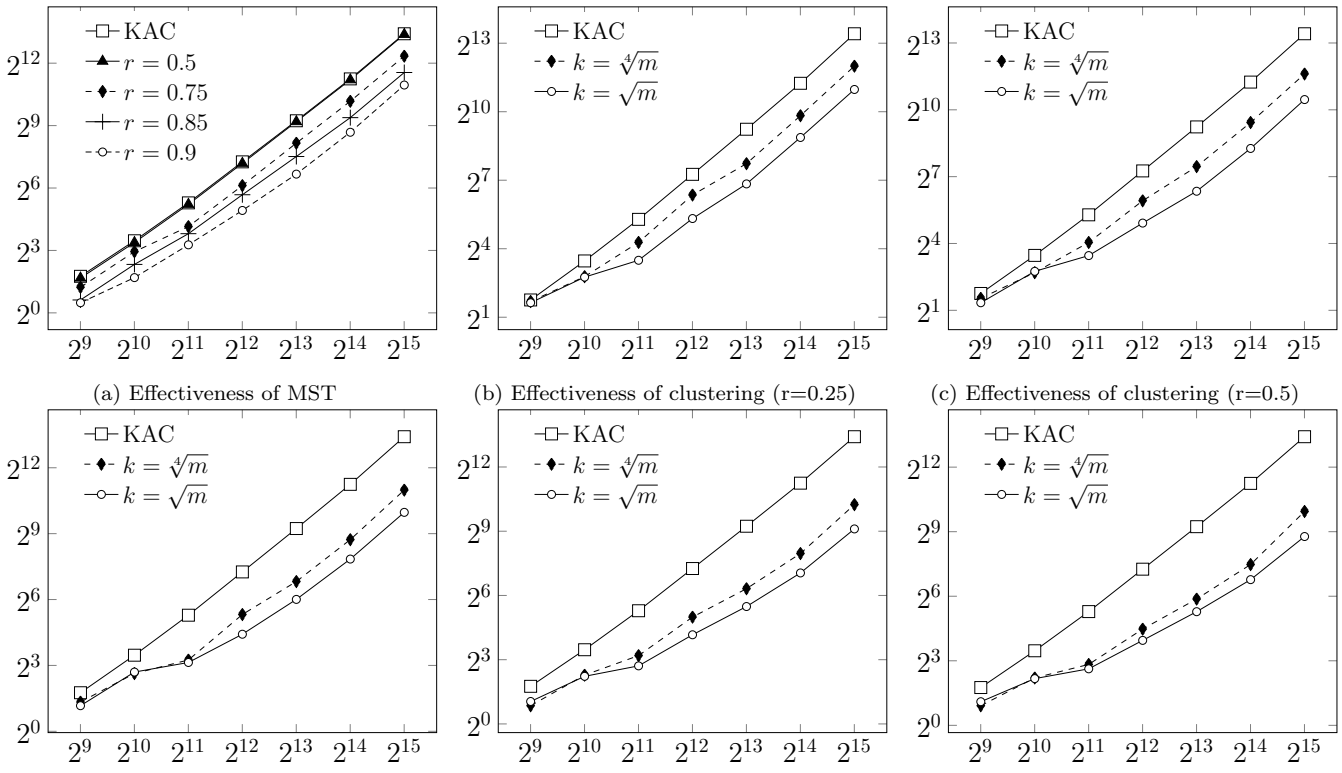
All experiments are performed on a system equipped with Intel Core-i5-4570u@3.2Ghz processor and 8GB of RAM. To simulate an environment with lower computation power such as mobile devices, we limit the cpu usages to 20% of the original using the tool `cpulimit`[2]. Our implementation employs Charm [3] and symmetric pairings over Type-A (supersingular) curves with a base field of 512 bits as underlying cryptographic framework and pairing group. We repeat each experiment 10 times and report average results, with time and size are measured in seconds and bytes respectively.

6.3 Experiment result

Encryption time..

Figure 4 compares the encryption time of the three alternatives under log-log scale, with the total number of samples range from 2^4 to 2^{18}. Since our approach employs KAC's encryption algorithm, the performance of the two are exactly the same. The experiment results agree with the analysis in the previous section. The cost of encryption incurred by KP-ABE is several times higher. For example, to encrypt 2^{18} items, KP-ABE needs 17 hours, while KAC only requires 70 minutes. Note that the main overhead of KP-

[2] `http://cpulimit.sourceforge.net/`



(a) Effectiveness of MST (b) Effectiveness of clustering (r=0.25) (c) Effectiveness of clustering (r=0.5)

(d) Effectiveness of clustering (r=0.75) (e) Effectiveness of clustering (r=0.85) (f) Effectiveness of clustering (r=0.9)

Figure 8: Reconstruction time (seconds) vs m, the number of keys, for a general query (i.e. Q3).

ABE's encryption lies in carrying out exponent operations, which directly depends on the total number of samples, and thus the overhead would be higher for larger datasets.

Ciphertext Size (storage cost)..

A main disadvantage of KP-ABE lies in its ciphertext size. Figure 5 reports storage space required with various n, the total number of samples to be encrypted. When $n = 2^{18}$, KP-ABE requires approximately 10 times more than KAC. This is so because KAC's ciphertext comprises of only three group elements, whereas KP-ABE's ciphertext contains $(3A + 2)$ group elements, where A is the number of attributes associated with a ciphertext. The value of A varies for different ciphertext, but its expected value is at least $\frac{1}{2} \log n$. Similar to the encryption time, gap between KP-ABE and KAC would be wider for larger datasets.

Aggregation time..

As shown in Table 1, KP-ABE's key aggregation time does not depend on m, the number of keys to be aggregated, but depends on n, the total number of samples. KAC, on the other hands, aggregates keys in $O(m)$ time. It turns out that, when m is less than 2^{15}, KP-ABE needs longer time compares to KAC (Figuer 6).

Reconstruction time..

Figure 7 shows the reconstruction time for the two-dimension and down-sampled range. For small m, reconstruction time incurred by ABE is higher than KAC, which is due to the expensive pairing operations. However, for larger m, we can observe the quadratic growth in KAC as the computation is now dominated by multiplications. Our proposed method, on the other hand, achieves linear reconstructing time. When $m = 2^{15}$, it can reconstruct the 2^{15} keys within 126 seconds, whereas KAC needs 3 hours, that is, a speedup of almost 90 times.

Figure 8(a) shows the reconstruction time for general queries of different densities. Unsurprisingly, the query density affects the speedup factor. When density is less than 0.5, the gain is negligible but more significant for larger density. At density 0.75, the speed-up is 2.5 times, and improves to 8 times when the density reaches 0.9.

We also evaluate the trade-off between number of aggregated keys and reconstruction time (Figures 8(b) to 8(f)). For a general query asking for m samples, with $\sqrt[4]{m}$ aggregated keys, reconstruction time can be speeded-up by 10 times. With a cost of $\sqrt{m}$ keys, we can achieve up to 24 times improvement. Note that for small queries, issuing more aggregated keys may increase reconstruction time, due to the increase in the number of pairing operations.

7. SYSTEM DESIGNS

In this section, we give two possible designs that incorporate key aggregation. We consider two types of sensors; one with Public-key Cryptosystem (PKC) capability, and the other that is only capable of performing standard symmetric key cryptosystem such as AES and SHA-1. We refer to the first category as PKC-enabled sensors and the later as low-powered sensors.

7.1 System with PKC-enabled sensors

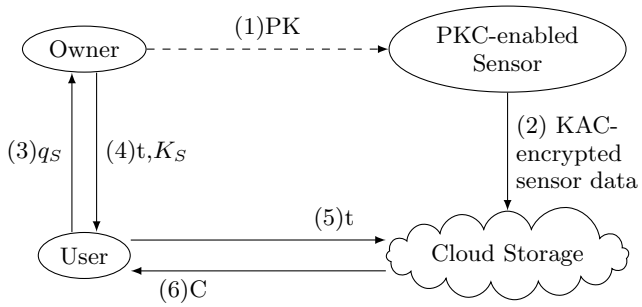


Figure 9: System model for PKC-enabled sensors

During system setup, the owner distributes the public key PK to all entities, and an unique identity ID to each sensor (Figure 9). The identity ID's are not secrets and are made public. For each sample (i, x), the sensor encrypts the sample value x with the index i using KAC's encrypt algorithm to obtain a ciphertext c. It then streams the c together with the index i to the storage server.

In situation where sensor samples are of large size, (e.g. images), they are encrypted using AES with a randomly generated key k, whereas the key k is being encrypted by KAC under an index i of the sensor sample (similar to sensor sample of small size). The two ciphertexts (encrypted sample and encrypted symmetric key) and the corresponding index are then streamed to the server.

When an user asks for access to a subset C, whose indices fall in S, he sends the query q_S to the owner. The owner issues an aggregated key K_S to the user, together with an authentication ticket t. The user presents the authentication ticket t to the storage server as a proof that he is authorised to access the C. Upon verification, the server sends the requested ciphertexts to the user, which are later decrypted using the aggregated key K_S. In case of large samples, she also needs to download corresponding encrypted symmetric keys. The encrypted keys are first reconstructed, and then used to decrypt the encrypted samples.

The incorporation of the authentication ticket can be based on standard protocol such as Kerberos [17, 20]. Although the authentication ticket is ineffective in scenario where the users collude with the server, it forms another layer of defence to prevent unauthorised downloading of the ciphertexts.

7.2 System with low-powered sensors

Figure 10 shows the system design for low-powered sensors, which are only capable of conducting non-expensive cryptographic operations such as AES or SHA-1. To address the resource constraints of these low-powered sensors, we introduce a trusted encryption proxy. This proxy also helps to relieve the owner's computation load.

During system setup, the owner broadcasts the public key PK to all entities except the low-powered sensors. The owner also distributes an unique identity ID and a shared secret key K_{ID} to each sensor. For each sensed sample (i, x), the sensor generates a symmetric encryption key $k_{i,ID}$ using a cryptographic pseudorandom function on input K_{ID} and the index i. The sensor then encrypts the sample value x obtaining c, and streams (i, c) to the storage server.

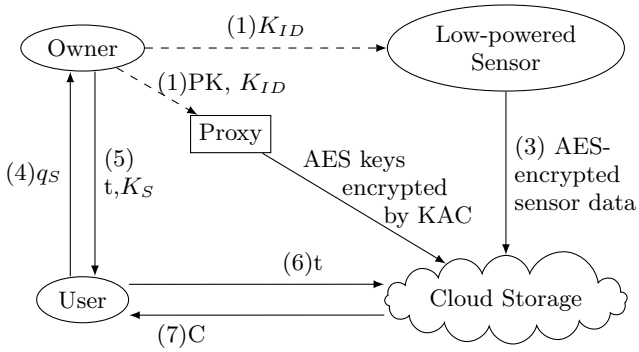


Figure 10: System model for low-powered sensors

All secret K_{ID} are also shared with the encryption proxy. Because the proxy (which actually represents the data owner) has knowledge of locations and frequencies at which sensor data are collected, it can infer the set of indices associated with the samples. With the knowledge of the indices and all sensors' secret keys, it can replicate a symmetric key $k_{i,ID}$. Each of these AES keys is encrypted with KAC under the corresponding sample index, giving $c_{i,ID}$.

The ciphertexts together with their indices, i.e. $(i, c_{i,ID})$'s, are then sent to the storage server. Note that this process need not be performed in realtime. Rather, the proxy can replicate, encrypt and send the encrypted AES keys to the cloud storage in batches well before the actual sensing. In addition, although the encryption proxy has the secret K_{ID}, it cannot derive the owner's secret key. The remaining steps (step (4) to (7) in Figure 10) are similar to the previous setting.

Compare to the PKC-enabled sensor, if a low-powered sensor ID is compromised, the secret K_{ID} could be revealed. With K_{ID}, the adversary can decrypt all previously encrypted sensor samples generated by that sensor.

8. RELATED WORK

Hierarchical access control..

Several cryptographic key assignment schemes exploit hierarchical structures, such as trees, to maintain keys for various sets of objects [24, 5]. A key for an internal node is used to derive keys for its descendant nodes. These approaches efficiently support aggregating key for simple access policies. Other schemes can support more complicated access policies, such as those that are described by cyclic or acyclic graphs [26, 4]. Benaloh et al. introduced an encryption scheme supporting delegating decryption capability with flexible hierarchy [6]. This scheme achieves constant-size aggregated decryption keys. However, it is not clear how to extend the schemes to maintain encryption keys for multidimensional objects whose access policies do not follow any hierarchical structure.

Key-policy Attributed-based Encryption..

KP-ABE enables various ciphertexts to be decrypted by one single key. This technique associates a set of attributes to a ciphertext and a policy to a decryption key, which is able to decrypt all ciphertext whose attributes conform to its policy [10, 16]. ABE obtains collusion-resistance at a cost of secret keys' compactness. More specifically, the key size is proportional to the number of attributes it is associated

with. Other alternative can reduce size of decryption keys, but inevitably increase ciphertext's size [14, 18]. These approaches requires many bilinear-mapping operations in their executions, which renders their performance cost prohibitive and thus impractical.

Multi-Dimensional Range Query over Encrypted Data..

Supporting complex queries over encrypted data is also of interest. Boneh *et al.* presented a primitive named Hidden Vector Encryption (HVE) to enable range and subset queries [9]. This scheme results in $O(dt)$ encryption time, ciphertext size and $O(d)$ decryption key size and decryption cost, where d is the number of dimensions and t the number of points. Shi *et al.* proposed a construction adopting a specialized data structure for range query evaluation [21]. Its encryption cost, ciphertext size and decryption key size are all $O(d\log(t))$ while decryption cost is $O((\log(t))^d)$. Because these schemes consider some security requirements which are not relevant in our application, such as secrecy of all attributes, they suffer from a poor performance and not applicable in our context.

9. CONCLUSION

Privacy has always been a serious concern in outsourced storage. The data stored on the potentially curious cloud storage should be protected by a strong cryptographic mean. However, to have data encrypted brings forth fundamental technical challenges in sharing the data with other entities, including sharing with compute-servers that are authorised to process selected data on a need-to-know basis. To support fine-grained access control, we need a mechanism that can efficiently aggregate and reconstruct large number of keys. Although there are many known key aggregation techniques, they are computationally intensive and not practical in our applications.

In this work, we focus on sensor data, especially time-series data that are continuously sensed, encrypted and streamed to the cloud. The temporal and spatial arrangements of these time-series data lead to queries of the form of multidimensional range and down-sampling that can be exploited for efficiency. We proposed a fast reconstruction algorithm for the known KAC. The enhancement is significant for range with down-sampling queries, for example, our approach can achieve 90 times speed-up in reconstructing 2^{15} ciphertexts. We also proposed reconstruction algorithms for general queries and trade-off between number of aggregated keys issued and reconstruction time. At an expense of issuing a reasonable number of extra aggregated keys, we can improve the reconstruction time of general queries by 24 times compared to original KAC. The proposed fast reconstruction addressed a main hurdle in adopting key aggregation in large datasets.

10. REFERENCES

[1] Intelligent transportation systems research data exchange. http://catalog.data.gov/dataset/intelligent-transportation-systems-research-data-exchange-pasadena-11-cctv-snapshots.

[2] Top 5 cities with the largest surveillance camera networks. http://www.vintechnology.com/journal/uncategorized/top-5-cities-with-the-largest-surveillance-camera-networks/.

[3] J. A. Akinyele, M. D. Green, and A. D. Rubin. Charm: A framework for rapidly prototyping cryptosystems. Cryptology ePrint Archive, Report 2011/617.

[4] M. J. Atallah, M. Blanton, N. Fazio, and K. B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 2009.

[5] G. Ateniese, A. D. Santis, A. L. Ferrara, and B. Masucci. Provably-secure time-bound hierarchical key assignment schemes. Cryptology ePrint Archive, Report 2006/225.

[6] J. Benaloh, M. Chase, E. Horvitz, and K. Lauter. Patient controlled encryption: Ensuring privacy of electronic medical records. In *CCSW 2009*.

[7] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO 2005*.

[8] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Advances in Cryptology–CRYPTO 2005*, pages 258–275. Springer, 2005.

[9] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *Theory of cryptography*. Springer, 2007.

[10] M. Chase and S. S. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *CCS 2009*.

[11] C.-K. Chu, S. S. M. Chow, W.-G. Tzeng, J. Zhou, and R. H. Deng. Key-aggregate cryptosystem for scalable data sharing in cloud storage. *IEEE TPDS*, 2014.

[12] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS 2006*.

[13] M. M. Hassan, B. Song, and E.-N. Huh. A framework of sensor-cloud integration opportunities and challenges. In *ICUIMC '09*.

[14] S. Hohenberger and B. Waters. Attribute-based encryption with fast decryption. In *PKC 2013*.

[15] W. Kurschl and W. Beer. Combining cloud computing and wireless sensor networks. In *Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services*, 2009.

[16] A. Lewko, A. Sahai, and B. Waters. Revocation systems with very small private keys. In *SP*, 2010.

[17] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The kerberos network authentication service (v5). RFC 4120, 2005.

[18] T. Okamoto and K. Takashima. Achieving short ciphertexts or short secret-keys for adaptively secure general inner-product encryption. Cryptology ePrint Archive: Report 2011/648, 2012.

[19] S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *Journal of the ACM (JACM)*, 2002.

[20] A. A. Pirzada and C. McDonald. Kerberos assisted authentication in mobile ad-hoc networks. In *Proceedings of the 27th Australasian conference on Computer science-Volume 26*, pages 41–46. Australian Computer Society, Inc., 2004.

[21] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *SP 2007*.

[22] R. Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[23] H. Takabi, J. Joshi, and G.-J. Ahn. Security and privacy challenges in cloud computing environments. *Security Privacy, IEEE*, 2010.

[24] W. G. Tzeng. A time-bound cryptographic key assignment scheme for access control in a hierarchy. *IEEE Trans. on Knowl. and Data Eng.*, 2002.

[25] M. Yuriyama and T. Kushida. Sensor-cloud infrastructure-physical sensor management with virtualized sensors on cloud computing. In *NBiS'10*.

[26] Q. Zhang and Y. Wang. A centralized key management scheme for hierarchical access control. In *In Proceedings of IEEE Global Telecommunications Conference*, 2004.