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

Hung Dang, Yun Long Chong, Francois Brun, Ee-Chien Chang
School of Computing
National University of Singapore

Abstract

We consider a sensor network setting in which the sensed samples are encrypted individually using different keys before being streamed to a cloud storage. For large systems with high capacity, e.g. generating several millions of samples per day, fine-grained sharing of encrypted samples is challenging. The straightforward solution is to send to a user all decryption keys of each and every shared samples. This approach does not scale up, for the number of keys to be shared would overwhelm the data owner's network resources. Existing solutions, such as Attribute-Based Encryption (ABE) and Key Aggregation Cryptosystem (KAC), can aggregate a number of keys into a single key of small size, addressing the problem to a certain extent. However, ABE generally incurs large overhead in ciphertext size, while KAC requires quadratic reconstruction time with respect to the number of keys to be reconstructed. These limitations render them impractical in our applications. In this paper, we first present an algorithmic enhancement for KAC that reduces its reconstruction time for the combination of range and down-sampling queries from quadratic to linear. Further, we generalize such enhancement and discuss various heuristics to boost the reconstruction time for arbitrary (general) queries. We also give a clustering-based method to trade-off the reconstruction time with the number of aggregated keys to be issued. These improvements address the main hurdle in adopting KAC for practical applications with large datasets. Our experimental studies show that given the query asking for 2^{15} keys, the proposed enhancement outperforms the original KAC by at least 90 times on range and down-sampling queries, and achieves 8 times speed up for general queries. It also shows that splitting the query into 16 sub-queries, each of which is associated with a separate aggregated key, can further reduce the reconstruction cost by 19 times.

1. INTRODUCTION

Incorporating cloud resources into wide-area sensor network [31, 16] has recently been of growing interest. 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 [28]. A common wisdom is to protect the sensitive data from potentially curious servers using strong cryptographic means. This, in turn, poses various technical challenges in fine-grained sharing of the encrypted data with multiple users. Although generic techniques such as Attributed-Based-Encryption (ABE) can facilitate fine-grained access control over encrypted data, adopting these techniques in large-scale systems remains challenging.

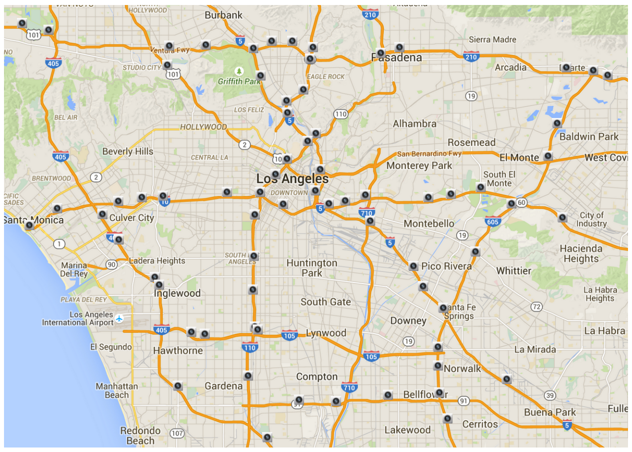


Figure 1: CCTV network in the City of Pasadena under the Real-Time Data Capture and Management Program. Each icon indicate location of a camera.

To illustrate the challenge, let us consider the following scenario. A *data owner* has a collection of sensors deployed along a city road network. The sensors continuously capture, encrypt the samples individually using different encryption keys before streaming the encrypted content to the storage servers. The size of each sample can be varied (e.g hundreds of Kbytes for images or only a few bytes of temperature reading), and sampling rates can also be different (e.g. ranging from 24 frames per second for video to single sample per second for temperature reading). In addition, each sample consists of multiple components; for example, scalable coding including different resolution layers. The data owner wants to share selected samples with other users, e.g. 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 necessary. 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 all the untrusted parties.

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 does not scale for it consumes significant computation and networking resources. Another method is to send all decryption keys corresponding to those samples to the users. The user can then use those keys to decrypt encrypted content which could be downloaded from the storage servers. However, the number of keys in consideration can be very large, equivalent to the number of samples to be shared. 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. Known techniques that "aggregate" all the keys into a single key of small size [10, 14, 9, 30, 3] can address this issue to a certain extent. 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, these techniques have their own limitations, rendering them impractical in our context. In particular, key-policy Attributed-Based Encryption (KP-ABE)[14, 9] would lead to large overhead on the ciphertext size, while Key-Aggregation Cryptosystem (KAC) [10] incurs quadratic key reconstruction time with respect to the number of keys to be reconstructed.

In this work, we place our focus on secure fine-grained sharing of encrypted 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 reported to have hundreds of thousands of surveillance cameras and sensors installed; and the numbers are rapidly growing. For these sensor data, we treat the spatial, temporal and other meta information as non-sensitive, whereas the confidentiality of the actual sensed samples is to be protected. Such assumption is reasonable, since after all, the storage server is probably able to derive the source and timing of the sensed data from the received network packets.

Our solution adopts Key-Aggregation Cryptosystem (KAC) [10] 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 in S. We make an observation that, for a large class of queries, there are computation redundancies in the reconstruction procedure. 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 give a fast reconstruction technique attaining optimal linear running time; i.e. $(O|S|)$. For general query, speeding up reconstruction time is also possible. We give two heuristics to derive the computation plan for those queries – one assumes no pre-computation and the other makes use of pre-computation and reuses intermediate values to further speed up the reconstruction. Informally, the computation plan describes a specific order in which a sequence of computation should be carried out. We also discuss a clustering-based method to trade-off the number of the aggregated keys being issued for the reconstruction time.

Our enhancement addresses computational aspects of KAC reconstruction (decrypt) algorithm while preserving other characteristics including its semantic security and collusion resistance. Thus, our system is provably secure. Experimental studies show that the proposed methods are efficient, outperforming 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 8). For general query asking for the same number of samples, our approaches achieve upto 8 times improvement in reconstruction cost. The speeding up is further increased to 19 times at the expense of splitting the query into 16 sub-queries, each of which is associated with one separate aggregated key.

2. BACKGROUND ON KEY-AGGREGATE ENCRYPTION

Key-Aggregate Encryption (KAC) [10] 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.

Similar to any typical public key cryptosystem, each ciphertext c_i can be decrypted using the private key. In addition, KAC supports key aggregation. 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)[5].

The KAC comprises of five basic functions, *Setup*, *KeyGen*, *Aggregate*[2] and *Decrypt*.

- $param$ ←**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) ←**KeyGen**(): Pick a value $\gamma \in_R \mathbb{Z}_p$, output the public and master-secret key pair: $(PK = v = g^\gamma, SK = \gamma)$.

- ζ ←**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 $\zeta = \langle g^t, (vg_i)^t, x \cdot e(g_1, g_n)^t \rangle$.

[1]http://catalog.data.gov/dataset/

[2]The function *Aggregate* is also known as *Extract* in the literature [10].

- K_S ←**Aggregate**(SK, S): Given a set S of indices j's, output the aggregated decryption key $K_S = \prod_{j\in S} g^{\gamma}_{n+1-j}$.
- $\{x, \perp\}$ ←**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 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.

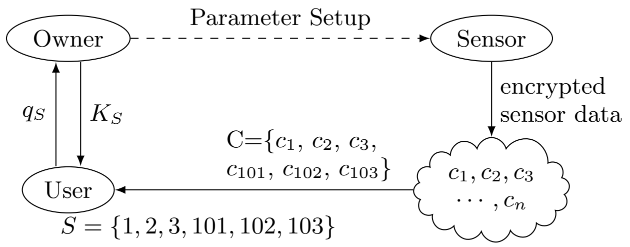
3. PROBLEM DEFINITION

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 value respectively. The sample value is the data captured by the sensor at a particular instance. Its size can be varied (e.g hundreds of Kbytes for images or only a few bytes for 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 non-sensitive. As such, they can be stored in plaintext in the storage server to facilitate efficient searching.

3.2 System Model



Parameter Setup
Owner
Sensor
q_S
K_S
encrypted sensor data
C={c_1, c_2, c_3, c_{101}, c_{102}, c_{103}}
User
$c_1, c_2, c_3 \cdots, c_n$
$S = \{1, 2, 3, 101, 102, 103\}$

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 and an optional "computation plan for reconstruction"[3] to the user. She can then download the encrypted samples in C from the storage server and follow the computation plan to reconstruct (decrypt) them using K_S[4]. 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 1st, and another aggregated key for road segment B on Feb 2nd, then he must not be able to obtain other images, including images captured on A during Feb 2nd. We follow the model by Boneh et. al [6] 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.

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[5]. In some applications (e.g. viewing of video stream), the reconstruction time has to meet the real-time requirement. As mentioned in the introduction, the known KAC scheme requires quadratic reconstruction time and thus is unacceptable.

Size of aggregated key. To reduce the communication between the owner and users, the size of the aggregated key

[3]The computation plan informs the user on specific sequence of computations should be conducted to reconstruct the encrypted data with at better computational cost than the naive approach of reconstructing each sample independently.

[4]To be accurate, the user first reconstruct the decryption keys (using K_S) which are then used to decrypt the encrypted samples. However, for brevity, we slightly abuse the language and simply say that the user reconstructs the encrypted samples using K_S.

[5]We stress that the cost of deriving the computation plan is not part of the reconstruction time.

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.

This 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.

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 these constraints in a single query by re-arranging and "lifting" the one-dimensional time component to multi-dimensions, i.e. 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.

Q2 - Down-sampling query.

This 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. The query may be constructed by listing down all the indices of the required samples. Alternatively, it can also be a 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.

As the distribution of the queries is application-specific, in this paper, we assume a simple distribution model for Q3 query: the set S contains $r\beta$ 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.

4.1 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 [25]. 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 $\{1, 4, 5, 7, 10\}$.

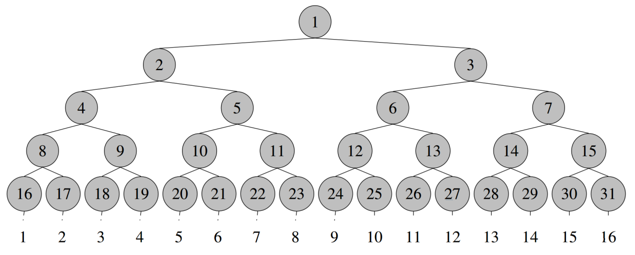


Figure 3: Tree based construction for one-dimensional data.

4.2 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)[19] 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 system's capacity (i.e. 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.

4.3 Multi-dimensional Range Query over Encrypted Data.

Shi et al. address Multi-dimensional Range Query over Encrypted Data (MRQED) problem [25]. The work can be viewed as an enhancement of the ABE-based construction, aiming 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 decrypt its encrypted content. Note that in our application, the indices are not considered secret and made publicly available. Thus, we do not enforce this security requirement. Similar to the ABE-based construction, MRQED admits an overhead of at least $\log(n)$ multiplicative factor in ciphertext size and aggregated key size, failing to meet our efficiency requirements.

5. PROPOSED FAST RECONSTRUCTION

Owing to the fact that KAC satisfies our security and two efficiency requirements (i.e. size of aggregated key and overhead in ciphertext size) put forth earlier in Section 3.2, we adopt its encryption and key aggregation procedures in our system. We remark that by adopting these two procedures, our system inherits KAC's proven security. Interesting readers are referred to [11] for further details on the security of the scheme.

However, as briefly discussed in Section 2, KAC reconstruction cost is expensive. In particular, reconstructing a single ciphertext with index i using an aggregated key K_S ($i \in S$) requires the following two values (Section 2):

$$\rho_i = \prod_{j \in S, j \neq i} g_{n+1+i-j} \qquad (1)$$

$$\hat{\rho} = \prod_{j \in S} g_{n+1-j} \qquad (2)$$

$\hat{\rho}$ is independent of i and can be computed only once for all ciphertexts in S. The computations of ρ_is ($i \in S$) are of more interest. A naive approach which computes each ρ_i independently — not exploiting their relationship — would incurs $O(|S|^2)$ group multiplications to compute all necessary ρ_i (i.e. for all i in S). We observe exploiting their relationship leads to a better computation cost.

In this section, we first introduce an algorithmic enhancement for KAC reconstruction specifically targeting Q1 and Q2 queries (Section 5.1). This enhancement reduces the reconstruction time from quadratic to linear. We later generalize the technique — using dynamic programming — to enable fast reconstruction for Q3 queries (Sections 5.2, 5.3).

5.1 Fast reconstruction for range and down-sampling queries

5.1.1 Main observation

For Q1 and Q2 queries (or their combination), the indices in S follows a specific pattern which straightforwardly permits fast computations. Let us first consider a 1-dimensional range $S = [1, m]$ for some m. 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 all R_is. 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.

We next show how to extend this observation to enable fast reconstruction for Q1 and Q2 queries.

5.1.2 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:

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

Let us define

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

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.

5.1.3 *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 $O(d|S|)$. Note that additional computations are required to transform the coordinate and they are significantly less expensive compare to the group multiplications. We refer reader to [17] for further details on the transformation which should be applied on the coordinate.

5.2 Fast reconstruction for General queries (Q3)

The technique we discuss above reuses common terms among different ρ_is to save computations. Such common terms are apparent in Q1 and Q2 queries, but not so for Q3 queries. An interesting question to consider is how to find a computation plan that evaluates all ρ_is with the least number of group multiplications for a given arbitrary S. Let us now formally define the computation plan.

Definition 1 (Computation Plan). *A computation plan to evaluate a set of elements D from the public parameter is a weighted sequence of computation steps $P = \langle p_1, p_2 \dots p_z\rangle$ such that p_j is to be carried out before p_{j+1}. Each p_j is a 3-value tuple $\langle v_n, v_p, I\rangle$ where v_n is an element in D to be computed at this step, v_p is another element in D computed previously, and I is a set of intermediate values. Other values required in the computation are to be picked from the public parameter. In addition, each computation step p_j is associated with a cost w_j; the total cost of the computation plan $P = \langle p_1, p_2 \dots p_z\rangle$ is $Cost(P) = \sum_{j=1}^{z} w_j$*

We are interested in deriving a computation plan $P = \langle p_1, p_2 \dots p_z\rangle$ that computes all required ρ_is with the minimum number of group multiplications. Based on the above definition, let us define the *minimum cost computation plan* problem.

Definition 2 (Minimum Cost Computation Plan). *Given a set of elements to be evaluated D, the Minimum Cost Computation Plan (MCCP) problem is to compute the computation plan $P = \langle p_1, p_2 \dots p_z\rangle$ whose cost (i.e., $Cost(P) = \sum_{j=1}^{z} w_j$) is minimized.*

With Definition 2, an optimal computation plan that evaluates all ρ_is is the solution of the MCCP problem in which D is the set of all ρ_is and $w_j$is the number of group multiplications required at each computation step p_j.

It turns out that the MCCP problem is difficult, as one may introduce intermediate values to reduce the number of operations (i.e. cost). For instance, observe that in section 5.1, the intermediate values $T_{(i_1,i_2)}$ help to significantly reduce the number of multiplications. Indeed, although we do not give a formal proof, we believe that this MCCP problem is reducible to the Steiner tree problem [13], which is known to be NP-complete and has a few approximations in the literature [8]. However, the reduction is not trivial, and the approximations often hardly scale up for large dataset. Instead, we opt to simplify the problem and give

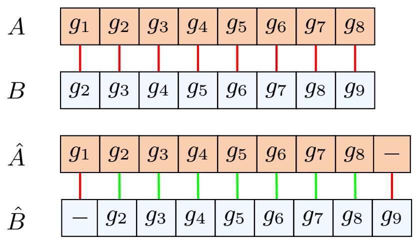

A
g_1 g_2 g_3 g_4 g_5 g_6 g_7 g_8
B
g_2 g_3 g_4 g_5 g_6 g_7 g_8 g_9
$\hat{A}$
g_1 g_2 g_3 g_4 g_5 g_6 g_7 g_8 $-$
$\hat{B}$
$-$ g_2 g_3 g_4 g_5 g_6 g_7 g_8 g_9

Figure 4: An example of sequence alignment. A green bar denotes zero penalty cost while a red bar represents penalty cost of one. By inserting "gaps" at the beginning of A and at the end of B, we obtain $\hat{A}$ and $\hat{B}$ which yield the optimal alignment with total penalty of only two.

two heuristics for the simpler version. We first consider the MCCP problem without any intermediate values. That is, ρ_i can either be computed from scratch (following Equation 1) or from another previously computed ρ_j. Next, we give a heuristic to determine which intermediate values should be pre-computed and re-used during the reconstruction.

5.2.1 *Minimum Spanning Tree based Strategy*

There is a similarity between the non intermediate value MCCP problem and the minimum spanning tree problem [12, 29]. Recall that ρ_i is a product of selected elements from the public parameter $\langle g_0, g_1, g_2, \dots\rangle$. Let s_i be the set of indices of these terms. For any two ρ_i, ρ_j, we can define their "distance" as follows:

$$\rho_i = \rho_j \cdot \prod_{a \in (s_i \setminus s_j)} g_a \cdot \prod_{b \in (s_j \setminus s_i)} g_b^{-1} \quad (3)$$

Hence, if the value of ρ_j is already known, one can derive ρ_i from ρ_j with $|s_i \setminus s_j| + |s_j \setminus s_i|$ group multiplications. On the other hand, one can also compute ρ_i following Equation 1. Consequently, $M(i,j) = \min(|S|-2, |s_i \setminus s_j| + |s_j \setminus s_i|)$ is the minimum number of multiplications required to obtain ρ_i given ρ_j. Based on this notion of $M(i,j)$, the problem of determining MCCP strictly without any intermediate values is reducible to the minimum spanning tree problem.

Let $G = (V,E)$ be a complete digraph in which V and E denotes the set of vertices and the set of directed edges, respectively. The set V comprises of $|S|$ vertices. Each value ρ_i maps to a vertex v_i. The set E contains $|S|(|S|-1)$ edges. Let us further denote by e_{ij} a unidirectional edge connecting vertex v_i to vertex v_j. Note that e_{ij} and e_{ji} are different edges, as G is directed. For each edge e_{ij}, we set its weight to $M(i,j)$.

With the reduction described above, we can use any algorithm which computes the minimum spanning tree of G to solves for the efficient computation plan. One of the well-known techniques computing minimum spanning tree in directed graph is the *Chu-Liu/Edmonds* algorithm [12]. The fast implementation of this algorithm runs in $O(V^2)$ for dense graph [29].

5.2.2 *Pre-computing and reusing intermediate values*

As one may expect, the computation cost can be further reduced if common intermediate values are pre-computed and reused. We provide here an intuitive heuristic to determine those common intermediate values.

Should one interpret each ρ_i as a sequence of $|S|-1$ elements, and intermediate values as shorter sequences comprising of x elements where $x < |S|-1$, the problem of finding the common intermediate values is reducible to the *local sequence alignment* problem [27] — which is tasked to determine "similar regions" between two sequences.

Sequence alignment is often considered as a textbook example for dynamic program. In the most basic version, the sequence alignment problem takes as input two sequences $A = a_1 \ldots a_m$ and $B = b_1 \ldots b_n$ over an alphabet Σ, together with penalty metrics $\alpha_{gap} \geq 0$ for inserting a "gap" and α_{ab} for matching an element a of one sequence against an element b of the other sequence (presumably $\alpha_{ab} = 0$ if $a = b$) and outputs an optimal "alignment" which minimizes the total penalty. In this work, we utilize the *Smith - Waterman* algorithm [27], which has various efficient and scalable implementations [21, 20], to solve for the sequence alignment problem.

Let us consider an example in Figure 4. The two input sequences are $A = g_1g_2g_3g_4g_5g_6g_7g_8$ and $B = g_2g_3g_4g_5g_6g_7g_8g_9$, $\alpha_{gap} = 1$, $\alpha_{ab} = 1$ if $a \neq b$ or $\alpha_{ab} = 0$ otherwise. Simply matching A and B as-is (without inserting any gap) incurs the penalty cost of eight. The optimal alignment is formed by inserting a gap to the end of A and another gap to the beginning of B, resulting in $\hat{A}$ and $\hat{B}$ whose alignment incurs the minimum penalty cost of two.

The most common intermediate values to be pre-computed and reused are the similar regions found in the the solution to the sequence alignment problem. For example, if we are to compute $\rho_1 = \prod_{j=1}^{8} g_j$ and $\rho_2 = \prod_{j=2}^{9} g_j$, we can represent ρ_1 and ρ_2 by the sequences A and B in the above example (Figure 4). Since the similar region of A and B is $R = g_2g_3g_4g_5g_6g_7g_8$, the intermediate value to be pre-computed is $v = \prod_{j=2}^{8} g_j$. Effectively, we can compute ρ_1 and ρ_2 from v with very low computation cost.

Let us denote the set of pre-computed intermediate values as I, the set of all necessary ρ_i as D, we can now compute the almost-optimal computation plan for D by solving the minimum spanning tree on a complete digraph $G = (V, E)$ in which V represents elements of $D \cup I$, while E contains $|V|(|V|-1)$ edges. The weight of an edge e_{ij} connecting vertices v_i to v_j is set to be the minimum number of group multiplications required to compute the value represented by v_i given the value represented by v_j.

Remarks.

The running time for computing the computation plan (Section 5.2.1) or determining common intermediate values (Section 5.2.2) are not of our concern. As it only involves public data, it may be done on the server with presumably powerful resource and is not counted toward the reconstruction time (Section 3.2).

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

We make another observation that the reconstruction time can be further reduced at the expense of splitting the query into smaller sub-queries and issuing one aggregated key for each of them. More specifically, one may partitions the set S into a collection of k clusters, where each cluster corresponds to one sub-query, and thus one aggregated key. Accordingly, one needs to issue k aggregated keys instead of one single key. However, the take is that the overall reconstruction cost is lower.

The partition should be performed in such a way that elements in each cluster is "close" to one another. Informally, let v be the common intermediate value of ρ_i and ρ_j, we define the distance between ρ_i and ρ_j by the number of extra computation it takes to compute both values from the common intermediate value v. In another word, the closer the two elements are, the less multiplications are required to compute them from v.

We employ the single-linkage clustering method [15] (implemented using *SLINK* algorithm [26]) to perform the clustering. In particular, each element in S is initially a cluster by itself. The clusters are then sequentially merged until only k clusters are left. The cluster merging are conducted based on a distance function in such a way that at every step, the two clusters with the shortest distance are merged together.

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 techniques described in the previous section (Section 5.2). For two clusters S_1 and S_2, their distance is simply $C(S_1 \cup S_2)$.

In the literature, the single-linkage clustering method is criticised to produce long thin clusters in which elements at opposite ends of a cluster are of far distance, which may lead to difficulties in defining classes subdividing the data. However, its other characteristic which is to have a distance of nearby elements residing in the same cluster small is of greater interest. Indeed, this feature will allows a ρ_i value to be computed efficiently from its nearby elements.

Table 1: Costs of encrypting one record, extracting aggregated key and reconstructing a range query of size m, where n is the system's capacity (i.e. maximum number of samples to be encrypted). The ciphertext size is measured by the number of group elements per sample.

		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

6. PERFORMANCE EVALUATION

This section compares performance of our proposed method with KP-ABE [19] and KAC [10] without the proposed fast reconstruction.

6.1 Performance Analysis

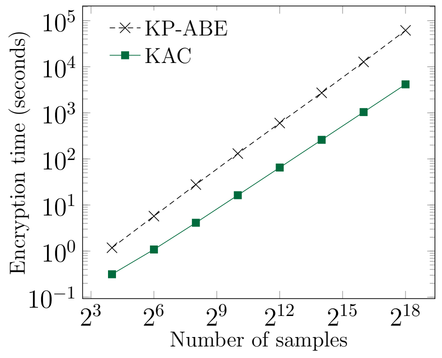


Figure 5: Encryption time

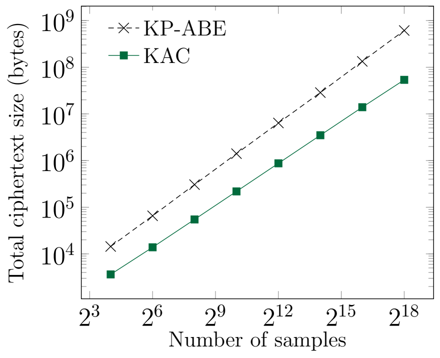


Figure 6: Total ciphertext size

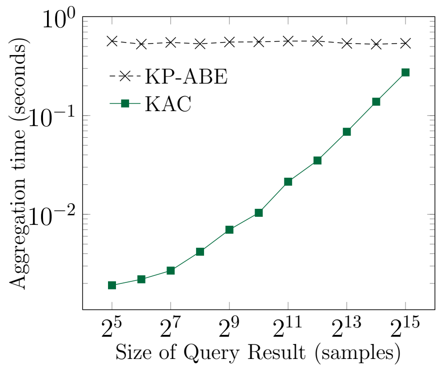


Figure 7: Aggregation time

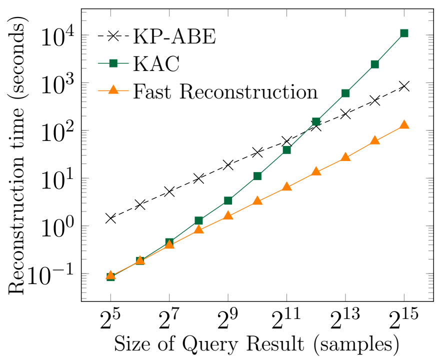


Figure 8: Reconstruction time for Q1 & Q2.

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. It also reports the size of the ciphertext with respect to the number of group elements. Observe that KP-ABE consistently suffers from a $O(\log n)$ overhead factor in comparison with KAC, 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), this large overhead is hardly 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. By adopting KAC encryption and key aggregation procedures, while introducing various techniques to boost-up its reconstruction cost, our system achieves favourable performance in all aspects. In particular, the proposed method reduces the number of multiplications to linear on Q1 and Q2 queries.

As an interesting note, we find that although KAC requires more group multiplications than KP-ABE during key aggregation, they require much fewer number of expensive exponentiations, and thus it is not clear which is more efficient in practice.

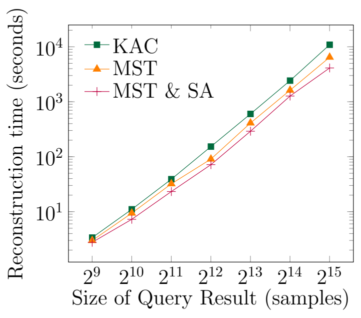
6.2 Experimental Setup.

In our 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 practice, 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, then apply the key aggregation on the symmetric keys.

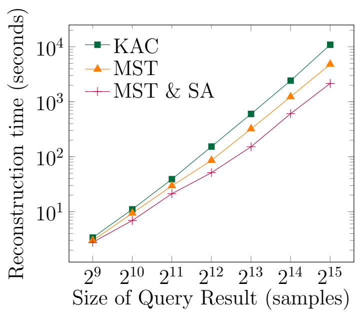
To measure the performance of aggregation and reconstruction on range (i.e. Q1) and down-sampling (i.e. Q2) queries, the total number of samples is fixed at 2^{18}, while the size of the query $m = |S|$ varies from 2^5 to 2^{15}. The query is a two-dimensional range (with the same width along both dimensions) with downsampling when testing on KAC and the proposed method. Since KP-ABE can only support down-sampling of limited rate, we only conduct experiment on the two-dimensional range query for KP-ABE.

For general queries (i.e. Q3), we conducted two set of experiments. In the first set, we study the effectiveness of our fast reconstruction techniques (Section 5.2). We evaluated the effectiveness of the computation plan with and without pre-computation and reusing of intermediate values against KAC's reconstruction. In the second experiment set, we examine the the speed up in reconstruction time at an expense of issuing more aggregated keys (Section 5.3). 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 sizes (m) and densities (r) 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 an arbitrary query, which could be a separate topic of interest.

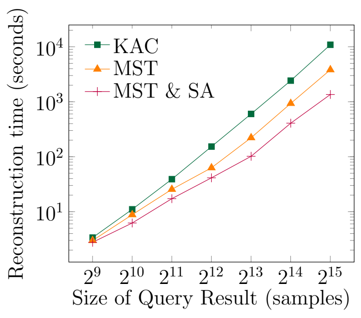
All experiments are performed on a system equipped with Intel Core-i5-4570u@3.2Ghz processor and 8GB of RAM. Our implementation employs Charm [1] and symmetric pairings over Type-A (supersingular) curves configured with 160-bit Solinas prime, offering 1024-bit of discrete-logarithm



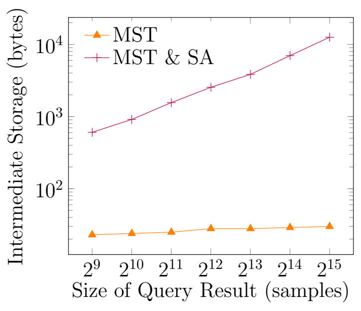
(a) Reconstruction time when r = 0.5



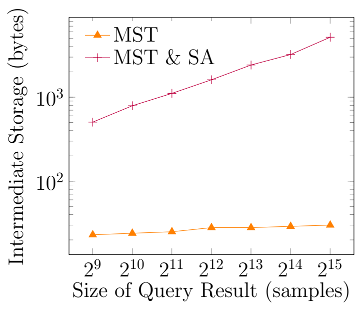
(b) Reconstruction time when r = 0.75



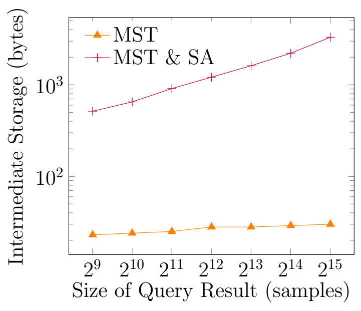
(c) Reconstruction time when r = 0.9



(d) Intermediate storage required in reconstruction when r = 0.5

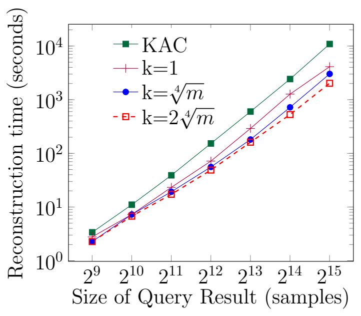


(e) Intermediate storage required in reconstruction when r = 0.75

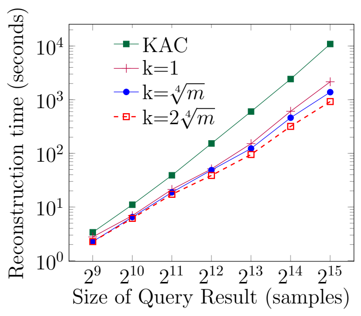


(f) Intermediate storage required in reconstruction when r = 0.9

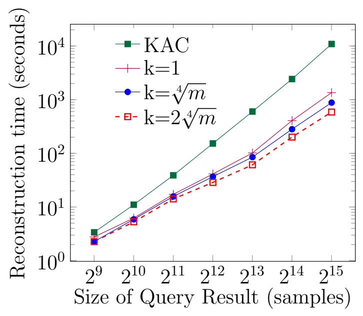
Figure 9: Fast reconstruction for Q3. MST indicates fast reconstruction using MCCP strictly without any intermediate values, while MST & SA represents MCCP with intermediate values being pre-computed and reused.



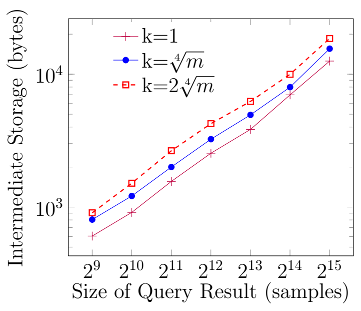
(a) Reconstruction time with $r = 0.5$



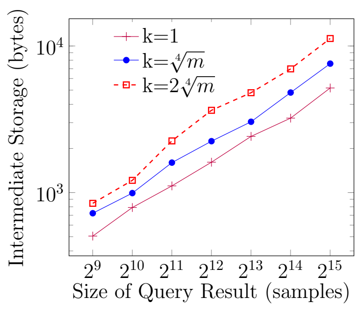
(b) Reconstruction time with $r = 0.75$



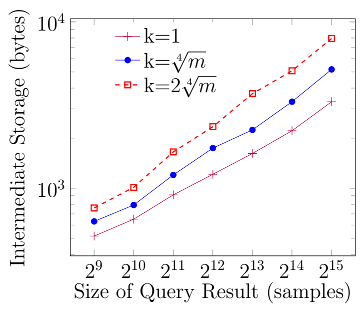
(c) Reconstruction time with $r = 0.9$



(d) Intermediate storage required in reconstruction when $r = 0.5$



(e) Intermediate storage required in reconstruction when $r = 0.75$



(f) Intermediate storage required in reconstruction when $r = 0.9$

Figure 10: Trade-off between number of aggregated keys and reconstruction cost for Q3. k is number of sub-queries, each sub-queries is associated with one aggregated key.

security[6]. For consistency, the KP-ABE implementation in our experiments is configured to provide the 80-bit security. Each experiment is repeated 10 times and average results, with time and size are measured in seconds and bytes respectively, are reported.

6.3 Experiment result

6.3.1 *Encryption time.*

Figure 5 compares the encryption time of KP-ABE and KAC (our system adopt KAC encryption procedure) under

[6]Though there exists no direct comparison, this level of security is approximate to the 80-bit security.

log-log scale, with the total number of samples range from 2^4 to 2^{18}. The experiment results agree with the analysis in the previous section. The cost of encryption incurred by KP-ABE is several times higher than that of KAC. For example, to encrypt 2^{18} items, KP-ABE needs 17 hours, while KAC only requires 70 minutes (i.e. faster by almost 15×). Note that the main overhead of KP-ABE's encryption lies in carrying out exponent operations, which directly depends on the total number of samples, and thus the overhead would be even higher for larger datasets.

6.3.2 Ciphertext Size

A main disadvantage of KP-ABE lies in its ciphertext size. Figure 6 reports total ciphertext size for various n - the total number of samples to be encrypted. When $n = 2^{18}$, KP-ABE produces ciphertext of size approximately 10× larger 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, the larger the dataset is, the more superior KAC is to KP-ABE in term of ciphertext size.

6.3.3 Aggregation time.

As shown in Table 1, KP-ABE's key aggregation time only 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 7).

6.3.4 Reconstruction time.

Figure 8 shows the reconstruction time for Q1 and Q2 queries. For small m, reconstruction time incurred by ABE is higher than KAC, which is due to the expensive pairing operations. However, for larger m, KAC starts to perform worse than KP-ABE because of the quadratic growth the number of required 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 (i.e. a speed-up of almost 90×).

For Q3 queries, we observe that the higher the query density is, the more effective our fast reconstruction techniques are. Though the gain is negligible when $r < 0.5$, it becomes more evident for larger r – achieving from 2.6× to 8× speedup over original KAC reconstruction cost. We also witness a better reconstruction time – upto 3× difference compared to fast reconstruction using computation plan strictly without any intermediate values – when intermediate values are pre-computed and reused (Figures 9a, 9b, 9c).

Figures 9d, 9e, 9f depict temporary storage required to save the pre-computed intermediate values during the reconstruction. At higher density, less temporary storage is needed, because the intermediate values are common to more elements. The requirement on temporary storage is at most a few KB (e.g. 12.5KB for reconstructing 2^{15} keys when $r = 0.5$).

We also evaluate the trade-off between number of aggregated keys and reconstruction time (Figures 10a, 10b, 10c). For a Q3 query asking for m samples, with $\sqrt[4]{m}$ aggregated keys, reconstruction time can be speeded-up by upto 13×. With a cost of $2\sqrt[4]{m}$ keys, upto 19× improvement can be achieved. Nevertheless, we note that for small queries, it is not worth issuing more aggregated keys, because the increase in the number of pairing operations may lengthen the reconstruction time.

Recall that intermediate values can only be shared among elements which are corresponding to the same aggregated keys. As more aggregated keys are issues, the reconstruction becomes more independent, and thus more intermediate values need to be stored. With $k = \sqrt[4]{m}$, as high as 15 KB of temporary storage is required, while that value is increased to 19 KB when $k = 2\sqrt[4]{m}$ are issued. In all of our experiments, the requirement on temporary storage is only a few KB, which is quite reasonable even for resource constrained devices.

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

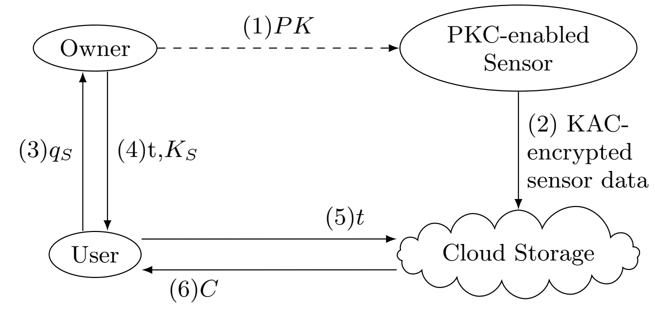


Figure 11: System model for PKC-enabled sensors

(1) During system setup, the owner distributes the public key PK to all entities, and an unique identity ID to each sensor (Figure 11). The identity ID's are not secrets and are made public. (2) 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.

(3) When an user asks for access to a subset C, whose indices fall in S, he sends the query q_S to the owner. (4) The owner issues an aggregated key K_S to the user, together with an authentication ticket t. (5) The user presents the authentication ticket t to the storage server as a proof that he is authorised to access the C. (6) 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 [22, 24]. 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

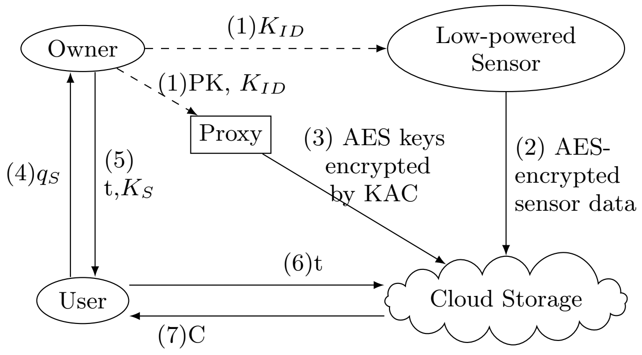


Figure 12: System model for low-powered sensors

Figure 12 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.

(1) 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. (2) 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.

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, (3) 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 12) 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 [30, 3]. 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 [32, 2]. Benaloh et al. introduced an encryption scheme supporting delegating decryption capability with flexible hierarchy [4]. 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 [9, 19]. 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 [18, 23]. 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 [7]. 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 [25]. 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 introduce algorithmic enhancement for the known KAC. The enhancement is significant for range and down-sampling queries, achieving 90 times speed-up in reconstructing 2^15 keys. Different heuristics for handling general queries attain upto 8 times speed-up in reconstructing the same number of keys. Finally, our clustering-based method for trading off between number of aggregated keys to be issued and reconstruction time is also proven to be efficient, as evidenced by the 19 times speed-up compared to original KAC. In sum, our proposed fast reconstruction techniques address a main hurdle in adopting key aggregation in large datasets.

10. REFERENCES

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

[2] 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.

[3] 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.

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

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

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

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

[8] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. An improved lp-based approximation for steiner tree. In *STOC*, 2010.

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

[10] 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.

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

[12] Y.-J. Chu and T.-H. Liu. On shortest arborescence of a directed graph. *Scientia Sinica*, 1965.

[13] M. R. Garey and D. S. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 1977.

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

[15] J. A. Hartigan. *Clustering algorithms*. 1975.

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

[17] M. H. Hayes. The reconstruction of a multidimensional sequence from the phase or magnitude of its fourier transform. *IEEE Trans Sig. Process*, 1982.

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

[19] A. Lewko, A. Sahai, and B. Waters. Revocation systems with very small private keys. In *IEEE S & P*, 2010.

[20] Y. Liu, B. Schmidt, and D. L. Maskell. Cudasw++ 2.0: enhanced smith-waterman protein database search on cuda-enabled gpus based on simt and virtualized simd abstractions. *BMC research notes*, 3, 2010.

[21] S. A. Manavski and G. Valle. Cuda compatible gpu cards as efficient hardware accelerators for smith-waterman sequence alignment. *BMC bioinformatics*, 9, 2008.

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

[23] 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.

[24] A. A. Pirzada and C. McDonald. Kerberos assisted authentication in mobile ad-hoc networks. In *ACSC*, 2004.

[25] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *IEEE S & P 2007*.

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

[27] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147, 1981.

[28] H. Takabi, J. Joshi, and G.-J. Ahn. Security and privacy challenges in cloud computing environments. *IEEE S & P*, 2010.

[29] R. E. Tarjan. Finding optimum branchings. *Networks*, 1977.

[30] W. G. Tzeng. A time-bound cryptographic key assignment scheme for access control in a hierarchy. *TKDE*, 2002.

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

[32] Q. Zhang and Y. Wang. A centralized key management scheme for hierarchical access control. In *GLOBECOM*, 2004.