Practical and Scalable Sharing of Encrypted Data in Cloud Storage with Key Aggregation

Hung Dang, Yun Long Chong, Francois Brun, Ee-Chien Chang
School of Computing, National University of Singapore
{hungdang,yunlong,francoisb,changec}@comp.nus.edu.sg

Abstract

We study a sensor network setting in which samples are encrypted individually using different keys and maintained on a cloud storage. For large systems, e.g. those that generate several millions of samples per day, fine-grained sharing of encrypted samples is challenging. Existing solutions, such as Attribute-Based Encryption (ABE) and Key Aggregation Cryptosystem (KAC), can be utilized to address the challenge, but only to a certain extent. They are often computationally expensive and thus unlikely to operate at scale. We propose an algorithmic enhancement and two heuristics to improve KAC's key reconstruction cost, while preserving its provable security. The improvement is particularly significant for range and down-sampling queries – accelerating the reconstruction cost from quadratic to linear running time. Experimental study shows that for queries of size 2^{15} samples, the proposed fast reconstruction techniques speed-up the original KAC by at least 90 times on range and down-sampling queries, and by eight times on general (arbitrary) queries. It also shows that at the expense of splitting the query into 16 sub-queries and correspondingly issuing that number of different aggregated keys, reconstruction time can be reduced by 19 times. As such, the proposed techniques make KAC more applicable in practical scenarios such as sensor networks or the Internet of Things.

1. INTRODUCTION

Incorporating cloud resources into wide-area sensor network [16] has 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 high risk of information leakage as demonstrated by many well-known incidents [26]. 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) and Key Aggregation Cryptosystem (KAC) can facilitate fine-grained access control over encrypted data, adopting these techniques in large-scale systems remains challenging.

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 and encrypt the samples individually using different encryption keys before streaming the encrypted content to the storage servers. The sample size and sampling rates are application specific (e.g hundreds of Kbytes per frame and 24 frames per second for video or only a few bytes per each sample and only one per second for temperature reading). In addition, each sample consists of multiple components; for example, scalable coding includes different resolution layers. The data owner wants to share selected samples with other users. The sharing policy may be quite complicated, e.g. sharing low resolution images captured by 100 cameras along a particular road segment, during every weekday from 6 am to 10 am at a reduced rate of one frame per second. The users can be third party cloud-service providers engaged by the data owner to perform certain processing, or personnels authorised to access certain sensors, etc. To handle multiple users while ensuring the principle of least privilege, 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 samples, 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 samples downloaded from the storage servers. However, for each sample is individually encrypted using different keys, 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 one 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, 29, 3] can address this issue to a certain extent. Unfortunately, these techniques are unlikely to operate at scale, and thus inapplicable in practical systems. In particular, key-policy Attributed-Based Encryption (KP-ABE)[14, 9] would lead to large overhead on the ciphertext

IHMMSec 2016, June 20-23, 2016, Vigo, Spain

DOI: http://dx.doi.org/10.1145/2909827.2930795

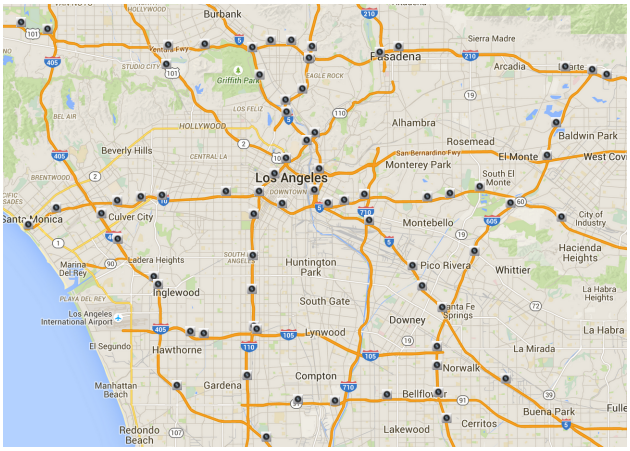


Figure 1: CCTV network in the City of Pasadena under the Real-Time Data Capture and Management Program. Each icon indicate location of a camera.

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 fine-grained sharing of encrypted data. The sharing mechanism in consideration should be not only secure but also practical and scalable. While the techniques that we propose are quite generic and applicable in a large body of application domains, hereafter we shall motivate and describe our approaches in the context of sensor data. 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 US Department of Transportations deployed a few hundreds cameras along roadway network in the City of Pasadena, California (Figure 1)[1]. Because of this spatial-temporal arrangement, the sensor data are often indexed by the their timestamps and sensors' locations. 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 and thus inherits its security. KAC enables aggregation of decryption keys for an arbitrary number of samples, say m, into a constant size key. Nevertheless, it incurs high cost in reconstruction, requiring $O(m^2)$ group multiplications to reconstruct all m samples. We make an observation that, for a large class of queries, the reconstruction time can be reduced by eliminating redundant or overlapping computations. In particular, we present a fast reconstruction technique attaining optimal linear running time; i.e. $O(m)$ reconstruction time 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 one sample per second instead of the original 24 samples per second) queries (of size m samples). In addition, we propose two heuristics to speed-up reconstruction time for queries of arbitrary form (general queries). The idea is to first approximate the optimal computation plan, and then perform the reconstruction following that computation plan. The computation plan describes a specific order in which a sequence of computations should be carried out so that redundant or overlapping computations can be avoided. In the other words, such computation plan would minimize the computation cost incurred in reconstructing the samples, resulting in a better performance as opposed to naively reconstructing each sample independently. Moreover, we also discuss a clustering-based method to trade-off the number of aggregated keys being issued for the reconstruction time. We remark that our proposed techniques address computational aspects of KAC reconstruction algorithm while preserving other characteristics of the scheme, including semantic security and collusion resistance. Therefore, our solutions are provably secure.

Experimental studies show that the proposed methods are efficient, outperforming relevant alternatives by significant factors. For queries of size 2^{15} samples, our fast reconstruction techniques attain at least 90 times speed-up over the original KAC on down-sampling and range queries, and eight times speed-up on queries of arbitrary form. The speed-up is 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.

The rest of this paper is organized as follows. We briefly review the KAC scheme in Section 2 and follow by stating our problem definition in Section 3. A few alternative constructions and their limitations are discussed in Section 4, before our fast reconstruction techniques are presented in Section 5. We report our experimental evaluation of the proposed techniques in Section 6. After that, we discuss two system designs in Section 7 and related works in Section 8 before finally concluding our work in Section 9.

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.

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 the aggregated key K_S, the user can decrypt any c_i as long as $i \in S$. However, she is unable to obtain any information on c_j for any $j \notin S$. KAC's security relies on decisional Bilinear Diffie-Hellman Exponent (BDHE)[5].

This cryptosystem comprises of five basic functions, *Setup*, *KeyGen*, *Encrypt*, *Aggregate*[2] and *Decrypt*.

- $param \leftarrow$**Setup**$(1^\lambda, n)$: Given security parameter λ and n, randomly pick 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 $\alpha \in_R \mathbb{Z}_p$, then compute $param = \langle g, g_1, g_2, \cdots, g_n, g_{n+2}, \cdots, g_{2n} \rangle$ where $g_i = g^{\alpha^i}$.

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

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

- $(PK, SK) \leftarrow$**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 $K_S = \prod_{j \in S} g_{n+1-j}^\gamma$.
- $\{x, \perp\} \leftarrow$**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 consists of one 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 S. 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.

While KAC has inspired various follow-ups [12, 28], to the best of our knowledge, we are the first to propose algorithmic enhancement for its key reconstruction, thus making it more applicable in practical systems.

3. PROBLEM DEFINITION

3.1 Sensor Data

We adopt the 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* $\mathcal{R}$ 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

Figure 2 illustrates our system model. To protect the confidentiality of sensor data, samples are individually encrypted using different keys before being streamed to the cloud. When a user wants to gain access to a set of encrypted sensor data C, which are indexed by a set S, she sends a query q_S to the owner. Upon approval, the owner issues an aggregated key K_S and an optional computation plan for reconstruction to the user. She can then download the set of requested encrypted samples C from the storage server and follow the computation plan to reconstruct (decrypt) them using K_S[3]. 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 defer the detailed discussion on system designs to Section 7.

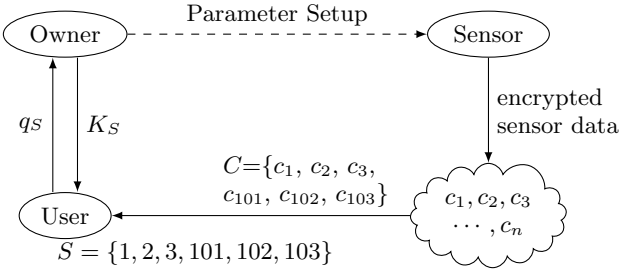


Figure 2: System model supporting fine-grained sharing of encrypted sensor data

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 maintained on the cloud storage. Nevertheless, she should not be able to learn the content of the encrypted samples without a permission granted by the data owner.

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 overheads incurred by the security mechanisms have 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:

[3]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.

Reconstruction time. Clearly, computation load of reconstructing the keys from the aggregated key K_S has to be low[4]. 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 for practical use.

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.

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 form a d-dimensional hyper-rectangle. For example, a request for images from cameras along a road segment during a certain period corresponds to a two-dimensional range query.

In some cases, it is possible to merge multiple range queries into one single range query. We can represent various constraints in one query by re-arranging and "lifting" one-dimensional component to multi-dimensions, e.g. 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 $\mathcal{L} = \{\sum_{i=1}^{t} a_i v_i | a_i \in \mathbb{Z}\} \cap \mathbb{L}_\Delta$ where each of the v_i is a d-dimensional vector and the basis $\{v_1, v_2, .., v_t\}$ is independent. This basis can also be used to represent the down-sampling query.

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 is not necessary a combination of range and down-sampling queries but rather asks for an arbitrary set of samples. 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, a Q3 query may asks for samples from an arbitrary set of sensors during all weekend's morning.

In this paper, we assume a simple distribution model for Q3 query: the set S (containing indices of all requested samples) contains $r\beta$ elements that are randomly selected from the interval $[1, \beta]$ where $r < 1$.

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

Remark.

Although we discuss the applications related to sensor networks and sensor data throughout the paper, our techniques can be straightforwardly applied to a wide range of applications which involve multidimensional data such as those that are related to the Internet of Things for example.

4. ALTERNATIVE CONSTRUCTIONS

In this section, we briefly discuss a few alternative cryptographic solutions and address 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 contains 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 [23]. 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 $\{1, 4, 5, 7, 10\}$.

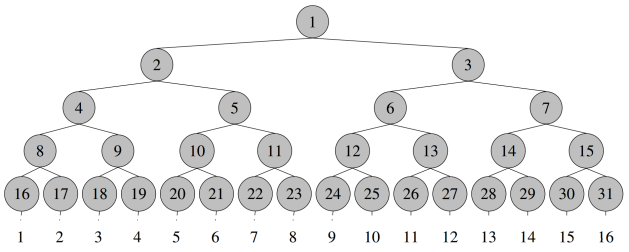


Figure 3: Tree based construction for one-dimensional data.

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 a 1 in the index's binary representation. For instance, the index $9 = 1001_2$ is represented by two 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 [23]. The work is 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 but rather 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 in Section 3.2, we adopt its encryption and key aggregation algorithms in our system. As such, our system inherits KAC's provable security. Interested 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):

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

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

$\hat{\rho}$ 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 incur $O(|S|^2)$ group multiplications to compute all necessary ρ_i (i.e. for all i in S). We observe that 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 A special recurrence relation

For Q1 and Q2 queries (or their combination), the indices of the requested samples follow specific patterns which straightforwardly permit fast computations. Let us first consider a one-dimensional Q1 query with 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 each $i \in S$, $\rho_i = \hat{g}_i^{-1} R_i$, and thus it can be easily computed from R_i. Now, we explore how to compute all R_is efficiently. Under the straightforward method, computing each R_i requires $|S| - 1$ multiplications, hence $|S|(|S| - 1)$ 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 two extra multiplications. This leads to a fast linear time algorithm that computes all R_is recursively, offering a significant speed-up over quadratic time as in the straightforward method.

We next show how to extend this observation on the recurrence relation to enable fast reconstruction for Q1 and Q2 queries. Interestingly, this can also be extended to improve computation cost of other cryptographic primitives whose constructions involve group multiplications, such as broadcast encryption [13] and redactable signatures [8]. This would be an interesting further extension.

5.1.2 Fast reconstruction for Q1 queries

Let us first consider two-dimensional lattice. Let $S = [1, m] \times [1, m]$ be a rectangular range in the two-dimensional lattice with bound n in both dimensions. As such, the indices are two-dimensional vectors. Let $\sigma(x_1, x_2) = x_1(n - 1) + x_2$ be the mapping function that maps the two-dimensional lattice to the one-dimensional lattice. It can be seen that decrypting a ciphertext with the index (i_1, i_2) requires the following value:

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

Similar to the simple one-dimensional example above, the term $n^2 + 1$ in the subscript is simply a fixed offset. Hence, the formula can be simplified by defining $\hat{g}_{(i_1,i_2)} = g_{n^2+1+\sigma(i_1,i_2)}$ and $R_{(i_1,i_2)}$ by:

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

It should be clear that obtaining the required $\rho_{(i_i,i_2)}$ from the corresponding $R_{(i_1,i_2)}$ is trivial. Exploiting the observation on the special recurrence relation that we make earlier, we can derive the following equation:

$$R_{(i_1+1,i_2)} = R_{(i_1,i_2)} \prod_{y=1}^{m} \hat{g}^{-1}_{(i_1,i_2)-(i_1-m,y)} \prod_{\bar{y}=1}^{m} \hat{g}_{(i_1,i_2)-(i_1,\bar{y})} \quad (3)$$

Let us dub the product of the first sequence $T_{(i_1,i_2)}$ and the second $\widetilde{T}_{(i_1,i_2)}$, Equation 3 becomes:

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

Now, observe that both $T_{(i_1,i_2)}$ and $\widetilde{T}_{(i_1,i_2)}$ in turn can also be expressed by recurrence relations, allowing the computations to be done in linear time. Consequently, evaluating all $R_{(i_1,i_2)}$s incurs only linear time, as opposed to quadratic time if the computation is to be done naively.

In general, for a d-dimensional range, the number of group multiplications required for computing all necessary ρ_i is in $O(d|S|)$. Since the number of dimensions (d) is deemed as a constant, we have derived a linear time approach to reconstruct Q1 queries.

5.1.3 Fast reconstruction for Q2 queries

Let us consider a Q2 query in two-dimension lattice (extending this to support higher dimension should be straightforward). Given a Q2 query represented by an independent basis, say $\{(3,0),(0,2)\}$ for example, one can first transform the coordinate system (e.g. transform (x,y) to $(x/3,y/2)$) such that indices of the required samples correspond to integer coordinates, and then apply the linear time reconstruction approach similar to that of Q1 queries on the transformed coordinate system and indices. We refer reader to [17] for further details on the transformation which could be applied on the coordinates.

In general, for a Q2 query that asks for samples in a d-dimensional range, the number of group multiplications required is also in $O(d|S|)$. Though additional computations are required to transform the coordinate, they are significantly less expensive than group multiplications. Therefore, we have derived a fast reconstruction method – running in linear time – for Q2 queries.

5.2 Fast reconstruction for Q3 queries

The techniques we discuss above reuses common terms and recurrence relations among different ρ_is to save computations. They 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 minimum computation cost (i.e., the least number of multiplications) for an arbitrary query. For the ease of exposition, we assume that some normalization (similar to the mapping function σ mentioned in Section 5.1.2) has been applied such that indices of queried samples are mapped to a one-dimensional set S. We shall use the set $S = \{1,2,3,6,9,10\}$ and system capacity (maximum number of ciphertexts that the system can support) $n = 20$ as the running example (depicted in Table 1).

Let us denote by $\mathcal{B}$ a set of singletons whose elements are indices of g in the public parameter $param$, and represent each ρ_i by a multi-set P_i comprising of indices of g in a sequence that computes ρ_i following Equation 1. Let $\mathcal{P}$ – the target collection – be a set of all such P_is. In the running example (Table 1), $\mathcal{B} = \{\{1\},\{2\},\ldots,\{40\}\}$, ρ_1 is represented by $P_1 = \{20,19,16,13,12\}$ and the target collection $\mathcal{P} = \{P_1,P_2,P_3,P_6,P_9,P_{10}\}$

The problem of computing all necessary ρ_i is now reducible to the problem of constructing $\mathcal{P}$ from the collection of singletons $\mathcal{B}$. Each multiplication is represented by a "computation step" performing either a union or a subtraction on two multi-sets.

Table 1: An exemplar Q3 query asking for a six samples whose indices are $S = \{1,2,3,6,9,10\}$. The system capacity is $n = 20$. To reconstruct these samples, six corresponding ρ_i ($i \in S$) have to be computed. The target collection is $\mathcal{P} = \{P_1,P_2,P_3,P_6,P_9,P_{10}\}$.

$\rho_1 = g_{20} \cdot g_{19} \cdot g_{16} \cdot g_{13} \cdot g_{12}$	$P_1 = \{20,19,16,13,12\}$
$\rho_2 = g_{22} \cdot g_{20} \cdot g_{17} \cdot g_{14} \cdot g_{13}$	$P_2 = \{22,20,17,14,13\}$
$\rho_3 = g_{23} \cdot g_{22} \cdot g_{18} \cdot g_{15} \cdot g_{14}$	$P_3 = \{23,22,18,15,14\}$
$\rho_6 = g_{26} \cdot g_{25} \cdot g_{24} \cdot g_{18} \cdot g_{17}$	$P_6 = \{26,25,24,18,17\}$
$\rho_9 = g_{29} \cdot g_{28} \cdot g_{27} \cdot g_{24} \cdot g_{20}$	$P_9 = \{29,28,27,24,20\}$
$\rho_{10} = g_{30} \cdot g_{29} \cdot g_{28} \cdot g_{25} \cdot g_{22}$	$P_{10} = \{30,29,28,25,22\}$

With these notions, we are ready to define the computation plan.

Definition 1 (Computation Plan). *Given $\mathcal{B}$ and the target collection $\mathcal{P}$, the computation plan is a sequence of computation steps $\mathcal{M} = \{m_1,m_2,\ldots,m_z\}$ that constructs a set of multi-sets $\mathcal{A}$ such that $\mathcal{P} \subset \mathcal{A}$.*

The set of multi-sets $\mathcal{A}$ is initiated to $\mathcal{B}$. Each computation step picks, with replacement, two multi-sets in $\mathcal{A}$, either unions or subtracts them from one another, and inserts the resulting multi-set to $\mathcal{A}$. The computation plan is optimal if it has the minimum number of computation steps.

In Section 5.1, we have seen that introducing $T_{(i_1,i_2)}$ significantly reduces the number of multiplications required in computing $R_{(i_1,i_2)}$, suggesting that introducing appropriate intermediate values serves as a good heuristic in evaluating the optimal computation plan[5]. Unfortunately, the space of possible intermediate values are exponentially large, making the choice of appropriate intermediate values difficult. Indeed, though we do not attempt to give a formal proof, we believe that computing the optimal computation plan may very well be NP-complete. We provide below heuristics to approximate the optimal computation plan.

5.2.1 Minimum Spanning Tree based Strategy

For any two P_i, P_j, let us define their distance as:

$$dist(i,j) = |P_i \setminus P_j| + |P_j \setminus P_i|$$

If P_j is already constructed, one can derive P_i from P_j with at most $dist(P_i,P_j)$ computation steps. In the best case where the two multi-sets $(P_i \setminus P_j)$ and $(P_j \setminus P_i)$ have already been inserted to $\mathcal{A}$, P_i can be derived from P_j with only two computation steps. Based on this notion of $dist(i,j)$, we can evaluate the computation plan by solving the following minimum spanning tree (MST) problem.

Let $G = (V,E)$ be a complete graph in which V and E denote the set of vertices and the set of edges, respectively. The set V comprises of $|\mathcal{P}|+1$ vertices, representing multi-sets in $\mathcal{P}$ and an additional empty set $\bar{P}$. Each P_i maps to a vertex v_i, and $\bar{P}$ maps to a special vertex $\bar{v}$. The set E contains $|V|(|V|-1)/2$ edges. Let us denote by e_{ij} an

[5]The computation plan need not be evaluated in real-time. Since queries are likely to be repeated, the computation plans can be computed in offline sessions, probably by the data owner or on the server with presumable powerful resource. On another note, parallelizing the reconstruction computation is possible so long as the users' computation resource allows so. In such situation, values that should be evaluated independently can also be inferred from the computation plan.

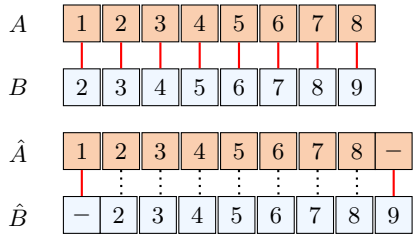


Figure 4: An example of sequence alignment. A dotted link denotes zero penalty cost while a solid link represents penalty cost of one. By inserting "gaps" at the beginning of A and at the end of B, we obtain $\hat{A}$ and $\hat{B}$ which yield the optimal alignment with total penalty of only two.

edge connecting vertex v_i to vertex v_j. For each edge e_{ij}, we set its weight to $dist(i, j)$. For edges originating from the special vertex $\bar{v}$, we set their weights to $|\mathcal{P}| - 2$.

With the reduction described above, we can use any minimum spanning tree algorithm, such as $Chu-Liu/Edmonds$ algorithm running in $O(|V|^2)$ time [27], to approximate the optimal computation plan.

5.2.2 *Introducing intermediate values*

As discussed earlier, the computation cost can be further reduced if common intermediate values are introduced properly. In particular, let us denote the set of those intermediate values as $\mathcal{I}$. We can approximate an even better computation plan by introducing these intermediate values to $\mathcal{P}$, obtaining $\mathcal{P}' = \mathcal{P} \cup \mathcal{I}$, and then apply the MST-based strategy described earlier to find the MST of the complete graph $G' = (V', E')$ in which V' represents the set of multi-sets $\mathcal{P}'$ and one additional empty set $\bar{P}$. Since the space of all possible intermediate values are exponentially large, it is not clear how to efficiently choose the most appropriate intermediate values. We provide here an intuitive heuristic to determine those common intermediate values.

Should one interpret each P_i as a sequence of elements, and intermediate values as shorter sequences, the problem of finding the common intermediate values is reducible to the *local sequence alignment* problem [25] — which is tasked to determine "similar regions" between sequences.

Sequence alignment is considered a textbook example for dynamic program. In the most basic version, the sequence alignment problem takes as input two sequences $A = a_1 \dots a_y$ and $B = b_1 \dots b_z$ 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 [20], to solve the sequence alignment problem.

Let us consider an example in Figure 4. The two input sequences are $A = 12345678$ and $B = 23456789$, $\alpha_{gap} = 1$, $\alpha_{ab} = 1$ 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 intermediate values are the similar regions found in the solution to the sequence alignment problem. For example, if we are to construct $P_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$ and $P_j = \{2, 3, 4, 5, 6, 7, 8, 9\}$, we can represent P_i and P_j by the sequences A and B in the above example (Figure 4). Since the similar region of A and B is $R = 2345678$, the intermediate multi-set is $I = \{2, 3, 4, 5, 6, 7, 8\}$. Effectively, we can construct both P_i and P_j from I with only two extra computation steps (one for each of them).

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

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. One may partition the set S into k clusters, each of which corresponds to one sub-query, and issue one aggregated key for each sub-query. Accordingly, one needs to issue k aggregated keys instead of one single key. At the expense of issuing k aggregated keys, the reconstruction time can be reduced.

The partition should be performed such that elements in each cluster are "close" to each other. In another word, for two neighbour indices i and j belonging to the same cluster, it should be the case that ρ_i can be derived from ρ_j with only a small number of group multiplications. Let us informally define the distance between two indices i, j in the same cluster as follows. Let v be the common intermediate value of ρ_i and ρ_j, their distance is the number of group multiplications it takes to compute both values ρ_i and ρ_j from the common intermediate value v.

We now define the distance function between two cluster S_a and S_b. Let $W(S_a)$ be the number of group multiplications required to compute all ρ_i where $i \in S_a$. This value can be determined via the computation plan discussed previously in Section 5.2. For two clusters S_a and S_b, their distance is simply $W(S_a \cup S_b)$. Note that $W(S_a \cup S_b)$ is not necessarily equal to the sum of $W(S_a)$ and $W(S_b 2)$.

We employ the single-linkage clustering method [15] (implemented using *SLINK* algorithm [24]) to perform the clustering. In particular, each element in S is initially a cluster by itself. Each step would choose two clusters with the shortest distance (using the distance function defined above) and merge them together. The clusters are sequentially merged until only k clusters are left.

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.

6. PERFORMANCE EVALUATION

In this section, we compare performance of our proposed fast reconstruction techniques with KP-ABE [19] and the original KAC [10].

6.1 Performance Analysis

Table 2 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

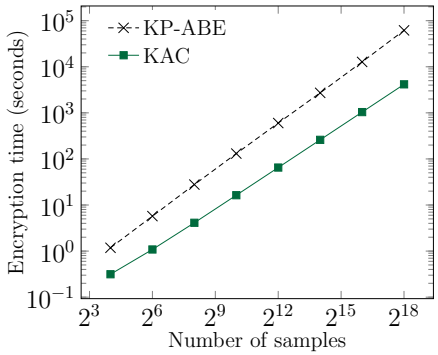


Figure 5: Encryption time

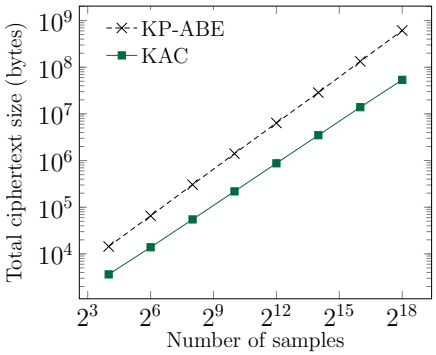


Figure 6: Total ciphertext size

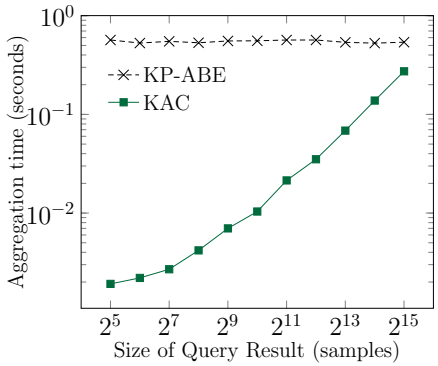


Figure 7: Aggregation time

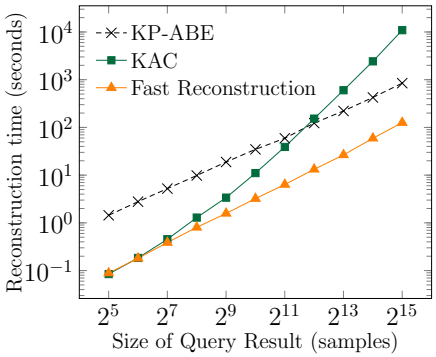


Figure 8: Reconstruction time for Q1 & Q2.

Table 2: Costs of encryption, 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

suffers from a $O(\log n)$ overhead factor in comparison with KAC, which is 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, the system of 100 sensors will generate almost a quarter billion samples every day), such 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, rendering the scheme impractical in our application. By adopting KAC encryption and key aggregation procedures, and introducing various techniques to improve 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, and achieve several times speed-up for other queries.

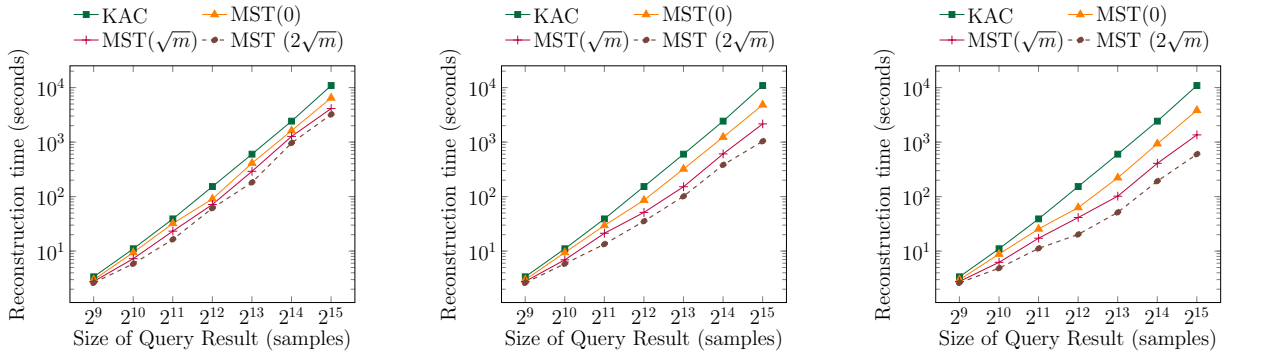
In key aggregation, KAC requires more group multiplications but fewer number of exponentiations compared to KP-ABE. Because exponentiation is more computational expensive than group multiplication, the efficiency of the two schemes depends on the scale at which they operate. In particular, KAC would perform better than KP-ABE in aggregating a small number of keys, but worse when the number of keys is large (see Figure 7).

6.2 Experimental Setup

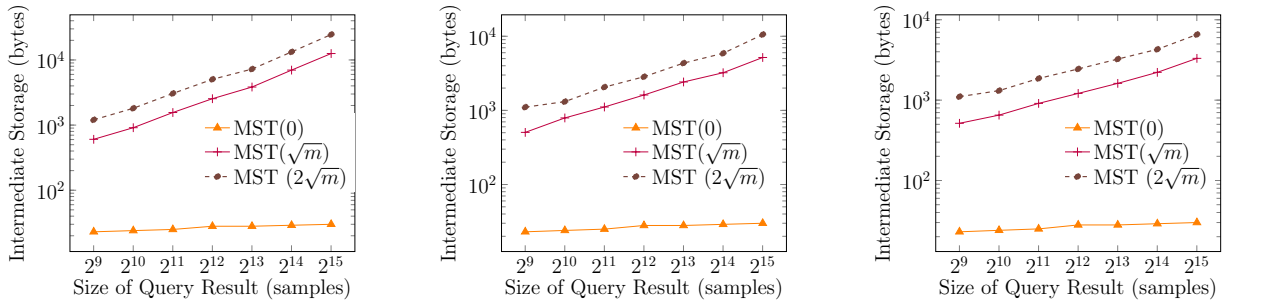
In practice, when the size of a single sample is large, it is more efficient to encrypt the sample using symmetric encryption such as AES with a randomly chosen key, then apply the key aggregation on the symmetric keys. We follow such fashion in our experiments.

To evaluate the performance of our fast reconstruction for Q1 and Q2 queries in comparison with KP-ABE and the original KAC, we fix the total number of encrypted samples at 2^{18}, while varying the query size $m = |S|$ from 2^5 to 2^{15}. The queries that we use in our experiments are two-dimensional range queries (having the same width along both dimensions) with down-sampling rate equal to 1/4.

For Q3 queries, we perform two set of experiments. In the first set, we study the effectiveness of the computation plan constructed with l intermediate values against KAC's reconstruction (Section 5.2). In the second experiment set, we examine the speed-up in reconstruction time at the expense of issuing more aggregated keys (Section 5.3). The queries are generated by selecting m indices randomly from

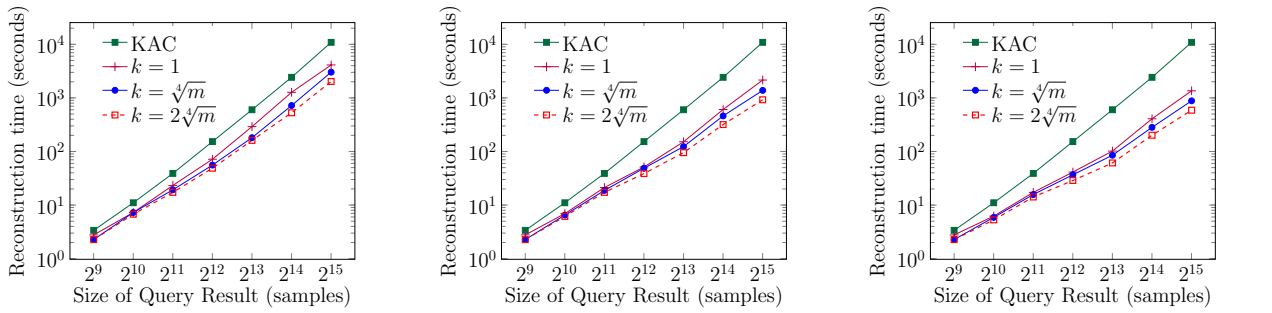


(a) Reconstruction time when r = 0.5 (b) Reconstruction time when r = 0.75 (c) Reconstruction time when r = 0.9

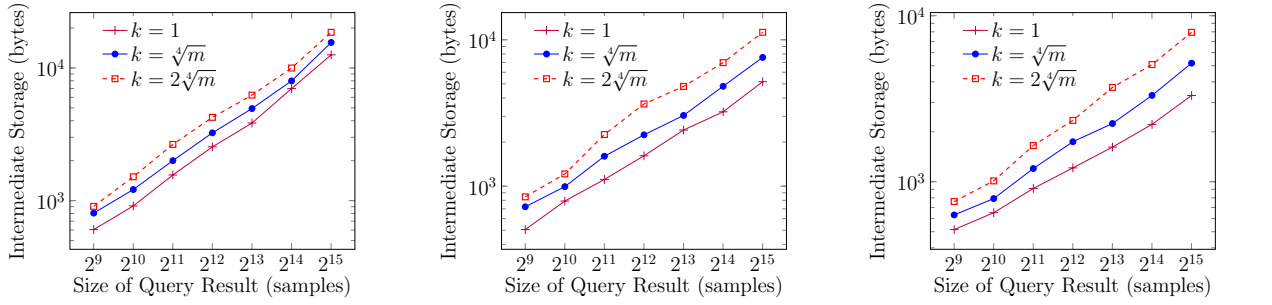


(d) Intermediate storage required in reconstruction when r = 0.5 (e) Intermediate storage required in reconstruction when r = 0.75 (f) Intermediate storage required in reconstruction when r = 0.9

Figure 9: Fast reconstruction for Q3. MST(l) indicates performance of the computation plan constructed l intermediate values. Presumably, $l = 0$ indicates introducing no intermediate values. m is the size of query result.



(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-query is associated with one aggregated key. m is the size of query result.

the range $[1, n]$. We define *query density* (r) as the ratio m/n. Various query sizes (i.e., m) and densities (i.e., 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] crypto-

graphic framework and utilizes symmetric pairings over Type-A (supersingular) curves. The KAC implementation and our fast reconstruction techniques are configured with 160-bit Solinas prime, offering 1024-bit of discrete-logarithm security. For consistency, the KP-ABE implementation is configured to provide 80-bit security. Although there exists no direct comparison between discrete-logarithm security and bit-security, 1024-bit of discrete-logarithm security is often considered to be equivalent to the 80-bit security. Each experiment is repeated ten times and average results — with time measured in seconds while storage and ciphertext size measured in bytes — 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 log-log scale, with the total number of samples varied 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 1.17 hours (i.e. faster by almost $15\times$). 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 $10\times$ 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 2, 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 compared 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 all the keys within 126 seconds, whereas KAC needs three hours (i.e. a speed-up of almost $90\times$).

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 $2.6\times$ to $8\times$ speed-up over original KAC reconstruction cost. We also witness a better reconstruction time – upto $3\times$ improvement as 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 for maintaining all intermediate values computed during the reconstruction. This temporary storage is at most a few KB (e.g. 12.5KB for reconstructing 2^{15} keys when $r = 0.5$). As can be seen from the figures, the higher the density is, the less temporary storage is required.

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 $13\times$. With a cost of $2\sqrt[4]{m}$ keys, upto $19\times$ 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.

In another note, the more aggregated keys are issued, the 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

(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 a 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 ticket t to the storage server as a proof that he is authorised to access 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

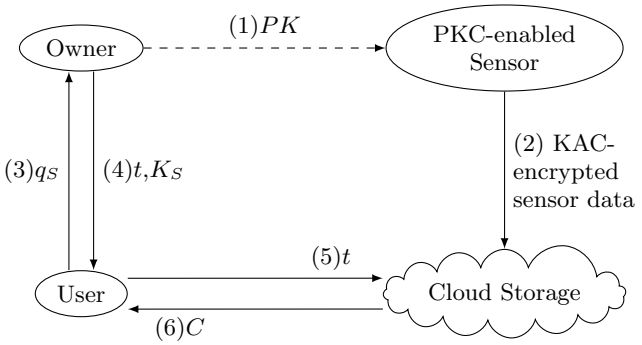


Figure 11: System model for PKC-enabled sensors

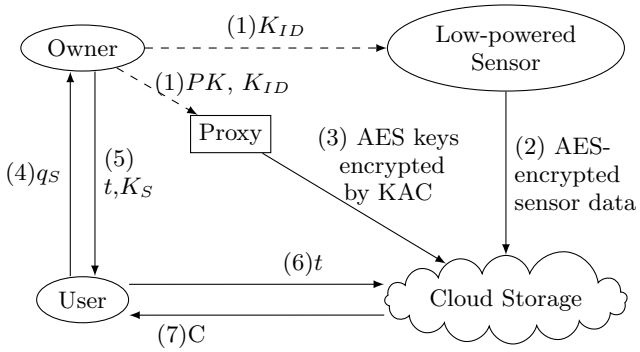


Figure 12: System model for low-powered sensors

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 [21, 22]. Although this cannot prevent the collusion between the users and the server, it forms another layer of defence to prevent unauthorised downloading of the ciphertexts.

7.2 System with 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 the system setup phase, 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 seed 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 using the secret seed K_{ID} and the index i. The sensor then encrypts the sample value x with encryption key $k_{i,ID}$, 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

Several cryptographic key assignment schemes exploit hierarchical structures (e.g. trees) to maintain keys for various sets of objects [29, 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 [2]. Benaloh et al. introduced an encryption scheme supporting delegating decryption capability with flexible hierarchy [4]. 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.

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. Such key can decrypt all ciphertexts whose attributes conform to its policy [9, 19]. ABE attains collusion-resistance at a cost of either increasing the secret keys size or ciphertext's size [18]. These approaches requires many bilinear-mapping operations in their executions, rendering their performance prohibitive and thus impractical.

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 [23]. 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.

While KAC has inspired various follow-ups [12, 28], those works have not yet focused on improving KAC's key reconstruction cost. To our knowledge, the techniques proposed in this paper are the first algorithmic enhancement to reduce the key reconstruction cost of KAC. This not only makes KAC more applicable in practical systems, but also benefits those follow-up works that employs KAC as the underlying cryptographic primitive.

9. CONCLUSION

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 in 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 Q1 and Q2 queries, achieving 90 times speed-up in reconstructing 2^{15} keys. To deal with more general applications, we generalize the technique and provide heuristics for handling arbitrary queries. These heuristics attain upto eight times speed-up over the original KAC. 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, evidenced by the 19 times speed-up.

Our proposed fast reconstruction techniques resolve the scalability issue in adopting key aggregation in practical application with large datasets. This makes the KAC more applicable in various scenario and system settings including the Internet of Things. More interestingly, the implication of our work is much more. Our observation on the recurrence relations, and the techniques we propose to exploit such relations for better computation cost can also be applied on other cryptographic primitives whose constructions involve group multiplications, such as broadcast encryption [13], redactable signatures [8].

10. ACKNOWLEDGMENT

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its National Cybersecurity RD Program (Award No. NRF2015-NCR-NCR002-001) and administered by the National Cybersecurity RD Directorate. All opinions expressed in this work are solely those of the authors.

11. 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*. 2007.

[8] E.-C. Chang, C. L. Lim, and J. Xu. Short redactable signatures using random trees. In *CT-RSA*. 2009.

[9] M. Chase and S. S. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *CCS*, 2006.

[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] H. Deng, Q. Wu, B. Qin, S. S. Chow, J. Domingo-Ferrer, and W. Shi. Tracing and revoking leaked credentials: accountability in leaking sensitive outsourced data. In *ASIACCS*, 2014.

[13] A. Fiat and M. Naor. Broadcast encryption. In *CRYPTO*, 1994.

[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*, 2009.

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

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

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

[23] 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.

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

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

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

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

[28] Y. Tong, J. Sun, S. S. Chow, and P. Li. Towards auditable cloud-assisted access of encrypted health data. In *CNS*, 2013.

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