A Light-Weight Permutation based Method for Data Privacy in Mobile Cloud Computing

Mehdi Bahrami[1] and Mukesh Singhal[2]

Cloud Lab
University of California, Merced, USA

[1]IEEE Senior Member, MBahrami@UCMerced.edu
[2] IEEE Fellow, MSinghal@UCMerced.edu

***Abstract*—Cloud computing paradigm provides virtual IT infrastructures with a set of resources that are shared with multi-tenant users. Data Privacy is one of the major challenges when users outsource their data to a cloud computing system. Privacy can be violated by the cloud vendor, vendor's authorized users, other cloud users, unauthorized users, or external malicious entities. Encryption is one of the solutions to protect and maintain privacy of cloud-stored data. However, encryption methods are complex and expensive for mobile devices. In this paper, we propose a new light-weight method for mobile clients to store data on one or multiple clouds by using pseudo-random permutation based on chaos systems. The proposed method can be used in the client mobile devices to store data in the cloud(s) without using cloud computing resources for encryption to maintain user's privacy. We consider JPEG image format as a case study to present and evaluate the proposed method. Our experimental results show that the proposed method achieve superior performance compared to over encryption methods, such as AES and encryption on JPEG encoders while protecting the mobile user data privacy. We review major security attack scenarios against the proposed method that shows the level of security.**

Keywords— Mobile Cloud Computing; Privacy; Permutation; Data storage; JPEG encryption; Chaos System;

I. INTRODUCTION

Cloud computing paradigm refers to a set of virtual machines (VM) that provide computing and storage services through the Internet [1, 2]. Cloud computing uses virtualization technology to provide VMs on the top of distributed computing systems. Cloud computing provides several advantages over traditional in-house IT [3], such as on-demand services, pay-per-use basis and elastic resources which have rapidly made cloud computing a popular technology in different fields, such as IT business, mobile computing systems and health systems [4, 5]. Huang [6] defines Mobile Cloud Computing (MCC) is rooted from mobile computing and cloud computing paradigm that allows mobile users offload their data and process data on cloud computing. MCC provides a big data storage for mobile users with a lower cost and more ease of use. As of today, some popular MCC cloud vendors for storage service, such as Google and Dropbox, provide a free standard storage service with 2GByte and 15GByte capacity, respectively[1].

MCC provides an online massive storage for mobile users, but it aggravates the user data privacy issues because users have to trust third-parties (cloud vendors and their partners). For instance, in a recent study, Landau [7] reports some challenges toward privacy issues when users trust cloud vendors, and the vendor shares data with a third-party or other unauthorized users could have access to these data. In another study, Kumar et al. [8] suggest that not all MCC applications can save energy on mobile devices by offloading data. Finally, in an earlier study on cloud computing, Ristenpart et al. [9] show a VM can be a vulnerable component when users use cloud shared resources.

These examples of challenges in MCC show a mobile user requires an encryption method to protect their data privacy. One of encryption methods that can be considered as a secure method is Advanced Encryption Standard (AES) [10-12]. However, mobile devices have limited resources, such as limited power energy, low speed CPU and small capacity of RAM, and it is impossible to use AES encryption method for each file when offload/download is required for each file. Another solution for this challenge is light-weight security methods that provide a balance between maintaining energy efficiency and security. A light-weight security method is based on simple operations, such as permutation, rather than using expensive operations, such as secret key or public-key encryptions,

In this paper, we propose a light-weight encryption method for mobile devices, such as smart phones, that uses permutation operation to protect data privacy. We investigate one of the popular image file formats as a case study, JPEG file formats because it is most world's widely used by smart phones for capturing pictures. We show that the proposed light-weight method provides a secure data privacy model based on chaos systems for JPEG file format in mobile devices. In addition, we evaluate the performance of the proposed method against other encryption methods, such as AES, and encryption on JPEG encoders, such as pixel and color encryptions. We investigate several attack scenarios against the proposed method.

The rest of the paper is organized as follows: in Section II, we briefly review background materials for this study. In Section III, we present the proposed method and its

[1] Recorded on November 13, 2014 from https://drive.google.com and https://www.dropbox.com

requirements. In Section IV, we implement the proposed method and present its experimental results. We also present a statistical security model of the proposed method to show the level of the security. In Section V, we investigate different scenario attacks against the proposed method. In Section VI, we review a comparison between the proposed method and existing methods. Finally, in Section VII, we conclude the proposed method and its evaluations.

II. BACKGROUND

In the proposed method, we use JPEG file format as a case study. In this section, we briefly review the background of JPEG file format and its encoder/decoder requirements.

Each JPEG file has a header file that defines metadata, such as the canvas of a JPEG file with a specific dimension (width and height), resolution, camera information, GPS information, and compression information.

Each JPEG file (including the header and the content) consists of several segments and each segments beginning with a marker. A raw data of a JPEG file includes several markers [13]. For example, each JPEG file begins with "0xFF0xD8" marker that represents this binary is a type to image and it allows an image viewer application to decode the binary file and show the image. Each marker begins with '0xFF'. Table 1 describes some important markers.

We use these markers in our proposed method to retrieve different segments of a JPEG files.

A JPEG image encoder uses a lossy form of compression which is based on the discrete cosine transform (DCT) method [13] to compress an image. A sequence byte of raw JPEG image file contains a multiple chunk of Minimum Codded Unit (MCU) [13]. Each MCU block stores 4*4 pixels of an image.

TABLE I. JPEG FILE FORMAT

Short Name	The description of marker	Bytes
SOI	Start Of Image	0xFF, 0xD8
SOF0	Start Of Frame (Baseline DCT)	0xFF, 0xC0
SOF1	Extended sequential DCT	0xFF, 0xC1
SOF2	Start Of Frame (Progressive DCT)	0xFF, 0xC2
DHT	Define Huffman Table(s)	0xFF, 0xC4
DQT	Define Quantization Table(s)	0xFF, 0xDB
DRI	Define Restart Interval	0xFF, 0xDD
SOS	Start Of Scan	0xFF, 0xDA
EOI	End Of Image	0xFF, 0xD9

III. THE PROPOSED METHOD

In this section, we present the proposed method to protect the privacy of data on MCC. As described in Section I, since mobile devices have limited resources, we have to provide a method with minimum overhead to protect data privacy that runs on a mobile device to store and retrieve image files on MCC. In this paper, we consider JPEG files as a case study because this format of photography by smart phones is popular at this time.

We assume the following requirements for the proposed method:

- The privacy model must satisfy a balance between computation overheads and maintaining the security.
- Unlike default MCC offloading methods that submit original files to MCC for encryption, the proposed method can be ran on mobile devices to provide data privacy, and then, the protected result will be stored on MCC.

The proposed method splits files into multiple files and uses a pseudo-random permutation to scramble chunks in each split file. The proposed method reads a JPEG file as a binary file rather than using a JPEG encoder/decoder to protect each pixel or the color of each pixel.

In this method, we have two phases to split files into multiple files and to recombine the files as follows:

- *Disassembling of an image* that splits an image file into multiple binary files, divide the original file into: (i) one file that contains the header of the original file, and (ii) multiple files that contain the content of the original file. The content of each split file consists of multiple chunks of original file. Chunks distribute through multiple files based on a *Pattern*, chunks in each file randomly scramble by using the chaos system. The output of this phase (split files) will be stored in MCC(s).
- *Assembly of split files* that recombine all split files to reorganize the original file. In this phase the following steps will be proceed: (i) read all scramble files from MCC(s); (ii) using the chaos system random arrays (which are used at the first phase) to reorder the chunks in each split file; and (iii) use the *Pattern* to reorganize the original files.

A. Disassembly Phase

We assume the proposed method requires to disassemble a series of images at the same time. The method divides each original file ($File_i$) into: (i) the header of the original file ($Header_i$), where i is the sequence number of original file in the file series that requires to be disassembled; (ii) the content of file that is divided into several chunks ($Chunk_{i,j}$) where j is the sequence number of the chunks of the original content file.

We divide the original JPEG file into header and the content because the header of the original file carries some important privacy information. This division also provides a complex method for recombining split files because it removes important JPEG markers from the original image file.

A split file is defined as follows:

$$File_i = Header_i + (\sum_{j=1}^{j=cmax} Chunk_{i,j}) \quad (1.)$$

where $cmax$ represents the maximum number of chunks in $File_i$ and is defined as follows:

$$cmax = \left\lceil \frac{Size_i}{Buffer} - HSize_i \right\rceil \quad (2.)$$

where $Size_i$ represents the size of $File_i$ (Byte), $Buffer$ represents the size of chunks (Byte), and $HSize_i$ represent the size of the header of the original $File_i$ (Byte).

Figure 1, illustrates a general view of the proposed method that allows a mobile device to split an original JPEG file into header file and three multiple files. The split files is submitted to two MCCs. A user can configure his/her application to set: (i) the number of split files, (ii) the size of chunks, and (iii) the cloud user account(s) information to upload files on MCC(s).

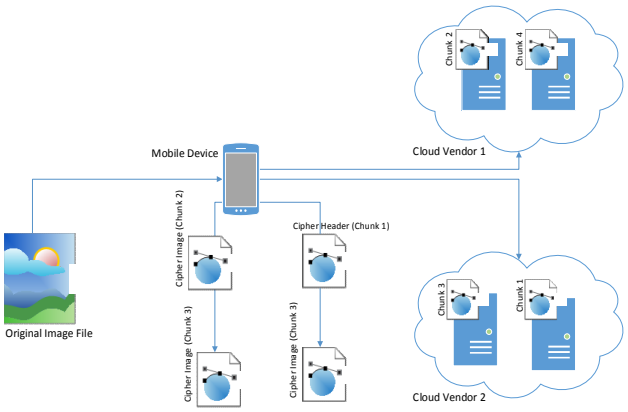


Fig. 1. A general view of the proposed Method

We use the JPEG markers to split the header of the original file from the content and to find important JPEG markers. If someone has an access to split files to assemble different files without accessing to the header, or if the person creates a new header for a JPEG file, still he/she cannot simply retrieve the file. For example, Figure 2 shows two pictures (a) and (b), with the same size and the same resolution that took by a smart phone. The last right frame (c) shows the result of assembling the header of the first file and the content of the second file. As shown in this figure, only the size and the resolution can be retrieved and still the assembler cannot retrieve the content of the image. To protect an image from a person who attempt to assemble a part of image by assembling files, first, we use a *pattern* to distribute chunks of the image to different files. The *pattern* can be defined as an input by a user to indicate, how to distribute a sequence of bytes in a split file. Second, we use chaos theory to randomly distribute each chunk in each split file.

The proposed method uses two steps to disassemble an original JPEG file to a number of chunks, and to scramble randomly each chunk in each split file as follows:

1) Pattern

The proposed method divides each file into $cmax$ chunks (binary codes) and it distributes chunks to multiple split files in different order as will be discussed in the next phase. At this phase, the method use *pattern* that aims to distribute chunks to multiple split files. A *pattern* can be defined as a key by a user or it can be selected randomly as a predefined method. A user can define different *patterns* to provide different strategy for distribution. For example, Figure 3 shows two different patterns for disassembling a JPEG file. In this figure, the original file includes a header, and nine chunks of content. In $Pattern_A$, the proposed method reads two consecutive chunks from the original file and stores each chunk in one of two files. The first chunk stores in $File_{1,2}$ and the second chunk stores in $File_{1,3}$. At the end of reading $Source\ File_1$, $File_{1,2}$ contains $\{B_1, B_3, B_5, B_7\}$ and $File_{1,3}$ contains $\{B_2, B_4, B_6, B_8\}$. In Figure 3, $Pattern_B$ shows another pattern to store chunks. In this pattern, each four consecutive chunks of original file store in two files. The first and the forth chunks store in $File_{2,2}$; the second and the third chunks store in $File_{2,3}$. At the end of reading $Source\ File_2$, $File_{2,2}$ contains$\{B_1, B_4, B_5, B_8, B_9\}$ and $File_{2,3}$ contains $\{B_2, B_3, B_6, B_7\}$.

If someone has an access to all contents, still he/she cannot assemble files because he/she needs an access to the pattern as a key to assemble split files.

2) Scrambling

We use the pseudo-random permutation (PRP) [14] with chaos [15] to scramble chunks in each split file. PRP uses chaos system which is defined as follows:

$$P_{k+1} = \mu P_k (1 - P_K) \quad (3.)$$

where $P \in \{0,1\}$ and μ is a parameter of this equation.

(a) Source File #1 (b) Source File #2 (c)Header of file #1 with the content of file #2

Fig. 2. Split the header of files and subsitute header of file #1 with file #2

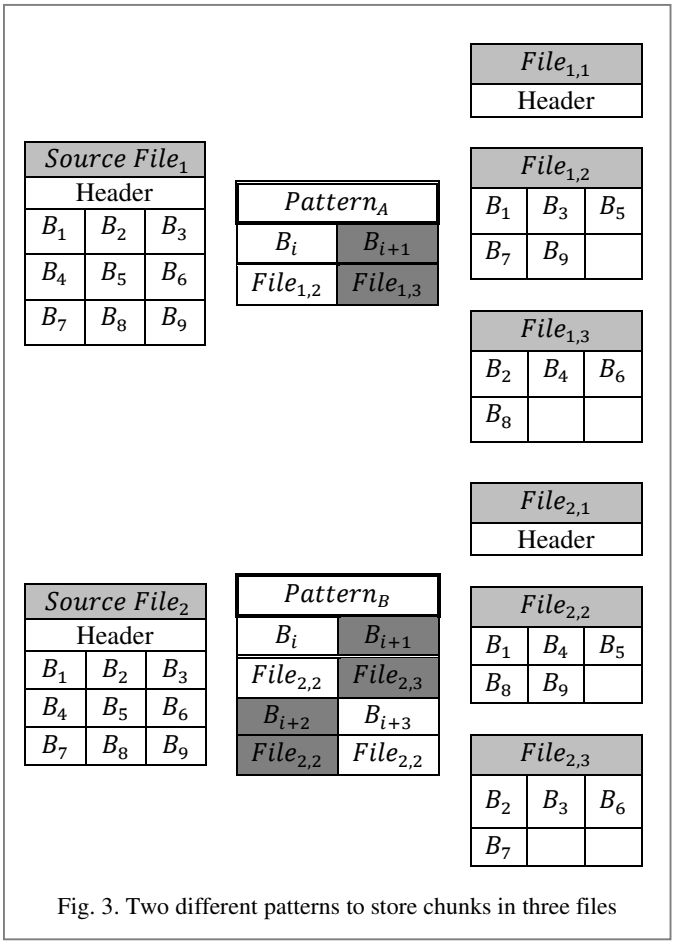


Fig. 3. Two different patterns to store chunks in three files

In the classic problem of the chaos system if selected μ is to be selected between $3.569945 \leq \mu \leq 4$, P can provide a complex chaos model.

Figure 4, shows 300 iterations of a chaos behavior that describes in Equation (2) when $\mu = 3.684$

The proposed method uses the following set to provide a non-convergent, non-periodic pseudo random numbers:

$$\{P_k\}_{k=0}^{\omega} \tag{4.}$$

where $\omega = cmax$ by an initial value of $P_0 = 0.9999$

The proposed method uses the following equation to find the location of $Chunk_k$ in a split file:

$$Pos_k = P_k * cmax \tag{5.}$$

where Pos_k represents the position of $Chunk_k$in each file.

Reading and writing a JPEG file as a binary adds more complexity to retrieve the image by unauthorized users. For example, let's assume that the original file has two consecutive bytes (i.e., 'FFD8' that indicates start of an image). First, we split these two consecutive bytes into two chunks (i.e., 'FF' and 'D8'). Second, distribute chunks in different location in a file (i.e, at 0 position and 2047 position); then a JPEG decoder cannot retrieve this file because the decoder cannot find JPEG makers. In this case, the JPEG decoder or an operating system cannot understand this binary file is a type of JPEG format.

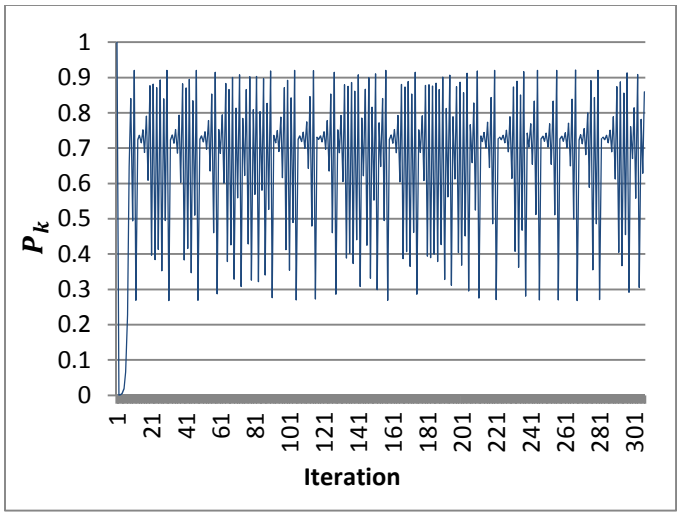


Fig. 4. Chaos behavior of $\{P_k\}_{k=0}^{300}$

The best option for selecting the buffer is: when Buffer $mod\ 2 = 1$ that splits two consecutive bytes into two chunks.

In the case of a collision between Pos_k and Pos_l in Equation 5, we develop a framework to find a new address. We use the following equations (6-11) to relocate a $Chunck$ from Pos_k to Pos_l in a file. In these equations $\{Avail\}$ represents positions that are not used in $\{P_k\}$.

$$Pos_l = Avail_m \tag{6.}$$
$$If(k = \max) \Rightarrow \exists k \in \max[\{Avail_k\}] \ where\ m < k$$

$$Pos_l = Avail_n \tag{7.}$$
$$If(k = \min) \Rightarrow \exists n \in \min[\{Avail_n\}] \ where\ n > k$$

$$Pos_l = Avail_o \tag{8.}$$
$$If(k \neq min) \wedge \exists o \in \min[\{Avail_o\}] \text{ where } o < k)$$
$$\Rightarrow \exists n \in \min[\{Avail_n\}] \ where\ o < k$$

$$Pos_l = Avail_p \tag{9.}$$
$$If(k \neq \min) \wedge \nexists p \in \min[\{Avail_p\}] \ where\ p < k$$
$$\Rightarrow \exists p \in \min[\{Avail_p\}] \ where\ p > k$$

$$Pos_l = Avail_q \tag{10.}$$
$$If(k \neq \max) \wedge \exists q \in \max[\{Avail_q\}] \text{ where } q > k)$$
$$\Rightarrow \exists q \in \max[\{Avail_q\}] \text{ where } q > k$$

$$Pos_l = Avail_r \quad (11.)$$
$$If(k \neq \max) \wedge \nexists p \in \max[\{Avail_r\}]$$
$$where\ r > k$$
$$\Rightarrow \exists r \in \max[\{Avail_r\}]\ where\ r < k$$

The procedure (Equations 6-11) basically extend the upper bound collisions and the lower bound collisions to upper and lower available addresses, respectively. If $Pos_{k-1} < Pos_k$ then, the procedure finds an upper level available position. If $Pos_{k-1} > Pos_k$ then, the procedure find a lower level available position. Some exceptions are listed as follows: (i) k is the maximum address number: the procedure finds a maximum position of l where $k>l$; (ii) k is the minimum address number: the procedure finds a minimum position of l where $k<l$; (iii) there is no any available upper bound position: the procedure finds the maximum number of l where $k<l$; (iv) there is no any available lower bound position: the procedure finds the minimum number of l where $k>l$.

Figure 5, shows the result of equations (6-11) when $\mu = 3.684$ and $P_0 = 0.9999$.

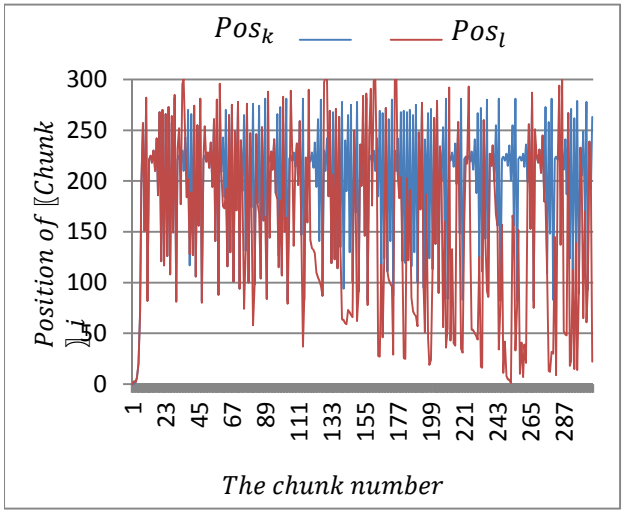


Fig. 5. A comparison between Pos_k and its relocation (Pos_k)

B. Assembly a file

At the first step, the method reads each chunk from each split file based on selected pattern in Section II.A.1, and then the proposed method uses the chaos model in Section II.A.2 to relocate each chunk to the original location of JPEG file. At this phase the proposed method retrieves the address of each chunk by one the following procedure: (i) use an array of PRP numbers which is used in assembling phase; (ii) uses μ and $P_0$to build the array of PRP numbers. The array can be defined by finding each P_k and its relocation (P_l), if there is a collision between P_k and P_l.

IV. Evaluation of the Proposed Method

One of the approaches to evaluate the proposed method is implementation that represents experimental results, such as the response time in a load and performance testing. Another approach is statistical model that describes a deviation between the original and the output. This section presents these two approaches to evaluate the proposed method.

A. Implementation

We use the proposed method to disassemble and store 21 JPEG files with different sizes that were taken from a smart phone. We compare the result of assembly phase against encryption methods and disassembly phase against decryption method.

1) Experimental Setup

In this experiment, we used 21 pictures with different qualities that were taken by a Samsung Galaxy III smart phone. The total size of our dataset was 24.9 Mbyte. We select different size of files to benchmark the proposed method with different input rates. Selected pictures' sizes are varying from 21 Kbyte to 2.86Mbyte with different dimensions (180*320 to 2448*3264 with 72dpi). In this experiment, the proposed method uses a buffer with the size of 4096 Byte. We use Rijndael cryptography with a key of 32 bit (minimum regular key size) and a buffer size of 4096 Byte, Initialization Vector (IV) of 16 Byte. The proposed method is implemented by Visual Studio 2013 in C#.Net programming language, with .Net framework 4.5. We used a well-known JPEG library, LibJPEG.Net package which is developed by BitMiracle, to evaluate the encryption on JPEG encoder in C#.Net programming language.

We assume our application have an access to a pre-defined $\{P_i\}$ to avoid the computation overhead.

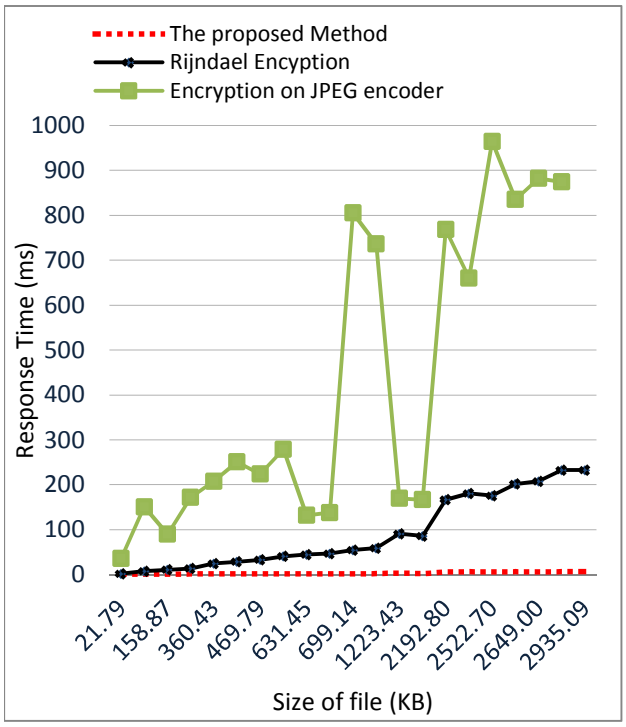
2) The result of the experiment

We implemented the proposed method with *pattern A* (according to the Figure 3) and Rijndael encryption method to compare the response times. As shown in Figure 6, the size of files increases, the proposed method provides a flat response time but the response time for Rijndael encryption increases linearly. As shown in this figure, encryption on JPEG encoder has more computation overhead over two other methods because it is required to read each pixel by a JPEG decoder, encrypt each pixel and use the JPEG encoder to write the result.

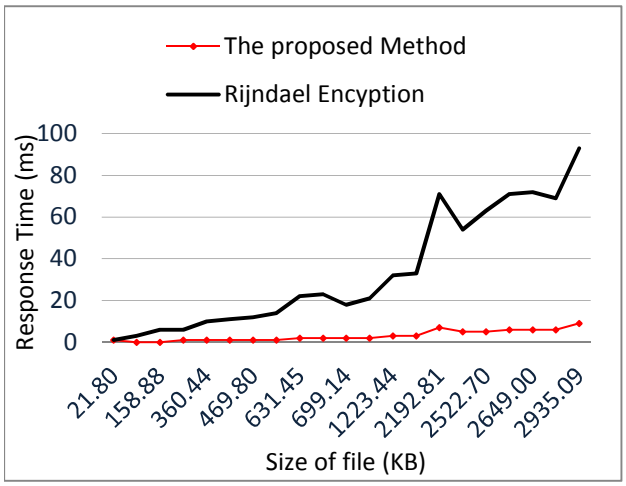
B. Statistical Model

Another approach to evaluate the proposed method is statistical model that presents a deviation of each chunk position between the position in original file and the position in scramble file.

To evaluate the proposed method, we select three different values of μ and we select the same other initial values as follows: $P_0 = .9999$, the maximum size of input file is 3057 Kbyte (the maximum size of our dataset images in the Section IV.2) and the size of each chunk (buffer size) is



A. A comparison of disassembling files with pattern A and encryption methods for 21 JPEG pictures



B. A comparison of assembling files with pattern A and Decryption method for 21 JPEG pictures

Fig. 6. Experimental results

10 Kbyte. Figure 7 shows a deviation of the original chunk position and the same chunk position in scrambled file with different parameters of (μ) values.

In Figure 7, X-axis represents eight selected Pos_k and Pos_{k+1} and the Y-axis represents the deviation of position of Pos_k and Pos_{k+1} in scrambled file. As shown in this diagram, each two consecutive positions in a scramble file has different deviations. The different values of μ provide different models of file scrambling. As shown in this figure, the deviation of each curve are different for different initial values of μ.

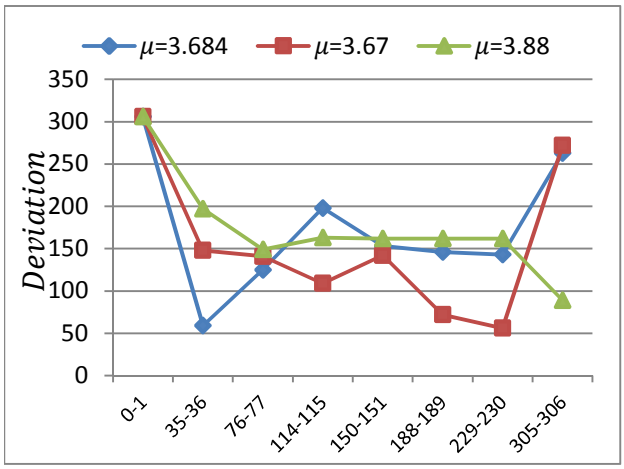


Fig. 7. A deviation of chunks positions in scramble file with different parameter values

We also compare the position of each chunk in the original file and in the scrambled file. We use the configuration of Figure 7 and we assume the original file is scrambled in one file that includes the header and the content of the original file. Figure 8, shows the deviation of a chunk position from the original file to the new position in the scrambled file. As shown in this figure, each chunk is relocated to different locations in scrambled file and the value of each chunk is vary chunk to chunk, randomly.

V. Security Attack Scenarios

In this section, we present different security attack scenarios that an attacker can implement against the proposed method.

An attacker requires assembling all split files and all chunks in each split file to retrieve an image. The proposed method provides a scrambled binary file that provides an obstacle for JPEG encoders to retrieve images because a JPEG encoder requires specific markers which are scrambled through split files.

We assume that the attacker wants to retrieve a JPEG file with a 2 MByte size and the attacker uses an Intel CPU i7 4770K with 127273 MIPS at 3.9 GHZ. The following scenarios can be implemented against the proposed method:

A. *Scenario 1*

Assumptions: the attacker who has access to all split files but dos not know $\{P_i\}$ and the size of each chunk

In this scenario, since the attacker does not have information of $\{P_i\}$, the attacker must run a brute-force attack to assemble split files and reorganize the scrambled chunks in split files. It requires minimum $O((n! - 1) - \partial)$, where ∂ is the number of similar bytes in all files and n is the size of file (byte). In this case, the attacker needs to try $2.229077716E + 9381$ permutation combinations to reconstruct the image ($\partial \approx 0$).

Using this scenario with this CPU configuration to retrieve an image is impossible and it is required

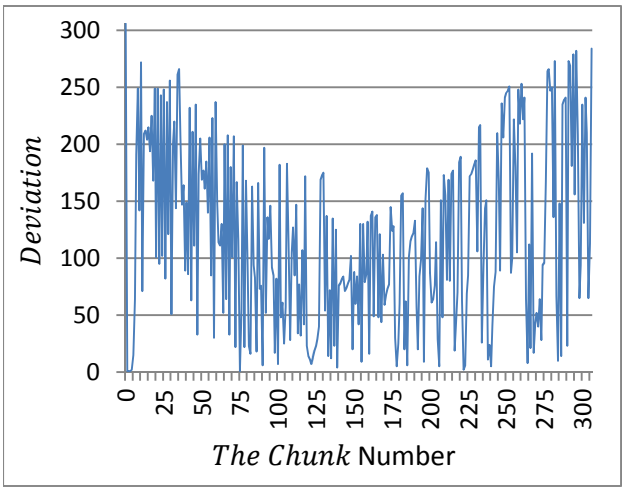


Fig. 8. A statistical deviation of position in original file and scramble files

2.027100061111045012201E+9371 years for processing the computations.

B. Scenario 2

> *Assumptions*: The attacker has access to all split files, knows the size of each chunks (10Kbyte) but does not know $\{P_i\}$.

In this scenario, the attacker needs to run a brute-force attack but the computation on size of the scrambled files (3070 Kbyte) is divided to the size of each chunk (10Kbyte). In this case, the attacker needs to try a minimum of $307! - 1$ permutation combinations to reconstruct scrambled file, needs an impossible computation (6.67732188350771659511 6E+621 years) to compute all permutation combinations.

C. Scenario 3

> *Assumptions*: The attacker has access to all split files, knows the size of chunks (10Kbyte) but does not know the method is based on chaos system.

In this scenario, the attacker needs to use a brute-force attack against the proposed method that requires a computation with $O(\frac{n}{10}! - 1) - \partial)$, where ∂ is the number of similar bytes in all files and n is the size of the original file (3070 Kbyte). In this case, the attacker needs $O(307!)$ computation.

D. Scenario 4

> *Assumptions*: The attacker has access to one file of the multiple split files.

In this scenario, since each two consecutives chunks stores in different clouds (i.e., using Pattern A in Figure 3), the attacker only could retrieve a part of an image by using a brute-force attack. We can estimate the probability of finding the size of each chunks as follows:

$$T = \sum_{j=1}^{j=size} \frac{Size!}{j} \quad (12.)$$

where $size$ represents the size of file (byte).

The worst case of this scenario is described as follows:

$$T = \sum_{j=1}^{j=cmax} \frac{Size!}{j} \quad (13.)$$

The attacker needs to try all $T!$ permutation combination to reconstruct a partial of an image.

VI. Related Works

Several studies [16-22] have been conducted to image encryption. Unlike these studies that address encryption methods based on JPEG encoders, our proposed method provides a light-weight data privacy based on binary file. As described in Section III, using JPEG encoders has computation overhead for mobile devices, such as smart phones. Our proposed method uses the binary file rather than using encrypt method on image pixels or color of a pixel.

Podesser et al. [17] proposed a selective encryption method for mobile devices to cipher a partial of an image. However, in this method, still a partial of image is visible to everyone. Choo et al. [18] proposed a light-weight method for real-time multimedia transmission. Although this method provides an efficient performance over AES encryption, the method still needs heavy computation on a smart phone to cipher an average 3Mbyte JPEG image file in a real-time (see Section V for the comparison).

Unlike existing studies, our proposed method provides three steps to reconstruct a JPEG image file as follows:

(i) *Recognizing the type of the scrambled files*: It is difficult to understand the type of file because the header of file (includes JPEG marker) splits from the content of file and all the chunks are scrambled in each file.

(ii) *Finding and assembling the scrambled files:* Since the split files distributed through two or multiple clouds, it is impossible to reconstruct an image file completely.

(iii) *Reconstructing the original file from split scrambled files*: Reconstructing split scrambled files requires heavy computations to retrieve partial or full image.

The proposed method hides JPEG markers from image decoders that does not allow JPEG encoders to retrieve the metadata of a scramble image. For example, Figure 9 shows: (a) an original image; (b) a scrambled JPEG file based on the proposed method (if we save the file with a JPG extension); (c) a cipher image based on JPEG encoder; (d) a cipher image based on AES. As shown in this figure, the proposed method and AES cannot retrieve the information of an image and the size of an image. However, metadata of an image can be retrieved for a cipher image based on a JPEG encoder. As described in Section IV, our proposed method provides a better performance over existing methods. The proposed method can be implemented for different applications in cloud computing systems such as [23] to collect information by a web crawler

and maintain the privacy of the information in a cloud computing system. The method can be applied to eHealth systems [24] to maintain data privacy.

VII. Conclusion

In this paper, we proposed a new data privacy method to store JPEG files on multi cloud computing systems. Since the method uses less complexity, we have shown, the implemented method provided a cost effective solution for mobile devices that do not have enough energy for resources, such as CPU and RAM. The proposed method splits each file to multiple chunks, distribute each chunk to multiple split files, and scramble chunks based on chaos system. The proposed method provides low computation overheads and it can efficiently run on a smart phone. It restricts unauthorized users including cloud vendors and their partners to reconstruct a JPEG image file. We compared our proposed method against other encryption methods to demonstrate its performance superiority over existing methods. Furthermore, we investigated some important security attack scenarios against the proposed method to evaluate the level of security.

Acknowledgement

The implementation work of the application was supported by Microsoft Windows Azure through Windows Azure Educator Award.

References

[1] Mehdi Bahrami and Mukesh Singhal, “The Role of Cloud Computing Architecture in Big Data”, Information Granularity, Big Data, and Computational Intelligence, Vol. 8, Chapter 13, Pedrycz and S.-M. Chen (eds.), Springer, 2014 *http://goo.gl/E1TmXu*

[2] Mukesh Singhal ,"A Client-centric Approach to Interoperable Clouds", International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, pp. 3-4, 2013, Doi: 10.7321/jscse.v3.n3.2

[3] Mukesh Singhal, Santosh Chandrasekhar, Gail-Joon Ahn, Elisa Bertino, Ram Krishnan, Ravi Sandhu and Ge Tingjian, “Collaboration in Multi-Cloud Systems: Framework and Security Issues”, IEEE Computer, Vol 46, No 2, February 2013, pp. 76-84.

[4] Adibi, Sasan, Nilmini Wickramasinghe, and C. Chan. "CCmH: The Cloud Computing Paradigm for Mobile Health (mHealth)" The International Journal of Soft Computing and Software Engineering [JSCSE] 3.3 (2013): 403-410.

[5] Rodrigues, Joel JPC, et al. "Distributed media-aware flow scheduling in cloud computing environment" Computer Communications 35.15 (2012): 1819-1827.

[6] Huang, Dijiang. "Mobile cloud computing" IEEE COMSOC Multimedia Communications Technical Committee (MMTC) E-Letter 6.10 (2011): 27-31.

[7] Landau, Susan. "Highlights from Making Sense of Snowden, Part II: What's Significant in the NSA Revelations" Security & Privacy, IEEE 12.1 (2014): 62-64.

[8] Kumar, Karthik, and Yung-Hsiang Lu. "Cloud computing for mobile users: Can offloading computation save energy?" Computer 43.4 (2010): 51-56.

[9] Ristenpart, Thomas, et al. "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds" Proceedings of the 16th ACM conference on Computer and communications security. ACM, 2009

[10] Daemen, Joan, and Vincent Rijmen, “The design of Rijndael: AES-the advanced encryption standard”, Springer, 2002.

[11] Osvik, Dag Arne, et al. "Fast software AES encryption" Fast Software Encryption. Springer Berlin Heidelberg, 2010.

[12] Yoshikawa, Masaya, and Hikaru Goto. "Security Verification Simulator for Fault Analysis Attacks", International Journal of Soft Computing and Software Engineering, vol.3, no.3, 71, March 2013.

[13] Digital Compression and Coding of Continuous-Tone Still- Images Requirements and Guidelines, T.81, ITU retrived on 11/11/2014 at http://www.w3.org/Graphics/JPEG/itu-t81.pdf

[14] Chakraborty, Debrup, and Palash Sarkar. "A new mode of encryption providing a tweakable strong pseudo-random permutation" Fast Software Encryption. Springer Berlin Heidelberg, 2006.

[15] Gharajedaghi, Jamshid, “Systems thinking: Managing chaos and complexity: A platform for designing business architecture”, Elsevier, 2011.

[16] Lian, Shiguo, Jinsheng Sun, and Zhiquan Wang. "A novel image encryption scheme based-on JPEG encoding." Information Visualisation, 2004. IV 2004. Proceedings. Eighth International Conference on. IEEE, 2004.

[17] Podesser, Martina, Hans-Peter Schmidt, and Andreas Uhl. "Selective bitplane encryption for secure transmission of image data in mobile environments." Proceedings of the 5th IEEE Nordic Signal Processing Symposium (NORSIG’02). 2002.

[18] Choo, Euijin, et al. "SRMT: A lightweight encryption scheme for secure real-time multimedia transmission." Multimedia and Ubiquitous Engineering, 2007. MUE'07. International Conference on. IEEE, 2007.

[19] Ye, Guodong. "Image scrambling encryption algorithm of pixel bit based on chaos map." Pattern Recognition Letters 31.5 (2010): 347-354.

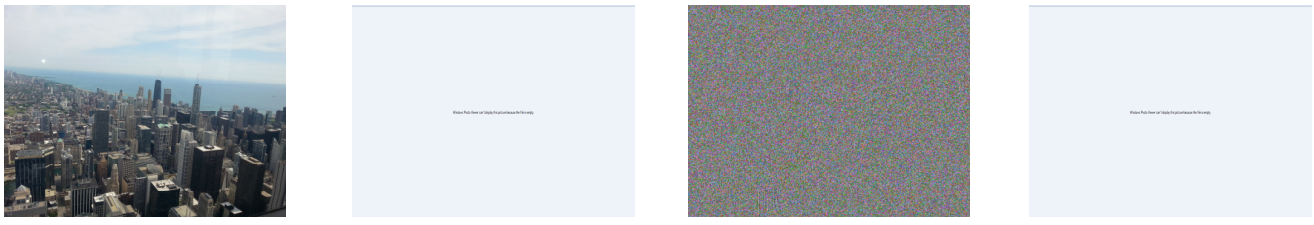
[20] Z. Liu and X. Li. Motion vector encryption in multimedia streaming. In Int. Conf. on Multimedia Modeling. IEEE, 2004.

[21] W. Zeng and S. Lei. Efficient frequency domain selective scrambling of digital video. IEEE Transactions on Multimedia, 5(1), Mar. 2003.

[22] Ra, Moo-Ryong, Ramesh Govindan, and Antonio Ortega. "P3: Toward Privacy-Preserving Photo Sharing" NSDI. 2013.

[23] Mehdi Bahra, Mukesh Singhal and Zixuan Zhuang, "A Cloud-based Web Crawler Architecture" in 2015 18th Int. Conf. Intelligence in Next Generation Networks: Innovations in Services, Networks and Clouds (ICIN 2015), Paris, France, IEEE, 2015.

[24] Rodrigues, Joel JPC, et al. "Analysis of the security and privacy requirements of cloud-based Electronic Health Records Systems" Journal of medical Internet research 15.8 (2013).



(a) (b) (c) (d)

Fig. 9. (a) the original image; (b) a scrambled image based on the proposed method; (c) a cipher image based on JPEG encoder; (d) cipher image based on AES encryption