This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585085, IEEE Transactions on Information Forensics and Security

2DCrypt: Image Scaling and Cropping in Encrypted Domains

Manoranjan Mohanty, Muhammad Rizwan Asghar, and Giovanni Russello

***Abstract*—The evolution of cloud computing and a drastic increase in image size are making the outsourcing of image storage and processing an attractive business model. Although this outsourcing has many advantages, ensuring data confidentiality in the cloud is one of the main concerns. There are state-of-the-art encryption schemes for ensuring confidentiality in the cloud. However, such schemes do not allow cloud datacenters to perform operations over encrypted images. In this paper, we address this concern by proposing *2DCrypt*, a modified Paillier cryptosystem-based image scaling and cropping scheme for multi-user settings that allows cloud datacenters to scale and crop an image in the encrypted domain. To anticipate a high storage overhead resulted from the naive per-pixel encryption, we propose a space-efficient tiling scheme that allows tile-level image scaling and cropping operations. Basically, instead of encrypting each pixel individually, we are able to encrypt a tile of pixels. *2DCrypt* is such that multiple users can view or process the images without sharing any encryption keys – a requirement desirable for practical deployments in real organizations. Our analysis and results show that *2DCrypt* is IND-CPA secure and incurs an acceptable overhead.When scaling a 512×512 image by a factor of two, *2DCrypt* requires an image user to download approximately 5.3 times more data than the un-encrypted scaling and need to work approximately 2.3 seconds more for obtaining the scaled image in plaintext.**

***Index Terms*—Image Outsourcing, Hidden Image Processing, Encrypted Scaling and Cropping, Paillier Cryptosystem.**

I. Introduction

Cloud computing is an attractive paradigm for accessing virtually unlimited storage and computational resources. With its pay-as-you-go model, clients access fast and reliable hardware, paying only for the resources they need to use without the risks of large upfront investments. Nowadays, building applications for multimedia content hosted in infrastructures managed by third-party cloud providers is common.

Images might contain highly sensitive and personal information. If not protected, sensitive information in the images (*e.g.,* MRI scan of a patient or G.I.S. maps) might be subject to unauthorized accesses by cloud providers.

A naive approach to protect confidentiality of outsourced images is to encrypt the images before they are stored in the cloud. However, once this is done, it may not be possible to perform basic image processing operations, such as scaling and cropping. For instance, a remote pathologist, accessing a large histopathology image, would require first to access a scaled-down version, and then perform scaling and cropping operations to get a proper resolution for the Region of Interest (ROI). With images that are encrypted using standard encryption techniques, such operations would require the client machine to download the full encrypted images, decrypt them on the local machine, and then perform the operations. This makes the workflow slow and inefficient because a huge amount of data is pre-fetched and processed.

A. Our Model

In our scenario, cloud providers are honest-but-curious. We assume they do not tamper with the applications deployed in the infrastructure, but data might be collected or leaked. A typical example is replacing old hard drives with new ones, where the data has not been properly wiped out. Also, we assume a full-fledged multi-user access model, where several authorized users access and modify the data stored in the cloud. In order to take full advantage of the cloud model, operations are offloaded as much as possible to cloud servers. However, to preserve confidentiality, operations are performed over encrypted images. In this work, we focus on dynamic scaling and cropping operations on encrypted images. These two operations can be combined to implement zooming and panning operations, which are necessary to navigate through large images (such as maps). In this way, no information contained in the images can be leaked to the cloud servers, and at the same time, users can fully exploit the cloud model by delegating most of the computation to the cloud.

B. State of the art

Ideally, one would like to use the fully homomorphic encryption scheme to perform any type of computations over encrypted data. However, the currently available fully homomorphic encryption scheme [1] is not computationally practical [2]. Thus, partial homomorphic encryption schemes, those supporting certain operations over encrypted data, are typically used for practical solutions.

Based on partial homomorphic Shamir's secret sharing [3], two main research works perform image scaling and cropping operations in the encrypted domain [4], [5]. By extending the seminal work of Thien and Lin [6], these works create multiple shares of the secret image and distribute the noise-like shared images among multiple cloud providers. To recover the original image, k out of n shared images have to be

Manoranjan Mohanty (email: manoranjan.mohanty@nyu.edu) is with Center for Cyber Security, New York University Abu Dhabi, UAE. Most of this work was completed when the author was with the Department of Computer Science, The University of Auckland, New Zealand. Muhammad Rizwan Asghar and Giovanni Russello are with the Department of Computer Science, The University of Auckland, New Zealand. They can be contacted by email: r.asghar@auckland.ac.nz and g.russello@auckland.ac.nz, respectively.

retrieved. The images are shared in such a way that scaling and cropping operations can be performed on encrypted images. However, these approaches suffer from two main drawbacks: (i) for each image, n shares are created and uploaded to the cloud, which increase the amount of storage required as well as the processing power (all the share images are processed and updated when an operation is performed); and (ii) there is no protection against collusion: if k datacentres collude then the original image can be retrieved.

When working on encrypted data, a lot of attention is paid to the actual scheme without considering the important aspect of key management. In a large organization, key management is essential as well as challenging. Employees want to share data, but issuing the same key to all employees is infeasible. Issuing the same key to all employees has also serious consequences, as in the event of an employee leaving the company, a new key must be generated and the data must be re-encrypted using the new key. In an ideal situation, each employee must have their own personal keys that can be used to access data encrypted by other employee's keys. This scenario is often referred to as the Full-Fledged Multi-User (FFMU) model. When the employee leaves the organization, the key must be revoked and the employee must not be able to access any data (even her own data). The data (including data of resigned employees) in the database must be accessible to employees having valid keys.

C. Our contributions

In this paper, we present *2DCrypt*, a practical cloud-based multi-user encrypted domain image scaling and cropping framework based on the modified Paillier cryptosystem. For practical deployment, we propose a novel space-efficient tiling scheme for tile-level encrypted domain scaling and cropping operations. The main contributions of our work can be summarized as follows:

- To the best of our knowledge, we are the first to provide a full-fledged multi-user scheme where users can view and process images without requiring any key sharing. Our key management approach suits the need of organizations having a dynamic workforce where managing shared keys is challenging;
- Unlike the state-of-the-art Shamir's secret sharing-based schemes, the modified Paillier-based cryptosystem scheme neither requires more than one datacenter nor assumes that an adversary cannot access more than certain number of datacenters at any time. Therefore, *2DCrypt* is more suitable for practical scenarios and it provides stronger defence against colluding attacks;
- To overcome high overheads resulted from encrypting an image, we propose a novel space-efficient tiling scheme that allows tile-level scaling and cropping operations. Using this scheme, we can encrypt a tile of pixels rather than encrypting each pixel independently. Furthermore, we optimize the cryptosystem to further limit its storage requirement. As a result, *2DCrypt* requires approximately 40 times less storage than the naive per-pixel encryption;
- *2DCrypt* supports any factor scaling and cropping on encrypted images. These operations can be combined to support zooming and panning operations, which are two key features in image streaming. Compared to similar approaches, *2DCrypt* does not create and store multiple copies of the same image. Moreover, from the cloud server to the user, only the requested processed part of the image is sent.

The rest of this paper is organized as follows. In Section II, we provide an overview of approaches for secret image processing. Section III describes our system model and threat model. Section IV provides an overview of *2DCrypt*. In Section V, we describe our Paillier-based image hiding scheme that allows scaling and cropping operations in outsourced environments. Construction details are given in Section VI, and security analysis is provided in Section VII. Section VIII explains results and performance analysis of our scheme. Some ideas for improving the performance of *2DCrypt* are discussed in Section IX. Finally, Section X concludes our work and provides directions for future work.

II. Related Work

The use of cryptosystems for hiding images is a well-studied area. A number of approaches, including but are not limited to, Public Key Cryptosystem (PKC) [7], watermarking [8], Shamir's secret sharing [6] and chaos-based encryption [9], have been proposed to protect images. These schemes provide confidentiality for cloud-based storage systems where a cloud datacenter does not perform any operation on the stored image.

To allow cloud datacenters to perform operations on the encrypted image, partial homomorphic cryptosystem-based solutions have been proposed [10]–[12]. A partial homomorphic cryptosystem exclusively offers either addition or multiplication operations. Paillier [13], Goldwasser-Micali [14], Benaloh [15], Shamir's secret sharing [3] are among partially homomorphic cryptosystems that support addition. Whereas, examples of partially homomorphic cryptosystems that offer multiplication are RSA [16] and ElGamal [17]. Thus, the choice of a partial homomorphic scheme is heavily dependent on the type of operations to be performed in the encrypted domain.

Early works have focused on retrieving encrypted text documents. For instance, [18] presented the first practical scheme for single keyword search on encrypted documents. To improve performance, [19] extended the encrypted search with indexing capability. Both works have been extended for searching using conjunctions of multiple keywords [20]. More recent works have focused on SQL-like queries supporting conjunctions and disjunctions [21].

Encrypted text-based search can also be applied to retrieval of encrypted images. However, the precision of the returned set is dependent on the quality of the keywords used for describing the content of an image. Few works have been proposed for searching encrypted images based on dynamic extraction of image features. In [10], Lu *et al.* proposed practical search based on feature/index randomization techniques that offer a good trade-off between privacy preservation and performance. [11] proposed a homomorphic-based SIFT (Scale-Invariant Feature Transform) extraction search that increases the accuracy of the search but also incurs from 2 to 4 orders of

magnitude more costs. A more recent work by [12] introduces a search scheme for encrypted images that is accurate and at the same time incurs computational overheads similar to a plaintext method. However, their scheme requires users to share the keys for accessing images.

Several works have been proposed for privacy-preserving face recognition [22]–[24] where one party tries to match a face image with a dataset hosted by another party and both parties are interested in keeping their data secret from each other.

Shamir's secret sharing has been used for allowing encrypted domain scaling and cropping [4], [5]. As discussed in Section I, Shamir's secret sharing-based schemes, however, can be infeasible for practical scenarios since they require n cloud servers. Moreover, these schemes are prone to collusion attack when k cloud servers collude. In contrast, *2DCrypt* uses the Paillier-based cryptosystem that requires only one cloud datacenter and is more robust to collusion attacks. The Paillier cryptosystem is homomorphic to additions and scalar multiplications [13] and can be modified to a proxy encryption scheme [25], [26].

III. System Model

In this work, we consider a distributed cloud-based image storage and processing system where a cloud server stores, scales, and crops an encrypted image on behalf of an image outsourcer. In the system model, we assume the following entities.

- **Image Outsourcer:** This entity outsources the storing and processing (*i.e.,* scaling and cropping) of images to a third-party cloud provider. It could be an individual or an organization, such as a hospital. In the latter case, several users can act as an Image Outsources. Typically, this entity owns the image. An Image Outsourcer is responsible for addressing security and privacy concerns attached to image outsourcing. To achieve this, the Image Outsourcer encrypts the image before sending it to the cloud datacenter. Further, the Image Outsourcer can store new images on a cloud server, delete/modify existing ones, and manage access control policies (such as read/write access rights) to regulate access to the images stored on the cloud server.
- **Cloud Server:** It is the part of infrastructure provided by a cloud service provider, such as Amazon S3[1], for storing and processing images. It stores encrypted images and access policies used to regulate access to the images. After making authorization checks, it retrieves a requested image from its image store. If the access request satisfies access policies, it scales and/or crops images in an encrypted manner, *i.e.,* without decrypting them.
- **Image User:** it is authorized by the Image Outsourcer to access the requested image stored in an encrypted form on the Cloud Server. Depending on authorization, an Image User can issue either *read request* or *process request* (*i.e.,* scaling and cropping operations). In both cases, the Image User decrypts the image returned by the request. Note that in a multi-user setting, (i) an Image User can modify an image that will be accessible by other Image Users, or (ii) an Image User can access images processed by other Image Users. In both cases, Image Users do not need to share any keying material.
- **Key Management Authority (KMA):** It generates and revokes keys. It generates a client and server key pair for each user, be it an Image Outsourcer or Image User. The client and the server side keys are securely transmitted to the user and the Cloud Server, respectively. Whenever required (say in key lost or stolen cases), the KMA revokes the keys from the system with the support of the Cloud Server.

[1]https://aws.amazon.com/s3/

Threat Model: We assume that the KMA is a fully trusted entity. Typically, the Image Outsourcer directly controls by the KMA. Since the KMA deals with the small amount of data, it can be easily managed and secured. We also assume that the KMA securely communicates the key sets to the Cloud Server and the Image User. This is achieved by establishing a secure channel. Except for managing keys, the KMA does not need to be involved in any operations. Therefore, it can be kept offline most of the times.

We consider an *honest-but-curious* Cloud Server. That is, the Cloud Server is trusted to honestly perform the operations on an image as requested by the Image User. However, it is not trusted to guarantee data confidentiality. The adversary can be either an outsider or even an insider, such as unfaithful employees serving the cloud service provider. Furthermore, we assume that there are mechanisms to deal with the image integrity and availability of the Cloud Server.

IV. Proposed Approach

In this section, we discuss the architecture and workflow of *2DCrypt*, a cloud-based multi-user image scaling and cropping system. *2DCrypt* is based on Paillier cryptosystem.

Figure 1 shows the architecture of *2DCrypt*. In *2DCrypt*, for each user (*i.e.,* be it an Image Outsourcer or Image User), the KMA generates two keys pairs by randomly splitting the master secret key into two parts: the user-side key sent to the user and the server-side key deployed to the server.

The Image Outsourcer stores an image and its access policies in the cloud server. For this job, the Image Outsourcer invokes its client module *Store Requester* by providing plaintext image and access policies as inputs (Step i). The Store Requester performs the first round of encryption on the input image, using the user-side key, and then sends the encrypted image along with its access policies to the *Store Keeper* module of the Cloud Server (Step ii). Note that when encrypting the image, the Store Keeper divides the image into multiple tiles and performs per-tile encryption. A detailed discussion about the tile-level encryption will be presented in Section V. The encrypted image, which is received by the Cloud Server, is not in the common format necessary for sharing in multi-user settings. At the Cloud Server-end, the Store Keeper performs the second round of encryption using the server-side key corresponding to the user, and stores

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585085, IEEE Transactions on Information Forensics and Security

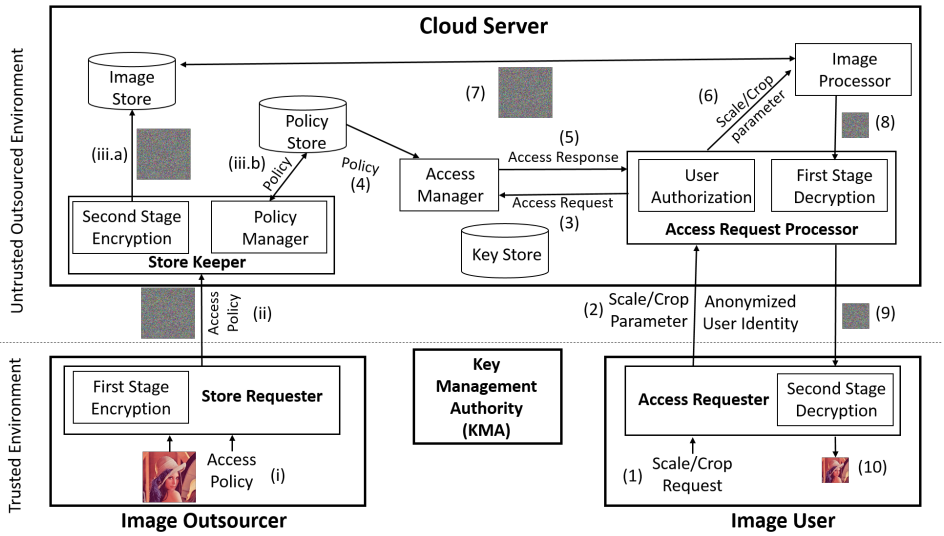


Fig. 1: The architecture of *2DCrypt*: a cloud-based secure image scaling and cropping system.

the encrypted image in an image store (Step iii.a). The Store Keeper also stores the access policies of the image in the *Policy Store* (Step iii.b).

Once an Image User expects the Cloud Server to process any image, its client module *Access Requester* receives its input (Step 1). The module *Access Requester* estimates the scaling and cropping parameters and forwards the request to the *Access Request Processor* module of the Cloud Server (Step 2). In the request, the Access Requester sends image scaling/cropping parameters (such as scaling factor and/or cropping ROI) and user credential (which can be anonymized) to the Access Request Processor. The Access Request Processor first performs a user authorization phase by forwarding an access request to the *Access Manager* (Step 3). The Access Manager fetches the access policies for the requesting user from the Policy Store (Step 4) and it matches the access policies against the access request. Finally, the access response is sent back to the Access Request Processor (Step 5). If the user is authorized to perform the requested operation, the *Image Processor* is invoked with scaling/cropping parameters as inputs (Step 6). The requested image is retrieved from the Image Store (Step 7) and the Image Processor performs scaling/cropping on the encrypted image. When the scaling/cropping operations are completed, the processed image is sent to the Access Request Processor (Step 8). The Access Request Processor performs the first round of decryption on the processed image using the key corresponding to the Image User and sends the image to the Access Requester module (Step 9). The Access Requester module on the Image User performs a second round of decryption and shows the processed image to the Image User (Step 10).

Note that the image after the first round of decryption on the Cloud Server is still encrypted and the Cloud Server cannot learn the secret information contained in the image. To access the image in clear-text, a second round of decryption is required using the user-side key of the Image User (or Image Outsourcer) for the final decryption round.

V. Solution Details

The main idea behind *2DCrypt* is to employ the Paillier cryptosystem-based proxy encryption to encrypt images before storing them in the cloud. This version of Pailler cryptosystem supports re-encryption [25]–[27], and is homomorphic to additions and scalar multiplications. Therefore, we can apply this cryptosystem to encrypt an image that will be bilinearly scaled, since bilinear scaling requires addition and scalar multiplication operations only. Cropping of the encrypted image is easy since this cryptosystem does not disturb the pixel position, *i.e.,* allowing us to obtain the corresponding pixel position after the decryption. In order to provide the multi-user support, we extend the modified Paillier cryptosystem [25]–[27] such that each user has her own key to encrypt or decrypt the images. Thus, adding a new user or removing an existing one will not require re-encryption of existing images stored in the cloud.

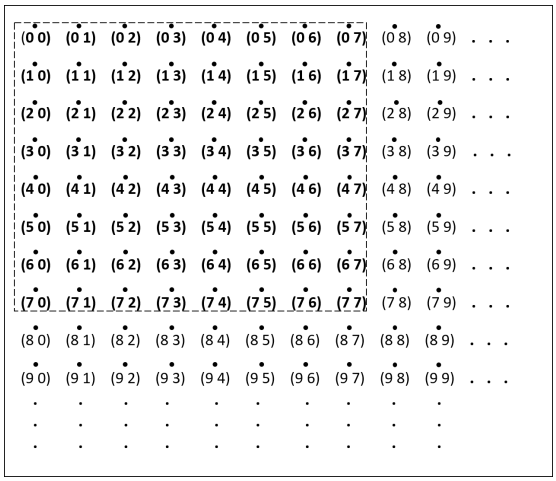
One could use a naive approach to encrypt each pixel independently.Using this pixel-by-pixel encryption, the 24 bits color (Red, Green and Blue - RGB in short- each requiring one byte) will be represented by $4k$ bits, where k is the key size of Pallier cryptosystem. Such an approach, however, is not feasible since the value of k is way larger than 24. For example, the current recommended value of k is 1024, implying that the 24 bit RGB values will be represented using 4096 bits after encryption. Therefore, when a pixel is encrypted, more than 170 times more storage is required to store the ciphertext.

To take full advantage of the input space allowed by the proposed cryptosystem, we introduce a concept of tiling to group a set of pixels. A tile can be encrypted instead of encrypting each pixel. Using the tiling in *2DCrypt*, we save the space and decrease the number of required encryptions and decryptions by a factor of the tile size.

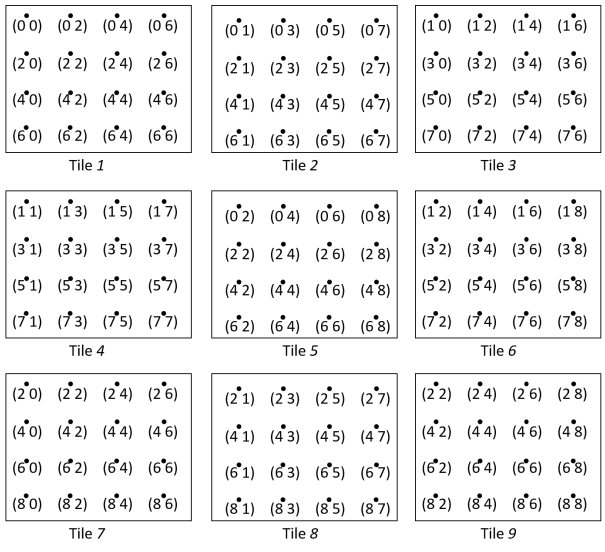
In the following sections, Section V-A discusses about our tiling scheme and tile-level scaling operation, Section V-B discusses about tile-level cropping operation, and Section V-C

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585085, IEEE Transactions on Information Forensics and Security

provides an overview of tile-level encryption process.



(a) The first 8×8 super-tile shown in the dashed rectangle.



(b) Tiles generated from the first 8×8 super-tile.

Fig. 2: Space-efficient tiling for tile-level bilinear scaling. We have created different tiles for pixels at coordinates: (i, j), $(i, j+1)$, $(i+1, j)$, and $(i+1, j+1)$; and have put pixels at coordinates: (i, j), $(i, j+2)$, $(i+2, j)$ in one tile.

A. Space-Efficient Tiling for Bilinear Image Scaling

We employ bilinear scaling to scale an image. Since the choice of scaling technique influences tiling, we first provide an overview of bilinear scaling below. Then, we discuss our tiling scheme.

Bilinear Scaling. Bilinear scaling scales an image by iteratively selecting four neighboring pixels and interpolating these pixels to compute pixel of the resulting image. For example, to scale a $H \times W$ image to a $h \times w$ image, in the input image, pixels at positions: (i', j'), $(i', j'+1)$, $(i'+1, j')$, $(i'+1, j'+1)$ (where $i' = \frac{H}{h} \times i$ and $j' = \frac{W}{w} \times j$) are interpolated to compute $(i, j)^{th}$ pixel of the output image. The interpolation can be represented as:

$$C = \sum_{m=0}^{1} \sum_{n=0}^{1} c_{i'+m, j'+n} C_{i'+m, j'+n}, \tag{1}$$

where C is color of $(i, j)^{th}$ pixel, $C_{i'j'}$ is color of $(i', j')^{th}$ pixel, and $0 \leq c_{i'j'} \leq 1$ is the interpolation factor (a constant).

Bilinear scaling involves floating point operations that are incompatible with the modular prime operations performed by most cryptosystems, including the Paillier cryptosystem. One way of addressing this issue is to modify floating point constant c (from brevity, we have dropped indices from $c_{i'j'}$) to a fixed point number c' by first rounding off c by d decimal places, and then multiplying 10^d to the round-off value [4]. In other words, we can replace c by

$$c' = (c + \epsilon_d) \times 10^d.$$

By doing this, we, however, introduce a round-off error that can result in information loss.

Lemma 1. *In integer version of bilinear scaling, information loss E_d in the scaled image due to rounding off an interpolation factor c by d decimal places is bounded by $\pm 51 \times 10^{1-d}$.*

Proof. From Equation 1, it is evident that information loss is due to the rounding off error only. Multiplication and division of 10^d to the round-off value have no impact on information loss as both of them are lossless operations.

By dropping indices from variables, we can simplify Equation 1 as:

$$\begin{aligned} C &= \sum_{m=0}^{3} C_m (c_m + \epsilon_{m,d}) \\ &= \sum_{m=0}^{3} C_m c_m + \sum_{m=0}^{3} C_m \epsilon_{m,d} \\ &= \sum_{m=0}^{3} C_m c_m + E_d, \end{aligned}$$

where C_m is the m^{th} color, c_m is m^{th} interpolating factor, and $\epsilon_{m,d}$ is the rounding error due to rounding off c_m by d decimal places.

We know that $\epsilon_{m,d}$ is bounded by $\pm 0.5 \times 10^{-d}$, and $0 \leq C_m \leq 255$ is a positive number. Thus, we get the lowest value of:

$$E_d = \sum_{m=0}^{3} C_m \epsilon_{m,d} \tag{2}$$

when each C_m is equal to 255 and each $\epsilon_{m,d}$ is equal to -5×10^{-d}. Similarly, the highest value of E_d is obtained when each C_m is equal to 255 and each $\epsilon_{m,d}$ is equal to 5×10^{-d}. Putting these values in Equation 2, we get $-51 \times 10^{1-d} \leq E_d \leq 51 \times 10^{1-d}$. □

Tiling. Tiling must be done in a way that, by interpolating tiles, neighboring pixels must be interpolated. Therefore, we distribute the pixels into tiles in such a way that four neighboring pixels are always put in four different tiles. Since a pixel is never operated with neighbor's neighbor, we put neighbor's

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585085, IEEE Transactions on Information Forensics and Security

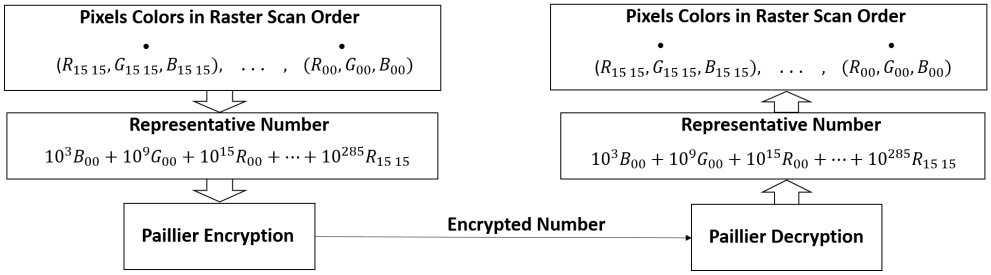
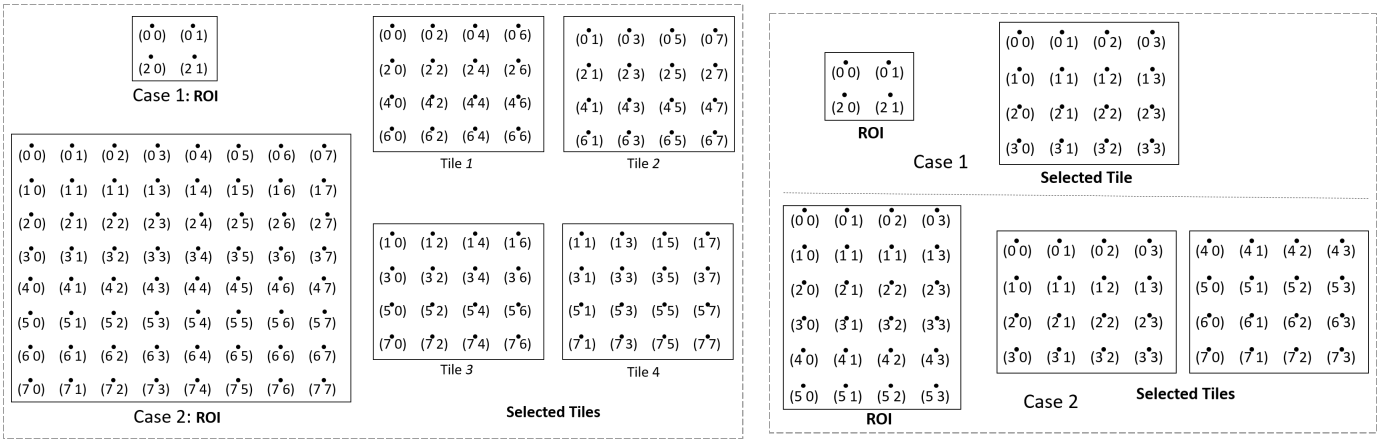


Fig. 3: Representation of a tile with a number.



(a) Cropping a non-scaled image: Same four tiles selected when ROI contains four neighboring pixels (Case 1) or a super-tile (Case 2).

(b) Cropping a scaled image: Number of tiles selected dependent on the ROI.

Fig. 4: Image cropping using *2DCrypt*.

neighbor in one tile. Figure 2 presents one such tiling example where we have assumed 4×4 size tiles. Our scheme, however, can work for any size tile. By requiring neighbor's neighbor, we are considering a $2t_h \times 2t_w$ size super-tile for $t_h \times t_w$ size tiles. A super-tile contains four neighboring tiles. In our tiling example, the 8×8 super-tile is marked with a dashed rectangle. Different super-tiles use the same tiling scheme. In the following, we focus on the tiling scheme of one super-tile.

A super-tile contains 9 overlapping tiles to cover all possibilities of bilinear scaling. For instance, Figure 2a shows the first super-tile in an image and the corresponding 9 tiles are shown in Figure 2b. The selection of tiles for scaling is dependent on the scaling factor. For example, we can perform scale-by-two scaling along super-tiles' height and width by interpolating *Tile 1, Tile 2, Tile 3*, and *Tile 4*. These four tiles are also sufficient to recover the unscaled super-tile. However, if we require scaling with any other scale factor, we may need other tiles. For example, for scaling by a factor of $\frac{2}{3}$ along width and scale-by-two along the height, we require *Tile 5* and *Tile 6* as additional tiles. Similarly, *Tile 7* and *Tile 8* may be required to scale by a factor other than two along the height and by a factor two along the width. When scaling by a factor other than two is required both along width and height, *Tile 9* may be required. When the scaling factor is less than the inverse of size of a super-tile, some of the tiles of a super-tile may be excluded, while the same tile of another super-tile may be needed. For example, for scale factor of $\frac{1}{8}$ along image width, we do not need *Tile 5* and *Tile 6* of the first super-tile, but *Tile 5* and *Tile 6* of the second super-tile along image width are required.

Tile Size: In *2DCrypt*, the tile size is determined from the key size of the cryptosystem and the rounding places in integer bilinear scaling.

The modified Paillier cryptosystem of k-bit key size can take an input having a maximum of k-bit size. Therefore, pixels of a tile can be represented as a k-bit number, which will be input to the modified Paillier cryptosystem. However, We must get back the pixel from this representative number. One way of ensuring this is to unroll the tile into a vector of pixels (*e.g.,* in the raster scan order), multiply 10^m (where m is an integer) to the color C (*i.e.,* RGB values) of a pixel, and add all $10^m C$s – one such scheme shown in Figure 3. At recovery time, we can get back Cs from $\sum 10^m C$ by using *divide-by-10* and *remainder* tricks. For example, we can get back C_1 and C_2 from $10^{m_1} C_1 + 10^{m_2} C_2 + ...$ by first dividing $10^{m_1} C_1 + 10^{m_2} C_2 + ...$ by 10^{m_1}, and then dividing the remainder by 10^{m_2}. For i^{th} pixel, the value of m is dependent on i and the rounding factor d (as C will be multiplied with 10^d during integer version of bilinear scaling). For the gray image, the value of m is $(i-1) \times (d+4) + (d+1)$, as three digits are required to represent $0 \leq C \leq 255$ and one place is reserved to accommodate carry from addition of 4

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585085, IEEE Transactions on Information Forensics and Security

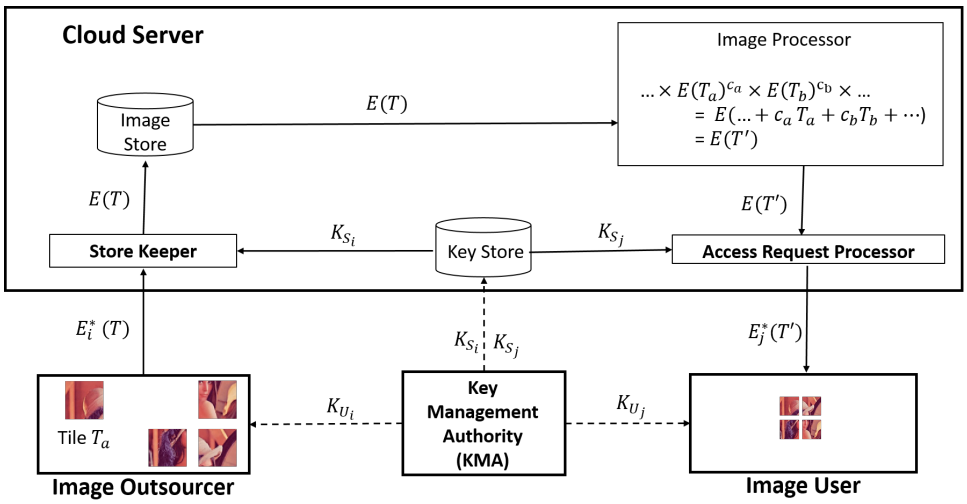


Fig. 5: Overview of encryption and decryption processes in *2DCrypt*.

neighboring Cs. For a color image, the value of m can be determined by considering the color image as a gray image of triple resolution, and by considering color components (*i.e.,* R, G and B) of a pixel independent to each other. The value of m therefore determines the tile size as for a tile having $3n$ number of pixels of a gray image or n number of pixels of a color image, the value $255 \times 10^{(3n-1)\times(d+4)+(d+1)}$ must be less than $2^k - 1$. Our tiling example of a color image considers $k = 1024$ bits and $d = 2$. Thus, we choose 4×4 size tiles as $255 \times 10^{47\times6+3}$ (maximum value of C is 255) evaluates to a 288 digit number, while 2^{1024} is a 309-digit number.

B. Tile-Level Image Cropping

Since the pixels' positions of a secret image are not hidden in the encrypted image, cropping in encrypted domains is easy. We can perform cropping by selecting the tiles containing pixels of an ROI. That is, when an Image User requests an ROI, the Cloud Server sends those tiles containing the pixels of the ROI. The Image User then fetches the required pixels from the received tiles and discards unnecessary pixels.

The procedure of tile selection, however, is dependent on whether scaling has been performed before or not as illustrated in Figure 4. The selection of tiles for a non-scaled original image is different than the selection of tiles for a scaled image. In a non-scaled image, any four neighboring pixels are present in four different tiles (*e.g., Tile 1, Tile 2, Tile 3*, and *Tile 4*). Therefore, we have to select four tiles when four neighboring pixels are part of ROI (Figure 4a). Since neighbor's neighbor is in one tile, these four tiles are sufficient to cover all neighbor's neighbor pixels in a super-tile: *Tile 1, Tile 2, Tile 3*, and *Tile 4* tiles are also sufficient when the super-tile is the ROI. In the case of a scaled image, the non-border neighboring pixels are part of one tile. Therefore, we can select one tile when four non-border pixels are part of an ROI (Figure 4b).

Note that cropping does not introduce round-off error since it does not round-off any floating point numbers (as no floating point operation is required).

C. Tile-Level Encryption

Figure 5 summarizes the process of tile-level encryption. Mathematical details of encryption and decryption (including proof of correctness) will be presented in Section VI.

For each user i (either acting as an Image Outsourcer or Image User), key pairs K_{U_i} and K_{S_i} generated by the KMA are securely transmitted to the user and the *Key Store* of the Cloud Server, respectively.

During the uploading phase, the Image Outsourcer splits the image into tiles. Next, the Image Outsourcer encrypts each tile T using the user-side key K_{U_i} of the user i. The Image Outsourcer yields the ciphertext $E_i^*(T)$, which is sent to the Store Keeper. The Store Keeper retrieves K_{S_i}, corresponding to the user i, and re-encrypts the ciphertext $E_i^*(T)$. The re-encrypted ciphertext $E(T)$ is finally stored in the Image Store. This is a one-time-only operation performed when the image is stored for the first time on the Cloud Server.

Upon request from the user, the Image Processor retrieves encrypted tiles from the Image Store, and performs bilinear scaling and/or cropping operation without decrypting them. These operations are shown in Equations 3 and 4, respectively.

$$E(T)^{c'} = E(c' * T) \tag{3}$$

and

$$E(T_1) * E(T_2) = E(T_1 + T_2) \tag{4}$$

where c' is the integer version of interpolating factor, and T, T_1 and T_2 denote the tiles.

The Image Processor returns each processed tile $E(T')$ to the Access Request Processor. The Access Request Processor pre-decrypts the processed tile $E(T')$, using key K_{S_j} corresponding to the Image User j. The pre-decrypted tile $E_j^*(T')$ can only be accessed by the Image User j. The Image User client receives each pre-decrypted tile $E_j^*(T')$ and finally decrypts the processed tile T', using the user-side key K_{U_j}.

Tile-Level Encryption vs Pixel-Level Encryption:. Although proposed tile-level encryption scheme *2DCrypt* can

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585085, IEEE Transactions on Information Forensics and Security

have less computational and storage overheads than the naive per-pixel encryption, the flexibility of selecting an individual pixel is lost.

Since *2DCrypt* encrypts a tile of pixels, it requires a fewer number of encryptions (and hence encryption cost) than the naive scheme. In *2DCrypt*, the number of decryptions, although dependent on the scaling and cropping factors, never exceeds the number of decryptions of the naive scheme. The worst case happens when one pixel in a tile is required by the scaling/cropping requester. On average, *2DCrypt* requires a fewer number of decryptions as typically multiple pixels are put together in a tile. As an obvious consequence, computational (as fewer encryptions and decryptions are required) and storage overheads (as fewer encrypted values are stored and communicated) of *2DCrypt* is lower than the conventional encryption. For instance, in our tiling example of 4×4 size tiles, the encryption of RGB values of 64 pixels of a super-tile results in maximum 36864 bits, requiring maximum 24 bytes of ciphertext to get one byte of plaintext color. In this way, *2DCrypt* has more than 21 times improvement than the naive approach that independently encrypts each color component of a pixel.

In *2DCrypt*, the flexibility of selecting an individual pixel is lost as a tile of pixels is considered when some of the pixels in the tile are indeed required. The unwanted pixels must be excluded by the Image User before displaying the image. We ignore the cost associated with excluding the unwanted pixels as it can be implemented using simple table lookup operation.

2DCrypt operates in integer domain by converting floating point numbers to integers. The float-to-integer conversion involves rounding off errors that can lead to degrade in quality of processed images (due to loss of per-pixel color information). Note that this quality degradation can also happen in per-pixel encryption. However, such degradation does not happen in the conventional un-encrypted domain scaling/cropping as the conventional scheme does not involve the float-to-integer conversion.

Theorem 1. *The loss of color information in 2DCrypt is bounded by $\pm 51 \times 10^{1-d}$, where each float is rounded off by d decimal places for converting it to an integer.*

Proof. From past discussion, it is evident that in *2DCrypt* information loss can happen only during the encrypted domain tile-level bilinear scaling as it is the only place where we perform the float-to-integer conversion. No information loss happens during tile-level cropping. Similarly, no information loss happens during encryption and decryption as these operations operate in integer domain.

As shown in Equations 3 and 4, the encrypted domain tile-level bilinear scaling uses the integer c' in place of float c, where c' is obtained by rounding off c by d decimal places (and multiplying 10^d to the round-off value). The round-off error ϵ_d has no impact on the decryption (and encryption) as considered Paillier encryption scheme is homomorphic to additions and scalar multiplications. Thus, the round-off error in encrypted domain tile-level scaling is the same as the rounding off error in un-encrypted domain tile-level scaling.

In un-encrypted domain tile-level scaling, the scaling operation is independent of the tiling operation. Tiling has no impact on rounding off the interpolating factors (c) and interpolating the colors of the pixels. Thus, information loss due to tile-level scaling is the same as information loss due to per-pixel scaling. As indicated in Lemma 1, the loss of color information in a pixel due to per-pixel scaling is bounded by $\pm 51 \times 10^{1-d}$. Thus, information loss in *2DCrypt* is bounded by $\pm 51 \times 10^{1-d}$. □

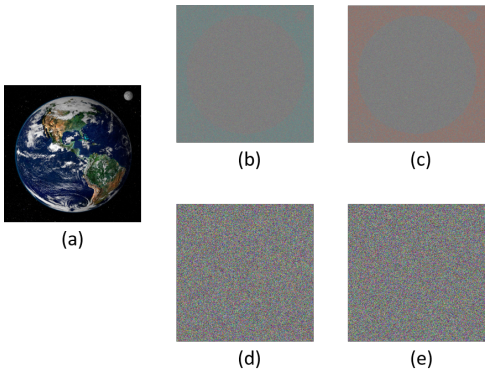


Fig. 6: Encryption of (a) the earth image with and without random number in a tile. Figures (b) and (c) show the images after the Image Outsourcer and Cloud Server encryptions, without using a random number (best viewed on a computer screen or printed in color). Figures (d) and (e) show the images after Image Outsourcer and Cloud Server encryptions using a random number in a tile.

D. Optimization of the Modified Paillier Cryptosystem

Although the tile-level encryption decreases computational overheads, it is still resource-intensive. To address this issue, in this section, we describe an optimization of the modified Paillier scheme.

The modified Paillier is represented as: $E(T) = (e_1, e_2)$, where $e_1 = g^r$ and $e_2 = g^{rx}.(1 + Tn)$ (explained in Section VI). It is important to note that e_1 is independent of plaintext. This component introduces randomness. By using a different e_1 for a different tile (typical case), we need $2k$ bits (where k is a security parameter) for storing e_1 of each tile, even if this cipher component alone cannot reveal any information about the plaintext. We propose to optimize this space requirement by using one e_1 for t number of tiles, requiring $2k$ bits for storing e_1 for all t tiles. This optimization can be achieved by using the same random number r for all t tiles. In this way, we almost half the storage requirement for storing ciphertext of t tiles. For example, if we use one r for all 9 tiles in a super-tile of our tiling example, then we need 107 bits of ciphertext for getting 8 bits of the plaintext color.

Our optimization, however, can leak color information when a tile is equivalent to another tile. When a tile is equivalent to another tile and they use the same random number r, then they have the same ciphertext. Therefore, their equivalence can be learnt from their ciphertexts by subtraction, that results to zero (see Section VI). This information loss is significant for an

image with equivalent tiles. For example, if the image uses the same color for all its background pixels, then the background information (although not background color) can be leaked. This information loss can lead to disclosing the shape of the image in the encrypted domain as shown in Figure 6.

To limit this information loss, we use a random number in each tile. Recall that a tile represents a set of pixels (as illustrated in Figure 3). The number of pixels that can fit into a tile is dependent on the security parameter. Generally, the size of each tile is greater than the number of pixels it can encompass. We explain the remaining bits in each tile to embed the random number to make it difficult for the Cloud Server to distinguish two same tiles. As part of the tile, the random number also undergoes the first round of encryption, making the Cloud Server unable to know its plaintext value.

The Image User, however, can know the random number after performing the second round of decryption. This random number then can be discarded by the Image User when recovering the image. The size of the random number is $k-NB(\sum 10^m C)$ (as it must fit into the tile after putting pixel colors in the tile), where $NB(x)$ denotes the number of bits required to represent x, and k is the security parameter. The location of the random number is implementation dependent. In our tiling example, we use 8-bit random numbers. A trade-off between optimization and security has been discussed in Section IX.

VI. Construction Details

Our proposed scheme is built on top of the schemes described in [25]–[27]. However, different from [25]–[27], our scheme is multi-user: each user has her own key to encrypt or decrypt the data. The proposed scheme consists of the following algorithms.

- **Init(1^k).** The KMA runs the initialization algorithm in order to generate public parameters $Params$ and a master secret key set MSK. It takes as input a security parameter k and generates two prime numbers p and q of bit-length k. It computes $n = pq$. The secret key is $x \in [1, n^2/2]$. The g is of order:

$$\begin{aligned}\frac{\phi(n)}{2} &= \frac{\phi(p)\phi(q)}{2} \\ &= \frac{(p-1)(q-1)}{2}\end{aligned}$$

  and can be easily found by choosing a random $a \in \mathbb{Z}^*_{n^2}$ and computing $g = -a^{2n}$. It returns:

$$\begin{aligned}Params &= (n, g) \text{ public parameters and} \\ MSK &= x \text{ the master secret key set.}\end{aligned}$$

  K_S represents the Key Store, initialised as $K_S \leftarrow \phi$.
- **KeyGen(MSK, i).** The KMA runs the key generation algorithm to generate keying material for users in the system. For each user i, this algorithm generates two key sets K_{U_i} and K_{S_i} by choosing a random x_{i1} from $[1, n^2/2]$. Then it calculates $x_{i2} = x - x_{i1}$, and transmits

$$\begin{aligned}K_{U_i} &= x_{i1} \text{ securely to user } i \text{ and} \\ K_{S_i} &= (i, x_{i2}) \text{ to the server.}\end{aligned}$$

  The server adds K_{S_i} to the Key Store as follows: $K_S \leftarrow K_S \cup K_{S_i}$.
- **ClientEnc(D, K_{U_i}).** A user i runs the data encryption algorithm to encrypt the data D using her key K_{U_i}. To encrypt the data $D \in \mathbb{Z}_n$, the user client chooses a random $r \in [1, n/4]$. It computes $E^*_i(D) = (\hat{e}_1, \hat{e}_2)$, where

$$\begin{aligned}\hat{e}_1 &= g^r \bmod n^2 \text{ and} \\ \hat{e}_2 &= \hat{e}_1^{x_{i1}}.(1+Dn) \bmod n^2 \\ &= g^{rx_{i1}}.(1+Dn) \bmod n^2.\end{aligned}$$

- **ServerReEnc($E^*_i(D)$, K_{S_i}).** The server re-encrypts the user encrypted data $E^*_i(D) = (\hat{e}_1, \hat{e}_2)$. It retrieves the key K_{S_i} corresponding to the user i and computes the re-encrypted ciphertext $E(D) = (e_1, e_2)$, where

$$\begin{aligned}e_1 &= \hat{e}_1 \bmod n^2 \\ &= g^r \bmod n^2 \text{ and} \\ e_2 &= \hat{e}_1^{x_{i2}}.\hat{e}_2 \bmod n^2 \\ &= g^{rx}.(1+Dn). \bmod n^2\end{aligned}$$

- **ServerSum($E(D_1)$, $E(D_2)$).** Given two encrypted values $E(D_1) = (e_{11}, e_{12})$ (where $e_{11} = g^{r_1}$ and $e_{12} = g^{r_1 x}.(1 + D_1 n)$ and $E(D_2) = (e_{21}, e_{22})$ (where $e_{21} = g^{r_2}$ and $e_{22} = g^{r_2 x}.(1 + D_2 n)$), the server calculates the encrypted sum $E(D_1 + D_2) = (e_1, e_2)$, where

$$\begin{aligned}e_1 &= e_{11}.e_{21} \bmod n^2 \\ &= g^{r_1+r_2} \bmod n^2 \text{ and} \\ e_2 &= e_{12}.e_{22} \bmod n^2 \\ &= g^{(r_1+r_2)x}.(1+(D_1+D_2)n) \bmod n^2.\end{aligned}$$

- **ServerScalMul(c, $E(D)$).** Given a constant scalar factor c and an encrypted value $E(D) = (e_1, e_2)$ where $e_1 = g^r$ and $e_2 = g^{rx}.(1 + Dn)$, the server calculates the encrypted scalar multiplication $E(c.D) = (e^*_1, e^*_2)$, where

$$\begin{aligned}e^*_1 &= e_1^c \bmod n^2 \\ &= g^{rc} \bmod n^2 \text{ and} \\ e^*_2 &= e_2^c \bmod n^2 \\ &= g^{rcx}.(1+cDn) \bmod n^2.\end{aligned}$$

- **ServerPreDec($E(D)$, K_{S_j}).** The server runs this algorithm to partially decrypt the encrypted data for the user j. It takes as input the encrypted value $E(D) = (e_1, e_2)$, where $e_1 = g^r$ and $e_2 = g^{rx}.(1 + Dn)$. The server retrieves the key K_{S_j} corresponding to the user j and computes the pre-decrypted data $E^*_j(D) = (\hat{e}_1, \hat{e}_2)$, where

$$\begin{aligned}\hat{e}_1 &= e_1 \bmod n^2 \\ &= g^r \bmod n^2 \text{ and} \\ \hat{e}_2 &= e_1^{-x_{j2}}.e_2 \bmod n^2 \\ &= g^{rx_{j1}}.(1+Dn) \bmod n^2.\end{aligned}$$

- **UserDec($E^*_j(D)$, K_{U_j}).** The user runs this algorithm to decrypt the data. It takes as input the pre-decrypted data $E^*_j(D) = (\hat{e}_1, \hat{e}_2)$ where $\hat{e}_1 = g^r$ and $\hat{e}_2 =$

$g^{rx_{j1}}.(1+Dn)$), and her key K_{U_j}, and retrieves the data by computing:

$$\begin{aligned} D &= L(\hat{e}_2.\hat{e}_1^{-x_{j1}}) \\ &= L(1+Dn), \end{aligned}$$

where

$$L(u) = \frac{u-1}{n}$$

for all $u \in \{u < n^2 | u = 1 \bmod n\}$.

- **Revoke(i).** The server runs this algorithm to revoke user i access to the data. Given the user i, the server removes K_{S_i} from the Key Store as follows: $K_S \leftarrow K_S \backslash K_{S_i}$.

VII. Security Analysis

In this section, we present the security analysis of *2DCrypt*. In general, a scheme is considered secure if no adversary can break the scheme with probability significantly greater than random guessing. The adversary's advantage in breaking the scheme should be a negligible function (defined below) of the security parameter.

Definition 1 (Negligible Function)**.** *A function f is negligible if for each polynomial $p(.)$, there exists N such that for all integers $n > N$ it holds that:*

$$f(n) < \frac{1}{p(n)}$$

We consider a realistic adversary that is computationally bounded and show that our scheme is secure against such an adversary. We model the adversary as a randomized algorithm that runs in polynomial time and show that the success probability of any such adversary is negligible. An algorithm that is randomized and runs in polynomial time is called a Probabilistic Polynomial Time (PPT) algorithm.

The scheme relies on the existence of a pseudorandom function f. Intuitively, the output a pseudorandom function cannot be distinguished by a realistic adversary from that of a truly random function. Formally, a pseudorandom function is defined as:

Definition 2 (Pseudorandom Function)**.** *A function $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ is pseudorandom if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $negl$ such that:*

$$|Pr[\mathcal{A}^{f_s(\cdot)} = 1] - Pr[\mathcal{A}^{F(\cdot)} = 1]| < negl(n)$$

where $s \rightarrow \{0,1\}^n$ is chosen uniformly randomly and F is a function chosen uniformly randomly from the set of function mapping n-bit string to n-bit string.

Our proof relies on the assumption that the Decisional Diffie-Hellman (DDH) is hard in a group $\mathbb{G}$, *i.e.,* it is hard for an adversary to distinguish between group elements $g^{\alpha\beta}$ and g^{γ} given g^{α} and g^{β}.

Definition 3 (DDH Assumption)**.** *The DDH problem is hard regarding a group g if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $negl$ such that:*

$$\begin{aligned} &|Pr[\mathcal{A}(\mathbb{G}, n, g, g^{\alpha}, g^{\beta}, g^{\alpha\beta}) = 1] - \\ &Pr[\mathcal{A}(\mathbb{G}, n, g, g^{\alpha}, g^{\beta}, g^{\gamma}) = 1]| < negl(k) \end{aligned}$$

where $g \leftarrow \mathbb{G}$ is a group of order $\phi(n)/2$ (where $n = pq$), and $\alpha, \beta, \gamma \in \mathbb{Z}_n$ are uniformly randomly chosen.

Theorem 2. *If the DDH problem is hard relative to $\mathbb{G}$, then the proposed Paillier-based proxy encryption scheme (let us call it **PPE**) is INDistinguishable under Chosen Plaintext Attack (IND-CPA) secure against the server S, i.e., for all PPT adversaries $\mathcal{A}$ there exists a negligible function $negl$ such that:*

$$\begin{aligned} &Succ^{\mathcal{A}}_{PPE,S}(k) = \\ &Pr\left[b' = b \;\middle|\; \begin{array}{c} (Params, MSK) \leftarrow Init(1^k) \\ (K_{U_i}, K_{S_i}) \leftarrow KeyGen(MSK, i) \\ d_0, d_1 \leftarrow \mathcal{A}^{ClientEnc(\cdot, K_{U_i})}(K_{S_i}) \\ b \xleftarrow{R} \{0,1\} \\ E^*_i(d_b) = ClientEnc(d_b, K_{U_i}) \\ b' \leftarrow \mathcal{A}^{ClientEnc(\cdot, K_{U_i})}(E^*_i(d_b), K_{S_i}) \end{array} \right] \\ &< \tfrac{1}{2} + negl(k) \end{aligned}$$

Proof. Let us consider the following PPT adversary $\mathcal{A}'$ who attempts to solve the DDH problem using $\mathcal{A}$ as a sub-routine. For the proof technique, we take inspiration from the one presented in [28]. Recall that $\mathcal{A}'$ is given $\mathbb{G}, n, g, g_1, g_2, g_3$ as input, where $g_1 = g^{\alpha}, g_2 = g^{\beta}$ and g_3 is either $g^{\alpha\beta}$ or g^{γ} for some uniformly chosen random $\alpha, \beta, \gamma \in \mathbb{Z}_n$. $\mathcal{A}'$ does for the following:

- $\mathcal{A}'$ sends n, g to $\mathcal{A}$ as the public parameters. Next, it randomly chooses $x_{i2} \in \mathbb{Z}_n$ for the user i and computes $g^{x_{i1}} = g_1.g^{-x_{i2}}$. It then sends (i, x_{i2}) to $\mathcal{A}$ and keeps all $(i, x_{i2}, g^{x_{i1}})$.
- Whenever $\mathcal{A}$ requires oracle access to ClientEnc(.), it passes the data d to $\mathcal{A}'$. $\mathcal{A}'$ randomly chooses $r \in \mathbb{Z}_n$ and returns $(g^r, g^{rx_{i1}}.(1+dn))$.
- At some point, $\mathcal{A}$ outputs d_0 and d_1. $\mathcal{A}'$ randomly chooses a bit b and sends $(g_2, g_2^{-x_{i2}} g_3.(1+d_b n))$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs b'. If $b = b'$, $\mathcal{A}'$ outputs 1 and 0 otherwise.

We can distinguish two cases:

Case 1. If $g_3 = g^{\gamma}$, we know that g^{γ} is a random group element of $\mathbb{G}$ because γ is chosen at random. $g_2^{-x_{i2}} g_3.(1+d_b n))$ is also a random element of $\mathbb{G}$ and gives no information about d_b. That is, the distribution of $g_2^{-x_{i2}} g_3.(1+d_b n))$ is always uniform, regardless of the value of d_b. Further, g_2 does not leak information about d_b. So, the adversary $\mathcal{A}$ must distinguish d_0 and d_1 without additional information. The probability that $\mathcal{A}$ can successfully output b' is exactly $\frac{1}{2}$, when b is chosen uniformly randomly. $\mathcal{A}'$ outputs 1 if and only if $\mathcal{A}$ outputs $b' = b$. Thus, we have:

$$Pr[\mathcal{A}'(\mathbb{G}, n, g, g^{\alpha}, g^{\beta}, g^{\gamma}) = 1] = \tfrac{1}{2}$$

Case 2. If $g_3 = g^{\alpha\beta}$, because

$$\begin{aligned} g_2 &= g^{\beta} \text{ and} \\ g_2^{-x_{i2}} g_3.(1+d_b n) &= g^{-\beta x_{i2}} g^{\alpha\beta}.(1+d_b n) \\ &= g^{\beta(\alpha - x_{i2})}.(1+d_b n) \\ &= g^{\beta x_{i1}}.(1+d_b n) \end{aligned}$$

Thus, $(g_2, g_2^{-x_{i2}} g_3.(1+d_b n))$ is a proper ciphertext encrypted under PPE. So, we have:

$$Pr[\mathcal{A}'(\mathbb{G}, n, g, g^{\alpha}, g^{\beta}, g^{\alpha,\beta}) = 1] = Succ^{\mathcal{A}}_{PPE,S}(k)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2585085, IEEE Transactions on Information Forensics and Security

If the DDH problem is hard relative to $\mathbb{G}$, then the following holds:

$$|Pr[\mathcal{A}'(\mathbb{G}, n, g, g^{\alpha}, g^{\beta}, g^{\alpha\beta}) = 1] - Pr[\mathcal{A}'(\mathbb{G}, n, g, g^{\alpha}, g^{\beta}, g^{\gamma}) = 1]| < negl(k)$$

$$Pr[\mathcal{A}'(\mathbb{G}, n, g, g^{\alpha}, g^{\beta}, g^{\alpha\beta}) = 1] < \tfrac{1}{2} + negl(k)$$

So, we have:

$$Succ^{\mathcal{A}}_{PPE,S}(k) < \frac{1}{2} + negl(k).$$

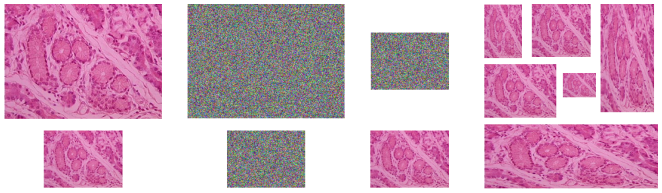
Informally, the theorem says that without knowing the user side keys, the proxy cannot distinguish the ciphertext in a chosen plaintext attack.

VIII. Results and Performance Analysis

In this section, we present the results of a performance analysis of *2DCrypt*. The experiments were performed using a PC powered by Intel i5-4670 3.40 GHz processor and 8 GB of RAM. We implemented the optimized modified Paillier cryptosystem and image scaling cropping operations in C language on Ubuntu 15.04 platform. We used the MIRACL cryptographic library to deal with big number cryptographic primitive operations (such as big number additions, multiplications, modular operations and inverse modular operations). In our implementation, we chose 1024 bits key size. We rounded off the interpolating factors of the bilinear scaling by 2 decimal places. By doing so, each tile contained 16 pixels along with an 8-bit random number. For the modified Paillier encryption, we chose one random number r for all tiles of an image, requiring one $e_1 = g^r$ (first cipher component) for all tiles. For brevity reasons, we present here the validation of *2DCrypt* with the Histo image (350×262) only.

In our implementation, we create 8×8 size non-overlapping super-tiles for an input image. From each super-tile, nine 4×4 size overlapping tiles are created by implementing the space-efficient tiling method discussed in Section V-A. Then, each tile is encrypted and stored in a file. Alternatively, we can also store all the tiles in a single file. For a scaling or cropping request, required tiles are determined (at the Cloud Server end) based on scaling and cropping parameters. These tiles are then fetched from the file for the scaling and cropping operations. For example, for scaling by a factor of two, only the first four tiles of a super-tile are fetched and processed. After decrypting the processed tiles, the required pixels (at the Image User end) of a tile are determined from the scaling and cropping parameters, and the decrypted pixels are grouped to render the requested scaled/cropped image.

Figures 7 and 8 show how *2DCrypt* provides perceptual security in the cloud: in both figures the images on the Cloud Server do not provide any information about the original image. Similarly, the images obtained from the Cloud Server side encrypted tiles also do not reveal any information about the original image. To retrieve an image, the Cloud Server performs the first round of decryption for those tiles sufficient to address the Image User's request. Figure 7 illustrates our encrypted domain scaling scheme; while Figure 8 illustrates our encrypted domain cropping scheme. In *2DCrypt*, the Cloud Server, which performs scaling and cropping, learns no information about the processed image. The Image User recovers the image by decrypting processed images.



(a) (b)

Fig. 7: Encrypted scaling. (a) 1^{st} column illustrates the user perspective - the original image (1^{st} row) and the scaled image (2^{nd} row) desired by the Image User; 2^{nd} column shows the Cloud Server perspective - the encrypted image (1^{st} row) and its scaled version (2^{nd} row); 3^{rd} column again shows the Image User side - the scaled image received, but before decryption (1^{st} row) and after decryption by the Image User (2^{nd} row). (b) *2DCrypt* supporting multiple scaling factors.

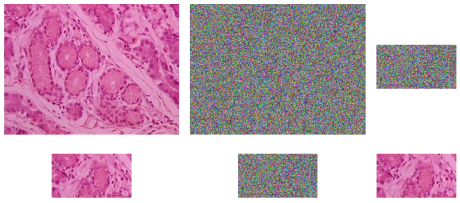


Fig. 8: Encrypted cropping. 1^{st} column illustrates the user perspective - the original image (1^{st} row) and the cropped image (2^{nd} row) desired by the Image User; 2^{nd} column shows the Cloud Server perspective - the encrypted image (1^{st} row) and its cropped version (2^{nd} row); 3^{rd} column again shows the Image User side - the cropped image received, but before decryption (1^{st} row) and after decryption by the Image User (2^{nd} row).

A. Performance Analysis

In this section, we analyze the computational and storage overheads of *2DCrypt*.

In *2DCrypt*, the processing by the Image Outsourcer and the encryption by the Cloud Server are one-time operations. We assume them being carried out in an *offline* manner. The overheads of these operations, however, are directly proportional to the image size as the size of the image determines the number of super-tiles to be created. In *2DCrypt*, we choose equal size super-tiles. Therefore, the size of an image encrypted by the Image Outsourcer or the Cloud Server is equal to the product of the size of an encrypted super-tile with the number of super-tiles. For example, given an 8×8 size super-tile (therefore, 4×4 size tile), then we have 4096 super-tiles and 36864 tiles (9 tiles per super-tile) for a 512×512 image (*e.g.,* the Lena image). Since the size of a super-tile is approximately 18 times of the Paillier security parameter k ($2k$ bits for a tile), for $k = 1024$, we need approximately 9.43 MB space for the encrypted data of the 0.78 MB 512×512 image (approximately 12 times

more data). Similar to the storage overhead, the computational overhead is also a function of the number of super-tiles in the image. To process a super-tile, the Image Outsourcer and the Cloud Server take approximately 3.84 and 2.26 millisecond (ms), respectively. Therefore, for processing the 512×512 image, the Image Outsourcer requires 15.7 seconds (which includes the time taken to create tiles and encrypt them), and the Cloud Server requires 9.256 seconds (which includes the encryption time only).

The image processing and the decryption of the processed images by the Cloud Server, and the decryption by the Image User are performed at runtime according to the scaling and cropping parameters provided by the Image User. The overhead of performing these operations affects the latency of accessing and interacting with the stored images. In *2DCrypt*, the amount of extra data sent from the Cloud Server to the Image User is directly proportional to the Image User's scaling and cropping parameters as these parameters decide how many output tiles (which are resultant of scaling or cropping) to be sent to the Image User. For example, when the 512×512 image is scaled by a scale factor of two, then 4096 tiles are generated from 16384 tiles. Therefore, the Cloud Server sends 1.05 MB (approximately 5.3 times more of the conventional non-encrypted scaling) of data to the Image User. Similarly, the computational overhead of the Cloud Server side processing and decryption, and the Image User side decryption are also directly proportional to the scaling and cropping parameters as they decide which tile to select and operate on. On the Cloud Server, the computational overhead of scaling and decryption, however, is more than the computational overhead of cropping and decryption as unlike scaling, cropping does not involve any mathematical operation. This observation is also evident from our analysis. In our analysis, the Cloud Server needs to work approximately 3.07 ms for scaling four tiles to one resultant tile and to decrypting the resultant tile. In the case of cropping, only decryption time counts, which is approximately 0.36 ms for one tile. Irrespective of scaling and cropping, the Image User needs approximately 0.56 ms for decrypting a tile: thus requiring approximately 2.3 seconds for decrypting a 256×256 image resulted from scaling a 512×512 image by a factor of two.

Note that these results have been collected using a non-optimized version of our code. We will discuss some of these optimizations in the next section. Our current implementation is based on MIRACL which implementation is also not optimized. Finally, worthwhile to note is that we expect more improvements in terms of latency when a more realistic top-of-the-line hardware configuration is used for the Cloud Server side.

B. Performance Analysis on a Parallelized Prototype

We have implemented a parallelized version of *2DCrypt* using OpenMP to be deployed on a cluster. All functions executed by the client and the server were tested on a single cluster with 64 Intel $i5$ 3.3 GHz processors with 256 GB RAM, running Ubuntu 14.08 Linux system. In our testing scenario, all operations ran on one cluster and we ignored the network latency.In the following, all the results are averaged over 10 trials using a 3872×2592 pixel image.

In these settings, the time taken for performing a client encryption is 5.75 seconds while the server second round of encryption takes 4.4 seconds. When the client requests the server to perform a scaling operation by a factor of 2 from the original image, this will take 10.7 seconds. This also includes the server pre-decryption operation. To decrypt the scaled image, the client takes 1.74 seconds.

IX. Discussion

A. Optimization vs Security

As discussed in Section V-D, we improve both storage and performance overheads by using the same random component, *i.e.,* $e_1 = g^r$, for all tiles. To make it difficult for the cloud to distinguish two same tiles, we exploit the remaining bits in the tile to embed a random number. From the provable security point of view, we understand that embedding this random number is not a substitute of g^r. Theoretically, the proposed optimized version is not semantically secure as the division of two secret tiles can be pre-computed. However, finding out their values from their division result is not feasible (almost same as brute-forcing) because they are large random numbers. Thus, we believe that the optimized version provides perceptual security. For provable security, we recommend using a different g^r for each tile. In terms of storage overhead, ensuring provable security will double the size of an encrypted image. Although, the computational overhead of this non-optimized version can be brought down to similar to the computational overhead of the optimized version by pre-computing a set of random components. That is, we can pre-compute $\{(r_1, g^{r_1}), (r_2, g^{r_2}), \cdots\}$. Nonetheless, if performance is more important than security, we recommend to use the optimized version. Otherwise, non-optimized version can be used.

B. Streaming Latency

In *2DCrypt*, the streaming latency is affected by several components: such as the time taken by the Cloud Server for processing the encrypted image (scaling/cropping) and the first round of decryption; the data transmission time; and the time taken by the Image User for the second round of decryption. In this section, we discuss how this latency can be decreased by adopting some optimizations.

Pre-Processing and Pre-Decryption. To decrease the time taken in image processing and decryption, the Cloud Server can predict the request, and perform these operations without waiting for the actual request. For prediction, we can use encrypted-domain machine learning techniques [29] to forecast requests from existing or new users. Upon a request, the forecasted scaling and cropping parameters can be compared with actual ones sent by Image Users and if their difference is less than a threshold (*i.e.,* with accepted error), then the pre-processed image can be sent to the Image User. Otherwise, if the difference exceeds the threshold, the Cloud Server computes the results on-the-fly.

To decrease the workload, the Cloud Server can use caching to store recently processed images. When a request is received,

the cache can first be looked up to find out if the request can be fulfilled by a cached image. If not, fresh operations are performed on the cached image to fulfill the request. That is, if the cache contains an image that fulfills the scaling request but does not fulfill the cropping request, then cropping can be performed on this image (no need to perform scaling and cropping from scratch). This caching can be handy for a multi-user setup, where a set of users can issue similar requests.

Decreasing Data Transmission Time. One way to decrease the data transmission time is to decrease the network RTT. *2DCrypt* is suitable for this purpose as we can employ Cloud Servers geographically closer to the Image User's premises. For example, when the Image Outsourcer and Image User are in different countries, the images can be stored on a Cloud Server in the Image User's region. Since we store images in an encrypted form, privacy concerns, which are the main hurdle in outsourcing data to different jurisdictions, are addressed.

2DCrypt is also suitable to residual image streaming, where instead of sending the whole image, we can send the difference to the previously sent image. For example, if scale-by-two has been requested after scale-by-four, then while addressing the scale-by-two request, the Cloud Server can send only those tiles that were not part of the scale-by-four request.

C. Enhancing Security for Image User

One of the key requirements of our scheme is to preserve the privacy of Image Users. An Image User must be anonymized to ensure her privacy, and the encryption key must be protected from being misused. We can hide the Image User's identity by using a standard user anonymization scheme, say using pseudonyms or anonymous communication networks (such as ToR). To protect encryption keys, we could exploit our encryption scheme to provide session keys. Since the user's key is independent of the stored images, the keys can be generated per-session, so that the user gets a different key for each different session. Therefore, when an adversary gets access to the user's key, she can only get information of a given session.

D. Collusion Attacks

Compared to state-of-the-art in supporting operations on encrypted images based on Shamir's secret sharing scheme, *2DCrypt* does not need to store multiple versions of the same image on different Cloud Servers. In fact, if k of these servers collude the image could be recovered. In our case, even if we stored the image on multiple Cloud Servers (for instance to reduce latency) these cannot collude because even putting together their server-side keys would not allow them to decrypt the image.

On the other hand, if an Image User colludes with the Cloud Server they could recover the master key by putting together their key shares. To avoid this issue, we could split the user side key into two shares: one share is provided to the user as her private key; the second share is provided to a *Trusted Authority* that is under control of the organization managing Image Outsourcers. In this way, even if a user and the Cloud Server collude, combining their shares would not allow them to retrieve the master key.

X. Conclusions and Future Work

Cloud-based image processing has data confidentiality issues, which can lead to privacy loss. In this paper, we addressed this issue by proposing *2DCrypt*, a modified Paillier cryptosystem-based scheme that allows a cloud server to perform scaling and cropping operations without learning the image content. In *2DCrypt*, users do not need to share keys for accessing the image stored in the cloud. Therefore, *2DCrypt* is suitable for scenarios where it is not desirable for the image user to maintain per-image keys. Furthermore, *2DCrypt* is more practical than existing schemes based on Shamir's secret sharing because it neither employs more than one datacenter nor assumes that multiple adversaries could collude by accessing a certain number of datacenters.

To make *2DCrypt* practical, we propose some improvements to decrease overheads resulted from the application of the modified Paillier cryptosystem. First, we proposed a space-efficient tiling scheme that allows the cloud to perform per-tile operations. In *2DCrypt*, we put a number of pixels in a tile, and encrypt the tile instead of encrypting each pixel independently. Furthermore, we optimized the modified Paillier scheme to limit its storage requirement. Due to these improvements, *2DCrypt* requires approximately 40 times less cloud storage than the naive per-pixel encryption. The computational overhead is also significantly reduced because of fewer encryptions and decryptions rounds. The exact computational overhead and the data required by the image user, however, are dependent on the image size and the user's scaling and cropping parameters. For example, when a 512×512 image is scaled by a factor of two, the user needs approximately 5.3 times more data and works 2.3 seconds more than the conventional processing.

We believe that *2DCrypt* can be extended in multiple ways. An obvious direction is to extend this work for compressed images. Another approach can be using our idea for addressing security issues in more specialized images, such as histopathology images and G.I.S maps. It will be interesting to investigate if we can utilize properties of these specialized images to further decrease overheads. Another possible future work can be extending our work to video processing in encrypted domains.

Acknowledgements

We would like to thank anonymous reviewers for providing useful comments that helped us in improving the quality of our work. Also, we are grateful to Prof. Steven Galbraith from the Department of Mathematics at The University of Auckland for proofreading this manuscript. This research is supported by STRATUS (Security Technologies Returning Accountability, Trust and User-Centric Services in the Cloud), a project funded by the Ministry of Business, Innovation and Employment (MBIE), New Zealand.

REFERENCES

[1] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, Stanford, USA, 2009.

[2] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, 2011, pp. 113–124.

[3] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, pp. 612–613, November 1979.

[4] M. Mohanty, W. T. Ooi, and P. K. Atrey, "Scale me, crop me, know me not: supporting scaling and cropping in secret image sharing," in *Proceedings of the 2013 IEEE International Conference on Multimedia & Expo*, San Jose, USA, 2013.

[5] K. Kansal, M. Mohanty, and P. K. Atrey, "Scaling and cropping of wavelet-based compressed images in hidden domain," in *MultiMedia Modeling*, ser. Lecture Notes in Computer Science, 2015, vol. 8935, pp. 430–441.

[6] C.-C. Thien and J.-C. Lin, "Secret image sharing," *Computers and Graphics*, vol. 26, pp. 765–770, October 2002.

[7] T. Bianchi, A. Piva, and M. Barni, "Encrypted domain DCT based on homomorphic cryptosystems," *EURASIP Journal on Multimedia and Information Security*, vol. 2009, pp. 1:1–1:12, January 2009.

[8] X. Sun, "A blind digital watermarking for color medical images based on PCA," in *Proceedings of the IEEE International Conference on Wireless Communications, Networking and Information Security*, Beijing, China, August 2010, pp. 421–427.

[9] N. K. Pareek, V. Patidar, and K. K. Sud, "Image encryption using chaotic logistic map," *Image and Vision Computing*, vol. 24, pp. 926–934, September 2006.

[10] W. Lu, A. L. Varna, and M. Wu, "Confidentiality-preserving image search: A comparative study between homomorphic encryption and distance-preserving randomization," *IEEE Access*, vol. 2, pp. 125–141, February 2014.

[11] C.-Y. Hsu, C.-S. Lu, and S.-C. Pei, "Image feature extraction in encrypted domain with privacy-preserving SIFT," *IEEE Transactions on Image Processing*, vol. 21, no. 11, pp. 4593–4607, 2012.

[12] J. Yuan, S. Yu, and L. Guo, "SEISA: Secure and efficient encrypted image search with access control," in *IEEE Conference on Computer Communications*, 2015, pp. 2083–2091.

[13] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology EUROCRYPT*, 1999, vol. 1592, pp. 223–238.

[14] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[15] J. Benaloh and D. Tuinstra, "Receipt-free secret-ballot elections (Extended Abstract)," in *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, 1994, pp. 544–553.

[16] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, February 1978.

[17] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in Cryptology*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1985, vol. 196, pp. 10–18.

[18] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *IEEE Symposium on Security and Privacy*, 2000, pp. 44–55.

[19] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology-Eurocrypt*, 2004, pp. 506–522.

[20] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Applied Cryptography and Network Security*, 2004, pp. 31–45.

[21] M. R. Asghar, G. Russello, B. Crispo, and M. Ion, "Supporting complex queries and access policies for multi-user encrypted databases," in *Proceedings of the ACM Workshop on Cloud Computing Security Workshop*, 2013, pp. 77–88.

[22] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft, "Privacy-preserving face recognition," in *Privacy Enhancing Technologies*. Springer, 2009, pp. 235–253.

[23] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "Efficient privacy-preserving face recognition," in *Information, Security and Cryptology–ICISC 2009*. Springer, 2010, pp. 229–244.

[24] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich, "SCiFI - A system for secure face identification," in *IEEE Symposium on Security and Privacy*, May 2010, pp. 239–254.

[25] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Transactions on Information and System Security*, vol. 9, pp. 1–30, February 2006.

[26] E. Ayday, J. L. Raisaro, J.-P. Hubaux, and J. Rougemont, "Protecting and evaluating genomic privacy in medical tests and personalized medicine," in *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society*, 2013, pp. 95–106.

[27] E. Bresson, D. Catalano, and D. Pointcheval, "A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications," in *Advances in Cryptology - ASIACRYPT 2003*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, vol. 2894, pp. 37–54.

[28] C. Dong, G. Russello, and N. Dulay, "Shared and searchable encrypted data for untrusted servers," *Journal of Computer Security*, vol. 19, pp. 367–397, August 2011.

[29] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *Network and Distributed System Security Symposium*, February 2015.

Manoranjan Mohanty is a Post-Doctoral Researcher in Center for Cyber Security, New York University Abu Dhabi, UAE. He received B.Sc. in Computer Science from Ravenshaw College, Cuttack, Odisha, India, Master of Computer Applications from Jawaharlal Nehru University, New Delhi, India, and Ph.D. in Computer Science from School of Computing, National University of Singapore, Singapore. His research interests include encrypted domain processing, digital forensics, multimedia security, and privacy-preserving biometrics.

Muhammad Rizwan Asghar is a Lecturer in the Department of Computer Science at The University of Auckland in New Zealand. Prior to joining this tenure-track faculty position, he was a Post-Doctoral Researcher at international research institutes including Saarland University in Germany and CREATE-NET in Trento Italy, where he also served as a Researcher. He received his Ph.D. degree from the University of Trento, Italy in 2013. As part of his Ph.D. program, he was a Visiting Fellow at the Stanford Research Institute (SRI), California, USA. He obtained his M.Sc. degree in Information Security Technology from the Eindhoven University of Technology (TU/e), The Netherlands in 2009. His research interests include access control, applied cryptography, security, privacy, cloud computing and distributed systems.

Giovanni Russello is an Associate Professor in the Department of Computer Science at the University of Auckland, New Zealand. He received his M.Sc. (summa cum laude) degree in Computer Science from the University of Catania, Italy in 2000, and his Ph.D. degree from the Eindhoven University of Technology (TU/e) in 2006. After obtaining his Ph.D. degree, he moved to the Policy Group in the Department of Computing at Imperial College London, UK. His research interests include policy-based security systems, privacy and confidentiality in cloud computing, smartphone security, and applied cryptography. He has published more than 60 research articles in these research areas and has two granted US Patents in smartphone security. He is an IEEE member.