LockPic: Privacy Preserving Photo Sharing in Social Networks

Carlos Pares-Pulido and Isaac Agudo(✉)

Network, Information and Computer Security Lab.,
University of Malaga, Malaga, Spain
carlosparespulido@gmail.com, isaac@lcc.uma.es

Abstract. There are many privacy concerns related to the use of social networks, in particular the posting of pictures and controlling who has access to them. In this paper we introduce a solution for the distribution of personal or sensitive pictures. Our aim is to provide a method for secure and privacy friendly picture sharing through social networks, that allows users to encrypt sensitive regions in pictures (particularly, faces) in a reversible, non-intrusive way, leaving the rest of the picture unaltered. This way, any image can be freely published and distributed on any social network, and viewed by as many users as the platform allows, while the protected parts are only accessible with the corresponding key. Once the key for a particular region has been acquired, the receiver of the picture can decrypt this region without downloading any additional information. The core of our proposal is a C library, which efficiently integrates an encryption/decryption algorithm with the encoding/decoding process. We have also released an Android application, LockPic, and a companion key server that showcase all the functionality mentioned in this work.

Keywords: Partial image encryption · Privacy in social networks · Reversible image scrambling

1 Introduction

Nowadays social networking is booming; and with it, the public sharing of personal photographs. From a privacy point of view, as soon as a user publishes a picture, he or she loses all control over it. Even if that content is later removed from the servers, other users that have stored it can share it again indefinitely, even contrary to the wishes of the original owner. The situation worsens when the picture involves other people who may not wish the content to be shared (or even be aware of it) or when legal restrictions apply to the subjects, as is the case for children in most countries; or when some of the subjects have certain notoriety.

Traditional protection mechanisms do not fit well in this scenario. Instead of protecting the whole picture only critical parts of it should be protected or hidden. This will allow other members of the social network to preview the picture without compromising the privacy of the users in the picture. Usually,

© Springer International Publishing Switzerland 2016
J. Garcia-Alfaro et al. (Eds.): DPM and QASA 2015, LNCS 9481, pp. 281–290, 2016.
DOI: 10.1007/978-3-319-29883-2_21

faces are the target for protection but other elements might need to be hidden too, e.g. number plates. The advantage of reversible hiding is that authorized users are able to recover the whole picture while non-authorized users only get the public parts.

There are three parameters that we need to take into account when proposing a solution:

- **Usability.** The hiding technique should be as unobtrusive as possible and should be lightweight enough to run smoothly in mobile platforms.
- **Interoperability.** It should be easy to integrate the hiding process with the photo sharing work flow.
- **Security.** It should be hard enough to recover the original picture without knowing the key material used for hiding the critical parts.

The centre of our proposal is a C library, based on OpenSSL and open-source JPEG codecs, which integrates cryptographically strong encryption (AES) in the encoding process of JPEG pictures. The LockPic app[1] is built upon this C library an is able to encrypt sensitive parts of a picture and decrypt them afterwards. We have also implemented a companion Key Server that is used to convey the cryptographic keys used to protect the pictures.

2 Related Work

There are many proposals for partial or selective encryption of pictures and videos in the literature [8]. If we focus on still pictures, most of them rely on the use of the JPEG2000 format [3], that provides a better basis for partial encryption but has a major drawback: it is mostly unsupported by current social networks and web browsers. In this section we focus on solutions based on JPEG [5].

In the following paragraph we briefly summarize how JPEG images are encoded, in order to explain how the encryption mechanism can be integrated. An image has between 1 and 3 channels: a luminance channel, and up to 2 chrominance channels. Each channel is broken down into square blocks, called MCU (Minimum Coded Units) upon which encoding is performed separately. Each MCU undergoes a discrete cosine transformation, and the 64 coefficients corresponding to the lowest frequencies (the ones which human eyes can distinguish best) are kept. These values are rounded, to maximize the number of zeros between them; this step, called quantization, introduces losses. Finally, this sequence of 64 numbers is compressed without loss, using Huffman encoding. This clearly points to an optimal place where to perform the encryption process as the set of 64 coefficients obtained after quantization suffer no further losses beyond this point.

In [2] authors present an approach that could be used for privacy preserving video surveillance. Later in [7] they adapted their approach to support JPEG

[1] https://www.nics.uma.es/lockpic.

and provide a prototype iOS application, Proshare. Their solution is based on flipping the sign of the coefficient in the encoded image, i.e. using any cryptographically secure pseudo random bit sequence generator, the sign of a coefficient is changed for a 1, or kept as a 0. Although this scheme works well in general, its application to high resolution images is limited because the scrambling becomes less noticeable for all proposed encryption levels, except for the ultra-high level that encrypts the DC components using a one time pad. Unfortunately, there is not enough information about what kind of PRNG is used for sign flipping and one time pad encryption in the ultra-high level in order to evaluate the real strength of this proposal. Another important aspect is that scrambling and descrambling on client application and automatic key distribution is not supported in the prototype at the time of writing.There are some other approaches [6,11] that also base their encryption algorithm on a pseudo-random shift on the JPEG coefficients, trying to keep enough information for the whole image to be "homogeneous" but at the same time trying to limit the chances to recognize the scrambled portion.

The main security risk of schemes based on flipping/shifting the signs of the JPEG coefficients based on some random patterns is that brute force attacks can reconstruct the pattern taking advantage of the characteristics of the image by looking at the neighbouring pixels of the scrambled area in the first phase and iterating from the edge to the centre. Furthermore, its visual output is fairly obtrusive, as seen in Fig. 1a.

To mitigate the visual impact, we can follow the idea described in [11]: always keeping the most significant coefficient (DC), and pseudorandomly shifting the remaining 63. The effects of this alteration make the output visually acceptable, as shown in Fig. 1b. However, this not only makes the algorithm less secure from a theoretical point of view (by exposing even more information), but also in practice. Encrypting only the signs of the AC coefficients has been proven insecure for video contents [4]. The DC, which holds colour information, remains unaltered. This means that the average colour in any encrypted MCU will be the same as prior to encryption. For a large enough image, an MCU basically behaves as a pixel; therefore making the encryption irrelevant to the naked eye (even if information is altered) as we mentioned before. This is shown in Fig. 1c. This is roughly the same effect that all encryption levels will suffer from, apart from the ultra-high in [7].

One of the key aspects of all these proposals is that the scrambled image contains all the information needed to recover the original image given the encryption key. From a practical perspective, this seemed to be a good starting point, since the algorithm for encryption and decryption is very simple and efficient. In [9], the authors present a solution that stores some encrypted information needed to recover the image in the Cloud. Their solution is based on a proxy that intercepts all the images downloaded from the social network and decrypts them on the fly, providing a transparent decryption service. Although the authors claim that their solution only adds minimal photo storage overhead in the Cloud, it is true that they add a new dependency. Also, the transparent proxy is harmless

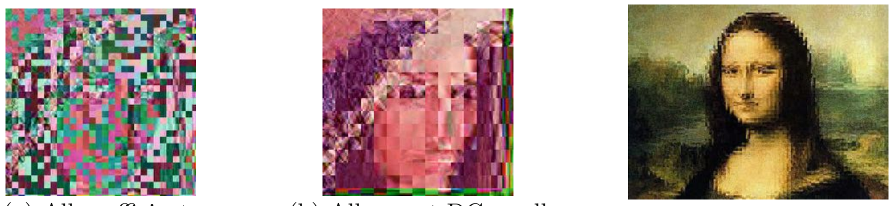

(a) All coefficients (b) All except DC small (c) All except DC medium

Fig. 1. Pseudorandomly shifted coefficient signs

in HTTP connections but can cause some security and trust issues when using HTTPS, which is the current standard in social networks.

In [10] they take a different approach, instead of encrypting some parts of the image they used a JPEG file as a container for an encrypted image. Their focus is to preserve recoverability of the container JPEG in common social networks, i.e. resistant to JPEG re-compression. Their solution is based on Javascript, providing browser side decryption of the image without requiring any server to store encrypted parts of the image but fails to protect only parts of the images, their proposal is encrypt all or nothing. As the container JPEG is not representative of the shared picture this scheme is somewhat equivalent to just sharing a link to the protected picture.

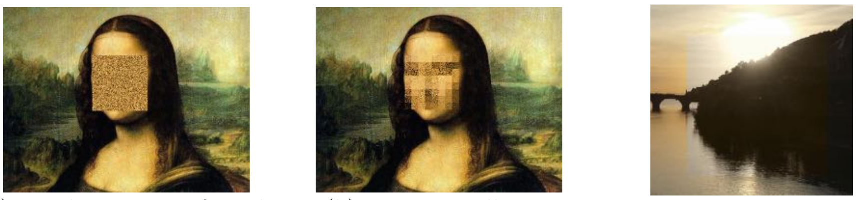
Another issue that is usually disregarded in most proposals is the usability of the application. Some applications assume there is a given box in the picture to be protected but how the coordinates are defined and how the protection is activated is not mentioned. In [1] the authors propose the use of QR codes as wearable Tags to activate the protection mechanism. In their approach, users who wear a *Privacy.Tag*, i.e. a printed QR code, are recognized when taking a picture and their face is automatically protected based on their preferences.

3 The Encryption/decryption Module

Our proposal is based on an C library that performs JPEG encoding/decoding at the same time as encryption/decryption. This core C library requires symmetric keys for the encoding and decoding of JPEG pictures.

We plan the following requirements for the encryption/decryption process: It has to be **reversible**; the output of the encryption must still be a **valid image** in a standard format; and the process should be **cryptographically secure**. We aim to distort, beyond recognition, any encrypted region in the picture, leaving the rest fundamentally unaltered. The encryption/decryption procedure should be lightweight, and produce encrypted files of a manageable size. Furthermore, ideally, obfuscated regions in the encrypted image should be visually unobtrusive with respect to the rest of the picture; i.e., colours and the rough outline of the picture should remain similar to the original. This latter requirement, in a way, conflicts with the desired distortion. The higher the distortion level gets, the more obtrusive the output becomes. We have tried to find a compromise in the distortion level, favouring security over aesthetics.

Our first approach was to use an encryption mechanism based on secure permutations of pixels in the regions to be obfuscated; but that meant relying on the BMP format, which posed several problems - principally speed and output size. In Fig. 2a we can see that the randomization of pixels preserve the colours of the original picture but not the shape of the underlying face. This can be partially solved by decomposing area of interest in small tiles, applying the pseudo randomization only in the tiles. In Fig. 2b we can see that the encryption is less intrusive in this case. As mentioned before the main problem of working at the bitmap level is the size of the pictures. If we later convert the resulting images into a compressed format we have some issues with colour distortion, Fig. 2c that can not be easily solved. After the evaluation of the pros and cons, we took a completely different approach to encryption, instead of considering the image as a matrix of pixels, we directly targeted JPEG-encoded images, and worked on the integration of the encryption process inside the JPEG codec.



(a) Randomization of pixels (b) Using tessellation (c) JPEG decoding halo

Fig. 2. Bitmap level encryption

In our proposal, the sequence of all non-zero coefficients in each MCU is encrypted using a standard symmetric cipher, specifically we use AES in OFB mode, resulting in a new set of coefficients. It is important to note that this scheme does not in any way respect the original distribution of colours. However, irrespective of the size of the image, the encrypted regions will remain fully obfuscated. In Appendix B there is a sample encryption output.

We decided to encrypt only the non-zero coefficients in order to preserve the effectiveness of the compression algorithm. Huffman encoding takes advantage of long runs of zeros, but those patterns are removed if we encrypt all coefficients. In practice, encrypting all coefficients resulted in JPEG images which, even if smaller than their BMP counterparts, were still too big to be conveniently manageable. We admit that this weakens the scheme (as opposed to also encrypting zeros) because we are openly revealing which coefficients are null and which are not. Still, this information is largely insufficient to reconstruct an image, and in exchange, it allows the encrypted images to be the size of a regular JPEG file.

This scheme fulfils most of the requirements we initially planned for the encryption procedure: it is indeed reversible, completely obfuscates the encrypted regions, and does not affect any other region in the image. The size of the encrypted files is almost the same as the original file and the encryption/decryption process is fast and lightweight. There are also some negative aspects to our proposal. Firstly, it implies some degree of quality loss (unavoidable due to the conversion to JPEG, which is always lossy). Secondly, the encrypted areas are very obtrusive in the picture. It should also be noted that this implementation is not tolerant to further compression of the image.

The fact that some social networks recompress pictures in order to optimise storage and network bandwidth may force us to use some external services for storing protected pictures (e.g. Flickr, ImageShack, Dropbox, etc.), sharing only the link to the picture instead of the picture itself. This way the social network will only store the preview of the picture, whereas the original picture can always be retrieved from the external service. However there are social networks that do not recompress the pictures and keep the original metadata, (e.g. Google+) in which we do not need an external service.

4 LockPic Elements and Security Model

We assume pictures are stored by their owner in a service of their choice. It can be a social network or a photo sharing service, but the user can also share pictures using email or other means. We assume this server will not alter the pictures but it might be interested in learning the protected contents.

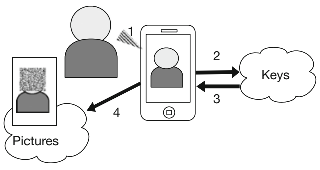
The key server is queried by the LockPic application using a secure channel, whenever a picture needs to be decrypted or encrypted. The key server does not know anything about the contents of the picture, only some metadata and the encryption keys. This way, pictures and the keys needed to encrypt/decrypt them, are stored in different trust domains.

The key server and the photo sharing service are considered honest-but-curious servers. We assume they will not collude to try to compromise the user's privacy. The Key Server will only provide keys to authorized users and will generate keys and IDs for pictures as instructed. The photo sharing service is trusted not to alter pictures but there are no additional requirements as to whom is granted access to the encrypted pictures. Other users can interact with both services using the standard APIs and are not supposed to be trusted. Users can collude among themselves and can have access to all pictures stored in the photo sharing service but can only get keys according to the access granted in the Key Server.

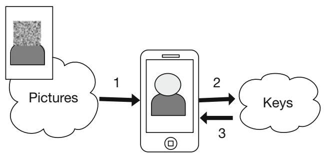
The encryption work flow (Fig. 3a) consists of four steps. It usually starts when a picture is taken using the LockPic application but users can also select a picture already stored in their phone. The application will then show the picture to the users and prompt them to select which areas need protection and who is going to be able to access them. After that, the Key Server is queried by the application using a secure channel, in order to get the encryption key for each area of interest. The application sends the name and the dimensions of the picture as well as the list of coordinates for each of the boxes that need to be encrypted together with the list of users that are granted access to them. The Key Server generates a random ID for the picture and stores the information about the protected boxes in the database. Then, the server sends back the ID and the list of encryption keys corresponding to each of the boxes requested, one key per box. We explain the details of the key generation in Appendix A.

The application passes the encryption keys, coordinates and ID to our modified JPEG encoding library that will encrypt the boxes using the corresponding keys and include the ID in the system of the picture. The ID needs to be present in the metadata of the picture in order to allow other users to query the decryption keys. Finally, the application shows the encrypted image and offers the user some sharing choices.

As shown in Fig. 3b the decryption work flow consists of three main steps and is triggered each time the user receives an encrypted picture. First, the picture is loaded by the LockPic application. The ID of the picture is extracted from the metadata. Second, the application authenticates the user against the Key Server and submits the picture ID in order to get the keys for the boxes they have access to. The Key Server gets from the data base the list boxes present in the picture based on the picture ID. For each box it checks whether the user has been granted access or not. Then, it returns to the application the list of boxes that the user has been granted access to, together with the corresponding decryption keys. Finally, the application passes on the list of boxes and keys to the modified JPEG decoding library.



(a) Encryption Work Flow



(b) Decryption Work Flow

Fig. 3. LockPic Encryption and Decryption Work Flows

Access rights can be modified by the owner of the picture at any time. Users can request from the Key Server the list of picture IDs that has been generated for them, together with the coordinates of the encrypted boxes and the users that are currently granted access. They can add new users or delete existing ones from current boxes, however they cannot create new boxes. With the current API, the application needs to request a new picture ID when the users want to modify or delete existing boxes, or create new ones.

The Key Server provides the encryption/decryption keys to the users based on the picture ID and the user ID, hence the first step is to be able to authenticate users against the Key Server. We have used Google Accounts for authentication in order to take full advantage of the Google ecosystems. This way, authentication of users is transparently managed by the Android operating system and the Google App Engine where we have deployed an instance of the Key Server.

5 Conclusions and Future Work

The LockPic application has been implemented as a prototype to showcase the functionality of the C library for simultaneous JPEG encryption/decryption and encoding/decoding. We have released the source code of the application and the key server in order to demonstrate the feasibility of our approach and help other built upon our work.

As for future work, we would like to explore other architectures, authentication flows, and key distribution methods that can work without the need of an on-line key server and include our functionality in messaging applications (e.g. Telegram). We can use an hybrid approach where the symmetric keys used to encrypt each region are encrypted with the public key of the intended recipients. Challenges in this scenario will be to optimize the size of the metadata needed to encode the cryptographic keys. We would also like to broaden the reach of our work - release the encryption module as a standalone C library, and release the application for other operating systems. Another important issue that is not tackled in the present paper is how to keep track of who is accessing your pictures. We think our approach can be easily adapted to provide a better view of who has accessed the pictures based on the release of decryption keys. However, this will require modifications both on the client - and server - side.

Acknowledgements. The research leading to these results has received funding from the Junta de Andalucía through the projects FISICCO (P11-TIC-07223) and PISCIS (P10-TIC-06334). We also thanks the anonymous reviewers for their valuable comments.

A Managing Encryption/decryption Keys

Apart from providing a proper security level and an efficient implementation, one relevant challenge is to properly manage all the encryption keys used in the system. We propose a centralised approach where all keys are stored in the trusted Key Server.

It is essential that the server is able to uniquely identify images in order to be able to generate unique keys for each picture and region in it. As we have mentioned, the Key Server randomly generates a unique identifier for each protected picture that is sent back to the LockPic application at encryption time. This unique ID is included in the metadata of the encrypted picture. Another approach could be to use the hash of the picture as ID. The problem of using the hash as the ID is that the hash has to be performed in the mobile application, which might be an expensive operation depending on the size of the picture, and could present security problems in the case that hash collisions are found. More importantly, the key server would be able to analyse some usage patterns as it would be able to recognize if two different users encrypt the same picture.

As mentioned, the key generation process is performed on the server side. Our initial approach was to generate a separate key for each protected region in every image. This, however, posed some problems because, due to the speed at which random numbers may be needed, the Random Number Generator (RNG) might act as a bottleneck. It would also be difficult to estimate the size of the key store as it would grow in proportion to the number of regions protected. Since having different keys for different regions is mandatory in order to allow for fine grain access control to regions, we have taken the following approach.

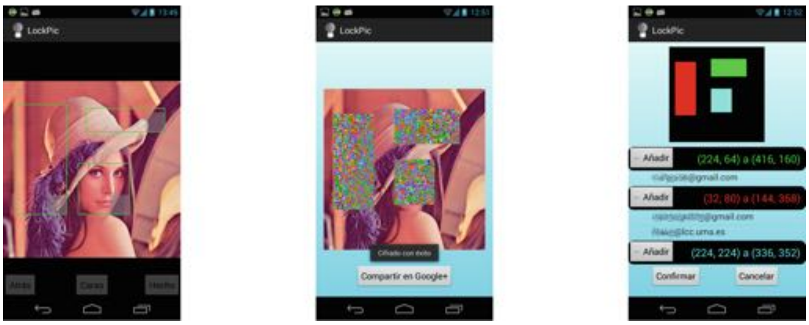
For each user, U, a master secret, MS_U is randomly generated at the first access. For every region to encrypt, this secret is concatenated with the picture identifier, ID, and the coordinates of the region, $r = \{x_0, y_0, x_1, y_1\}$; a secure hash function is subsequently applied on this string of bits, and its output is used as the encryption key for the region, i.e.

$$key_{U,ID,r} = hash(MS_U \parallel ID \parallel x_0 \parallel y_0 \parallel x_1 \parallel y_1)$$

The main advantage of this design is that it only uses the RNG once per user and that the number of keys managed by the Key Servers is linear on the number of users, thus independent from the number of pictures or encrypted boxes.

B The LockPic App

The LockPic App uses a very simple user interface with three different choices: Encrypt, Decrypt and My Pictures. The first choice triggers the encryption mechanisms, users are prompted to choose a picture from the gallery and are required to select which regions need to be protected. The selection of protected (Fig. 4a) areas can be performed manually, by placing a box over the desired regions and scaling it by dragging the lower-right corner. Another option is to rely on Android face detection APIs in order to get boxes over the detected faces. In any case, boxes can be easily rearranged and scaled with one finger movement.



(a) Regions selection (b) Encrypted result (c) My Pictures

Fig. 4. LockPic user interface

Once the regions have been selected, the user is prompted to select which contacts are authorized to decrypt each of the regions. This step can be skipped and new permissions can be set up later on. Then, the encrypted image (Fig. 4b) that will be stored in the LockPic folder is shown.

Decryption is performed by checking the picture ID included in the metadata and requesting from the key server the corresponding decryption keys. The decrypted image is shown to the user but never stored in the file system. LockPict also provides users with the opportunity to review their access control policies (Fig. 4c). It retrieves from the key server all picture IDs created by the user together with their associated encrypted regions and the list of authorized users and gives the user the choice to modify (add or remove) the users allowed to view each of the regions.

References

1. Bo, C., Shen, G., Liu, J., Li, X.-Y., Zhang, Y., Zhao, F.: Privacy.tag: privacy concern expressed and respected. In: Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems, SenSys 2014, pp. 163–176 (2014)
2. Dufaux, F., Ouaret, M., Abdeljaoued, Y., Navarro, A., Vergnenègre, F., Ebrahimi, T.: Privacy enabling technology for video surveillance. In: Defense and Security Symposium, International Society for Optics and Photonics (2006)
3. Engel, D., Sttz, T., Uhl, A.: A survey on JPEG2000 encryption. Multimedia Syst. **15**(4), 243–270 (2009)
4. Hofbauer, H., Unterweger, A., Uhl, A.: Encrypting only AC coefficient signs considered harmful. In: IEEE International Conference on Image Processing (2015)
5. ITU. Iso/iec 10918–1: (e) ccit recommendation t.81 (1993)
6. Khan, M.I., Jeoti, V., Khan, M.A.: Perceptual encryption of JPEG compressed images using DCT coefficients and splitting of DC coefficients into bitplanes. In: International Conference on Intelligent and Advanced Systems (ICIAS) (2010)
7. Korshunov, P., Ebrahimi, T.: Scrambling-based tool for secure protection of JPEG images. In: IEEE International Conference on Image Processing (ICIP) (2014)
8. Massoudi, A., Lefebvre, F., De Vleeschouwer, C., Macq, B., Quisquater, J.-J.: Overview on selective encryption of image and video: challenges and perspectives. EURASIP J. Inf. Secur. **2008**(1), 179290 (2008)
9. Ra, M.-R., Govindan, R., Ortega, A.: P3: toward privacy-preserving photo sharing. In: Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation, NSDI 2013 (2013)
10. Tierney, M., Spiro, I., Bregler, C., Subramanian, L.: Cryptagram: photo privacy for online social media. In Proceedings of the First ACM Conference on Online Social Networks, COSN 2013 (2013)
11. Van Droogenbroeck, M., Benedett, R.: Techniques for a selective encryption of uncompressed and compressed images. In: Advanced Concepts for Intelligent Vision Systems (ACIVS) (2002)