# Thumbnail Preserving Encryption for JPEG

Charles V. Wright Wu-chi Feng Feng Liu

Portland State University
Portland, Oregon USA
{cvwright, wuchi, fliu}@cs.pdx.edu

## ABSTRACT

With more and more data being stored in the cloud, securing multimedia data is becoming increasingly important. Use of existing encryption methods with cloud services is possible, but makes many web-based applications difficult or impossible to use. In this paper, we propose a new image encryption scheme specially designed to protect JPEG images in cloud photo storage services. Our technique allows efficient reconstruction of an accurate low-resolution thumbnail from the ciphertext image, but aims to prevent the extraction of any more detailed information. This will allow efficient storage and retrieval of image data in the cloud but protect its contents from outside hackers or snooping cloud administrators. Experiments of the proposed approach using an online selfie database show that it can achieve a good balance of privacy, utility, image quality, and file size.

## Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption; I.4.2 [**Image Processing and Computer Vision**]: Compression (Coding)

## Keywords

Multimedia encryption; Image security; Privacy

## 1. INTRODUCTION

As network connectivity continues to increase, cloud services have become perhaps the most common tools for storing and sharing photos. As early as 2010, Facebook was already receiving tens of millions of new images every day [2]. The idea of moving all one's personal files to the cloud is increasingly popular because it offers a number of benefits over earlier approaches, including low cost, worldwide availability, built-in redundancy against hardware or network failure, and nearly infinite storage capacity.


*IH&MMSec'15,* June 17 - 19, 2015, Portland, OR, USA

ACM 978-1-4503-3587-4/15/06...$15.00
DOI: http://dx.doi.org/10.1145/2756601.2756618

At the same time, the security and privacy of today's cloud services leaves much to be desired. Users must trust the operators of the cloud infrastructure to protect their data against malicious outsiders seeking to steal secrets, as in the 2014 compromise of Apple iCloud accounts [7], in which intimate photos were stolen from hundreds of victims and posted to the Internet. Users must also trust that the cloud service itself (or its employees) will not misuse its access for its own ends, as when Facebook used its members' photos to build a face recognition database [13].

A simple way to secure one's multimedia data in the cloud would be to encrypt all files before uploading them into the cloud, using a key unknown to the cloud service. But the use of encryption presents its own challenges. First, traditional file encryption tools like PGP or `openssl` produce opaque binary byte streams as output, whereas the cloud photo services expect to receive only valid image files. Format-compliant encryption schemes can encrypt the image data while maintaining valid file formats. Recent work on P3 [12] and Cryptagram [17] has proposed using similar schemes to protect images posted to social networking sites. However, even format-compliant encryption still breaks much of the useful functionality that made the cloud services popular in the first place, because the cloud service can no longer perform meaningful computations on the data.

One particularly painful limitation that encryption creates for photo management services is that the service can no longer create useful low-resolution "thumbnail" versions of the images it stores. Each user may store hundreds or thousands of photos, each of which may be a few megabytes in size. Normally, the cloud service generates reduced resolution versions of each photo so the user can preview her collection without downloading hundreds of megabytes every time she visits the page. Without the thumbnails, even a simple task like finding a certain photo becomes much more difficult; it does not help that cameras tend to create images with non-informative filenames like `DSC0293.JPG`.

In other applications, it may be useful to allow the cloud service to perform some analysis on the uploaded images. For example, some security cameras can upload a snapshot to the cloud whenever they detect motion. The cloud service then uses its greater processing power to perform more intensive image processing to look for the presence of a human in the image. It would be nice if we could give the cloud service just enough information to do its job, while still allowing a user to log in from her mobile device to retrieve and decrypt the full-resolution image that triggered an alarm.

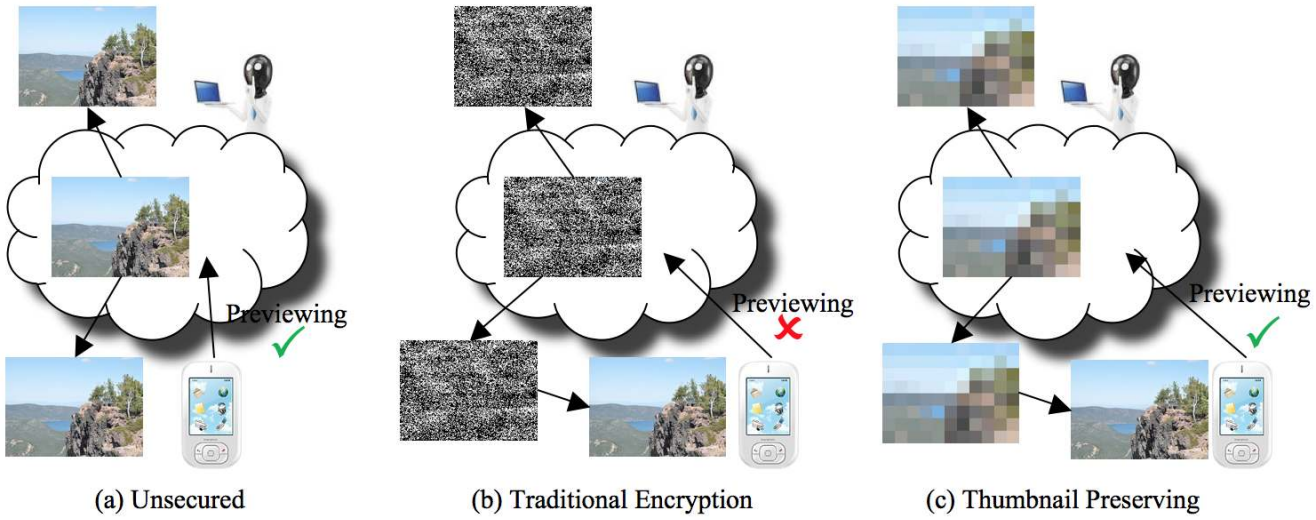



Figure 1: Different Approaches to Cloud Storage of Images. (a) Images stored in plaintext are available to both user and hacker. (b) Conventionally encrypted. Less usable for the client but secure from attackers. (c) Thumbnail-preserving. Attacker has access to the obscured version only. Users can preview thumbnails, then download at full resolution and decrypt locally.

In this paper, we propose an image encryption scheme for images that allows the cloud service to create accurate, reduced resolution thumbnails from the encrypted images, but prevents the extraction of more detailed information. Our goal is not to provide perfect security, but rather to protect privacy in images stored online, while still allowing convenient use of today's cloud services. To be useful in practice, any such scheme must:

1. Support the image formats used in popular cloud services like Flickr, Apple iCloud, Google Drive, and Facebook. In practice, this means support for JPEG.

2. Obscure the image data sufficiently to prevent the cloud service or an attacker from recovering fine details from the image, for example revealing the faces (or other body parts) of people in the photo.

3. Preserve coarse perceptual features of the image so that the user can effectively manage large photo collections online, using low-resolution versions of the images generated by the cloud service. A user who knows what the original photo looks like can recognize it trivially from the thumbnail.

4. Enable the user to reverse the encryption to recover a high-quality representation of the original image from the encrypted image.

5. Degrade gracefully when the cloud service compresses the obfuscated image to save on storage space. Facebook and other social networking services are known to apply aggressive compression to all uploaded photos.

Additionally, we would like to minimize the increase in file size incurred by the obfuscation, because many users access cloud services via mobile devices over slow or expensive wireless links. We also avoid reliance on trusted third parties as in P3 [12].

We experiment with and demonstrate the utility of the proposed approach with user-generated images from an online selfie dataset. Our results show that our approach can obscure interesting features in real user-generated images from the web, while maintaining acceptable image quality and file size. In the remainder of this paper, we provide some necessary background and related work. We then follow with a description of the proposed approach, some experimentation, and a discussion.

## 2. BACKGROUND AND RELATED WORK

### 2.1 Securing Data

Typically, to protect the confidentiality of sensitive data against a nosy third party, the best approach is to encrypt it using a rigorously vetted encryption scheme and a secret, hard-to-guess key. Many symmetric encryption schemes are built on block ciphers, for example AES-GCM is the block cipher AES [13] in Galois-counter mode [14].

There are a number of desirable properties for any encryption scheme. Most importantly, it should be computationally infeasible for anyone who does not have the key to recover the original data (the "plaintext") from the encrypted "ciphertext"; this is called *one-wayness*. Most current encryption schemes also aim to provide much stronger security, so that the adversary cannot learn anything at all—not even a single bit—about the plaintext from the ciphertext, even if he has access to a large number of (plaintext, ciphertext) pairs encrypted under the same key [15]; this is called *semantic security*.

In this work, our goal is a scheme that is one-way but not semantically secure. We want to prevent the adversary from learning the true values of our pixels, but we intentionally reveal coarser features, including the average color in each block of the image. We note that the approach presented here also reveals the set of pixel values in each block of the image. Because this technique reveals so much more

information than a typical encryption scheme, the reader may find it more intuitive to think of this as a reversible *obfuscation* method.

## 2.2 Multimedia Cryptography

Multimedia cryptography techniques typically focus on taking advantage of the fact that the image data is (or is going to be) highly compressed and more or less random in byte distribution. As such, lighter-weight encryption techniques can be employed to encrypt and decrypt the stream [8, 14, 16].

More closely related to our approach is work on partial image encryption [3, 20, 19, 15, 10]. Stütz and Uhl's transparent encryption technique for JPEG2000 [15] is thumbnail-preserving and uses a traditional encryption method (AES) to achieve very good security; unfortunately it only works with the relatively rare JPEG2000 format. Unterweger and Uhl's bitstream encryption for JPEG [19] is also similar to our approach in many ways. Because they operate on the compressed JPEG bitstream rather than on the pixels, they incur no increase in file size. At the same time, their technique does create some visual artifacts, and it is not guaranteed to preserve an accurate thumbnail.

# 3. THUMBNAIL PRESERVING ENCRYPTION

Our encryption scheme is essentially a special purpose, wide block, tweakable block cipher [9] for image data. Given an image of $M$x$N$ pixels, a secret key $k$, and a block size $B$, we divide the image spatially into blocks of neighboring pixels, ie $B$x$B$ squares. The goal is to encrypt each block such that the cipher image reveals the average pixel value in the block but nothing more. This allows an untrusted third party who does not posess the key to re-construct an accurate $M/B$ x $N/B$ pixel thumbnail, where each block in the cipher image corresponds to a single pixel in the thumbnail. This has the desired effect of preserving coarse features of the image—those larger than the block size—while obscuring fine details smaller than the block size.

Naturally, a user who has the key can decrypt the cipher image to recover a full resolution $M$x$N$ image. Our encryption scheme is carefully designed so that, even if the cipher image has been lossily compressed with JPEG, the decrypted image still maintains high fidelity to the original, and the image quality degrades gracefully as the level of JPEG quantization increases.

## 3.1 Key Derivation

Given an input image and a secret passphrase, we use the passphrase to derive a secret symmetric key $K$ using a password based key derivation function [11]. We also use some public information about the image, such as its filename, as a "salt" in the key derivation function; this makes it more difficult for an adversary to guess our key and ensures that we derive a different key for each unique filename.

## 3.2 Color Space Transformation

In the following sections, we describe techniques for operating on images as two-dimensional arrays of greyscale pixel values. We treat a color images as a collection of three such 2-d arrays or "planes", one for each dimension of the color space, for example R,G,B or Y,U,V.

JPEG uses the YUV color space, where pixel values are represented by a luminance value, Y, and two chrominance values, U and V. Because the human eye is more sensitive to changes in brightness than in color, JPEG typically stores the U and V channels at half the resolution of the Y channel. Given a JPEG image as input, we simply decompress it to extract the Y, U, and V planes. If we are given an RGB image as input, we first convert it to the YUV colorspace and subsample the chrominance channels at 4:1:1 just like a JPEG encoder would.

We encrypt each plane independently using a unique key. We use the password-derived key $K$ to compute a key for each color plane as a pseudorandom function (PRF) of the name of the plane. In our prototype implementation, we use HMAC-SHA1 as our PRF, so we compute the key for the Y plane as $K_Y = HMAC_K(\text{“Y”})$.

After encryption, we recombine the Y, U, and V planes to arrive at a raw uncompressed YUV representation of the cipher image. Finally, the cipher image YUV array (with its subsampled UV planes) is passed to the JPEG encoder, which then compresses the data in the regular way.

We take this approach to better align the encryption with the JPEG compression process. In early experiments, we investigated encrypting by operating on the full R,G,B or Y,U,V pixels as a single plane. While this produced acceptable results for lossless PNG images, with JPEG it led to significant artifacts and loss of color in the decrypted image. The reason was that dissimilar pixels in the original image were often placed next to each other in the permutation step. Then, when the JPEG encoder performed its subsampling on the U and V planes, much of the original color detail was lost.

## 3.3 Block Encryption

Then, if our block size is $B$, we divide the image into blocks of $B$x$B$ pixels, and we encrypt the pixels in each block by first permuting the order in which they appear. Because each block can be encrypted independently of all the others, this design allows for a very fast implementation in the future using parallel processing on a GPU.

We use the $(x, y)$ coordinates of the block as a "tweak" in our block encryption algorithm [9]. The tweak ensures that, even if the same block of pixel values appears at several different locations in the image, it will be encrypted differently at each location. For the block located at $(x, y)$ in an image plane $P$ having key $K_P$, we seed a cryptographically secure pseudorandom number generator with the seed $s = HMAC_{K_P}(x||y)$. (The $||$ operator denotes concatentation.) We then use the PRNG to permute the pixels in the block as follows.

We shuffle the locations of the pixels within the block, using our cryptographically secure PRNG to drive a random shuffle algorithm from Fisher and Yates [5]. We chose the Fisher-Yates shuffle because, when used with a good pseudorandom number generator, it makes all permutations of the input equally likely. As a result, the locations of the pixels in the cipher image give the adversary no information about their original locations. If he is to reconstruct the image, he must rely on the values of the pixels to infer which ones go where. With a sufficiently large block size, shuffling alone may actually give us some reasonable security (see Section 3.5). Figure 2 shows an example 320x320-pixel image encrypted with 32x32 blocks.

Figure 2: (left) Original and (right) 32x32 block-shuffled

### 3.4 Alternative: Recursive Block Encryption

As an alternative to the baseline approach above, we also propose the following scheme for use with large images, or in applications where efficiency and image quality are prioritized over high security.

As before, we first derive a secret symmetric key for the image and divide the image up into $B$x$B$-pixel blocks. Next, we divide each BxB block into a $(B/b)$x$(B/b)$ grid of smaller $b$x$b$-pixel sub-blocks. (Because JPEG uses 8x8-pixel blocks in its compression algorithm, we set $b = 8$.) We seed the PRNG as before and use the Fisher-Yates algorithm to permute the location of the sub-blocks within the block. Finally, we encrypt the pixels in each $b$x$b$ sub-block using the algorithm from Section 3.3.

Figure 3 shows an example of this technique, using 32x32-pixel blocks and 16x16-pixel sub-blocks. Part (A) shows an intermediate result after the sub-blocks have been rearranged, but before the pixels have been encrypted. Part (B) shows the final encrypted image. Note that we chose a very large sub-block size for this image to illustrate the technique. In practice, the sub-blocks should be made much smaller to achieve good JPEG compression.

Figure 3: (left) After shuffling 16x16 sub-blocks within 32x32 blocks and (right) after shuffling pixels in sub-blocks

### 3.5 Security

It's not clear what is the best way to evaluate the security of our scheme. As a first approximation of the difficulty of reconstructing the image, we note that the number of possible permutations increases as the factorial of the number of pixels in the block. So for example, a 32x32 block contains 1024 pixels; there are therefore 1024! $\approx 4 \times 10^{2567}$ ways to re-order its pixels. In comparison, there are only $2^{256} \approx 10^{77}$ possible AES-256 keys. This is the general approach used by Unterweger and Uhl for evaluating the security of their partial image encryption scheme in [19]. However, this approach is problematic because it focuses only on brute force attacks, ignoring the possibility of cryptanalytic or statistical attacks.

The most promising attack method may be to start with the low-resolution thumbnail and use super resolution techniques [6, 21] to increase its effective resolution. Super resolution can commonly be used to double the resolution of an image; if our attacker can do this, then he can replace each pixel in our thumbnail with a 2x2 grid of pixels. Then, he can go back to the ciphertext image and examine the pixels in the block corresponding to the given pixel in the thumbnail. He can then attempt to use the 2x2 grid of superresolution pixel values to place each pixel in the block back into the quadrant where it started. If he is successful, he can repeat this process several times to recover a much higher resolution version of the image.

Fortunately, it appears that this attack may be computationally quite hard. When the attacker attempts to put the pixels back into the correct quadrant of the block, for an $N$x$N$ block he must find $N^2/4$ pixels whose average value is the value of the super resolution pixel for that quadrant. This is an instance of the $d$-SUM problem, a generalization of the more well-known 3SUM problem. The best known lower bound [4] for $d$-SUM given $n$ numbers is approximately $O(n^{d/2})$. When $d = n/4$ as we have here, $d$-SUM is approximately $O(n^{n/8})$.

## 4. EXPERIMENTATION

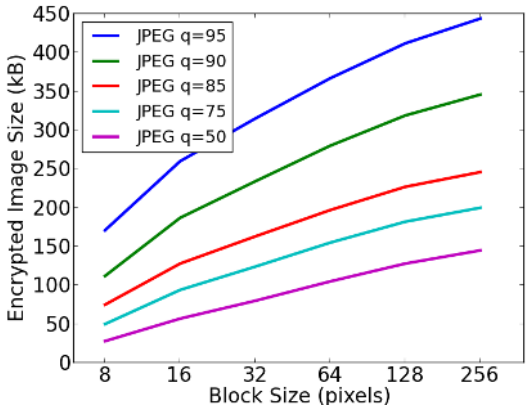
Knowing that our techniques tend to create cipher images with more high frequency components than in natural images, we are interested in the impact of our encryption on the encrypted file size and the quality of decrypted images. In the remainder of this section, we describe the image dataset used and the results of our initial empirical evaluation.

### 4.1 Image Dataset

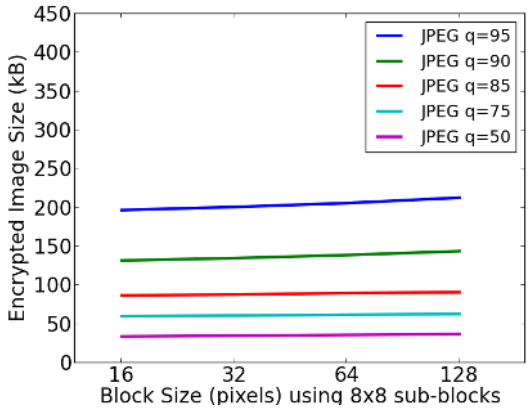
To study the performance impact of our techniques, we require an image dataset. Because the block size of permutation is currently fixed across the entire image, we sought an image set with roughly the same image content per pixel density. This allows us to look at the effect of encryption on images in a more meaningful way. For this reason, we chose an online database of "selfies" collected from Instagram by the Selfiecity project [18]. With selfies, the distance to camera and the object of the image (the person in this case) will be roughly constant. From this database, we randomly selected 50 images for this initial study. The selfies are 640x640 pixel JPEGs; 48 of the 50 are color images and 2 are black and white. Most were taken with mobile phones.

### 4.2 File Size and Image Quality

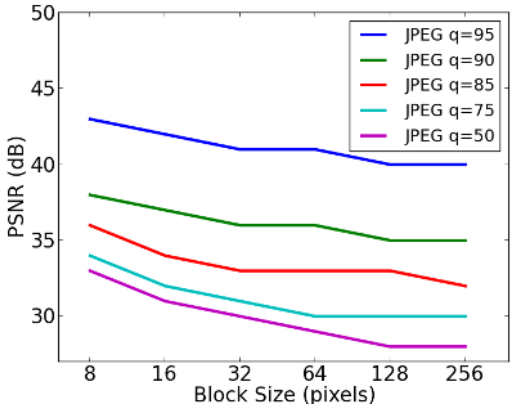
We encrypted each of the 50 images using the two techniques described in Section 3, with block sizes ranging from 8x8 up to 256x256. Each encrypted image was then JPEG compressed using the ImageMagick `convert` utility on Linux, with quality parameter ranging between 10 and 95. Finally, each cipher image was decrypted to recover the pixels of the original image, albeit with some loss due to JPEG.



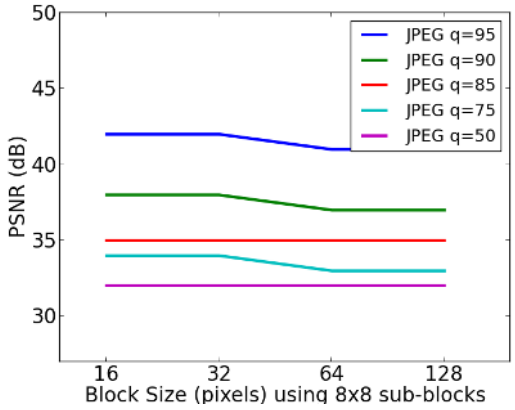

(a) File size for block-permuted images




(b) File size for recursively permuted images




(c) Quality of decrypted images with block-based permutations




(d) Quality of decrypted images with recursive permutations

Figure 4: Experimental Results

#### 4.2.1 *Size of the Encrypted Images*

Figure 4a and Figure 4b show the average size of images encrypted using the simple block encryption and the recursive encryption technique, respectively. The average file size of the original plaintext images was 56 kB.

With the simple block-based approach, as the block size increases, it becomes more and more likely that the permutation destroys spatial locality in the image by placing pixels from distant parts of the image next to each other. This creates high-frequency noise in the ciphertext image, which causes a large increase in file size, particularly for less-aggressive JPEG settings.

#### 4.2.2 *Quality of the Decrypted Images*

Figure 5 shows two example images decrypted after the ciphertext was JPEG compressed with q=85. Figure 4c and Figure 4d show how the average peak signal-to-noise ratio (PSNR) for the decrypted images changes as the block size increases. When we use the simple block-based encryption scheme, doubling the block size tends to decrease the PSNR by about 1-2 dB. With the recursive block encryption scheme, the PSNR is relatively unaffected by changes in block size. This is because JPEG operates on 8x8 blocks, and the pixels within each 8x8 sub-block in the encrypted image are still roughly similar to one another even after being permuted and perturbed.

Figure 5: Decrypted images (left) 16x16 blocks and (right) 64x64 blocks

## 5. CONCLUSIONS AND FUTURE WORK

We have presented a new image encryption algorithm for protecting JPEG image data stored in the cloud. It provides tunable privacy through an adjustable block size and allows the untrusted cloud service to reconstruct an accurate low-resolution thumbnail of the encrypted photo. Experiments

with a data set of real selfie images from the web show that, by selecting an appropriate block size, we can achieve a good balance between convenience, privacy, image quality, and file size.

In future work, to make the adversary's job even more difficult, we will apply additional transformations to modify the pixel values after shuffling. The key property for these transformations is that they must (approximately) preserve the average value of the pixels in the block—otherwise the encryption will no longer be thumbnail-preserving. To achieve reasonable compression and high quality on natural images, we also approximately preserve the standard deviation of the pixels in each block. The simplest way to perturb pixel values without shifting the mean is to add some small noise to each pixel, where the noise is drawn from a statistical distribution with mean zero and relatively small standard deviation. For example, in additive white Gaussian noise (AWGN) with shape parameter $\sigma$, we set each pixel $p_i$ equal to $p_i = p_i + n_i$, where $n_i \sim N(0, \sigma)$.

We will also investigate another approach, which also preserves the sample standard deviation. We first compute the mean value $m$ of the pixels in the block and the difference between each pixel $p_i$ and the mean, $d_i = p_i - m$. Then we set the new value for each pixel value to be:

$$p_i = \begin{cases} m + d_i, & \text{with probability } 0.5 \\ m - d_i, & \text{with probability } 0.5 \end{cases}$$

Each pixel has a 50% chance of being "reflected" across the sample mean and a 50% chance of staying the same. In either case, the mean is unchanged, and each value in the sample stays the same distance away from the mean, leaving the sample standard deviation unchanged. This way, blocks that are very uniform or low-frequency, e.g. a patch of clear blue sky, can still be compressed extremely efficiently, while blocks containing more interesting features are more strongly perturbed.

Finally, we will also develop a better understanding of the relationship between our encryption parameters (eg. block size) and the cloud service's ability to perform various analyses on the uploaded images. For example, work on using super resolution to improve face recognition [1] indicates that relatively large blocks may be required to prevent the recognition of people in our photos.

# 6. REFERENCES


[1] Simon Baker and Takeo Kanade. Limits on super-resolution and how to break them. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1167–1183, September 2002.

[2] Doug Beaver, Sanjeev Kumar, Harry C Li, Jason Sobel, Peter Vajgel, et al. Finding a needle in haystack: Facebook's photo storage. In *OSDI*, volume 10, pages 1–8, 2010.

[3] H. Cheng and Xiaobo Li. Partial encryption of compressed images and videos. *Signal Processing, IEEE Transactions on*, 48(8):2439–2451, Aug 2000.

[4] Jeff Erickson. Lower bounds for linear satisfiability problems. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '95, pages 388–395, Philadelphia, PA, USA, 1995.

[5] Ronald Aylmer Fisher and Frank Yates. *Statistical tables for biological, agricultural and medical research*. 3rd edition, 1949.

[6] William T Freeman, Egon C Pasztor, and Owen T Carmichael. Learning low-level vision. *International Journal of Computer Vision*, 40(1):25–47, 2000.

[7] Andy Greenberg. The police tool that pervs use to steal nude pics from Apple's iCloud. *Wired*, September 2014.

[8] Lala Krikor, Sami Baba, Thawar Arif, and Zyad Shaaban. Image encryption using DCT and stream cipher. *European Journal of Scientific Research*, 32(1):47–57, 2009.

[9] Moses Liskov, Ronald L Rivest, and David Wagner. Tweakable block ciphers. In *Advances in Cryptology—CRYPTO 2002*, pages 31–46. Springer, 2002.

[10] Benoit M Macq and Jean-Jacques Quisquater. Cryptology for digital TV broadcasting. *Proceedings of the IEEE*, 83(6), 1995.

[11] Niels Provos and David Mazieres. A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track*, pages 81–91, 1999.

[12] Moo-Ryong Ra, Ramesh Govindan, and Antonio Ortega. P3: Toward privacy-preserving photo sharing. In *NSDI*, pages 515–528, 2013.

[13] Somini Sengupta and Kevin J. O'Brien. Facebook can ID faces, but using them grows tricky. *The New York Times*, September 2012.

[14] Changgui Shi and Bharat Bhargava. An efficient mpeg video encryption algorithm. In *Reliable Distributed Systems, 1998. Proceedings. Seventeenth IEEE Symposium on*, pages 381–386. IEEE, 1998.

[15] Thomas Stütz and Andreas Uhl. On efficient transparent JPEG2000 encryption. In *Proceedings of the 9th ACM Workshop on Multimedia & Security*, pages 97–108. ACM, 2007.

[16] Lei Tang. Methods for encrypting and decrypting mpeg video data efficiently. In *Proceedings of the Fourth ACM International Conference on Multimedia*, MULTIMEDIA '96, pages 219–229, 1996.

[17] Matt Tierney, Ian Spiro, Christoph Bregler, and Lakshminarayanan Subramanian. Cryptagram: Photo privacy for online social media. In *Proceedings of the First ACM Conference on Online Social Networks*, COSN '13, pages 75–88. ACM, 2013.

[18] Alise Tifentale and Lev Manovich. *Selfiecity: Exploring Photography and Self-Fashioning in Social Media*. Palgrave Macmillan, Forthcoming.

[19] Andreas Unterweger and Andreas Uhl. Length-preserving bit-stream-based jpeg encryption. In *Proceedings of the 14th ACM Workshop on Multimedia and security*, pages 85–90. ACM, 2012.

[20] Marc Van Droogenbroeck. Partial encryption of images for real-time applications. *Fourth IEEE Benelux Signal Processing*, pages 11–15, 2004.

[21] Qiang Wang, Xiaoou Tang, and Harry Shum. Patch based blind image super resolution. In *Tenth IEEE International Conference on Computer Vision*, volume 1, pages 709–716. IEEE, 2005.