P^3: Privacy-Preserving Photo Sharing

Moo-Ryong Ra Ramesh Govindan Antonio Ortega
University of Southern California

Abstract

With increasing penetration of mobile devices, photo sharing services are experiencing a resurgence. Aside from providing storage, photo sharing services enable bandwidth-efficient downloads to mobile devices by performing server-side image transformations (resizing, cropping). On the flip side, photo sharing services have raised privacy concerns such as leakage of photos to unauthorized viewers and the use of face recognition technologies by providers. To address these concerns, we propose a privacy-preserving photo encoding algorithm that extracts and encrypts a small, but significant, component of the photo, while preserving the remainder in a standards-compatible form. These two components can be separately stored. This technique significantly reduces the signal-to-noise ratio and the accuracy of automated detection and recognition on the publicly available photo, while preserving the ability of the provider to perform server-side transformations to conserve download bandwidth usage. Our prototype privacy-preserving photo sharing system, P^3, works with Facebook and Flickr, and can be extended to other services as well. P^3 requires no changes to existing services or mobile application software, and adds minimal photo storage overhead.

1 Introduction

With the advent of mobile devices with high-resolution on-board cameras, photo sharing has become popular again. Users can share photos either through photo sharing services like Flickr or Picasa, or popular social networking services like Facebook or Google+. These *photo sharing service providers* (PSPs) now have a large user base, to the point where PSP photo storage subsystems have motivated interesting systems research [9].

However, this development has generated privacy concerns (Section 2) Private photos have been leaked from a prominent photo sharing site [12]. Furthermore, widespread concerns have been raised about the application of face recognition technologies in Facebook [2]. Despite these privacy threats, it is not clear that the usage of photo sharing services will diminish in the near future. This is because photo sharing services provide several useful functions that, together, make for a seamless photo browsing experience. In addition to providing photo storage, PSPs also perform several server-side image transformations (like cropping, resizing and color space conversions) designed to improve user perceived latency of photo downloads and, incidentally, bandwidth usage (an important consideration when browsing photos on a mobile device).

In this paper, we explore the design of a privacy-preserving photo sharing algorithm (and an associated system) that *ensures photo privacy without sacrificing the latency, storage, and bandwidth benefits provided by PSPs*. This paper makes two novel contributions that, to our knowledge, have not been reported in the literature (Section 6).

- The design of the P^3 algorithm (Section 3), which prevents leaked photos from leaking *information*, and reduces the efficacy of automated processing (e.g., face detection, feature extraction) on photos, while still permitting a PSP to apply image transformations. It does this by splitting a photo into a public part, which contains most of the *volume* (in bytes) of the original, and a secret part which contains most of the original's *information*.
- The design of the P^3 system (Section 4), which requires no modification to the PSP infrastructure or software, and no modification to existing browsers or applications. P^3 uses interposition to transparently encrypt images when they are uploaded from clients, and transparently decrypt and reconstruct images on the recipient side.

Evaluations (Section 5) on two commonly used image data sets, as well as micro-benchmarks on an implementation of P^3, reveal several interesting results. Across these data sets, there exists a "sweet spot" in the parameter space that provides good privacy while at the same time preserving the storage, latency, and bandwidth benefits offered by PSPs. At this sweet spot, the public part of the image has low PSNR and algorithms like edge detection, face detection, and SIFT feature extraction are completely ineffective; *no* faces can be detected from the public part, *no* correct features can be extracted, and a very small fraction of pixels defining edges are correctly estimated. P^3 image encryption and decryption are fast, and it is able to reconstruct images accurately even when the PSP's image transformations are not publicly known.

P^3 is proof-of-concept of easily deployable privacy preserving photo storage. Adoption of this technology

will be dictated by economic incentives: for example, PSPs can offer privacy preserving photo storage as a premium service offered to privacy-conscious customers.

2 Background and Motivation

The focus of this paper is on PSPs like Facebook, Picasa, Flickr, and Imgur, who offer either direct *photo sharing* (e.g., Flickr, Picasa) between users or have integrated photo sharing into a social network platform (e.g., Facebook). In this section, we describe some background before motivating privacy-preserving photo sharing.

2.1 Image Standards, Compression and Scalability

Over the last two decades, several standard image formats have been developed that enable interoperability between producers and consumers of images. Perhaps not surprisingly, most of the existing PSPs like Facebook, Flickr, Picasa Web, and many websites [5, 6, 36] primarily use the most prevalent of these standards, the JPEG (Joint Photographic Experts Group) standard. In this paper, we focus on methods to preserve the privacy of JPEG images; supporting other standards such as GIF and PNG (usually used to represent computer-generated images like logos etc.) are left to future work.

Beyond standardizing an image file format, JPEG performs lossy compression of images. A JPEG encoder consists of the following sequence of steps:

Color Space Conversion and Downsampling In this step, the raw RGB or color filter array (CFA) RGB image captured by a digital camera is mapped to a YUV color space. Typically, the two chrominance channels (U and V) are represented at lower resolution than the luminance channel (Y) reducing the amount of pixel data to be encoded without significant impact on perceptual quality.

DCT Transformation In the next step, the image is divided into an array of blocks, each with 8×8 pixels, and the Discrete Cosine Transform (DCT) is applied to each block.

Quantization The result of the DCT step is a set of DCT coefficients. In this step, these coefficients are quantized; this is the only step in the processing chain where information is lost. For typical natural images, information tends to be concentrated in the lower frequency coefficients (which on average have larger magnitude than higher frequency ones). For this reason, JPEG applies different quantization steps to different frequencies. The degree of quantization is user-controlled and can be varied in order to achieve the desired trade-off between quality of the reconstructed image and compression rate. We note that in practice, images shared through PSPs tend to be uploaded with high quality (and high rate) settings.

Entropy Coding In the final step, redundancy in the quantized coefficients is removed using variable length encoding of non-zero quantized coefficients and of runs of zeros in between non-zero coefficients.

Beyond storing JPEG images, PSPs perform several kinds of transformations on images for various reasons. First, when a photo is uploaded, PSPs statically resize the image to several fixed resolutions. For example, Facebook transforms an uploaded photo into a thumbnail, a "small" image (320x320) and a "big" image (720x720). These transformations have multiple uses: they can reduce storage[1], improve photo access latency for the common case when users download either the big or the small image, and also reduce bandwidth usage (an important consideration for mobile clients). In addition, PSPs perform *dynamic* (i.e., when the image is accessed) server-side transformations; they may resize the image to fit screen resolution, and may also *crop* the image to match the view selected by the user. (We have verified, by analyzing the Facebook protocol, that it supports both of these dynamic operations). These dynamic server-side transformations enable low latency access to photos and reduce bandwidth usage. Finally, in order to reduce user-perceived latency further, Facebook also employs a special mode in the JPEG standard, called *progressive* mode. For photos stored in this mode, the server delivers the coefficients in increasing order (hence "progressive") so that the clients can start rendering the photo on the screen as soon as the first few coefficients are received, without having to receive all coefficients.

In general, these transformations *scale* images in one fashion or another, and are collectively called image scalability transformations. Image scalability is crucial for PSPs, since it helps them optimize several aspects of their operation: it reduces photo storage, which can be a significant issue for a popular social network platform [9]; it can reduce user-perceived latency, and reduce bandwidth usage, hence improving user satisfaction.

2.2 Threat Model, Goals and Assumptions

In this paper, we focus on two specific threats to privacy that result from uploading user images to PSPs. The first threat is unauthorized access to photos. A concrete instance of this threat is the practice of *fusking*, which attempts to reverse-engineer PSP photo URLs in order to access stored photos, bypassing PSP access controls. Fusking has been applied to at least one PSP (Photobucket), resulting in significant privacy leakage [12]. The second threat is posed by automatic face recognition technologies, by which PSPs may be able to in-

[1] We do not know if Facebook preserves the original image, but high-end mobile devices can generate photos with 4000x4000 resolution and resizing these images to a few small fixed resolutions can save space

fer social relationships not explicitly specified by users. Facebook's deployment of face recognition technology has raised significant privacy concerns in many countries (e.g., [2]).

We are now ready to state the goal of this paper, which is *to design and implement a system that enables users to ensure the privacy of their photos (with respect to the two threats listed above), while still benefiting from the image scalability optimizations provided by the PSP.*

Implicit in this statement are several constraints, which make the problem significantly challenging. The resulting system must not require any software changes at the PSP, since this is a significant barrier to deployment; an important implication of this constraint is that the image stored on the PSP must be JPEG-compliant. For a similar reason, the resulting system must also be transparent to the client. Finally, our solution must not significantly increase storage requirements at the PSP since, for large PSPs, photo storage is a concern.

We make the following assumptions about trust in the various components of the system. We assume that all local software/hardware components on clients (mobile devices, laptops etc.) are completely trustworthy, including the operating system, applications and sensors. We assume that PSPs are completely untrusted and may either by commission or omission, breach privacy in the two ways described above. Furthermore, we assume eavesdroppers may attempt to snoop on the communication between PSP and a client.

In this paper, we describe the design of P^3, an algorithm and an associated system that is designed to counter the privacy threats posed by photo storage on PSPs, to respect the constraints and the trust assumptions described above.

3 P^3: The Algorithm

In this section, we describe the P^3 algorithm for ensuring privacy of photos uploaded to PSPs. In the next section, we describe the design and implementation of a complete system for privacy-preserving photo sharing.

3.1 Overview

One possibility for preserving the privacy of photos is end-to-end encryption. *Senders*[2] may encrypt photos before uploading, and *recipients* use a shared secret key to decrypt photos on their devices. This approach cannot provide image scalability, since the photo representation is non-JPEG compliant and opaque to the PSP so it cannot perform transformations like resizing and cropping. Indeed, PSPs like Facebook reject attempts to upload fully-encrypted images.

[2]For lack of a better term, we use "sender" to denote the user of a PSP who uploads images to the PSP.

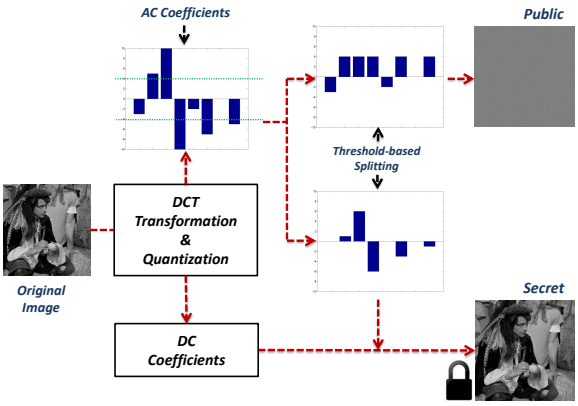


Figure 1: Privacy-Preserving Image Encoding Algorithm

A second, more subtle approach, is to leverage the JPEG image compression pipeline. Current image compression standards use a well-known *DCT dictionary* when computing the DCT coefficients. An alternative approach is to use a *private* dictionary [8], known only to the sender and the authorized recipients, and upload the resulting coefficients to the PSP. Using these coefficients, it may be possible for PSPs to perform image scaling transformations. However, as currently defined, these coefficients result in a non-JPEG compliant bit-stream, so PSP-side code changes would be required in order to make this approach work.

A third strawman approach might selectively hide faces by performing face detection on an image before uploading. This would leave a JPEG-compliant image in the clear, with the hidden faces stored in a separate encrypted part. At the recipient, the image can be reconstructed by combining the two parts. However, this approach does not address our privacy goals completely: if an image is leaked from the PSP, attackers can still obtain significant information from the non-obscured parts (e.g., torsos, other objects in the background etc.).

Our approach on privacy-preserving photo sharing uses a *selective encryption* like this, but has a different design. In this approach, called P^3, a photo is divided into two parts, a *public* part and a *secret* part. The public part is exposed to the PSP, while the secret part is encrypted and shared between the sender and the recipients (in a manner discussed later). Given the constraints discussed in Section 2, the public and secret parts must satisfy the following requirements:

- It must be possible to represent the public part as a JPEG-compliant image. This will allow PSPs to perform image scaling.
- However, intuitively, most of the "important" *information* in the photo must be in the secret part. This would prevent attackers from making sense of the public part of the photos even if they were able to access these photos.

It would also prevent PSPs from successfully applying recognition algorithms.

- Most of the *volume* (in bytes) of the image must reside in the public part. This would permit PSP server-side image scaling to have the bandwidth and latency benefits discussed above.
- The combined size of the public and secret parts of the image must not significantly exceed the size of the original image, as discussed above.

To our knowledge, this combination of requirements has not been considered in the image compression literature (Section 6).

In the following subsections, we discuss the P^3 algorithm, which satisfies these requirements. The algorithm has two components: a sender side encryption algorithm, and a recipient-side decryption algorithm.

3.2 P^3 Sender-Side Encryption

JPEG compression relies on the *sparsity* in the DCT domain of typical natural images: a few (large magnitude) coefficients provide most of the information needed to reconstruct the pixels. Moreover, as the quality of cameras on mobile devices increases, images uploaded to PSPs are typically encoded at high quality. P^3 leverages both the sparsity and the high quality of these images. First, because of sparsity, most information is contained in a few coefficients, so it is sufficient to degrade a few such coefficients, in order to achieve significant reductions in quality of the public image. Second, because the quality is high, quantization of each coefficient is very fine and the least significant bits of each coefficient represent very small incremental gains in reconstruction quality. P^3's encryption algorithm encode the most significant bits of (the few) significant coefficients in the secret part, leaving everything else (less important coefficients, and least significant bits of more important coefficients) in the public part. We concretize this intuition in the following design for P^3 sender side encryption.

The selective encryption algorithm is, conceptually, inserted into the JPEG compression pipeline after the quantization step. At this point, the image has been converted into frequency-domain quantized DCT coefficients. While there are many possible approaches to extracting the most significant information, P^3 uses a relatively simple approach. First, it extracts the DC coefficients from the image into the secret part, replacing them with zero values in the public part. The DC coefficients represent the average value of each 8x8 pixel block of the image; these coefficients usually contain enough information to represent thumbnail versions of the original image with enough visual clarity.

Second, P^3 uses a threshold-based splitting algorithm in which each non-DC coefficient $y(i)$ whose value is above a threshold T is processed as follows:

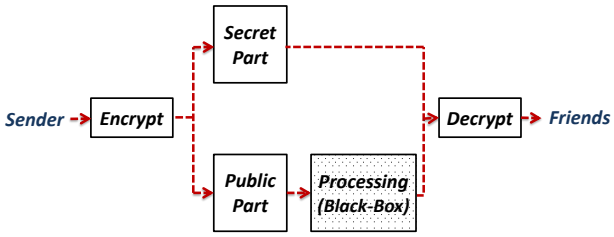


Figure 2: P^3 Overall Processing Chain

- If $\mathsf{y(i)} \leq T$, then the coefficient is represented in the public part as is, and in the secret part with a zero.
- If $\mathsf{y(i)} > T$, the coefficient is replaced in the public part with T, and the secret part contains the magnitude of the difference as well as the sign.

Intuitively, this approach clips off the significant coefficients at T. T is a tunable parameter that represents the trade-off between storage/bandwidth overhead and privacy; a smaller T extracts more signal content into the secret part, but can potentially incur greater storage overhead. We explore this trade-off empirically in Section 5. Notice that both the public and secret parts are JPEG-compliant images, and, after they have been generated, can be subjected to entropy coding.

Once the public and secret parts are prepared, the secret part is encrypted and, conceptually, both parts can be uploaded to the PSP (in practice, our system is designed differently, for reasons discussed in Section 4)). We also defer a discussion of the encryption scheme method to Section 4.

3.3 P^3 Recipient-side Decryption and Reconstruction

While the sender-side encryption algorithm is conceptually simple, the operations on the recipient-side are somewhat trickier. At the recipient, P^3 must decrypt the secret part and reconstruct the original image by combining the public and secret parts. P^3's selective encryption is *reversible*, in the sense that, the public and secret parts can be recombined to reconstruct the original image. This is straightforward when the public image is stored unchanged, but requires a more detailed analysis in the case when the PSP performs some processing on the public image (e.g., resizing, cropping, etc) in order to reduce storage, latency or bandwidth usage.

In order to derive how to reconstruct an image when the public image has been processed, we start by expressing the reconstruction for the unprocessed case as a series of linear operations.

Let the threshold for our splitting algorithm be denoted T. Let $\mathbf{y}$ be a block of DCT coefficients corresponding to a 8×8 pixel block in the original image. Denote $\mathbf{x_p}$ and

$\mathbf{x_s}$ the corresponding DCT coefficient values assigned to the public and secret images, respectively, for that same block[3]. For example, if one of those coefficients is such that $abs(y(i)) > T$, we will have that $x_p(i) = T$ and $x_s(i) = sign(y(i))(abs(y(i)) - T)$. Since in our algorithm the sign information is encoded either in the public or in the secret part, depending on the coefficient magnitude, it is useful to explicitly consider sign information here. To do so we write $\mathbf{x_p} = \mathbf{S_p} \cdot \mathbf{a_p}$, and $\mathbf{x_s} = \mathbf{S_s} \cdot \mathbf{a_s}$, where $\mathbf{S_p}$ and $\mathbf{S_s}$ are diagonal matrices with sign information, i.e., $\mathbf{S_p} = diag(sign(\mathbf{x_p})), \mathbf{S_s} = diag(sign(\mathbf{x_s}))$. Now let $\mathbf{w}[i] = T$ if $\mathbf{S_s}[i] \neq 0$, where i is a pixel index, so $\mathbf{w}$ marks the positions of the above-threshold coefficients.

The key observation is that $\mathbf{x_p}$ and $\mathbf{x_s}$ cannot be directly added to recover $\mathbf{y}$ because the sign of a coefficient above threshold is encoded correctly *only* in the secret image, i.e., even though the public image conveys sign information for that coefficient, it might not be correct. As an example, let $y(i) < -T$, then we will have that $x_p(i) = T$ and $x_s(i) = -(abs(y(i)) - T)$, thus $x_s(i) + x_p(i) \neq y(i)$. For coefficients below threshold $y(i)$ can be recovered trivially since $x_s(i) = 0$ and $x_p(i) = y(i)$. Note that incorrect sign in the public image occurs only for coefficients $y(i)$ above threshold, and by definition, for all those coefficients the public value is $x_p(i) = T$. Note also that removing these signs increases significantly the distortion in the public images and makes it more challenging for an attacker to approximate the original image based on only the public one.

In summary, the reconstruction can be written as a series of linear operations:

$$\mathbf{y} \quad = \quad \mathbf{S_p} \cdot \mathbf{a_p} + \mathbf{S_s} \cdot \mathbf{a_s} + \left(\mathbf{S_s} - \mathbf{S_s}^2\right) \cdot \mathbf{w} \qquad (1)$$

where the first two terms correspond to directly adding the correspondig blocks from the public and secret images, while the third term is a correction factor to account for the incorrect sign of some coefficients in the public image. This correction factor is based on the sign of the coefficients in the secret image and distinguishes three cases. If $x_s(i) = 0$ or $x_s(i) > 0$ then $y(i) = x_s(i) + x_p(i)$ (no correction), while if $x_s(i) < 0$ we have

$$y(i) = x_s(i) + x_p(i) - 2T = x_s(i) + T - 2T = x_s(i) - T.$$

Note that the operations can be very easily represented and implemented with if/then/else conditions, but the algebraic representation of (1) will be needed to determine how to operate when the public image has been processed. In particular, from (1), and given that the DCT is a linear operator, it becomes apparent that it would be possible to reconstruct the images in the pixel domain. That is, we could convert $\mathbf{S_p} \cdot \mathbf{a_p}$, $\mathbf{S_s} \cdot \mathbf{a_s}$ and $\left(\mathbf{S_s} - \mathbf{S_s}^2\right) \cdot \mathbf{w}$ into the pixel domain and simply add these three images pixel by pixel. Further note that the third image, the correction factor, does not depend on the public image and can be completely derived from the secret image.

We now consider the case where the PSP applies a linear operator $\mathbf{A}$ to the public part. Many interesting image transformations such as filtering, cropping, scaling (resizing), and overlapping can be expressed by linear operators. Thus, when the public part is requested from the PSP, $\mathbf{A} \cdot \mathbf{S_p} \cdot \mathbf{a_p}$ will be received. Then the goal is for the recipient to reconstruct $\mathbf{A} \cdot \mathbf{y}$ given the processed public image $\mathbf{A} \cdot \mathbf{S_p} \cdot \mathbf{a_p}$ and the unprocessed secret information. Based on the reconstruction formula of (1), and the linearity of $\mathbf{A}$, it is clear that the desired reconstruction can be obtained as follows

$$\mathbf{A} \cdot \mathbf{y} = \mathbf{A} \cdot \mathbf{S_p} \cdot \mathbf{a_p} + \mathbf{A} \cdot \mathbf{S_s} \cdot \mathbf{a_s} + \mathbf{A} \cdot \left(\mathbf{S_s} - \mathbf{S_s}^2\right) \cdot \mathbf{w} \qquad (2)$$

Moreover, since the DCT transform is also linear, these operations can be applied directly in the pixel domain, without a need to find a transform domain representation. As an example, if cropping is involved, it would be enough to crop the private image and the image obtained by applying an inverse DCT to $\left(\mathbf{S_s} - \mathbf{S_s}^2\right) \cdot \mathbf{w}$.

3.4 P^3 Properties

Privacy Properties. By encrypting significant signal information, P^3 can preserve the privacy of images by distorting them and by foiling detection and recognition algorithms (Section 5). Given only the public part, the attacker can guess the threshold T by assuming it to be the most frequent non-zero value. If this guess is correct, the attacker knows the positions of the significant coefficients, but not the range of values of these coefficients. Crucially, the sign of the coefficient is also not known. Sign information tends to be "random" in that positive and negative coefficients are almost equally likely and there is very limited correlation between signs of different coefficients, both within a block and across blocks. It can be shown that if the sign is unknown, and no prior information exists that would bias our guess, it is actually best, in terms of mean-square error (MSE), to replace the coefficient with unknown sign in the public image by 0.

Finally, we observe that *proving* the privacy properties of our approach is challenging. If the public part is leaked from the PSP, proving that no human can extract visual information from the public part would require having an accurate understanding of visual perception. Instead, we rely in our evaluation (Section 5) on metrics commonly used in the signal processing community. We note that the prevailing methodology in the signal processing community for evaluating the efficacy of image and video privacy is empirical subjective evaluation using user studies, or objective evaluation using met-

[3]For the purpose of this discussion, we represent these blocks as size 64 column vectors

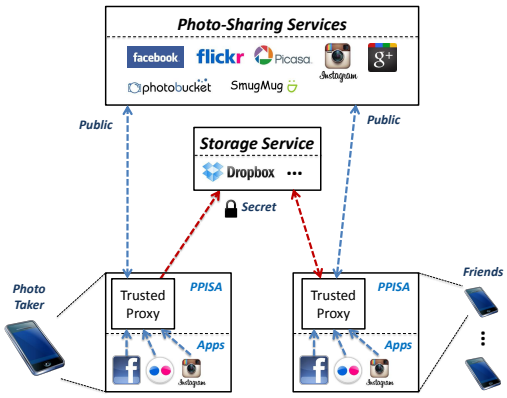


Figure 3: P^3 System Architecture

rics [14]. In Section 5, we resort to an objective metrics-based evaluation, showing the performance of P^3 on several image corpora.

Other Properties. P^3 satisfies the other requirements we have discussed above. It leaves, in the clear, a JPEG-compliant image (the public part), on which the PSP can perform transformations to save storage and bandwidth. The threshold T permits trading off increased storage for increased privacy; for images whose signal content is in the DC component and a few highly valued coefficients, the secret part can encode most of this content, while the public part contains a significant fraction of the volume of the image in bytes. As we show in our evaluation later, most images are sparse and satisfy this property. Finally, our approach of encoding the large coefficients decreases the entropy both in the public and secret parts, resulting in better compressibility and only slightly increased overhead overall relative to the unencrypted compressed image.

However, the P^3 algorithm has an interesting consequence: since the secret part cannot be scaled (because, in general, the transformations that a PSP performs cannot be known a priori) and must be downloaded in its entirety, the bandwidth savings from P^3 will always be lower than downloading a resized original image. The size of the secret part is determined by T: higher values of T result in smaller secret parts, but provide less privacy, a trade-off we quantify in Section 5.

4 P^3: System Design

In this section, we describe the design of a system for privacy preserving photo sharing system. This system, also called P^3, has two desirable properties described earlier. First, it requires no software modifications at the PSP. Second, it requires no modifications to client-side browsers or image management applications, and only requires a small footprint software installation on clients. These properties permit immediate deployment of privacy-preserving photo sharing.

4.1 P^3 Architecture and Operation

Before designing our system, we explored the protocols used by PSPs for uploading and downloading photos. Most PSPs use HTTP or HTTPS to upload messages; we have verified this for Facebook, Picasa Web, Flickr, PhotoBucket, Smugmug, and Imageshack. This suggests a relatively simple interposition architecture, depicted in Figure 3. In this architecture, browsers and applications are configured to use a local HTTP/HTTPS *proxy* and all accesses to PSPs go through the proxy. The proxy manipulates the data stream to achieve privacy preserving photo storage, in a manner that is transparent both to the PSP and the client. In the following paragraphs, we describe the actions performed by the proxy at the sender side and at one or more recipients.

Sender-side Operation. When a sender transmits the photo taken by built-in camera, the local proxy acts as a middlebox and splits the uploaded image into a public and a secret part (as discussed in Section 3). Since the proxy resides on the client device (and hence is within the trust boundary per our assumptions, Section 2), it is reasonable to assume that the proxy can decrypt and encrypt HTTPS sessions in order to encrypt the photo.

We have not yet discussed how photos are encrypted; in our current implementation, we assume the existence of a symmetric shared key between a sender and one or more recipients. This symmetric key is assumed to be distributed out of band.

Ideally, it would have been preferable to store both the public and the secret parts on the PSP. Since the public part is a JPEG-compliant image, we explored methods to embed the secret part within the public part. The JPEG standard allows users to embed arbitrary application-specific *markers* with application-specific data in images; the standard defines 16 such markers. We attempted to use an application-specific marker to embed the secret part; unfortunately, at least 2 PSPs (Facebook and Flickr) strip all application-specific markers.

Our current design therefore stores the secret part on a cloud storage provider (in our case, Dropbox). Note that because the secret part is encrypted, we do not assume that the storage provider is trusted.

Finally, we discuss how photos are named. When a user uploads a photo to a PSP, that PSP may transform the photo in ways discussed below. Despite this, most photo-sharing services (Facebook, Picasa Web, Flickr, Smugmug, and Imageshack[4]) assign a unique ID for all variants of the photo. This ID is returned to the client, as part of the API [17, 19], when the photo is updated.

[4]PhotoBucket does not, which explains its vulnerability to fusking, as discussed earlier

P^3's sender side proxy performs the following operations on the public and secret parts. First, it uploads the public part to the PSP either using HTTP or HTTPS (e.g., Facebook works only with HTTPS, but Flickr supports HTTP). This returns an ID, which is then used to name a file containing the secret part. This file is then uploaded to the storage provider.

Recipient-side Operation. Recipients are also configured to run a local web proxy. A client device downloads a photo from a PSP using an HTTP get request. The URL for the HTTP request contains the ID of the photo being downloaded. When the proxy sees this HTTP request, it passes request on to the PSP, but also initiates a concurrent download of the secret part from the storage provider using the ID embedded in the URL. When both the public and secret parts have been received, the proxy performs the decryption and reconstruction procedure discussed in Section 3 and passes the resulting image to the application as the response to the HTTP get request. However, note that a secret part may be reused multiple times: for example, a user may first view a thumbnail image and then download a larger image. In these scenarios, it suffices to download the secret part once so the proxy can maintain a cache of downloaded secret parts in order to reduce bandwidth and improve latency.

There is an interesting subtlety in the photo reconstruction process. As discussed in Section 3, when the server-side transformations are known, nearly exact reconstruction is possible[5] In our case, the precise transformations are not known, in general, to the proxy, so the problem becomes more challenging.

By uploading photos, and inspecting the results, we are able to tell, generally speaking, what kinds of transformations PSPs perform. For instance, Facebook transforms a baseline JPEG image to a progressive format and at the same time wipes out all irrelevant markers. Both Facebook and Flickr statically resize the uploaded image with different sizes; for example, Facebook generates at least three files with different resolutions, while Flickr generates a series of fixed-resolution images whose number depends on the size of the uploaded image. We cannot tell if these PSPs actually store the original images or not, and we conjecture that the resizing serves to limit storage and is also perhaps optimized for common case devices. For example, the largest resolution photos stored by Facebook is 720x720, regardless of the original resolution of the image. In addition, Facebook can dynamically resize and crop an image; the cropping geometry and the size specified for resizing are both encoded in the HTTP get URL, so the proxy is able to determine those parameters. Furthermore, by inspecting the JPEG header, we can tell some kinds of transformations that may have been performed: e.g., whether baseline image was converted to progressive or vice a versa, what sampling factors, cropping and scaling etc. were applied.

However, some other critical image processing parameters are not visible to the outside world. For example, the process of resizing an image using down sampling is often accompanied by a filtering step for antialiasing and may be followed by a sharpening step, together with a color adjustment step on the downsampled image. Not knowing which of these steps have been performed, and not knowing the parameters used in these operations, the reconstruction procedure can result in lower quality images.

To understand what transformations have been performed, we are reduced to searching the space of possible transformations for an outcome that matches the output of transformations performed by the PSP[6]. Note that this reverse engineering need only be done when a PSP re-jiggers its image transformation pipeline, so it should not be too onerous. Fortunately, for Facebook and Flickr, we were able to get reasonable reconstruction results on both systems (Section 5). These reconstruction results were obtained by exhaustively searching the parameter space with salient options based on commonly-used resizing techniques [24]. More precisely, we select several candidate settings for colorspace conversion, filtering, sharpening, enhancing, and gamma corrections, and then compare the output of these with that produced by the PSP. Our reconstruction results are presented in Section 5.

4.2 Discussion

Privacy Properties. Beyond the privacy properties of the P^3 algorithm, the P^3 system achieves the privacy goals outlined in Section 2. Since the proxy runs on the client for both sender and receiver, the trusted computing base for P^3 includes the software and hardware device on the client. It may be possible to reduce the footprint of the trusted computing base even further using a trusted platform module [40] and trusted sensors [27], but we have deferred that to future work.

P^3's privacy depends upon the strength of the symmetric key used to encrypt in the secret part. We assume the use of AES-based symmetric keys, distributed out of band. Furthermore, as discussed above, in P^3 the storage provider cannot leak photo privacy because the secret

[5]The only errors that can arise are due to storing the correction term in Section 3 in a lossy JPEG format that has to be decoded for processing in the pixel domain. Even if quantization is very fine, errors maybe introduced because the DCT transform is real valued and pixel values are integer, so the inverse transform of $\left(\mathbf{S_s} - \mathbf{S_s}^2\right)\mathbf{w}$ will have to be rounded to the nearest integer pixel value.

[6]This approach is clearly fragile, since the PSP can change the kinds of transformations they perform on photos. Please see the discussion below on this issue.

part is encrypted. The storage provider, or for that matter the PSP, can tamper with images and hinder reconstruction; protecting against such tampering is beyond the scope of the paper. For the same reason, eavesdroppers can similarly potentially tamper with the public or the secret part, but cannot leak photo privacy.

PSP Co-operation. The P^3 design we have described assumes no co-operation from the PSP. As a result, this implementation is fragile and a PSP can prevent users from using their infrastructure to store P^3's public parts. For instance, they can introduce complex nonlinear transformations on images in order to foil reconstruction. They may also run simple algorithms to detect images where coefficients might have been thresholded, and refuse to store such images.

Our design is merely a proof of concept that the technology exists to transparently protect the privacy of photos, without requiring infrastructure changes or significant client-side modification. Ultimately, PSPs will need to cooperate in order for photo privacy to be possible, and this cooperation depends upon the implications of photo sharing on their respective business models.

At one extreme, if only a relatively small fraction of a PSP's user base uses P^3, a PSP may choose to benevolently ignore this use (because preventing it would require commitment of resources to reprogram their infrastructure). At the other end, if PSPs see a potential loss in revenue from not being able to recognize objects/faces in photos, they may choose to react in one of two ways: shut down P^3, or offer photo privacy for a fee to users. However, in this scenario, a significant number of users see value in photo privacy, so we believe that PSPs will be incentivized to offer privacy-preserving storage for a fee. In a competitive marketplace, even if one PSP were to offer privacy-preserving storage as a service, others will likely follow suit. For example, Flickr already has a "freemium" business model and can simply offer privacy preserving storage to its premium subscribers.

If a PSP were to offer privacy-preserving photo storage as a service, we believe it will have incentives to use a P^3 like approach (which permits image scaling and transformations), rather than end to end encryption. With P^3, a PSP can assure its users that it is only able to see the public part (reconstruction would still happen at the client), yet provide (as a service) the image transformations that can reduce user-perceived latency (which is important an important consideration for retaining users of online services []).

Finally, with PSP co-operation, two aspects of our P^3 design become simpler. First, the PSP image transformation parameters would be known, so higher quality images would result. Second, the secret part of the image could be embedded within the public part, obviating the need for a separate online storage provider.

Extensions. Extending this idea to video is feasible, but left for future work. As an initial step, it is possible to introduce the privacy preserving techniques only to the I-frames, which are coded independently using tools similar to those used in JPEG. Because other frames in a "group of pictures" are coded using an I-frame as a predictor, quality reductions in an I-frame propagate through the remaining frames. In future work, we plan to study video-specific aspects, such as how to process motion vectors or how to enable reconstruction from a processed version of a public video.

5 Evaluation

In this section, we report on an evaluation of P^3. Our evaluation uses objective metrics to characterize the privacy preservation capability of P^3, and it also reports, using a full-fledged implementation, on the processing overhead induced by sender and receiver side encryption.

5.1 Methodology

Metrics. Our first metric for P^3 performance is the *storage overhead* imposed by selective encryption. Photo storage space is an important consideration for PSPs, and a practical scheme for privacy preserving photo storage must not incur large storage overheads. We then measure the efficacy of privacy preservation using PSNR (peak signal-to-noise ratio), a metric commonly used in signal processing. While the shortcomings of this metric in terms of quantifying perceptual quality are well known, it does provide a simple objective way of quantifying degradation. Note also that the public images that we are highly degraded with values of PSNR that will be commonly agreed to represent very poor quality. To complement PSNR, we also present the visual representation of the public part of an image, to let the reader judge the efficacy of P^3; lack of space prevents us from a more detailed exposition. We then evaluate the efficacy of privacy preservation by measuring the performance of state-of-the-art edge and face detection algorithms, and the SIFT feature extraction algorithm on P^3. We conclude the evaluation of privacy by discussing the efficacy of guessing attacks. Finally, we quantify the reconstruction performance, bandwidth savings and the processing overhead of P^3.

Datasets. We evaluate P^3 using three image datasets. First, as a baseline, we use the "miscellaneous" volume in the USC-SIPI image dataset [7]. This volume has 44 color and black-and-white images and contains various objects, people, scenery, and so forth, and contains many canonical images (including Lena) commonly used in the image processing community. Our second data set from INRIA [3], and contains 1491 full color images from va-

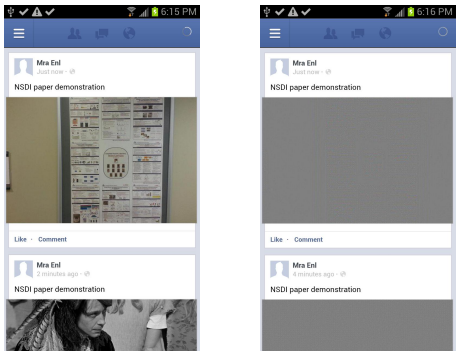


Figure 4: Screenshot of Facebook page, with and without decryption

cation scenes including a mountain, a river, a small town, other interesting topographies, etc. This dataset contains has greater diversity than the USC-SIPI dataset in terms of both resolutions and textures; its images vary in size up to 5 MB, while the USC-SIPI dataset's images are all under 0.5 MB.

We also use the Caltech face dataset [1] for our face detection experiment. This has 450 frontal color face images of about 27 unique faces depicted under different circumstances (illumination, background, facial expressions, etc.). All images contain at least one large dominant face, and zero or more additional faces.

Implementation. We also report results from an implementation for two PSPs: Facebook [18], and Flickr [21]. We chose the Android 4.x mobile operating system as our client platform, since the bandwidth limitations together with the availability of camera sensors on mobile devices motivate our work.

The *mitmproxy* software tool [33] is used as a trusted man-in-the-middle proxy entity in the system. To execute a mitmproxy tool on Android, we used a *kivy/python-for-android* software [26]. Our algorithm described in Section 3 is implemented based on the code maintained by the Independent JPEG Group, version 8d [25]. We report on experiments conducted by running this prototype on Samsung Galaxy S3 smartphones.

Figure 4 shows two screenshots of a Facebook page, with two photos posted. On the left is the view seen by a mobile device which has our recipient-side decryption and reconstruction algorithm enabled. On the right is the same page, without that algorithm (so only the public parts of the images are visible).

5.2 Evaluation Results

In this section, we first report on the trade-off between the threshold parameter and storage size in P^3. We then evaluate various privacy metrics, and conclude with an evaluation of reconstruction performance, bandwidth, and processing overhead.

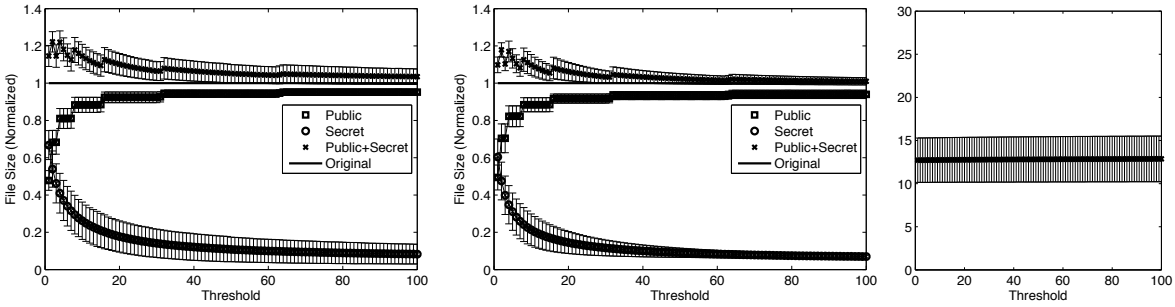
5.2.1 The Threshold vs. Storage Tradoff

In P^3, the threshold T is a tunable parameter that trades off storage space for privacy: at higher thresholds, fewer coefficients are in the secret part but more information is exposed in the public part. Figure 5 reports on the size of the public part (a JPEG image), the secret part (an encrypted JPEG image), and the combined size of the two parts, as a fraction of the size of the original image, for different threshold values T. One interesting feature of this figure is that, despite the differences in size and composition of the two data sets, their size *distribution as a function of thresholds is qualitatively similar*. At low thresholds (near 1), the combined image sizes exceed the original image size by about 20%, with the public and secret parts being each about 50% of the total size. While this setting provides excellent privacy, the large size of the secret part can impact bandwidth savings; recall that, in P^3, the secret part has to be downloaded in its entirety even when the public part has been resized significantly. Thus, it is important to select a better operating point where the size of the secret part is smaller.

Fortunately, the shape of the curve of Figure 5 for *both datasets* suggests that operating at the knee (with a threshold in the range 15/20), where the secret part is about 20% of the original image, and the *total storage overhead is about 5-10%*. Figure 6, which depicts the public and secret parts (recall that the secret part is also a JPEG image) of a canonical image from the USC-SIPI dataset, shows that for thresholds in this range virtually no visual information is present in the public part, with all of it being stored in the secret part. We include these images to give readers a visual sense of the efficacy of P^3; we conduct more detailed privacy evaluations below. This suggests that a threshold between 10-20 might provide a good balance between privacy and storage. We solidify this finding below.

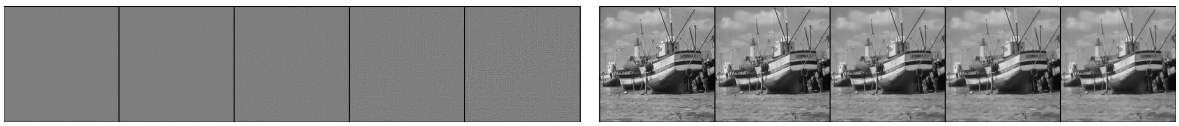
5.3 Privacy

PSNR. One of the earliest objective metrics for evaluating the quality of image reconstruction is the peak signal-to-noise ratio (PSNR). In Figure 5(c), we present normalized average PSNR of the the public part of the INRIA dataset (results for the USC-SIPI dataset are similar), as a function of different thresholds, when compared to the original image, for our two data sets. It is encouraging that the PSNR values are all around 15-20 dB, and interesting that they increase only slightly with threshold. The extraction of the DC component into the secret part plays a major part in leading to such low PSNR values. Images with a PSNR of about 35 to 40 dB are generally considered to be perceptually lossless (i.e., it will be difficult for untrained viewers to identify differences between the original image and the decoded one). Conversely, for the



(a) USC-SIPI (b) INRIA (c) INRIA

Figure 5: Threshold vs. Size and PSNR



(a) Public Part (b) Secret Part

Figure 6: Baseline: Encryption Result (Thresholds: 1,5,10,15,20)

range of (low) PSNRs that we observe here (e.g., around 20dB) it is widely accepted that quality is so degraded that these images are practically useless.

However, this alone is not an indication that P^3 preserves privacy; an examination of the public part of threshold 100 (not shown) reveals some of the features in the original image. At lower thresholds these features are no longer visible (Figure 6), but the difference in PSNR between a threshold of 10 and 100 is negligible.

For this reason, we consider using several other metrics to quantify the privacy obtained with P^3. These metrics quantify the efficacy of automated algorithms on the public part; *each automated algorithm can be considered to be mounting a privacy attack on the public part.*

Edge Detection. Edge detection is an elemental processing step in many signal processing and machine vision applications, and attempts to discover discontinuities in various image characteristics. We apply the well-known Canny edge detector [11] and its implementation [22] to the public part of images in the USC-SIPI dataset, and present images with the recognized edges in Figure 10. For space reasons, we only show edges detected on the public part of 4 canonical images for a threshold of 20. These images do reveal several "features", and signal processing researchers, when told that these are canonical images from a widely used data set, can probably recognize these images. However, a layperson who has not seen the image before very likely will not be able to recognize any of the objects in the images (the interested reader can browse the USC-SIPI dataset online to find the originals). We include these image is to point out that visual privacy is a highly subjective notion, and depends upon the beholder's prior experiences. If true privacy is desired, end-to-end encryption must be used. P^3 provides "pretty good" privacy together with the convenience and performance offered by photo sharing services.

It is also possible to quantify the privacy offered by P^3 for edge detection attacks. Figure 7 plots the fraction of matching pixels in the image obtained by running edge detection on the public part, and that obtained by running edge detection on the original image (the result of edge detection is an image with binary pixel values). At threshold values below 20, *barely 20% of the pixels match*; at very low thresholds, running edge detection on the public part results in a picture resembling white noise, so we believe the higher matching rate simply results from spurious matches. We conclude that, for the range of parameters we consider, P^3 is very robust to edge detection.

Face Detection. Face detection algorithms detect human faces in photos, and were available as part of Facebook's face recognition API [16], until Facebook shut down the API [2]. To quantify the performance of face detection on P^3, we use the Haar face detector from the OpenCV library [4], and apply it to the public part of images from Caltech's face dataset [1]. The efficacy of face detection, as a function of different thresholds, is shown in Figure 8. The y-axis represents the average number of

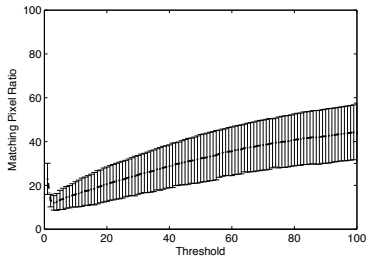


Figure 7: Edge Detection Results

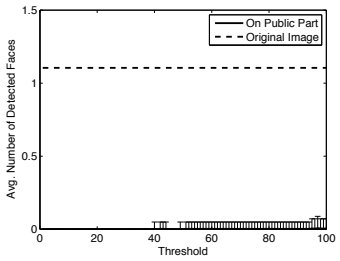


Figure 8: Face Detection Results

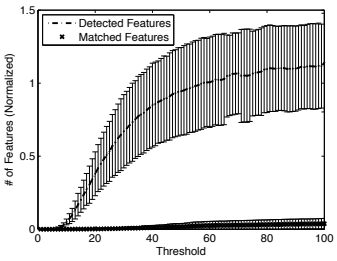


Figure 9: SIFT Feature Extraction

faces detected; it is higher than 1 for the original images, because some images have more than one face. P^3 *completely foils face detection*, for thresholds below 20; at thresholds higher than about 35, faces are occasionally detected in some images.

SIFT feature extraction. SIFT [30] (or Scale-invariant Feature Transform) is a general method to detect features in images. It is used as a pre-processing step in many image detection and recognition applications from machine vision. The output of these algorithms is a set of feature vectors, each of which describes some statistically interesting aspect of the image.

We evaluate the efficacy of attacking P^3 by performing SIFT feature extraction on the public part. For this, we use the implementation [29] from the designer of SIFT together with the default parameters for feature extraction and feature comparison. Figure 9 reports the results of running feature extraction on the USC-SIPI dataset[7] This figure shows two lines, one of which measures the total number of features detected on the public part as a function of threshold. This shows that as the threshold increases, predictably, the number of detected features increases to match the number of features detected in the original figure. More interesting is the fact that, below the threshold of 10, *no SIFT features are detected*, and below a threshold of 20, only about 25% of the features are detected.

However, this latter number is a little misleading, because we found that, in general, SIFT detects *different* feature vectors in the public part and the original image. If we count the number of features detected in the public part, which are less than a distance d (in feature space) from the nearest feature in the original image (indicating that, plausibly, SIFT may have found, in the public part, of feature in the original image), we find that this number is far smaller; up to a threshold of 35, a very small fraction of original features are discovered, and even at the threshold of 100, only about 10% of the original features may have been discovered. We use the default parameter for the distance d in the SIFT implementation; changing the parameter does not change our conclusions.[8]

Finally, we have left to future work an evaluation of a face recognition attack. However, we're confident that face recognition attacks on the public part will fail, especially for thresholds below 20. In this regime, face detection and SIFT feature extraction fail, and these are in some sense simpler tasks than face recognition; second, prior work has shown that a slightly different encryption approach can foil face recognition [14].

5.4 What is Lost?

P^3 achieves privacy but at a cost along three different dimensions: reconstruction accuracy, bandwidth overhead and processing cost.

Reconstruction Accuracy. As discussed in Section 3, the reconstruction of an image for which a linear transformation has been applied should, in theory, be perfect. In practice, however, quantization effects in JPEG compression can introduce very small errors in reconstruction. Most images in the USC-SIPI dataset can be reconstructed, when the transformations are known a priori, with an average PSNR of 49.2dB. In the signal processing community, this would be considered practically lossless. More interesting is the efficacy of our reconstruction of Facebook and Flickr's transformations. In Section 4, we described an exhaustive parameter search space methodology to *approximately* reverse engineer Facebook and Flickr's transformations. Our methodology is fairly successful, resulting in images with PSNR of 34.4dB for Facebook and 39.8dB for Flickr. To an untrained eye, images with such PSNR values are generally blemish-free. Thus, using P^3 does not significantly degrade user experience in terms of the accuracy of the reconstructed images.

Bandwidth usage cost. In P^3, suppose a recipient downloads, from a PSP, a resized version of an uploaded image. The total bandwidth usage for this download is the

[7]The SIFT algorithm is computationally expensive, and the INRIA data set is large, so we do not have the results for the INRIA dataset. (Recall that we need to compute for a large number of threshold values). We expect the results to be qualitatively similar.

[8]Our results use a distance parameter of 0.6 from [29]; we used 0.8, the highest distance parameter that seems to be meaningful ([30], Figure 11) and the results are similar.

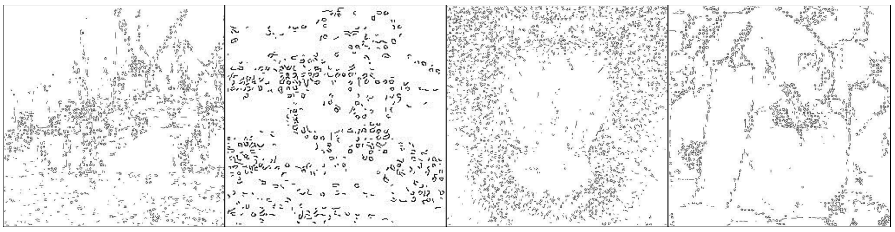


Figure 10: Canny Edge Detection on Public Part (Threshold 20)

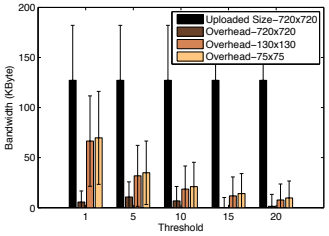


Figure 11: Bandwidth Usage Cost

size of the resized public part, together with the complete secret part. Without P^3, the recipient only downloads the resized version of the original image. In general, the former is larger than the latter and the difference between the two represents the bandwidth usage cost, and important consideration for usage-metered mobile data plans. This cost, as a function of the P^3 threshold, is shown in Figure 11 for the INRIA dataset (the USC dataset results are similar). For thresholds in the 10-20 range, this cost is modest: 20KB or less across different resolutions (these resolutions are the ones Facebook statically resizes an uploaded image to). As an aside, the variability in bandwidth usage cost represents an opportunity: users who are more privacy conscious can choose lower thresholds at the expense of slightly higher bandwidth usage. Finally, the observe that this additional bandwidth usage can be reduced by trading off storage: a sender can upload multiple encrypted secret parts, one for each known static transformation that a PSP performs. We have not implemented this optimization.

Processing Costs. On a Galaxy S3 smartphone, for a 720x720 image (the largest resolution served by Facebook), it takes on average 152 ms to extract the public and secret parts, about 55 ms to encrypt/decrypt the secret part, and and 191 ms to reconstruct the image. These costs are modest, and unlikely to impact user photo browsing experience.

6 Related Work

We do not know of prior work that has attempted to address photo privacy for photo-sharing services. Our work is most closely related to work in the signal processing community on image and video privacy. Early efforts at image privacy introduced techniques like region-of-interest masking, blurring, or pixellation [14]. In these approaches, typically a face or a person in an image is represented by a blurred or pixelated version; as [14] shows, these approaches are not particularly effective against algorithmic attacks like face recognition. A subsequent generation of approaches attempted to ensure privacy for surveillance by scrambling coefficients in a manner qualitatively similar to P^3's algorithm [14]. However, this line of work has not explored designs under the constraints imposed by our problem, namely the need for JPEG-compliant images at PSPs to ensure storage and bandwidth benefits, and the associated requirement for relatively small secret parts.

This strand is part of a larger body of work on selective encryption in the image processing community. This research, much of it conducted in the 90s and early 2000s, was motivated by ensuring image secrecy while reducing the computation cost of encryption [32, 28]. This line of work has explored some of the techniques we use such as extracting the DC components [39] and encrypting the sign of the coefficient [38, 35], as well as techniques we have not, such as randomly permuting the coefficients [39, 37]. Relative to this body of work, P^3 is novel in being a selective encryption scheme tailored towards a novel set of requirements, motivated by photo sharing services. In particular, to our knowledge, prior work has not explored selective encryption schemes which permit image reconstruction when the unencrypted part of the image has been subjected to transformations like resizing or cropping. Finally, a pending patent application by one of the co-authors [34] of this paper, includes the idea of separating an image into two parts, but does not propose the P^3 algorithm, nor does it consider the reconstruction challenges described in Section 3.

Tangentially related is a body of work in the computer systems community on ensuring other forms of privacy: storage privacy [20, 31, 10], and privacy and anonymity for mobile systems [15, 23, 13]. None of these techniques directly apply to our setting.

7 Conclusions

P^3 is a privacy preserving photo sharing scheme that leverages the sparsity and quality of images to store most of the information in an image in a secret part, leaving most of the volume of the image in a JPEG-compliant public part, which is uploaded to PSPs. P^3's public parts have very low PSNRs and are robust to edge detection, face detection, or sift feature extraction attacks. These benefits come at minimal costs to reconstruction accuracy, bandwidth usage and processing overhead.

References

[1] Caltech computational vision group, http://www.vision.caltech.edu/html-files/archive.html.

[2] Facebook Shuts Down Face Recognition APIs After All, http://www.theregister.co.uk/2012/07/09/facebook_face_apis_dead/.

[3] Inria holidays dataset, http://lear.inrialpes.fr/~jegou/data.php.

[4] Open source computer vision, http://opencv.willowgarage.com/wiki/.

[5] Usage of image file formats for websites, http://w3techs.com/technologies/overview/image_format/all.

[6] Usage of JPEG for websites, http://w3techs.com/technologies/details/im-jpeg/all/all.

[7] Usc-sipi image database, http://sipi.usc.edu/database/.

[8] M. Aharon, M. Elad, and A. Bruckstein. K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. *Signal Processing, IEEE Transactions on*, 54(11):4311–4322, Nov 2006.

[9] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel. Finding a needle in haystack: facebook's photo storage. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[10] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. Depsky: dependable and secure storage in a cloud-of-clouds. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 31–46, New York, NY, USA, 2011. ACM.

[11] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.

[12] CNN. Photobucket leaves users exposed: http://articles.cnn.com/2012-08-09/tech/tech_photobucket-privacy-breach_1_photobucket-social-media-privacy-settings.

[13] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos. Anonysense: privacy-aware people-centric sensing. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, MobiSys '08, pages 211–224, New York, NY, USA, 2008. ACM.

[14] T. Ebrahimi. Privacy Protection of Visual Information.

[15] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[16] Facebook. Face recognition api.

[17] Facebook. http://developers.facebook.com/docs/reference/api/photo/.

[18] Facebook. http://www.facebook.com.

[19] Facebook. http://www.flickr.com/services/api/upload.api.html.

[20] A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten. Sporc: group collaboration using untrusted cloud resources. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–, Berkeley, CA, USA, 2010. USENIX Association.

[21] Flickr. http://www.flickr.com.

[22] B. C. Haynor. A fast edge detection implementation in c, http://code.google.com/p/fast-edge/.

[23] B. Hoh, M. Gruteser, R. Herring, J. Ban, D. Work, J.-C. Herrera, A. M. Bayen, M. Annavaram, and Q. Jacobson. Virtual trip lines for distributed privacy-preserving traffic monitoring. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, MobiSys '08, pages 15–28, New York, NY, USA, 2008. ACM.

[24] ImageMagick Resize or Scaling. http://www.imagemagick.org/Usage/resize/.

[25] Independent JPEG Group. http://www.ijg.org/.

[26] Kivy. python-for-android, https://github.com/kivy/python-for-android.

[27] H. Liu, S. Saroiu, A. Wolman, and H. Raj. Software abstractions for trusted sensors. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, pages 365–378, New York, NY, USA, 2012. ACM.

[28] X. Liu and A. M. Eskicioglu. Selective encryption of multimedia content in distribution networks: challenges and new directions. In *Conf. Communications, Internet, and Information Technology*, pages 527–533, 2003.

[29] D. Lowe. Sift keypoint detector, `http://www.cs.ubc.ca/~lowe/keypoints/`.

[30] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[31] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish. Depot: Cloud Storage with Minimal Trust. In *OSDI 2010*, Oct. 2010.

[32] A. Massoudi, F. Lefebvre, C. De Vleeschouwer, B. Macq, and J.-J. Quisquater. Overview on selective encryption of image and video: challenges and perspectives. *EURASIP J. Inf. Secur.*, 2008:5:1–5:18, Jan. 2008.

[33] mitmproxy. `http://mitmproxy.org`.

[34] A. Ortega, S. Darden, A. Vellaikal, Z. Miao, and J. Caldarola. Method and system for delivering media data, Jan. 29 2002. US Patent App. 20,060/031,558.

[35] C. ping Wu and C.-C. J. Kuo. Fast Encryption Methods for Audiovisual Data Confidentiality. In *in Multimedia Systems and Applications III, ser. Proc. SPIE*, pages 284–295, 2000.

[36] R. Pingdom. New facts and figures about image format use on websites. `http://royal.pingdom.com/`.

[37] L. Qiao, K. Nahrstedt, and M.-C. Tam. Is MPEG encryption by using random list instead of zigzag order secure? In *Consumer Electronics, 1997. ISCE '97., Proceedings of 1997 IEEE International Symposium on*, pages 226 –229, Dec 1997.

[38] C. Shi and B. K. Bhargava. A Fast MPEG Video Encryption Algorithm. In *ACM Multimedia*, pages 81–88, 1998.

[39] L. 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, New York, NY, USA, 1996. ACM.

[40] Trusted Computing Group. TPM Main Specification, `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`.