Encrypted Cloud Photo Storage Using Google Photos

John S. Koh
koh@cs.columbia.edu
Columbia University

Jason Nieh
nieh@cs.columbia.edu
Columbia University

Steven M. Bellovin
smb@cs.columbia.edu
Columbia University

ABSTRACT

Cloud photo services are widely used for persistent, convenient, and often free photo storage, which is especially useful for mobile devices. As users store more and more photos in the cloud, significant privacy concerns arise because even a single compromise of a user's credentials give attackers unfettered access to all of the user's photos. We have created Easy Secure Photos (ESP) to enable users to protect their photos on cloud photo services such as Google Photos. ESP introduces a new client-side encryption architecture that includes a novel format-preserving image encryption algorithm, an encrypted thumbnail display mechanism, and a usable key management system. ESP encrypts image data such that the result is still a standard format image like JPEG that is compatible with cloud photo services. ESP efficiently generates and displays encrypted thumbnails for fast and easy browsing of photo galleries from trusted user devices. ESP's key management makes it simple to authorize multiple user devices to view encrypted image content via a process similar to device pairing, but using the cloud photo service as a QR code communication channel. We have implemented ESP in a popular Android photos app for use with Google Photos and demonstrate that it is easy to use and provides encryption functionality transparently to users, maintains good interactive performance and image quality while providing strong privacy guarantees, and retains the sharing and storage benefits of Google Photos without any changes to the cloud service.

CCS CONCEPTS

• **Security and privacy** → **Key management**; *Public key encryption*; **Mobile and wireless security**; **Usability in security and privacy**; **Privacy protections**; Software security engineering; Distributed systems security; • **Computer systems organization** → Cloud computing; • **Human-centered computing** → *Ubiquitous and mobile devices*; • **Information systems** → Web services.

KEYWORDS

Image encryption; key management; usable security; Google Photos

ACM Reference Format:
John S. Koh, Jason Nieh, and Steven M. Bellovin. 2021. Encrypted Cloud Photo Storage Using Google Photos. In *The 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '21), June 24–July 2, 2021, Virtual, WI, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3458864.3468220

MobiSys '21, June 24–July 2, 2021, Virtual, WI, USA

ACM ISBN 978-1-4503-8443-8/21/06.
https://doi.org/10.1145/3458864.3468220

1 INTRODUCTION

The rapid proliferation of smartphones with increasingly high-quality built-in cameras is driving enormous growth in the number of photos being taken, with well over a trillion photos captured each year [28]. Since smartphones often have low storage capacity and are prone to accidental damage and loss, many users use the cloud to permanently store their photos online via cloud photo services such as those offered by Google, Apple, Flickr, and others. Google Photos is particularly popular with over a billion users [37]. However, users' photo collections often represent a gold mine of personal information which is valuable not only to the services, but to attackers as well. Even if users trust cloud photo services with their data, the threat of attackers compromising user accounts and data is tangible. External attackers often target one of the weakest points of account security, passwords, to gain access to personal photos such as in the case of the 2014 celebrity nude photo hacks [34]. Because passwords are such a weak defense that is often compromised through social engineering, phishing, or password leaks, many services augment account security with two-factor authentication (2FA), but this is still not enough. One important reason is because adversaries may be internal, such as rogue employees at cloud services abusing their access privileges to snoop on user data [9, 14–16], and bugs or errors may reveal user data to unintended recipients, such as the recent case of Google Photos accidentally sharing users' private videos with other completely unrelated users [3]. In all cases, it takes only a single compromise of a user's account to expose their entire photo collection.

Encryption offers a well-known solution to this problem: if users encrypt their photos, then they become indecipherable even if an attacker compromises user accounts. However, existing encryption schemes are incompatible with cloud photo services. Google Photos expects uploaded files to be valid images, and compresses them to reduce file sizes. Image compression is incompatible with general and photo-specific encryption techniques, causing corruption of encrypted images. Even if image compression were compatible, mobile users expect to quickly browse through identifiable photo thumbnails which are typically generated by cloud photo services; this is not possible with any existing photo encryption schemes. Finally, encrypting data and managing keys is too complicated for most users especially if public key cryptography is involved [49]. This is made more difficult for modern users who use multiple mobile devices which each must decrypt their photos. Some third-party photo services promise image encryption and user privacy, and others propose new external secure photo hosting services [27, 41, 42, 54, 56, 57, 59], but they all require users to abandon existing widely-used cloud photo services such as Google Photos.

To address this problem, we have created Easy Secure Photos (ESP), a system that enables mobile users to use popular cloud photo services such as Google Photos while protecting their photos against account compromises. ESP encrypts uploaded photos so that attackers cannot decipher them, yet the encryption is transparent to

authorized users who can visually browse and display images as if they were unencrypted images. ESP achieves this by introducing a new client-side encryption architecture that is compatible with and requires no changes to cloud photo services such as Google Photos, and has no reliance on any external third-party system or service provider. The architecture includes three key components: a format-preserving image encryption algorithm, an encrypted thumbnail display mechanism, and an easy-to-use key management system.

ESP's image encryption algorithm works for lossy and lossless image formats such as JPEG and PNG, is compatible with image compression and is efficient enough for mobile devices. ESP converts an image to RGB, encrypts it in RGB color space using a block-based Fisher-Yates shuffle [19] with a per-image encryption key, splits the RGB channels into three separate grayscale ciphertext images, then converts them back into the original image format, including compression for JPEG. This novel encrypted grayscale approach maintains the original image dimensions and ensures compatibility with standard JPEG compression, widely-used compression techniques such as Guetzli [4, 5] JPEG encoding, and JPEG chroma subsampling.

ESP provides an encrypted thumbnail display mechanism that is compatible with cloud photo services. ESP uploads client-generated, encrypted thumbnails and redirects the client image browser to view the uploaded thumbnails rather than server-generated ones. This makes it easy to interactively browse and view thumbnail galleries.

ESP includes easy-to-use key management that supports multiple devices, yet eliminates the need for users to know about and move private keys from one device to another. ESP uses verified self-generated keypairs that build up a chain of trust from one device to another by leveraging the cloud photo service itself as a communication channel for a QR code-based message protocol.

We have implemented ESP on Android in the Simple Gallery app, a popular photo gallery app with millions of users, adding the ability to use Google Photos for encrypted photo storage. ESP consists of only client-side modifications that use the Google Photos API and requires no changes to the Google Photos cloud service. We have evaluated ESP using images from Google's Open Images Dataset V5 [23]. Our experimental results show that ESP (1) works seamlessly with Google Photos even with their image compression techniques, (2) produces encrypted images with quality comparable to the original images, (3) provides strong security against practical threats including account compromises and machine learning analysis, (4) provides fast encrypted image upload, download, and browsing times with modest overhead, and (5) is easy to use across multiple devices. ESP is also compatible with other popular cloud photo services such as Flickr and Imgur.

The contributions of this work are the design, implementation, and evaluation of a new system for encrypting images stored on existing cloud photo services, with no server-side modifications, requiring no trust in the photo services or their servers. The system includes a novel format-preserving image encryption scheme coupled with a key management solution for users. To the best of our knowledge, we are the first to address practical issues such as key management, usability, image sharing, and compatibility with existing cloud photo services without needing to trust them, all in a single system.

2 THREAT MODEL

ESP protects the privacy of images stored remotely in cloud services with no changes to software or protocols other than the installing a client-side ESP app. Attackers may be inside or outside the cloud photo service, and may compromise user accounts by obtaining passwords or abusing privileged access. They may be sophisticated, but not at the level of a nation-state intelligence agency; they do not have the computing resources to break encryption via brute force.

We assume that user devices with ESP clients are secure and trustworthy. Protecting users' devices is an orthogonal concern that should be managed by device hardware or at the operating system level. A compromise of a user's device would mean the attacker has access to the private keys that can be used to decrypt any encrypted images belonging to the user.

3 USAGE MODEL

ESP is easy to use. A user installs an ESP photos app, then authenticates it with a cloud photo service such as Google Photos. Our Android ESP app's only setup step is to select the Google account to use. The app appears like any regular photos app, except that it automatically and transparent encrypts images before uploading them to Google Photos and decrypts them on download. Users are free to perform common image operations, such as viewing thumbnail galleries, moving photos to albums, assigning labels, modifying metadata, editing pictures, and sharing them with others.

ESP assumes that reasonable users only access encrypted images on trusted devices; it is uncommon (and inadvisable) to view photos on untrusted devices such as public computers. In other words, ESP is not compatible with using untrusted computers such as those at an Internet cafe, nor should it be if users care about their privacy.

Any device that a user trusts can decrypt and view images from the cloud photo service. Users are free to use ESP on as many devices as desired. Each app installation on a new device after the first requires a short and simple setup step to verify it with any previously configured device. This process appears conceptually similar to device pairing: a user verifies his new ESP device using one of his existing ESP devices. In contrast to normal pairing, the user only needs to complete this verification step *once* per new device with only one other existing ESP device. Verification involves (1) the user configuring a new device with ESP, (2) it displaying a random phrase, and (3) the user copying the random phrase to *one* other existing ESP device. Successful completion of these steps ends the new device's setup. The device can then upload and download encrypted images on the user's chosen cloud photo service.

Users can remove devices from their ESP ecosystem to revoke device access to their photos. Any configured ESP device can be used to remove any other device. If the user indicates that a device is being removed because it was lost, ESP informs the user that his photos will be re-encrypted for security reasons.

ESP intrinsically prevents users from losing access to their encrypted photos if they have more than one ESP device. A user may lose one, but can access his encrypted photos on the remaining devices. However, if a user loses all of them, a recovery password is needed to regain access; this is also standard on popular operating systems for disk encryption schemes [8, 38].

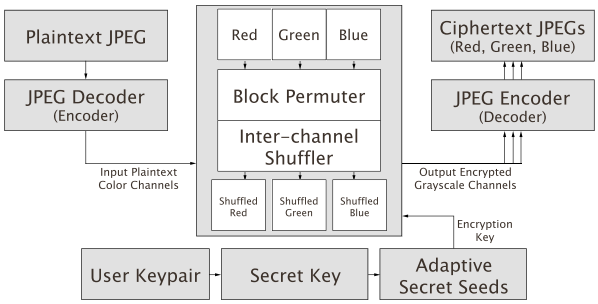

Figure 1: The ESP encryption architecture for JPEGs.

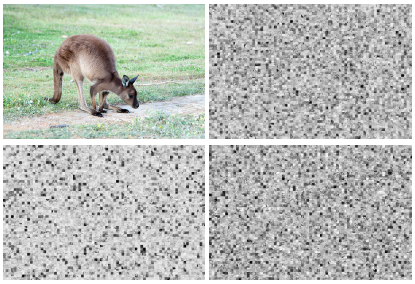

Figure 2: An image and its encrypted RGB components.

Since ESP apps encrypt the images stored in the cloud photo provider, apps which do not support ESP will be unable to decrypt them. The encrypted images, however, can be opened in image apps and editors for viewing the ciphertext, and are valid JPEG files. Users of services with web browser interfaces may install ESP browser extensions to decrypt and view their photos.

4 ARCHITECTURE

ESP is designed for compatibility with existing cloud photo services, preservation of image formats when encrypted, and end-user usability with respect to browsing images and managing keys. While ESP works with multiple cloud photo services, we focus on Google Photos given its popularity. Google has no stipulations in its terms of service that prohibit users from encrypting their data [24].

4.1 Format-Preserving Encrypted Images

ESP's image encryption algorithm is compatible with cloud photo services which support standard image formats including JPEG, PNG, WebP, and RAW. Compatibility means uploading encrypted images without them being rejected, and decrypting them with minimal loss of quality beyond any compression by the service. We focus on JPEG since it is the most commonly used image format, though ESP's encryption method works with others. We also focus on Google Photos' free tier of service which compresses and processes uploaded images. Google Photos does not publicly specify its processing pipeline, but we observe it to downsample the JPEG chroma format to 4:2:0, and to apply compression with possibly a noise filter. The compression is likely a standard JPEG quantization plus Google's own Guetzli JPEG encoder [4, 5]. ESP encryption must account for these techniques or images will be corrupted. It must also be fast and efficient, as many users use resource-constrained mobile devices.

To understand how ESP's encryption works, it is necessary to understand JPEG compression. Images rendered on a user's screen generally consist of pixels, each of which has red (R), green (G), and blue (B) components; each component is an 8-bit value in the range 0 to 255. Converting RGB data to JPEG format takes four steps. (1) The RGB components are converted into luminance, chroma-blue, and chroma-red (YCbCr) components. Sometimes the chroma is subsampled to reduce the sizes of Cb and Cr which are less important for image quality. An image with full size, half size, and quarter size CbCr are in 4:4:4, 4:2:2, and 4:2:0 format, respectively. (2) YCbCr components are transformed using a discrete cosine transform (DCT) to output DCT coefficients. (3) The DCT coefficients are quantized to reduce the number of bits representing them. Quantization is the main lossy compression step of JPEG, is controlled via a JPEG quality parameter, and is performed on 8×8 blocks of pixels. (4) Lossless compression techniques further reduce the image file size. Cloud photo services' compression practices are problematic for encrypted images due to data loss caused by quantization and chroma subsampling. We have confirmed images encrypted using common DCT coefficient diffusion and confusion techniques experience visual corruption and are unusable.

As shown in Figure 1, ESP is an encrypt-then-compress mechanism that shuffles 8×8 pixel blocks. It encrypts by scrambling the order of 8×8 pixel blocks in the image, then performs image compression on the scrambled image. Because it does not modify the values or positions of pixels within the 8×8 blocks, ESP's encryption is unaffected by any intra-block pixel-based lossy operations, making it robust in the presence of JPEG compression. Decryption means moving each block back to its original position, so any lossy compression of pixels within each block is unrelated to the decryption. In contrast, standard image encryption methods modify pixel values and shuffle pixels within blocks. Lossy compression further modifies these encrypted pixel values, but in a non-uniform manner, making it impossible to reconstruct the original values. This resulting corruption appears visually, and is even worse for color images.

As shown in Figure 1, encryption and decryption are performed on RGB data. On encryption, the image is (1) decoded and decompressed to RGB color space, (2) separated into three grayscale images, one for each of the RGB color channels, (3) encrypted by shuffling the pixel blocks of the grayscale images, and (4) JPEG compressed, resulting in three separate grayscale ciphertext JPEG images; users may choose a JPEG quality setting that balances preservation of visual quality with larger ciphertext file sizes. Since the ciphertext JPEGs are legitimate JPEGs, they can be displayed like regular JPEGs but do not reveal the original images. Figure 2 shows a sample image encryption produced by ESP. Grayscale ciphertext images are also immune to chroma subsampling because they have no chroma components, only a luminance (Y) component. One beneficial side effect of this is that ESP images can retain higher resolution chroma channels in the decrypted JPEGs compared to unencrypted images, which may have their chroma subsampled by Google Photos. On decryption, the encrypted image is (1) decoded by decompressing the three ciphertext JPEGs to their grayscale values, (2) moving each pixel block back to its original position, and (3) combining the grayscale images into an RGB image, which can then be rendered for viewing or compressed again to be stored on disk.

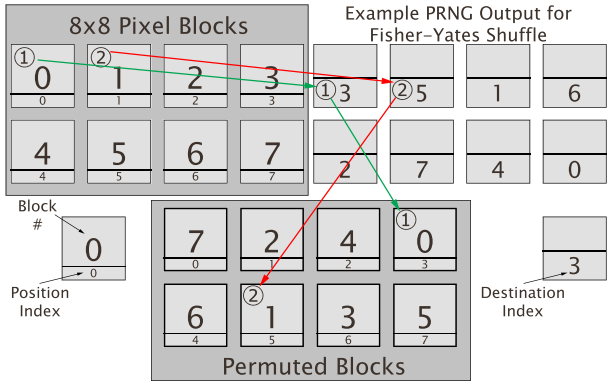

Figure 3: Example of 8×8 block permutation.

To encrypt, ESP shuffles pixel blocks within and across the grayscale images representing the RGB components of the image using pseudo-random number generators (PRNGs) with known secret seed values. The encryption method, ESP-FY, computes three initial secret seed values, (s_R,s_G,s_B), one for each RGB component, to derive (s'_R,s'_G,s'_B) values with the appropriate seed length to use as inputs to the PRNGs. (s_R,s_G,s_B) are unique per image, making them adaptive secret seeds generated from a user's secret key and properties of the image. An identifer for the algorithm used to encrypt an image is stored in its metadata, which can be EXIF format, and used on decryption to identify the decryption algorithm to use. This is similar to how widely-used cryptosystems such as TLS support multiple algorithms, and allows ESP to also support alternative or new crypto methods.

Figure 3 depicts an example of ESP-FY's encryption process applied to a 32×16 pixel image, which consists of 4×2 blocks of 8×8 pixels. ESP pads image dimensions to the nearest multiple of 8—the same strategy as the JPEG format. ESP-FY first does a Fisher-Yates shuffle for all 8×8 blocks in each RGB channel. The shuffle is driven by three PRNGs seeded by the (s'_R,s'_G,s'_B) 256-byte secret seeds, which may be truncated if they are too large for the PRNG. It then shuffles blocks across the RGB components by pseudo-randomly swapping the ith block in each iteration of the Fisher-Yates shuffle with the ith block of the other channels. In other words, in the ith iteration of the Fisher-Yates shuffle, blocks (i_R,i_G,i_B) in each of the RGB channels are shuffled. Which swaps occur is determined by $s'_{RGB}=s'_R\oplus s'_G\oplus s'_B$ as the input to a PRNG. Then using this seeded PRNG, generate a uniformly distributed random integer $k_i\in[0,5]$ on every block iteration. Each k_i represents a unique ordering of (i_R,i_G,i_B) of which there are $3!=6$ permutations. Assume that $k_i=0$ is mapped to the ordering (i_R,i_G,i_B) with no swaps. Now consider $k_i=1$ mapped to (i_R,i_B,i_G), meaning block i_G is swapped with block i_B, i.e. the ith green block is swapped with the ith blue block. Therefore, each $k_i\in[0,5]$ generated by each round of shuffling is mapped to a unique ordering of (i_R,i_G,i_B).

The encryption algorithm is therefore a Fisher-Yates block shuffle in each RGB channel, which on each iteration pseudo-randomly swaps blocks (i_R,i_G,i_B) across channels. Decryption is the inverse in reverse order: reversing the inter-channel shuffle on each iteration over the images' blocks while reversing the Fisher-Yates shuffle of the 8×8 blocks of pixels.

The security of shuffle-based encryption scales with the image dimensions. Smaller images have fewer permutations, and larger images have more. Block-based shuffling is not ideal for very simple images, such as ones where the visual data is aligned to the blocks, e.g., solid colors or simple patterns. However, from a privacy perspective, simple images are less likely to contain sensitive information. Like all encryption schemes, an encrypted image which has been resized cannot be decrypted. ESP therefore resizes images larger than the maximum allowed sizes for cloud photo services prior to encryption. Google Photos' maximum size is 16 MP, so ESP scales images to at most 16 MP before encrypting them. This is no worse than a user uploading an unencrypted image larger than 16 MP since it will be resized in any case.

A crucial component of PRNG-based encryption is the selection of the secret seed values; easily guessed ones are a security risk. ESP adaptively generates unique seed values (s_R,s_G,s_B) per encrypted image. A user's initial secret key S is a sufficiently long, say 256-bit, string of cryptographically secure random bits. 384-bit adaptive seed values (s_R,s_G,s_B) are randomly generated separately for each image, then encrypted using S as $E_S(s_R,s_G,s_B)$. This is stored in the encrypted grayscale images' metadata either as EXIF format or via another method such as Google Photo's metadata fields. The 256-byte secret seeds are derived from the 384-bit adaptive seed values.

4.2 Encrypted Thumbnails

A difficulty in supporting existing cloud services is constructing a seamless and fast experience so that users can quickly browse through thumbnail galleries of their photos. Without designing for this, a user would only be shown thumbnails created from the ciphertexts, which are unusable, or would have a sluggish experience with high bandwidth usage using a naive approach that generates thumbnails as a user is browsing through them by downloading and decrypting full resolution images. An alternative would be to continuously check for newly encrypted images in the background and create thumbnails even when the user is not browsing, but this is a laborious process involving all of a user's ESP clients downloading all the images at full resolution, decrypting them, resizing them, and generating thumbnails.

ESP introduces a simple solution by generating thumbnails when uploading new images. This removes thumbnail generation from the critical path of browsing. It also hides its overhead because uploading is all done in the background, so the user does not need to wait for it and is free to continue using the app, and thumbnail generation is cheap compared to encrypting the image itself. ESP prepares two encrypted images (six grayscale ciphertext images). The first is the original encrypted image, and the second is an encrypted, resized thumbnail. The encrypted original is stored in the user's chosen location on Google Photos while the encrypted thumbnail is uploaded to an album specifically for encrypted thumbnails; the ESP client hides this album from the user under normal operation.

When the uploads complete, the ESP client maps encrypted images' Google Photos media IDs to the corresponding encrypted thumbnails' media IDs. Thus, when the user is browsing an encrypted album for the first time, the client requests the smaller encrypted thumbnails rather than the full-size originals, and requests and downloads full-size encrypted images at a lower priority. However, it may prioritize requesting full-size images that the user is likely to view next, for example, if the user is swiping through individual

images. The user's other ESP clients that do not have the original-to-thumbnail mappings recreate them independently by periodically scanning the special album for new encrypted thumbnails.

4.3 Image Sharing

ESP users can share their images and albums with others normally or securely. Normally sharing images appears the same as sharing a regular unencrypted image. If the source image is available locally, it is uploaded unencrypted as a separate copy to the service and shared with the recipient via the service's normal mechanism. If the image is not available locally, the image in Google Photos is downloaded, decrypted, and re-uploaded to be shared with the recipient. The user is notified that normal sharing is insecure and may compromise the image. However, the willingness to share a photo already represents a security risk as no guarantee can be made that the recipient is trustworthy. For example, the recipient may copy the image and distribute it; the sender has no control over this.

ESP users may securely share images and albums with other ESP users. Suppose Alice wishes to share an encrypted photo album with Bob. If this is their first time sharing albums with each other, they perform a one-time handshake which is a public key exchange. Alice asks Bob to share his handshake link (URL) generated by his ESP app. Bob's ESP public key and other metadata are encoded into this URL; it is then shortened, for example, via Google Firebase Dynamic Links, so that if Alice opens it on Android or iOS, it will be routed to the ESP app and add Bob's public key. Alice also shares her handshake URL with Bob whose ESP app performs the same process. This completes the public key exchange without them knowing what a public key is.

Now Alice begins sharing $E(Album_A)_{S_A}$, the encryption of $Album_A$ using her secret key (S_A). Alice selects $Album_A$ to share and chooses Bob from her list of known ESP users. Alice's client creates a new album $Album_{AB}$, invisible to her, to share with Bob; the original $Album_A$ is not shared. Alice's client then generates S_{AB}, a new secret key shared by Alice and Bob. Alice decrypts $E(Album_A)_{S_A}$, and re-encrypts and stores it in $Album_{AB}$, resulting in $E(Album_{AB})_{S_{AB}}$. Alice then encrypts S_{AB} using both her own and Bob's public keys, resulting in $E(S_{AB})_{(Alice,Bob)}$. Next, if Alice's cloud photo service supports its own sharing mechanism, Alice's client ensures that $Album_{AB}$ is viewable by Bob and retrieves the service's URL for $URL(Album_{AB})$. Finally, Alice's client presents her with a shortened share URL $U_{Alice,Bob}$ to send to Bob. When expanded, $U_{Alice,Bob} = \{URL(Album_{AB}), E(S_{AB})_{(Alice,Bob)}\}$. Bob's device receives and automatically opens it in ESP. Bob's client uses its private key to decrypt $E(S_{AB})_{(Alice,Bob)}$ and retrieve S_{AB} to decrypt the album.

Alice and Bob's clients also support viewing $Album_{AB}$ on multiple devices. When Alice's client constructs $U_{Alice,Bob}$, it is also synchronized among all of Alice's devices in the form of a IKM broadcast message, the protocol for which is discussed in Section 4.5. Any of Alice's devices which trust the broadcasting device will accept the message, decrypt the contained encrypted shared seeds, and record the seeds' association with $URL(Album_{AB})$. Bob's client performs the same steps. If Alice shares $Album_A$ with multiple people, ESP adds their public keys to the list of keys used to encrypt the shared seeds $E(S_{AB...})(Alice,Bob,...)$.

Alice revokes Bob's access to $Album_{AB}$ by removing him from $Album_A$. Alice's client first revokes any granted access controls via the cloud service itself, and then deletes $Album_{AB}$ from the service. ESP cannot prevent Bob from accessing any copies of $Album_{AB}$ that he may have saved. Alice revoking Bob's access only prevents Bob from viewing the album via the cloud service. If Alice has shared it with multiple people, then she may only revoke all of their access at once. If Alice desires granular access revocation, she must share her album with each user separately.

4.4 Other Features and Limitations

ESP is incompatible with cloud photo services' features that rely on server-side access to photo image data. This includes facial recognition and detection, machine learning-based labeling and classification, image searching, and other similar functionality. For example, Google Photos uses server-side machine learning to classify images ostensibly for supporting search. However, this feature is incompatible with encrypted images, as it is run on Google's servers which cannot decrypt ESP images. Moreover, server-side classification of images compromises their privacy. Consequently, ESP utilizes client-side image classification, similar to what Apple does on their Photos app on mobile devices, to protect their users' privacy [6]. In fact, both Google and Apple provide on-device classification models, Google via ML Kit [25] and Apple via Core ML [7]. Although ML Kit provides fewer labels for on-device classification compared to Google's cloud-based one, Apple's Core ML has no such limitations. Once labeled, images can be searched.

Users may wish to edit images such as by adding filters or cropping them. Images are edited locally by first decrypting the image, applying the modifications, then re-encrypting the image. ESP can be supported on web browsers via browser extensions. Such an extension implements all the features of a normal ESP client. Depending on local storage constraints, decrypted images may be cached locally. If storage is constrained, then the extension fetches ESP encrypted thumbnails to ensure a smooth user experience.

Existing cloud photo services could change their systems or format requirements in ways that impact ESP users. ESP assumes that services adhere to existing image standards and will not deviate from them, i.e. services will not arbitrarily convert users' photos to different formats. This is unlikely to happen, but if it does, ESP clients would not overwrite their local encrypted ESP images with the copies that the cloud photo service has converted. Since ESP clients keep mappings of photo identifiers, these records can also be used to detect arbitrary modifications by services and retain the copies of the images prior to their modifications, allowing ESP to re-encrypt the images to the new format. However, this requires users to maintain local copies of their photos, making this a potential limitation of ESP if services decide to arbitrarily convert users' photo file formats. Another possibility is that services may release support for encrypted images themselves which would compete with ESP. This is actually a desirable outcome. The motivation of ESP is to satisfy users' desires for privacy via a client-side solution with no reliance on any third-party services, since cloud photo services do not provide this feature themselves. If services begin to provide features similar to ESP, this would be an overall win for users as the user experience, integration, and feature support can be further improved with support from the services themselves.

4.5 Key Management for Multiple Devices

ESP's secret key used to encrypt and decrypt images is never exposed to the user and never leaves a ESP device without being encrypted. A user's ESP device maintains self-generated public/private keys such as PGP keypairs, which have no reliance on PKI. The keypairs are used for encrypting and decrypting secret key S. The encrypted S, denoted as $E(S)$, is also stored remotely in the cloud photo service as a QR code image. Any user's ESP device can recover S from $E(S)$. The challenge lies in how ESP should manage keypairs for users, including granting and revoking access to $E(S)$ for multiple devices, since public key cryptography and key management are extraordinarily confusing for users [49].

To address these problems, ESP introduces Image-based Key Management (IKM), building on our previous work on key management for encrypted email [35]. IKM has a simple usage model for users, hiding the many confusing aspects of key management and exposing to users only the concept of device management, which they readily understand. IKM's primary principle is that a user should not need to synchronize a private key across their devices, but instead only synchronize their devices' public keys. Thus, when a user installs ESP on a new device, it generates a new keypair and uses the IKM verification process to gain the trust of another existing ESP device. This already-trusted device then broadcasts the new device's public key in a signed message encoded as a QR code uploaded to the cloud photo service. The user's other ESP devices accept the broadcast since they already trust the announcing device. So adding a new device builds up a chain of trust in which devices that trust each other also trust the other devices they each trust. Device revocation is handled similarly; trusted devices broadcast the removal of another device.

IKM uses the cloud photo service as the communication channel for messages. For example, with Google Photos, ESP creates a special album named #IKM-QR-CODES, which is invisible to the user in the app, and uploads IKM QR code messages to it. We chose QR codes as they can be represented as images which are robust against image compression and resizing. IKM messages are not encrypted but are signed and verified by every trusted ESP client.

Adding and verifying a new ESP device so that it becomes trusted happens through a synchronization process reminiscient of Bluetooth pairing. It is carried out either via a platform-independent verification step that works with any device that has a display screen and a connection to the cloud photo service, or via near-field communication (NFC) when supported. Adding a new device begins when the user sets up ESP on it. ESP displays a random phrase to the user, which the user must input on any of his existing ESP devices. The random phrase may be 3 to 5 random words from curated lists such as the PGP Word List [33]. This random phrase—conveyed by the user, who acts as a secondary, out-of-band communication channel—is the password input to a Password-Authenticated Key Exchange (PAKE) [2] completed over the primary communication channel, the cloud photo service. The PAKE allows the two devices to independently generate a secret session key for constructing an encrypted tunnel. Since the user copied the random phrase—the password—out of band, the photo service cannot learn the secret session key or intercept the tunnel. Once the encrypted tunnel has been established, the two devices can exchange public keys and authenticate them without risk of eavesdropping or man-in-the-middle attacks. The existing ESP device then uses the IKM protocol as normal to broadcast the new device's public key.

Next, the existing device decrypts $E(S)$ and re-encrypts S using the new device's public key plus all other existing ones. The resulting $E'(S)$ is stored as normal, completing the new device setup. The new device can now decrypt $E'(S)$ to get S, which it can use to decrypt any encrypted image. Deleting a device is similar: the user selects it from one of his existing devices, then the remotely stored $E(S)$ is re-encrypted using the public keys of all devices except for the deleted one. This simple deletion mode is suitable when the user is still in control of the deleted device. However, if the device to delete was lost, this requires more work since the goal is to ensure that the lost device cannot continue to download and decrypt images from the cloud photo service; a lost device still has access to previously locally stored images. We make a distinction between a lost device and a compromised device. ESP does not defend against device compromise where the attacker has bypassed the hardware or OS security because then the attacker has access to the secret key regardless of the security or encryption scheme. The purpose of device deletion is to revoke device access.

Deleting a lost device replaces $E(S)$ with a new secret key S', which is then used to re-encrypt all of the user's images. S' is encrypted using all public keys except for the key of the lost device. The other devices detect this change and update their local caches. The devices then begin re-encrypting the user's images using S'. The device which initiated the deletion request chooses how to parallelize the re-encryption effort among other devices by broadcasting distribution of work. For example, it can partition images by album or time ranges.

In extreme cases, a user may lose all of his ESP devices, thus necessitating recovering access to his encrypted images. What the user has lost is all of his private keys which can decrypt $E(S)$. Therefore, when a new ESP client does not detect an existing one in the cloud photo service, it also provides the user with a recovery key that they must write down. The recovery key is a symmetric key that can decrypt a copy of $E(S)$.

5 SECURITY ANALYSIS

ESP acts as a significant security barrier to compromises of privacy. As the following security analysis shows, it is difficult to break encryption for even one photo. If a user encrypts many photos, the difficulty of breaking encryption for all those photos becomes almost impossible. Adversaries need to expend immense computing resources to break the encryption of even one photo, and more so with many photos since each photo has a unique random seed for its encryption. Even then, the output of attack attempts would need additional resources, either computing or manual human verification, to determine whether the attempts at decrypting photos are correct or not. It therefore becomes intractable for an attacker to successfuly decrypt many encrypted photos uploaded by a user.

Given significant resources, an attacker might break the encryption of a photo, but it may be more likely that the adversary would attempt to break the encryption via other means, such as stealing a user's trusted device or exploiting vulnerabilities on a user's device. As we discuss in Section 2, defenses against attacks like these are orthogonal to the core of ESP design, and would be better handled at the hardware, OS, or even application platform level, i.e. Android

and iOS providing mechanisms for secure key management resilient against attackers. Even though ESP considers device security to be out of scope, ESP still represents a significant improvement to cloud photo service security that attackers must overcome as it is much more difficult to obtain users' ESP secret keys compared to the current norm of needing only user account names and passwords to access their photos.

Our security analysis focuses on ESP's two main security guarantees: (1) ESP's device management ensures that photos are encrypted using only authenticated public keys. (2) ESP's encryption protects the confidentiality of photos. We describe ESP's properties to use as building blocks for these claims.

Property 1. *The first configured ESP device is trusted.*

ESP relies on a trust on first use approach since all keypairs are self-generated. This first device's keypair is used as the starting point for adding further devices.

Property 2. *Attackers cannot add their malicious devices to a user's chain of trust.*

To compromise a user's trusted device chain, an attacker could try to force one of the victim's trusted ESP devices to authenticate a malicious one. The attacker would need access to the user's cloud photo service account and initiate the protocol to add a new device. Next, the attacker needs the victim to authenticate the attacker's malicious device. However, since the user did not initiate the protocol, they will not recognize the request or know the correct verification phrase, so the malicious device will not be authenticated.

Another approach is to wait for the user to legitimately add a new device and then perform a man-in-the-middle attack. However, this is not possible due to the use of the PAKE, in which the user manually inputs the random phrase as the password to the PAKE. Since the PAKE's password is never exposed to the network, the attacker cannot intercept it and learn it remotely. To compromise the PAKE, the attacker would need to have either already compromised one of the user's devices, or be physically watching the user to learn the password. Guessing the PAKE password is also intractable because the PAKE protocol detects failed attempts and aborts when they occur. If this happens, the user must restart the protocol, and a new random phrase will be selected. The attacker must therefore guess the PAKE password with replacement and also with innate rate-limiting, as it will take more than a few seconds for the user to restart the protocol after every failure.

Property 3. *Any device which is trusted by the first ESP device is also a trusted device.*

This follows from Properties 1 and 2. If the first device authenticates a new device, then it is trusted by definition.

Property 4. *Any device trusted by a given trusted device is also trusted by every other device.*

This is the generalized version of Property 3, and holds true due to trust by transitivity. If a device trusts another one, then the first also trusts every device trusted by the second.

Property 5. *ESP incrementally builds up trust among devices using only a single completed verification process per device.*

When a user adds a new device, it does not need to be individually verified by every other existing device. Instead, it is only verified by one other device. We prove this property by induction. Suppose a user has a single device, then by Property 1, it is trusted. If the user obtains a second device, then the first is trusted, and the second device is verified by the first and becomes trusted. The result is that both devices trust each other and have exchanged their public keys.

Now the user adds a third device and has devices A, B, and C. A and B already trust each other. Then, the user uses B to verify C, after which point they trust each other. Since B trusts A, B knows A's public key and provides it to C, so C can also trust A despite having never interacted with it. The question is then how A learns about C's public key. B publishes its set of trusted public keys, signed by itself and containing C's public key, by uploading it to the photo service. A will detect the signed set of trusted public keys, and since A trusts B, it will therefore accept C's public key. A, B, and C now know every trusted public key and trust each other.

This same logic applies even if the user has N devices and adds a new device Z, resulting in $N+1$ devices. The user adds new device Z by using trusted device $K \in \{N\}$ to verify Z. Afterwards, K and Z trust each other, and Z has received the public keys of all N devices. K publishes an updated set of trusted public keys containing Z's public key, signed by K. The remaining $\{N-K\}$ devices observe Z's new public key in the set, and seeing that it was signed by K, accept it, thereby also accepting Z's public key. In summary, the user has performed only a single verification step for new device Z in order for it to join their ecosystem of trusted ESP devices.

Property 6. *ESP devices that do not participate in the verification step only add authenticated public keys.*

This follows from Property 4. Trusted devices only accept public key lists from known, trusted devices, so they reject any messages or public keys signed by an untrusted public key.

Property 7. *ESP's device management ensures that photos are encrypted using only authenticated public keys.*

This follows from Properties 1–6.

Property 8. *ESP's encryption is robust and secure.*

ESP is resistant to brute force attacks on the secret seed values and images. As described in Section 4.1, (s_R, s_G, s_B) are 384-bit seed values, so there are $(2^{384})^3$ possible values for (s_R, s_G, s_B)—a sufficiently large key space. Even if an attacker only wishes to brute force the seed value for a single channel, the complexity is still 2^{384}, and does not even account for the additional effort still needed to reverse the inter-channel shuffle.

A brute force attack could also be performed on the images. Such an attack must find both the correct permutation of 8×8 blocks and ordering of inter-channel swapped RGB blocks to reconstruct the image. There are consequently $O(B_R! \times B_G! \times B_B!)$ possible permutations of blocks to reconstruct the entire original image, so brute force attacks are impractical for all but the smallest of images. For example even a small (by modern standards) 1280×720 pixel image contains 14400 blocks, for a search space of $14400!^3$.

An extension of the brute force attack is to treat shuffle-based encrypted images as if they were unsolved jigsaw puzzles. A jigsaw puzzle solver attack leverages perceivable outlines within an image's shuffled blocks to try to re-assemble them into recognizable features. The solver running time increases exponentially with the number of blocks, and shuffling blocks across color components significantly reduces the output quality [12]. Using a small block size such as 8×8 pixel blocks also acts as an important defense [13]. Even when the

Figure 4: The image and its two ciphertexts that do not pass the NPCR randomness test by a slim margin.

solvers run to completion, there is often a low or zero reconstruction rate of the source image's recognizable features.

Attackers may not need the correct position of every block to gain useful information. For example, an image may contain sensitive data in only a small portion of it which a puzzle solver or outline counting attack may reconstruct, or a "close enough" guess can reveal the context of the image. Since more than 1 billion images were uploaded to Google Photos in 2019 [37], ESP is a significant barrier to adversaries like the cloud service itself analyzing user images. It is less clear what happens if the adversary is targeting just a few images, e.g. in the context of a specific investigation; this is the subject of ongoing research.

ESP is resistant to known plaintext attacks due to its adaptive key scheme: the encryption keys (seed values) used to encrypt every image is different. It is also robust against differential cryptanalysis [52]. We evaluate this using a commonly used measurement, the number of pixel change rate (NPCR) [11, 52], which compares two ciphertext images C_1 and C_2 for a source image which has been modified by 1 pixel. We computed the NPCR for 100 images we selected from the Open Images Dataset V5 [23] and encrypting the original and modified versions. Since ESP splits RGB images into three encrypted grayscale images, the inputs C_1 and C_2 were constructed by combining each of their three grayscale ciphertext images into a single RGB ciphertext image. 99 of 100 images passed the NPCR randomness test such that $\text{NPCR}(C_1,C_2) > \text{NPCR}*_{\alpha=0.05}$ for each pair of images C_1 and C_2, suggesting that ESP's encryption scheme is substantially resistant to differential cryptanalysis. Figure 4 shows the one image that was very close to passing this test but did not; we believe that minor improvements to our algorithm will bring it to a passing score.

Property 9. *ESP guarantees the confidentiality of its secret key and secret seed values.*

ESP uses standard public key encryption for protecting the secret keys which encrypt the secret seed values per image that drive the ESP-FY encryption scheme. The asymmetric public keys from each of a user's devices encrypt the ESP secret key, which is a high entropy, randomly generated symmetric AES key, and this key encrypts the secret seed values for each encrypted image.

Property 10. *ESP's encryption protects the confidentiality of photos.*

This follows from Properties 8–9.

6 IMPLEMENTATION

We implemented ESP on Android by modifying Simple-Gallery, a popular open-source image gallery app used by millions of users [44], and the Android Fresco image loading library [18], which uses the libjpeg-turbo [36] library written in C. Simple-Gallery was originally an offline image gallery app, so we modified it to support Google Photos. We implemented IKM by adapting a separate Android key management app, OpenKeychain. We implemented the JPEG encryption algorithm in C/C++. We modified Fresco's image pipeline to invoke the encryption and decryption routines when requested.

Since Simple-Gallery had no support for online services, nearly all the added code, about 4.5K lines of code (LOC), was for interfacing with Google Photos. The Java code added to the Fresco library mainly consisted of abstraction layers orthogonal to encryption to accommodate the software design patterns for their image pipeline. The encryption algorithm written in C/C++ only required about 1K LOC, not including the GNU Multiple Precision Arithmetic (GMP) Library [20] for arbitrary precision floating point values.

7 EXPERIMENTAL RESULTS

We evaluate ESP in terms of its compatibility with popular photo services, robustness against ML labelers, its performance overhead, and its usability. Unless otherwise indicated, we ran our tests with 2500 JPEG images selected from the Open Images Dataset V5 [23]. The selection process consisted of randomly choosing $N = 2500$ rows from the dataset CSV file, discarding the D_N number with dead links, then selecting $N - D_N$ new images and repeating this process until the number of unique images totaled N. To avoid Google Photos resizing images, ESP resized any greater than 16 MP (4920×3264) using bilinear downsampling, and saved the result as an 85 quality JPEG. All performance tests were executed on a Samsung Galaxy S7 smartphone with a Snapdragon 820 processor and 4 GB RAM on Android 8.0 using our modified Simple-Gallery app retrofitted with the Fresco and libjpeg-turbo libraries. Internet access was via the smartphone's WiFi connected to a Verizon Quantum Gateway G1100 5 GHz WiFi router with a Verizon FiOS 300/300 Mbps residential fiber optic connection.

7.1 Compatibility and Interoperability

We ensured that ESP is compatible with popular cloud photo services, namely Google Photos, Flickr, and Imgur. We randomly selected 100 of the 2500 images randomly selected from Open Images Dataset and encrypted them using ESP. For each service, we we uploaded the encrypted images, waited a period of time to ensure the images were processed by the service, such as applying compression, then downloaded, decrypted and manually imspected each image. In general, ESP images are compatible with any photo hosting service if they are not resized or if the full resolution versions are available. Flickr displays small resized images but also provides links to the originals. Links to full size versions can be used by ESP to correctly decrypt images. Most services have an arbitrary maximum limit on file size or image dimensions but this has little bearing on ESP which can simply resize images to each service's limits. All services apply compression of varying strengths, but images remain compatible with ESP in the sense that they do not suffer from visual artifacts or corruption beyond what is normally caused by JPEG compression. We manually inspected the images on a high resolution display and found that the differences in quality for most images compared to the source images were imperceptible unless we greatly magnified them, and even then it was difficult to say which looked definitively better from a psychovisual perspective; it was more a matter of individual preference. For the remaining experiments, we focused on Google Photos.

For Google Photos, we also confirmed that ESP has acceptable image quality across all 2500 images in our sample set. To provide a

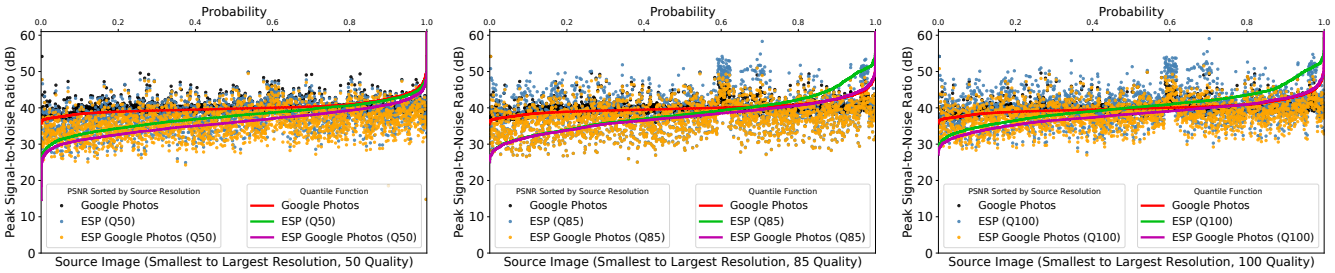

Figure 5: Image quality measured using PSNR relative to source images.

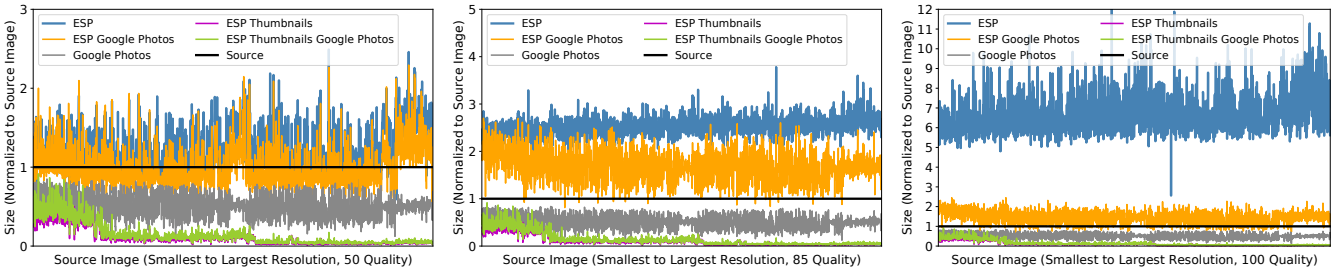

Figure 6: Image file size overhead normalized against source images.

quantitative measure of image quality, we measured the peak signal-to-noise ratio (PSNR) to compare the following against the source images: source images processed by Google Photos (*Google Photos*), decrypted ESP images (*ESP*), and ESP images processed by Google Photos before decryption (*ESP Google Photos*). For ESP, we obtained measurements for encrypting images using three levels of JPEG quality, 50, 85, and 100. Figure 5 shows the PSNR for each; a higher PSNR suggests that the level of noise in the image is more similar to the original and is therefore of better image quality. Each graph shows PSNR for each individual image ordered from smallest to largest image resolution, as well as a quantile function, which is an inverse cumulative distribution function (CDF). *Google Photos*'s average and median PSNRs were 40 dB. *ESP*'s average and median PSNRs were both 38 dB for 50 quality, 39 dB and 38 dB for 85 quality, and both 40 dB for 100 quality. *ESP Google Photos*'s average and median values were both 36 dB for 50 quality, both 37 dB for 85 quality, and both 38 dB for 100 quality.

Although ESP has some effect on image quality from compressing the grayscale ciphertext images, the PSNRs for *ESP* and *ESP Google Photos* were on average not that different from using Google Photos directly. Industry recommendations indicate that a PSNR between 30 to 50 dB is considered good, with higher being better [47]. On average, both *ESP* and *ESP Google Photos* provide PSNRs within that range. As the results indicate, ESP users may choose higher JPEG quality settings for better image quality as measured by the higher PSNRs for higher levels of JPEG quality.

A small minority of images for *ESP Google Photos* images had PSNRs below 30 dB. For example, the worst case PSNR was 25 dB for 85 quality. We manually inspected the images with PSNRs lower than 30 dB and observed two things. First, the visual quality of these images compared to the source images was not noticeably different from the other images we manually inspected as part of our compatibility testing. Second, the lower PSNRs occurred for images that could be described as being low quality images in the first place, in the sense that the source images were generally low resolution and blurry. This suggests that the lower PSNRs are unlikely to occur for real photos of interest that are encrypted using ESP and stored using Google Photos.

For some images, *ESP* has a lower PSNR compared to *ESP Google Photos*. In these cases, the noise introduced by ESP's intermediate compression was specifically reduced by Google Photos' processing pipeline, which suggests that a noise filter is applied to uploaded images. ESP clients could also use noise filters or similar algorithms to improve image quality if an unusually noisy picture is detected.

Since one of ESP's threats is the ML classifiers used by cloud services, we ensured that they fail to correctly label ESP images. We ran Google's ML Kit image labeler on our test images and their ESP-encrypted versions. We then compared the labels to verify if any matched. ML Kit labeled the encrypted images with "Pattern" which none of the images contained. Some encrypted images also experienced other false positives, having labels unrelated to the originals.

7.2 Performance Overhead

We compared the performance of using Google Photos directly versus using Google Photos with ESP. First, we compared performance in terms of the image file sizes uploaded and downloaded from Google Photos. For ESP, we used the same encrypted images at the three levels of JPEG quality, 50, 85, and 100, discussed in Section 7.1. ESP also creates an encrypted thumbnail for each image, which was done by first scaling source images to the 1/8 factor nearest to a target dimension of 400×400 pixels; resizing JPEGs this way is significantly faster than downsampling them to precisely fit within a 400×400 pixel box. Thumbnails were encrypted at JPEG quality 50. We only created thumbnails for source images larger than 800×800 pixels.

Figure 6 shows the measurements for the 2500 images, with file sizes normalized to the respective source image file sizes; smaller is better. When using Google Photos directly, the uploaded image file is the source image (*Source*), and the downloaded image file is after Google Photos compresses the image (*Google Photos*). When using Google Photos with ESP, the uploaded image files consist of the encrypted ESP image files (*ESP*) and the encrypted ESP thumbnails (*ESP Thumbnails*), and the downloaded image files consist of the encrypted ESP images after they are compressed by Google Photos (*ESP Google Photos*), and encrypted ESP thumbnails after they are processed by Google Photos (*ESP Thumbnails Google Photos*). Although ESP generates three separate grayscale JPEG images, we treat them as one and measure the sum total of all the image file sizes.

Using ESP, file size overhead increases with ESP's JPEG quality setting. The average file size for *ESP* was 1.2, 2.5, and 6.5 times the source image file size for 50, 85, and 100 quality, respectively, quantifying the file size overhead for the encrypted image that is uploaded to Google Photos. File size overhead for *ESP Thumbnails* was negligible except for the lowest resolution images since in those cases, thumbnail resolution and file size were no longer insignificant compared to the source images. The file size for *ESP Thumbnails* was on average less than a tenth of the source image size, but in the worst case, it was .6 times the source image file size for the lowest resolution image.

For ESP, other than the JPEG quality setting for encryption, the primary determining factors of the encrypted image file sizes were the permutation of pixel blocks for a given encrypted image and the properties of the source image itself. A shuffled image is unlikely to have sizable regions of consistent color and generally appears close to random noise, thus preventing efficient compression, so encrypted image file sizes are larger. Other factors include properties of the source image, including its original JPEG quality and chroma subsampling format. If the source image itself was saved with a high JPEG quality, i.e. higher than 85, then converting its RGB data to three grayscale images with 85 quality results in greater file size compression. However if the source image is already saved with low JPEG quality, there is little gain from compressing it further, resulting in a larger file size overhead. Similarly, if the source image uses chroma subsampling such as 4:2:0, the CbCr components are a quarter of the size of the Y component but the output encrypted RGB grayscale images are all full size and not downsampled. In contrast, if the source image has full size CbCr components (4:4:4 format), then the output encrypted grayscale images are effectively the same resolution as all of the YCbCr components in the original image, resulting in less inefficiency.

Figure 6 also shows the file size overhead for encrypted images downloaded from Google Photos, which is different from uploaded images because Google Photos compresses them. The average file size for *ESP Google Photos* was .9, 1.7, and 1.4 times the source image file size for 50, 85, and 100 quality, respectively, quantifying the file size overhead for encrypted images downloaded from Google Photos. File size overhead for *ESP Thumbnails Google Photos* was negligible except for the lowest resolution images. In comparison, the average file size for *Google Photos* was half of the source image file size.

The ESP-encrypted images processed by Google Photos are sometimes larger than the ESP images before they are processed by Google Photos. One explanation for this phenomenon is an apparent oversight by Google Photos' image processing and compression pipeline at the time of writing. The original unprocessed encrypted grayscale images output by ESP are true grayscale JPEGs with only one color component, the luminance (Y) channel. However, Google Photos seemingly processes all images as if they are color JPEGs with the YCbCr colorspace. In other words, Google Photos converts true grayscale JPEGs (Y) to color JPEGs (YCbCr), and needlessly populates the Cb and Cr components with the luminance data. Although Google Photos also forces its output images to use 4:2:0 chroma subsampling meaning that the CbCr components are downsampled, they still represent extraneous overhead. An optimization for ESP's encryption would be to output each encrypted grayscale JPEG as a color JPEG while only keeping the useful data in the Y channel, and populating the Cb and Cr channels with zeros.

Although source images are typically not saved with JPEG quality 100, the measurements suggest that this quality setting may be useful for ESP because Google Photos appears to more aggressively compress the large JPEG images with 100 quality compared to the lower JPEG quality settings. For *ESP Google Photos* the average file size overhead for 100 quality was surprisingly less than that for 85 quality. The main downside to using the 100 quality setting would be that it could take much longer to upload the photos to Google if only a low bandwidth network connection is available, since the average file size to upload is much larger for 100 quality than for 85 quality. However, using 100 quality would not need to consume much more local storage space if only the original unencrypted images are retained locally. Note that the file size overhead can greatly vary depending on the properties of the source image. For example, one *ESP* photo in the graph showing JPEG quality 100 size overhead has a distinctly lower file size of about 2.4 times the source image file size, compared to the average of 6.5 times, because it is a high resolution photo of the night sky with nearly no stars visible, making it an almost solid black JPEG photo.

Next, we compared performance in terms of the time to upload to and download from Google Photos. For these measurements, we used the 100 images randomly selected from the 2500 image sample set which we originally used for manually testing compability. When uploading an image, ESP first concurrently encrypts the source image and a thumbnail, which results in three separate grayscale JPEG images for the image and three more for the thumbnail, which are then concurrently uploaded to Google Photos by invoking a Google Photos API to register the uploaded images to Google Photos as Google media items. We measure the entirety of ESP's encryption and uploading time for all six images together (*ESP (Upload)*). When downloading an image, ESP separately downloads and decrypts the encrypted images and thumbnails, so we can measure their respective download and decrypt times (*ESP (Download)* and *ESP Thumbnails (Download)*) separately. For comparison, we also measure the time to upload the unmodified source image to Google Photos (*Google Photos (Upload)*) and download the respective image from Google Photos after it has been compressed and processed (*Google Photos (Download)*).

Figure 7 shows the upload and download times for ESP for 50, 85, and 100 quality, respectively. Although ESP's upload and download times are slightly higher with higher quality, the difference is small, suggesting that, at least for a fast residential Internet connection, the choice of JPEG quality setting for ESP should be based on factors other than upload and download times.

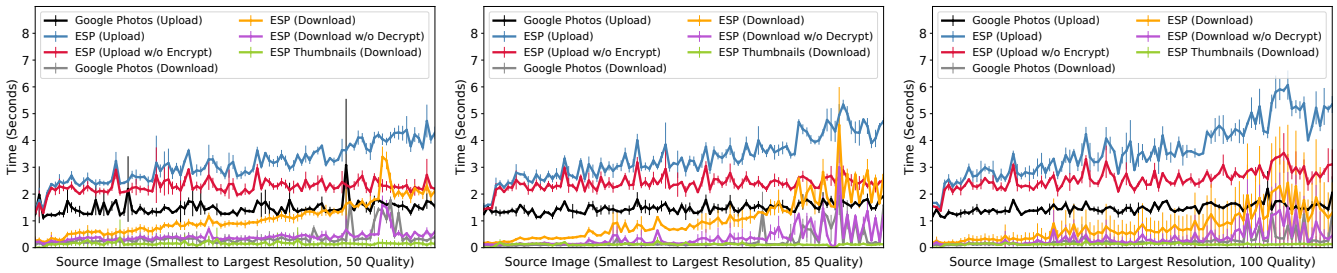

Figure 7: Image upload and download times using Google Photos.

ESP upload and download times are larger than directly using Google Photos, which is not surprising given the added encryption and decryption costs and the fact that the image files being transferred between client and server are also larger. Nevertheless, the difference in both upload and download times between directly using Google Photos and using ESP is at most a few seconds in all cases, though the difference is larger for uploading than downloading. Although the encrypt and upload times are larger than the download and decrypt times, encrypting and uploading occurs in the background and is not in the critical path of the user, who is free to continue using the app and perform other actions. This usage model is no different from the official Google Photos app, which also does background uploads. ESP's higher upload and download times, especially given that uploading can be done in the background, is arguably worth the additional security benefit it provides.

Figure 7 also shows the upload and download times (*ESP (Upload w/o Encrypt)* and *ESP (Download w/o Decrypt)*) without including the time to encrypt and decrypt the grayscale JPEG images, respectively. Comparing *ESP (Upload)* and versus (*ESP (Upload w/o Encrypt)*, we can see that most of the time is spent on uploading rather than encrypting, though encryption costs as a percentage of the total time increases at larger image resolutions. On the other hand, comparing *ESP (Download)* and versus *ESP (Download w/o Decrypt)*, we can see that most of the time is spent on decrypting rather than downloading. While downloading the encrypted images is not much different from downloading the processed source image from Google Photos, it is so fast that decrypting the full size images adds significant overhead.

Downloading and decrypting ESP thumbnails (*ESP Thumbnails (Download)*) is faster than downloading unencrypted images from Google Photos on average. This highlights the importance of leveraging encrypted thumbnails; they exhibit far lower overheads, generally less than 250 ms, to download and decrypt. ESP stores images locally in plaintext after decrypting them so downloading ESP images and thumbnails are a one-time cost per device, but the usage of encrypted thumbnails is still a critical point for providing a smooth user experience when browsing newly synchronized images. A cleverly implemented app can aggressively fetch and decrypt thumbnails, and lazily fetch full size images as the user selects them. Then adjacent images can also be fetched and decrypted in the background. The user will then only notice loading times if swiping quickly through full size images that have not yet been cached locally on the device. The overhead can also be eliminated from the user's perspective if images are preloaded and decrypted locally ahead of time in the background while the user is looking at an image or the app is idle.

Mean	Stdev.	Min.	Q1	Median	Q3	Max
79	18	43	67	83	92	100

Table 1: System Usability Scale summarized scores.

7.3 Usability

ESP's daily operation is transparent to users: the experience is the same as with a regular image app. The main difference in ESP is new device configuration, which includes setting up the first ESP device and adding any others. Either case requires the only significant new interaction from the user compared to a normal app. We consequently performed a small pilot user study of ESP with a focus on the configuration steps necessary for key management.

We administered a user study approved by our institutions' Institutional Review Board (IRB) with 18 study participants who used the IKM system for ESP. Due to the government and IRB enforced prohibition of human interaction in response to the on-going COVID-19 pandemic, we were limited in our ability to hold sessions with more users. Participants were allotted 60 minutes but none used the entire time. Twelve were 20 to 29 years old, four were 30 to 39, one was 40 to 49, and the last was 50 to 59. We asked them to setup the app on two of their Android devices if possible, otherwise we provided them with either a Samsung Galaxy S7 and a Huawei Honor 5X, or an LG K9 and Google Pixel 2 XL. We supplied Google accounts, which we helped users set up on their devices, to use for Google Photos. Users set up the app on one device and completed any required configuration steps. Next, they were asked to repeat this identical process on their second device and then perform the verification step. In our study, users used the verification phrase method. All users finished in 5 to 10 minutes.

Finally, users completed the System Usability Scale (SUS) [48], an industry-standard survey used to evaluate the overall usability of a system. The SUS scores are summarized in Table 1; a higher score correlates with better usability. For example, a score of 85 to 100 suggests *Excellent* usability, 71 to 85 means *Good*, and 51 to 70 means *OK* [1]. Although ESP's median SUS score is 83, in fact 9, or half, of the users gave SUS scores of 85 or higher and therefore rated the system as having *Excellent* usability, while the remainder felt it had between *OK* and *Good* usability. Only one user gave a score lower than 51; however this user expressed disinterest in the concept of photo security and encryption which may have biased their survey choices. Other user comments such as "That was easy" during the sessions suggests that the primary usability overhead of ESP, new device configuration, is simple and intuitive.

8 RELATED WORK

Many approaches have been explored for encrypting images, especially JPEGs. Much of this work in computer vision and signal and image processing was inspired by [21], which first introduced the idea of using chaotic maps such as the logistic map to drive image encryption. Different kinds of chaotic maps have been tried, such as Arnold's cat map [11, 22, 53, 58, 60], and maps have been combined to generate improved distributions [32]. All of these approaches either assume a raw bitmap without accounting for inefficient JPEG compression, or modify and permute DC and AC coefficients in ways that break JPEG compression algorithms.

The concept of format-preserving encryption, where an encryption outputs a ciphertext which retains the formatting and length of the original plaintext [10], has been used for encrypting images [31, 46]. A common approach is to scramble JPEGs within the constraints of its format by modifying DCT coefficients, directly obscuring [17, 30, 50] or scrambling them [55]. Some approaches specifically preserve JPEG image file sizes [26, 39, 40]. While there are many secure JPEG scrambling schemes [26, 30, 32, 39, 50], they are not compatible with existing cloud photo services. Our own experience implementing and testing these encryption schemes confirm that they are incompatible with services such as Google Photos.

While some approaches are designed for cloud storage, they often break the JPEG format and therefore require a third-party service exclusively tailored for their ciphertext format [27, 42, 54, 56, 57]. Others introduce unreasonably large performance overheads [45, 51, 53, 60] or sacrifice significant image quality [51]. Some build on format-preserving encryption by not only encrypting the original image but also outputting recognizable encrypted thumbnails [45, 51], but suffer from performance issues. Others encrypt only specific regions of interest (ROI) within images to obfuscate identities or sensitive material [27, 42, 54, 56, 57]. These approaches tend to encrypt the ROIs of an image, extract them from the remaining unencrypted parts, and store them separately in either generic cloud storage offerings or their own servers. They are not compatible with existing cloud photo services such as Google Photos.

The security of ROI encryption is not well understood [45]. In lay terms, the privacy guarantees are unclear because there are no well-defined models for judging whether an encrypted ROI protects users. An ROI approach could define ROIs as human faces and obscure them, but the remaining visible portions of the image may yield sensitive information such as location, time, relationships, etc. Some solutions ask users to themselves select the ROIs in an image, which is not only tedious but also unreliable as users do not understand the privacy and security implications of ROIs. Although not strictly an ROI-based approach, Fawkes [43] allows users to "cloak" their uploaded photos to shield them against facial recognition software, which ostensibly is also effective against ML labelers in general. Fawkes' strategy resembles ROI approaches in some ways, as it focuses on obscuring human subjects' faces rather then encrypting entire images. Fawkes therefore suffers from potential privacy issues from other threats beyond facial recognition software since environmental and contextual information is left uncloaked. One of Fawkes' important claims is that the cloak consists of changes at the pixel level which are imperceptible to the human eye, but independent evaluations observe significant unwanted visual modifications to photos [29]. In contrast to Fawkes, ESP encrypts entire photos, thereby shielding users against any kind of adversary, with negligible effects on visual quality.

ESP's encryption algorithm is inspired in part by a previous encrypt-then-compress strategy [13], but that approach suffers from two major problems that make it unworkable for use with cloud photo services. First, it is not compatible with services such as Google Photos because it modifies pixel values in a way that results in corruption after compression. Second, it results in a massive increase in file size and a 3× increase in image dimensions. The increase in image dimensions of their ciphertext images reduces users' effective maximum upload dimensions for Google Photos from 16 MP to 5.3 MP. ESP introduces a new encryption algorithm which is compatible with Google Photos' compression and does not have file and image size problems, avoiding prematurely bumping into the 16 MP Google Photos limit. ESP also introduces other key features to support cloud photo services, such as encrypted thumbnails, photo sharing, and end-user key management.

Previous image encryption approaches do little, if anything, for key management. Some briefly suggest that the secret (private key or password) used to decrypt images on other devices—the user's own or another person's—for viewing should be distributed via a secondary out-of-band channel which is never specified or evaluated [42, 45, 46]. IKM builds on E3 [35], which was introduced for email encryption. However, E3 relies on the trustworthiness of the email server used for communications and requires re-encrypting all of a user's emails every time a device is added or removed. ESP requires no trust in servers, and only re-encrypts photos when a user loses a device.

9 CONCLUSIONS

Easy Secure Photos (ESP) makes it possible for users to encrypt their images and use them with existing cloud photo services such as Google Photos, thereby providing privacy against cloud providers and other adversaries. ESP achieves this with purely client-side modifications by introducing a format-preserving encryption method for JPEG images and a unique key management solution which leverages the cloud photo service itself rather than any third parties. Moreover, the encryption method is compatible with compression algorithms used by real services such as Google Photos. We have implemented ESP by integrating it in an existing Android photos app and evaluated its security, performance, and usability with existing cloud photo services such as Google Photos. Our results show that ESP (1) is compatible with popular cloud photo services, including Google Photos and Flickr, (2) is resistant to various attacks including differential cryptanalysis, (3) maintains good image quality for encrypted images even after being processed through Google Photos' image processing and compression pipeline, (4) incurs only modest overhead on upload and download times when used with Google Photos, and (5) is easy to use as encryption and decryption is transparent to users, setting up a device to use ESP is simple, and everyday usage of ESP is no different from a regular photos app.

10 ACKNOWLEDGMENTS

This work was supported in part by NSF grants CNS-1717801, CNS-1563555, CCF-1918400, and CNS-2052947.

REFERENCES

[1] J. M. Aaron Bangor, Philip Kortum. Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. In *Journal of Usability Studies*, volume 4, pages 114–123. JUS, May 2009.

[2] M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *Proceedings of the 2005 International Conference on Topics in Cryptology*, CT-RSA'05, pages 191–208, Berlin, Heidelberg, 2005. Springer-Verlag.

[3] L. Abrams. Google Bug Sent Private Google Photos Videos to Other Users. https://www.bleepingcomputer.com/news/google/google-bug-sent-private-google-photos-videos-to-other-users/, Feb. 2020.

[4] J. Alakuijala, R. Obryk, O. Stoliarchuk, Z. Szabadka, L. Vandevenne, and J. Wassenberg. Guetzli: Perceptually Guided JPEG Encoder. *CoRR*, abs/1703.04421, Mar. 2017.

[5] J. Alakuijala, R. Obryk, O. Stoliarchuk, Z. Szabadka, L. Vandevenne, and J. Wassenberg. google/guetzli: Perceptual JPEG encoder. https://github.com/google/guetzli, Jan. 2020.

[6] Apple Inc. Photos - Private, on-device technologies to browse and edit photos and videos on iOS and iPadOS. https://www.apple.com/ios/photos/pdf/Photos_Tech_Brief_Sept_2019.pdf, Sept. 2019.

[7] Apple Inc. Core ML - Machine Learning - Apple Developer. https://developer.apple.com/machine-learning/core-ml/, Jan. 2020.

[8] Apple Inc. Use FileVault to encrypt the startup disk on your Mac - Apple Support. https://support.apple.com/en-us/HT204837, Jan. 2020.

[9] BBC News. Trend Micro rogue employee exposes customer data - BBC News. https://www.bbc.com/news/technology-50315544, Nov. 2019.

[10] J. Black and P. Rogaway. Ciphers with Arbitrary Finite Domains. In *Cryptographers' Track at the RSA Conference*, pages 114–130. Springer, Feb. 2002.

[11] G. Chen, Y. Mao, and C. K. Chui. A symmetric image encryption scheme based on 3D chaotic cat maps. *Chaos, Solitons & Fractals*, 21(3):749–761, July 2004.

[12] T. Chuman, K. Kurihara, and H. Kiya. On the Security of Block Scrambling-Based EtC Systems against Extended Jigsaw Puzzle Solver Attacks. *IEICE Transactions on Information and Systems*, 101(1):37–44, Jan. 2018.

[13] T. Chuman, W. Sirichotedumrong, and H. Kiya. Encryption-Then-Compression Systems Using Grayscale-Based Image Encryption for JPEG Images. *IEEE Transactions on Information Forensics and Security*, 14(6):1515–1525, June 2019.

[14] J. Cox. Snapchat Employees Abused Data Access to Spy on Users - VICE. https://www.vice.com/en_uk/article/xwnva7/snapchat-employees-abused-data-access-spy-on-users-snaplion, May 2019.

[15] J. Cox. Ring Fired Employees for Watching Customer Videos - VICE. https://www.vice.com/en_us/article/y3mdvk/ring-fired-employees-abusing-video-data, Jan. 2020.

[16] J. Cox and M. Hoppenstedt. Sources: Facebook Has Fired Multiple Employees for Snooping on Users - VICE. https://www.vice.com/en_us/article/bjp9zv/facebook-employees-look-at-user-data, May 2018.

[17] F. Dufaux and T. Ebrahimi. *Toward a secure JPEG*, volume 6312 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, page 63120K. Society of Photo-Optical Instrumentation Engineers, 2006.

[18] Facebook Open Source. Fresco - an image management library. | fresco. https://frescolib.org/, Jan. 2020.

[19] R. A. Fisher and F. Yates. *Statistical Tables: For Biological, Agricultural and Medical Research*. Oliver and Boyd, 1938.

[20] Free Software Foundation. The GNU MP Bignum Library. https://gmplib.org/, Jan. 2020.

[21] J. Fridrich. Image Encryption Based On Chaotic Maps. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, volume 2, pages 1105–1110, Oct. 1997.

[22] C. Fu, J.-B. Huang, N.-N. Wang, Q.-B. Hou, and W. Lei. A Symmetric Chaos-Based Image Cipher with an Improved Bit-Level Permutation Strategy. *Entropy*, 16, Jan. 2014.

[23] Google, Inc. Download Open Images V5. https://storage.googleapis.com/openimages/web/download.html, Jan. 2020.

[24] Google, Inc. Google Terms of Service – Privacy & Terms – Google. https://policies.google.com/terms, Jan. 2020.

[25] Google, Inc. ML Kit | Google Developers. https://developers.google.com/ml-kit, Jan. 2020.

[26] J. He, S. Huang, S. Tang, and J. Huang. JPEG Image Encryption With Improved Format Compatibility and File Size Preservation. *IEEE Transactions on Multimedia*, 20(10):2645–2658, Oct. 2018.

[27] J. He, B. Liu, D. Kong, X. Bao, N. Wang, H. Jin, and G. Kesidis. PUPPIES: Transformation-Supported Personalized Privacy Preserving Partial Image Sharing. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 359–370, June 2016.

[28] S. Heyman. Photos, Photos Everywhere - The New York Times. https://www.nytimes.com/2015/07/23/arts/international/photos-photos-everywhere.html, July 2015.

[29] K. Hill. This Tool Could Protect Your Photos From Facial Recognition - The New York Times. https://www.nytimes.com/2020/08/03/technology/fawkes-tool-protects-photos-from-facial-recognition.html?action=click&module=News&pgtype=Homepage, Aug. 2020.

[30] T. Honda, Y. Murakami, Y. Yanagihara, T. Kumaki, and T. Fujino. Hierarchical Image-scrambling Method with Scramble-level Controllability for Privacy Protection. *2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1371–1374, Aug. 2013.

[31] Hongjun Wu and Di Ma. Efficient and secure encryption schemes for JPEG2000. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages V–869, May 2004.

[32] Z. Hua, Y. Zhou, and H. Huang. Cosine-transform-based chaotic system for image encryption. *Information Sciences*, 480:403 – 419, 2019.

[33] P. Juola and P. Zimmermann. Whole-word phonetic distances and the pgpfone alphabet. In *Proceeding of 4th International Conference on Spoken Language Processing (ICSLP '96)*, volume 1, pages 98–101, Philadelphia, PA, USA, Oct. 1996. IEEE.

[34] J. Kastrenakes. Apple denies iCloud breach in celebrity nude photo hack - The Verge. https://www.theverge.com/2014/9/2/6098107/apple-denies-icloud-breach-celebrity-nude-photo-hack, Sept. 2014.

[35] J. S. Koh, S. M. Bellovin, and J. Nieh. Why Joanie Can Encrypt: Easy Email Encryption with Easy Key Management. In *Proceedings of the 14th EuroSys Conference 2019*, pages 2:1–2:16, Dresden, Germany, Mar. 2019.

[36] libjpeg-turbo. libjpeg-turbo | main / libjpeg-turbo. https://libjpeg-turbo.org/, Jan. 2020.

[37] H. McCracken. How Google Photos reached a billion users. https://www.fastcompany.com/90380618/how-google-photos-joined-the-billion-user-club, July 2019.

[38] Microsoft. Finding your BitLocker recovery key in Windows 10. https://support.microsoft.com/en-us/help/4530477/windows-10-finding-your-bitlocker-recovery-key, Jan. 2020.

[39] K. Minemura, Z. Moayed, K. Wong, X. Qi, and K. Tanaka. JPEG Image Scrambling Without Expansion in Bitstream Size. In *2012 19th IEEE International Conference on Image Processing*, pages 261–264, Sept. 2012.

[40] X. Niu, C. Zhou, J. Ding, and B. Yang. JPEG Encryption with File Size Preservation. In *2008 International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 308–311, Aug. 2008.

[41] P. Privacy. P3 image. https://www.p3image.com/, Nov. 2019.

[42] M.-R. Ra, R. Govindan, and A. Ortega. P3: Toward Privacy-preserving Photo Sharing. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI '13)*, pages 515–528, Berkeley, CA, USA, Apr. 2013. USENIX Association.

[43] S. Shan, E. Wenger, J. Zhang, H. Li, H. Zheng, and B. Y. Zhao. Fawkes: Protecting Privacy against Unauthorized Deep Learning Models. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1589–1604. USENIX Association, Aug. 2020.

[44] SimpleMobileTools. SimpleMobileTools/Simple-Gallery: Browse your memories without any interruptions with this photo and video gallery. https://github.com/SimpleMobileTools/Simple-Gallery, Jan. 2020.

[45] K. Tajik, A. Gunasekaran, R. Dutta, B. Ellis, R. B. Bobba, M. Rosulek, C. V. Wright, and W. Feng. Balancing Image Privacy and Usability with Thumbnail-Preserving Encryption. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*, Feb. 2019.

[46] M. Tierney, I. Spiro, C. Bregler, and L. Subramanian. Cryptagram: Photo Privacy for Online Social Media. In *COSN 2013 - Proceedings of the 2013 Conference on Online Social Networks*, pages 75–87. Association for Computing Machinery, Oct. 2013.

[47] I. T. Union. Objective perceptual multimedia video quality measurement in the presence of a full reference. Recommendation, International Telecommunication Union, Geneva, CH, Aug. 2008.

[48] U.S. Department of Health & Human Services. System usability scale (sus), 2015.

[49] A. Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *Proceedings of the 8th USENIX Security Symposium*, pages 169–184, Washington, D.C., USA, Aug. 1999. USENIX Association.

[50] K. Wong and K. Tanaka. DCT based scalable scrambling method with reversible data hiding functionality. In *2010 4th International Symposium on Communications, Control and Signal Processing (ISCCSP)*, pages 1–4, Mar. 2010.

[51] C. V. Wright, W.-c. Feng, and F. Liu. Thumbnail-Preserving Encryption for JPEG. In *Proceedings of the 3rd ACM Workshop on Information Hiding and Multimedia Security*, IH & MMSec '15, pages 141–146, New York, NY, USA, June 2015. ACM.

[52] Y. Wu, J. P. Noonan, and S. Agaian. NPCR and UACI Randomness Tests for Image Encryption. *Cyber Journals: Multidisciplinary Journals in Science and Technology, Journal of Selected Areas in Telecommunications (JSAT)*, 1(2):31–38, Apr. 2011.

[53] W. Xingyuan and Z. Hongyu. Cracking and Improvement of an Image Encryption Algorithm Based on Bit-Level Permutation and Chaotic System. *IEEE Access*, 7:112836–112847, 2019.

[54] L. Yuan, P. Korshunov, and T. Ebrahimi. Privacy-Preserving Photo Sharing based on a Secure JPEG. In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, volume 2015, Apr. 2015.

[55] L. Yuan, P. Korshunov, and T. Ebrahimi. Secure JPEG Scrambling Enabling Privacy in Photo Sharing. In *2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, volume 04, pages 1–6, May 2015.

[56] L. Yuan, D. McNally, A. Küpçü, and T. Ebrahimi. Privacy-preserving photo sharing based on a public key infrastructure. In A. G. Tescher, editor, *Applications of*

Digital Image Processing XXXVIII, volume 9599, pages 515 – 527. International Society for Optics and Photonics, SPIE, Sept. 2015.

[57] L. Zhang, T. Jung, C. Liu, X. Ding, X.-Y. Li, and Y. Liu. POP: Privacy-Preserving Outsourced Photo Sharing and Searching for Mobile Devices. In *2015 IEEE 35th International Conference on Distributed Computing Systems*, pages 308–317, June 2015.

[58] Y.-Q. Zhang and X. Wang. Analysis and Improvement of a Chaos-based Symmetric Image Encryption Scheme Using a Bit-level Permutation. *Nonlinear Dynamics*, 77:687–698, Aug. 2014.

[59] M. Zhu, T. Moataz, and S. Kamara. Pixek. https://pixek.io/, Nov. 2019.

[60] Z.-l. Zhu, W. Zhang, K.-w. Wong, and H. Yu. A Chaos-Based Symmetric Image Encryption Scheme Using a Bit-Level Permutation. *Inf. Sci.*, 181(6):1171–1186, Mar. 2011.