Cryptagram: Photo Privacy for Online Social Media

Matt Tierney, Ian Spiro, Christoph Bregler, Lakshminarayanan Subramanian
New York University
{tierney, spiro, bregler, lakshmi}@cs.nyu.edu

http://cryptagr.am

ABSTRACT

While Online Social Networks (OSNs) enable users to share photos easily, they also expose users to several privacy threats from both the OSNs and external entities. The current privacy controls on OSNs are far from adequate, resulting in inappropriate flows of information when users fail to understand their privacy settings or OSNs fail to implement policies correctly. OSNs may further complicate privacy expectations when they reserve the right to analyze uploaded photos using automated face identification techniques.

In this paper, we propose the design, implementation and evaluation of Cryptagram, a system designed to enhance online photo privacy. Cryptagram enables users to convert photos into encrypted images, which the users upload to OSNs. Users directly manage access control to those photos via shared keys that are independent of OSNs or other third parties. OSNs apply standard image transformations (JPEG compression) to all uploaded images so Cryptagram provides an image encoding and encryption mechanism that is tolerant to these transformations. Cryptagram guarantees that the recipient with the right credentials can completely retrieve the original image from the transformed version of the uploaded encrypted image while the OSN cannot infer the original image. Cryptagram's browser extension integrates seamlessly with preexisting OSNs, including Facebook and Google+, and currently has over 400 active users.

Categories and Subject Descriptors

K.4.1 [**Public Policy Issues**]: Privacy; K.6.5 [**Security and Protection**]: Unauthorized Access

Keywords

photo privacy; online social media

1. INTRODUCTION

Petabytes of imagery data have been posted by users to Online Social Networks (OSNs) with Facebook alone receiving over 250 million photo uploads per day [24], storing 10,000 times more photos than the Library of Congress [14]. Users feel the need internally and externally (peer pressure) to share photos on OSNs given the convenience of usage and their immense popularity [21, 2, 25, 27, 34]. Users share personal and potentially sensitive photos on OSNs, thereby exposing users to a wide range of privacy threats from external entities and the OSN itself [9, 33, 31]. We consider two basic factors that trigger privacy concerns for end-users in OSNs.

COSN'13, October 7–8, 2013, Boston, Massachusetts, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2084-9/13/10 ...$15.00.
http://dx.doi.org/10.1145/2512938.2512939.

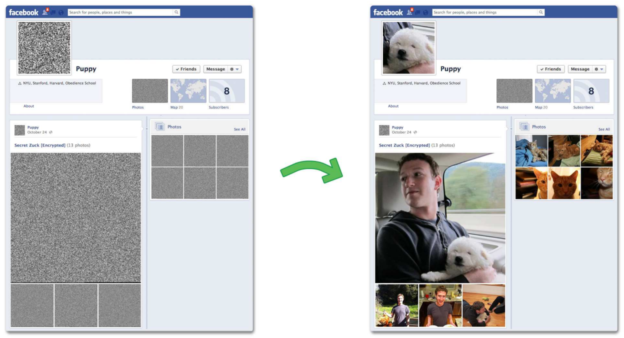


Figure 1: Example Cryptagram user experience. On the left, we show a social network with embedded Cryptagrams, uploaded by a user. A browser extension decrypts the images in place as shown on the right.

User/System Errors: A user who uploads an image to an OSN may wish to share it with only a select group of people, which OSNs partially satisfy with privacy settings. Contextual integrity [23] would state that the user is attempting to implement her own notion of appropriate information flows. But a recent study confirmed that Facebook users' impression of their sharing patterns and their true privacy settings are often inconsistent [18]. Moreover, an OSN may fail to correctly enforce their privacy settings, such as the case when Facebook exposed its own CEO's private photos in a systemwide glitch [9].

Face Identification: A passive observer or a hosting OSN could extract large volumes of online photo uploads, indexing and discovering images within a corpus that belong to a specific user [33]. Mining of photo corpora can lead to the unexpected disclosure of individuals' locations or their participation in events. Facial data mining incidents have resulted in litigation against OSNs and further weakened the integrity of the relationship between the social network and the individual [31].

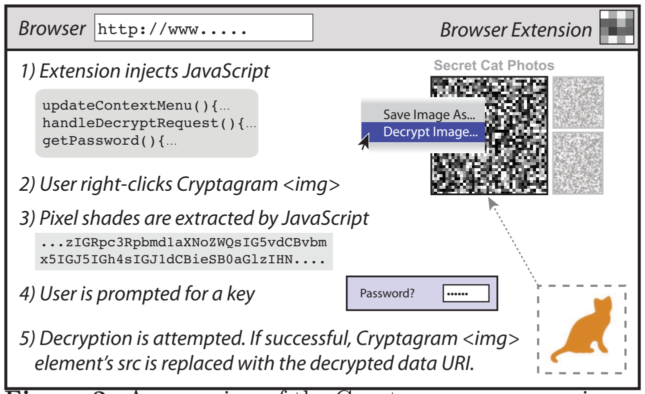
In this paper, we present the design, implementation and evaluation of Cryptagram, a system designed to address these photo privacy concerns in OSNs. A basic design goal of Cryptagram is to build a usable solution that can offer strong privacy guarantees for end-users that remains backwards compatible with existing OSN user interface designs. To maintain the conventional feel of an OSN, Cryptagram uses the abstraction of an image interface (RGBA pixels) to manipulate the core image formats used by OSNs. Cryptagram leverages an end-to-end encryption system to transport images, which are uploaded to OSNs. Figure 1 illustrates a specific example of how Cryptagram represents normal images as encrypted images.[1]

A challenge in the design of such an end-to-end image encryption/decryption mechanism is to be resilient to image transformations by the OSN. For instance Facebook converts all uploaded photos, regardless of original format, to JPEG, choosing quality settings without user input. The recipient of a Cryptagram image must be able to retrieve the original image from the OSN-transformed version. In this paper, we describe the notion of q, p-Recoverability (Section 3.1) which formalizes the aforementioned property that enables the assessment of embedding protocol designs. We describe a class of JPEG embedding protocols that can achieve the q, p-Recoverability property for different JPEG quality transformation levels. The top-down techniques that we discuss for designing q, p-Recoverable protocols can also be applied to lossless image compression formats.

Cryptagram addresses a problem that is fundamentally different from conventional image steganography. While steganography aims to hide data in plainsight and *avoid detection* [11], Cryptagram makes obvious that it is hiding data with the added aim of efficiently transporting bits in the image medium while being robust to image transformations. Despite the differences in problem definition, steganography does have the same mechanical use as Cryptagram for transporting bits in an image. When comparing the effective efficiency of our approach to steganography, Cryptagram packs many more bits per pixel (Section 6).

Cryptagram differs significantly from the recent work on photo privacy, P3 [29]. Unlike P3, Cryptagram operates completely in encrypted bit space and does not reveal sensitive cleartext data of photos to external entities (Section 8). Cryptagram also does not rely on third-party providers for providing photo privacy.

We present several key results in our evaluation. For JPEG Cryptagram images uploaded to Facebook, we find that JPEG compression quality for those high entropy images is in the range of 76 to 86 (for natural images, usually quality is 74). Given these recompression target ranges, we demonstrate JPEG embedding protocols that, in tandem with error-correcting codes, achieve an effective efficiency of 3.06 bits per pixel, which is 2.68× greater than the best related work. We also summarize a study of recoverability when recompressng already compressed images. We further illustrate a design point comparison of recoverability versus filesize expansion when comparing JPEG and webp lossy compression formats.



1) Extension injects JavaScript

```
updateContextMenu(){...
handleDecryptRequest(){...
getPassword(){...
```

2) User right-clicks Cryptagram <img>

3) Pixel shades are extracted by JavaScript

```
...zIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbm
x5IGJ5IGh4sIGJ1dCBieSB0aGlzIHN....
```

4) User is prompted for a key

5) Decryption is attempted. If successful, Cryptagram <img> element's src is replaced with the decrypted data URI.

Figure 2: An overview of the Cryptagram user experience.

Our end-to-end Cryptagram system has been deployed to the web. Figure 2 summarizes a user's experience with the current decoder. Our decoder browser extensions integrate seamlessly with existing OSNs including Facebook and Google+ while being compatible with their complicated DOM structures. We have nearly 400 active users of our decoder extension and over 300 users have agreed to our IRB-approved study through which they submit high-level data about their on-going image encrypting and decrypting habits with Cryptagram.

2. SYSTEM MODEL

2.1 Problem Statement

The basic problem that Cryptagram aims to address can be stated as follows: Two users U and V are members in an OSN and have a shared key k (e.g., password), independent of the OSN. U wants to use the OSN to share an image I with V but does not intend to upload I to the OSN since the OSN or other unauthorized users may also be able to view I. Instead, U needs an encryption mechanism that can transform I into an encrypted image I', which V can retrieve, decrypt and obtain I (using the shared key k). The key challenge is that when U uploads an encrypted image I', the OSN can apply image transformations and the image V downloads may significantly differ from I'. Hence, the sharing mechanism needs to be resilient to image transformations.

To better understand the transformation-resilient image encryption problem, we outline the basic image encoding and decoding steps used by Cryptagram and the property that Cryptagram aims to achieve:

- A user U encrypts a to-be-shared cleartext image, I, using a strong block cipher with a secret key, k, to produce a byte sequence, $E(I, k)$. This k may be human-readable (a password) or part of a hybrid cryptosystem in which the k is generated and shared using public key cryptography.
- An image encoding protocol, C, embeds $E(I, k)$ into the spatial domain of an image, $I_m = C(E(I, k))$ which the OSN transforms as $T(I_m)$. In this paper, we restrict T to be the identity transformation (lossless format) or a standard lossy image compression. We use JPEG for the lossy format

[1]In this example, a user has uploaded a single Cryptagram image per image in this Figure. OSNs typically recompress and resize images within their backend infrastructure, presenting the most bandwidth-friendly version (thumbnails) as they deem appropriate. In order to render the decrypted Cryptagrams for thumbnails, Cryptagram infers from URL of the thumbnail how to fetch the full-size image, which Cryptagram fetches and decompresses when a user indicates our extension should do so.

in much of the evaluation since that is a commonly used standard across OSNs.

• An OSN user V who is an authorized recipient of the image needs to be aware of the block cipher secret key k. Using this key k and an image decoding algorithm, V should be able to successfully decode I from a transformed version $T(I_m)$. Here, we aim to achieve *recoverability* (intuitively, data integrity of the embedded message), which in the case of a lossless format is tautologically true sans other transformations. For JPEG, we aim for q, p-Recoverability property: given a minimum quality level q of the transformed image $T(I_m)$, the decoding protocol should enable V to decode the original image I with high probability, p. We denote this recoverability probability using p, where the ideal case is when $p = 1$; however, achieving $p = 1$ may not always be feasible.

• Adversary, Eve, who passively observes $T(I_m)$ should not learn anything about I.

2.2 Design Goals

The aforementioned problem statement highlights two key design goals of Cryptagram: ***data confidentiality*** and ***probabilistic data integrity for lossy images***. For data confidentiality, Cryptagram leverages trusted algorithms that users may use to ensure that data has been encoded with a strong proofs of security. For probabilistic data integrity, since Cryptagram aims to create images for use on OSNs, we can relax the constraint of traditional information security data integrity [10] for lossy image formats such as JPEG. This relaxation enables the Cryptagram design to incorporate features that demonstrate a spectrum of data integrity when a social network transforms uploaded images.

But given these goals, should U and V use an OSN to share personal images at all? We accept as a constraint that many users desire the convenience of social networks [21]. This "convenience" constraint raises the following design goals that Cryptagram meets:

Usable: We aim to offer a system that affords users intuitive privacy on top of the images that they share on OSNs. While several offline approaches exist to preserve privacy (e.g., PGP [39]), Cryptagram makes it possible for users to create and share private photos without disrupting the OSN experience.

Widely Deployable and Applicable: To gain wide adoption, we have created a cross-browser, cross-platform, cross-image format system that enables Cryptagram to be used as both an encoder and decoder. The reduced friction to creating and accessing Cryptagram images removes the barrier to broader use of the technology.

Efficient: Compared to alternative methods, we present a system that offers significantly more data storage for a given file size or image dimensions.

2.3 Security Overview

2.3.1 Threat #1: Facial Data Mining

In this threat, the adversary is the OSN, whose aim is to execute facial data mining algorithms on uploaded images. In recent years, as social networks' corpi of images have grown dramatically, this is a serious concern for the privacy-conscious individual.

Approach. We have devised a scheme that reveals no information about the original image to the OSN. As we discuss in Section 4, the use of our embedding algorithm by nature thwarts facial data mining by embedding the cleartext (e.g., facial) data indirectly as encrypted bits in the spatial domain of the transport image.

Security Guarantees. With the use of a block cipher to transform the secret message, Cryptagram retains the strength of the underlying security properties of the chosen algorithm. With the use of public key cryptography users can retain cryptographic strength while leveraging a trusted, separate channel to bootstrap their sharing.

2.3.2 Threat #2: Misconfigured Privacy Controls

OSNs may fail to correctly deploy access control policies or users may accidentally misconfigure confusing access controls. The use of Cryptagram creates a separate channel of communication to ensure, with cryptographic strength, that only intended recipients see the cleartext photo. With the correct use of Cryptagram, an OSN could suffer a full system breach and encrypted images would remain private.

2.3.3 Limitations

Detecting and Blocking Cryptagram Images. Cryptagram does not address the problem of an OSN detecting and inhibiting the upload of all Cryptagram images. Steganography may be proposed in this scenario but problem redefinition and the tradeoff in efficiency make steganography an inappropriate application.

Unsupported Transformations. Though Cryptagram images are robust to varying degrees of JPEG compression, they does not support many other transformations. For example, cropping or rescaling a Cryptagram will generally break its encoding.

Brute-Force Cryptographic Attack. Cryptagram users who choose weak passwords in the symmetric key scheme can be attacked with dictionary or brute-force techniques. To address this limitation, we encourage users to abide by strong password creation techniques [22, 28] when using symmetric key credentials. When using public key cryptography, we encourage users to leverage the use of a public key infrastructure that is coupled with Cryptagram as we follow Key Continuity Management practices [13].

Copy and Paste. Users who gain access to cleartext images can copy and paste those images to whomever they choose. We believe this problem will persist despite any attempts, short of controlling all hardware at the disposal to humans accessing social networks.

3. IMAGE FORMATS IN OSNS

Several image formats are used across OSNs. While Facebook uses only the JPEG format to store images (and, moreover, strips uploaded images of EXIF data), Google+ and other networks allow for a variety of lossless (e.g., PNG) and lossy (e.g., webp) formats. Our goal is to design a generic photo privacy solution that can work across different image formats. While lossless compression techniques are relatively easier to handle, determining an image encryption/decryption mechanism in the face of a lossy transformation is much more challenging. Given the popularity and broad use of JPEG, we use JPEG as our primary image format to describe the design of Cryptagram. We show how Cryptagram can be easily applied for other image formats including lossy image formats like webp.

Our design primarily focuses on embedding data in the spatial dimensions of an image. We define an embedding protocol to be an algorithm that describes the creation of a sequence of bits and how those bits are embedded into the spatial domain (pixels) of an image. We design embedding algorithms that work in a top-down fashion; that is, the data to be embedded is written into the spatial domain of an image on a pixel level rather than in any protocol-specific manner. We believe that a top-down approach allows us to meet the aim for wide deployablility and applicability in terms of implementation, testing and future image formats. The top-down API means that the design of codecs can apply or be tested across multiple formats with ease. When codec design depends on DCT coefficients, for instance, there are non-intuitive programming interfaces that would be required to make that facility addressable to the PNG format and not just JPEG, webp, and other DCT-coefficient based compression algorithms.

Assuming a passive adversary, this approach is a valid solution to the security threats that we outlined in the previous section. This is an especially prudent design choice considering that lossy image transformations will most intuitively aim to preserve higher order features of an image rather than its bit-wise representation.

The generic interface to the image is thus the four possible color channels red, green, blue, and alpha as well as their corresponding bit value. For JPEG, this means up to eight bits per the first three color channels. For PNG, we have up to 16 bits per channel for all four possible channels.

3.1 Defining q, p-Recoverability for JPEG

JPEG image transformations are inherently lossy in nature. With the aim of probabilistic data integrity, we make concrete the goal of relaxing the constraints of traditional notions of information security data integrity [10] for embedding data in JPEG.

We define the *q, p-JPEG Recoverability* (or, simply *q, p-Recoverability*) property of embedding protocols as follows: given a minimum quality level q that an OSN preserves in a transformation T of an uploaded image, an authorized recipient should be able to decode the original image with high probability p, where in the ideal case $p = 1$. The concept of q, p-Recoverability can also be applied to other lossy image transformations though the corresponding attainable values of q and p are dependent on transformation T.

In the context of JPEG images, we define a Cryptagram protocol as a message-JPEG encoder G and JPEG-message decoder G'. Given an input image[2] I, the first step in Cryptagram encoding is to convert the image into an encrypted sequence of bits $m = E(I, k)$, for clear-text image I and a block cipher key k. We refer to the input to the JPEG encoder as a sequence of bits m. Given m, the protocol encodes m in the spatial domain of a JPEG image, $I_m = G(m)$. JPEG (denoted by the function T, its inverse for decompression is T') compresses I_m at quality q to produce a sequence of bits, $T(I_m, q)$.

The recipient uses a two step decoding mechanism to retrieve an encrypted set of bits m': (a) the first step involves using the decompression step T' to produce $T'(T(I_m, q))$; (b) the second step involves using the JPEG-message decoder G' to retrieve an encrypted sequence of bits $m' = G'(T'(T(I_m, q)))$. Ideally, m' should match m; if they do, the recipient can use the secret key k to decrypt m' to retrieve the original input message. However given the lossy nature of the transformations, the message-JPEG encoding and JPEG-message decoding steps may not always succeed. Here, we use the term p to denote the probability that the algorithm successfully decodes the input bit sequence m. Mathematically, we denote this as: $m' =_p m$. If this constraint holds, then we define the protocol to be q, p-Recoverable. By considering a large sample set of input messages, we can statistically estimate the value of p for a given quality threshold q. The aim of Cryptagram is to identify q, p-Recoverable protocols that attain p close to one for low quality values and a high bits per pixel ratio. We summarize these ideas in Figure 3.

[2]We mean that I is a sequence of bits that represent an image format that a browser can render. Notably, Cryptagram's embedding can be a used with any arbitrary message I for delivering a message via the spatial domain of a transport image.

$$I_m = G(m)$$

$$G'(T'(T(I_m, q))) = m' =_p m \implies$$

$$G \text{ is } q, p\text{-Recoverable}$$

Figure 3: q, p-Recoverability in a nutshell.

4. SYSTEM DESIGN

4.1 Lossy Images

To discuss how to embed data into a lossy image, we focus on the JPEG compression algorithm, though our design principles apply to other lossy formats.

How should one embed bits into the spatial domain of an image? To approach this challenge, we develop a mapping of bits to colors for specific pixels in an image. Intuitively, when choosing points (coordinates) in the color space to represent bits, we leverage the observation that the lossy codec may *shift* an initially embedded point (pixel's color) during encoding and decoding an image; however, the sphere in the color space within which that point may move does not overlap with other point-centered spheres. This is to say that when choosing what values to embed and how to coordinate pixel values, protocol designers must be sensitive to the assumption that the lossy codec will shift values within spheres in a color space. This intuition guides our JPEG design discussion below but, more importantly, is the generally applicable principle for Cryptagram protocol design.

The principal unit of embedding in Cryptagram is the Cryptagram pixel block (CPB). Multiple CPBs must fill or pack a 8×8 JPEG pixel block (JPB) for each channel of JPEG (luminance, chrominance red and chrominance blue), which is the "atomic unit" of pixels that undergoes JPEG compression [36]. We consider how to pack bits into the spatial domain of JPEG given two goals: *(1) efficient bit packing (increasing the number of bits per pixel)* and *(2) q, p-Recoverability.*

4.1.1 Embedding in the Spatial Domain

Cryptagram Pixel Blocks.

For protocol design we examine how to manipulate 64 pixels to embed bits efficiently. We embed symbols into the 64-pixel JPBs for each YC_bC_r channel with multiple Cryptagram pixel blocks (CPBs) per JPB. A CPB could be any shape that packs into a JPB. For our discussion, we consider 1×1 and 2×2 CPBs.

The composition of a CPB thus is a shape description, $w \times h$ (width, height) and a set of rules, R, for translating a symbol, x, or set of bits ($x = b_0, \ldots, b_{|x|}$) into red, green, and blue tuples (r, g, b) that we embed in each pixel of the CPB according to the appropriate $RGB \rightarrow YC_bC_r$ conversion. For simplicity, we represent the CPB embeddings for each channel as $L^{R_L}_{w\times h}$, where R_L are the rules that correspond to the channel, L, how to embed x to color values for L.

Because JPEG compression applies different downsampling and quantization matrices to luminance and chrominance channels (but applies the same compression to the two chrominance channels), we express the embedding protocol for a CPB as:

$$\left(Y^{R_Y}_{w_Y\times h_Y}, C^{R_C}_{w_C\times h_C}\right)$$

where Y corresponds to luminance and C to chrominance channels.

The rule set, R provides a large space for Cryptagram protocol designers. Intuitively, the composition of rules becomes a choice of three parameters: (1) how many bits to embed, (2) the number of discretizations to use (for which the number of bits to embed determines the lower-bound) in the color space, and (3) the choice of which discretization values from the color space to use. In short, we determine how many colors to use, what values they represent, and the resulting bitrate.

The JPEG compression algorithm compresses least the luminance channel of the three yielded by the $RGB \rightarrow Y'C_bC_r$ transformation. If we choose only to discretize values in luminance, we have an effective range of $[0, 255]$, which corresponds to "grayscale". We denote this scenario as $\left(Y^{R_Y}_{w_Y\times h_Y}, C^0\right)$, using C^0 to denote that chrominance is not used.

On Chrominance Embedding. When considering the use of the chrominance channels in the embedding protocol, there are several complications to address in this proposal. As described in the JPEG specification, the two chrominance channels are stored with significantly less fidelity than luminance. Both chrominance channels are down-sampled (by default in `libjpeg`, 2:1) and a more aggressive quantization table is used to further reduce the number of bits that need to be stored [36]. Intuitively, the chrominance channels are less efficient for embedding data in the spatial domain.

Encoding Algorithm.

We demonstrate the embedding algorithm in Figure 4. As we discuss the embedding algorithm at a high-level, we will refer to the concrete demonstration in that figure.

The first step of the encoding algorithm transforms the input clear-text image I into a sequence of encrypted bits m using a shared key k such that $m = E(I, k)$. Here, we use a standard block cipher algorithm AES in CCM mode (128 bit keys and 64 bit tag size). The encoding algorithm from this point chooses a small collection of bits at a time and converts these bits into Cryptagram pixel blocks. Figure 4 Step A shows how our example encrypted output message m is the sequence of characters, "Security." Using the base64 representation of the character, we know that the sequence

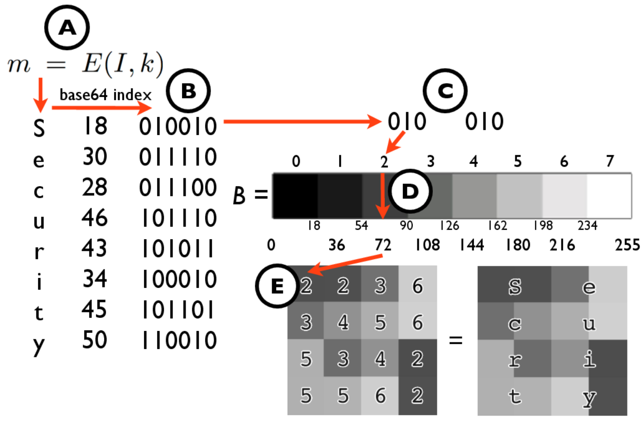


Figure 4: Encoding algorithm illustration. We demonstrate how Cryptagram maps an encrypted message's sequence of bits (Steps A-C) to color values (Step D) and how those correspond to embedded pixels (Step E).

of bits for each character is shown under Step B. We then show in Step C how the sequence of bits for the first character (S's representation as 010010) can be split into two three-bit sequences, the aforementioned "small collection of bits." Using a chunked grayscale color spectrum, we map the three-bits to an index in the array of values. The index's group representative (in this case at Step D, it's the grayscale value of 72) is what is embedded for the appropriate pixels, as shown in Step E. In this example, we continue to pluck off three bits at a time, for Steps B and C, then map those three bits values to grayscale values in Step D. Finally, we continue to embed the values left-to-right, top-to-bottom in this simple example for Step E. We have used a 2×2 CPB for this illustration, which packs perfectly into a standard 8×8 JPB. An alternative format could have used 1×1 CPBs, shading one pixel instead of four in Step E.

Figure 5 illustrates at a high-level where data is embedded in a typical Cryptagram image format.

We make the above example more concrete in the following formalism. An embedding algorithm, a, assumes that b bits will be embedded per CPB. Given b, a has a bijective mapping $B_L : \{0, 1, \ldots, 2^b\} \rightarrow L_a$ where $L_a \subseteq [0, 255]$. Given a bit sequence, s, a uses B_L to map b-length substrings of s to the corresponding L channel values that will be embedded at that particular pixel. Given the notation we have introduced, B is a more specific case of the notion of rule sets R_L we presented earlier. B_L mappings underpin the designs we present in this paper.

We can measure the efficiency of B_L based on b and the size of the channel's CPB to which the output of B_L mapped as $\frac{b}{|\text{CPB}|}$ bits per pixel, where $|\text{CPB}|$ is the number of pixels occupied by the CPB: $|\text{CPB}| = w \times h$.

From our discussion of the embedding protocol and the notion of q, p-Recoverability, we have laid the groundwork for how the designer's choice of protocol parameters ($d_{w,h}$, B, etc.) adjust the effective efficiency of the end-to-end protocol.

Example Encodings for Reasoning about q, p-Recoverability. To demonstrate the tradeoff between efficiency (number of discretizations in B per CPB size) and q, p-Recoverability that we must consider in protocol design, we present two

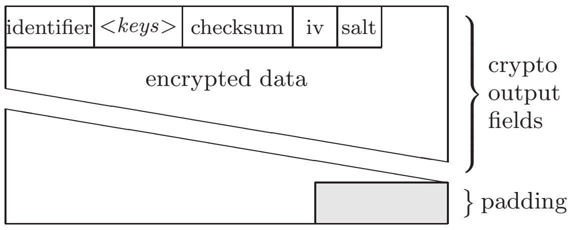


Figure 5: Layout of a Cryptagram. Each box is a sequence of shaded pixels representing the range of values for a particular protocol.

Name	Notation	Luminance-only B Mapping
Bin	$\left(Y^1_{1\times1}, C^0\right)$	$B : \{0,1\} \to \{0,255\}$
Quad	$\left(Y^2_{1\times1}, C^0\right)$	$B : \{0,1,2,3\} \to \{0,85,170,255\}$
Oct	$\left(Y^3_{1\times1}, C^0\right)$	$B : \{0,1,...,7\} \to \{0,36,73,...,255\}$
Hex	$\left(Y^3_{1\times1}, C^0\right)$	$B : \{0,1,...,15\} \to \{0,17,34,...,255\}$

Table 1: We present the B mappings for luminance-only embeddings in order to introduce the Y^n notation as well as illustrate the corresponding luminance values embedded in a Cryptagram using that mapping for the embedding protocol.

examples. The first example uses a $\left(Y^B_{1\times1}, C^0\right)$ CPB. As we translate (according to B) bits from m to successive CPBs color values, we fill the JPB from left-to-right, top-to-bottom, starting in the top-left of the 64 square pixel JPB, covering each channel independently. We can explore multiple color mappings B in order to see how q,p-Recoverability is affected by the $\left(Y^B_{1\times1}, C^0\right)$ CPB and B interaction.

We consider three mappings for B as shown in Table 1. The simplified representations for luminance will be used through this paper. The superscript is the number of bits that can be embedded given the use of equally space values in $[0, 255]$, including extremal values.

Figure 6 illustrates the q,p-Recoverability of these choices. In comparing the best of binary, quadrature, octature, and hexature bits per pixel discretizations for the $\left(Y^B_{1\times1}, C^0\right)$ CPB, we have a sense of how the mapping choices perform relative to one another. Given a social network quality value (for JPEG recompression), we want to choose an embedding that allows for p very close to 1. If we choose the target quality to be 86%, then the values that are actually at $p = 1$ are the Quad and Bin mappings. These yield two and one bits per pixel, respectively. Because we are apt to conservatively choose a quality threshold assuming an OSN may lower their thresholds slightly (e.g., the OSN finds they save enough disk space without causing user experience to suffer too much), we opt to use the Bin approach: $\left(Y^1_{1\times1}, C^0\right)$.

Figure 7 shows the results of our exploration of the chrominance CPB size and the impact of embedding in luminance and chrominance concurrently. We must use 2×2 CPBs in chrominance channels to embed one bit per channel's block (or a cumulative 0.5 bits per pixel gain). We can thus embed in chrominance as a function of the corresponding luminance values.[3] With this approach, we find that embedding more than two values per chrominance channel suffers low q,p-Recoverability. Thus while 4×4 appears to illustrate good q,p-Recoverability in the binary embedding case, we

[3]Notably, if the luminance values are at the extremes (0 or 255), then we do not embed a chrominance value in that pixel since no valid chrominance value exists.

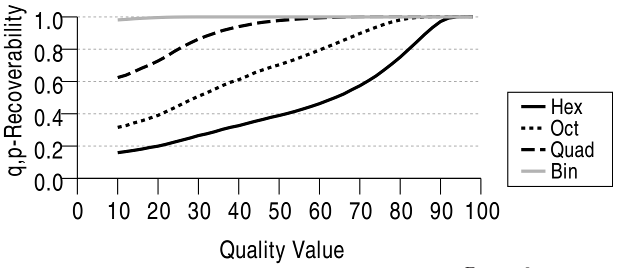


Figure 6: The relative performance of $\left(Y^B_{1\times1}, C^0\right)$ CPBs. We see that more discretizations results in weaker q,p-Recoverability as the quality to which we subject the JPEG to decreases. The tradeoff we must consider is what q,p-Recoverability we want to achieve (what minimum quality do we want a probabilistic guarantee) and how efficient we want for our embedding protocol.

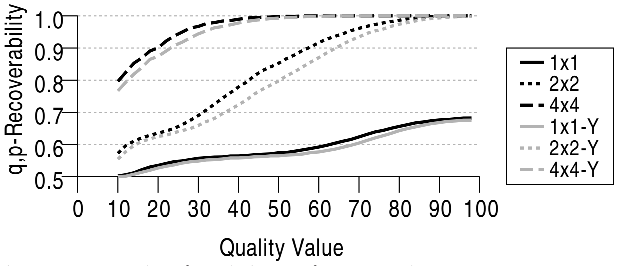


Figure 7: The feasibility of using chrominance to gain additional bits for embedding. All lines correspond to a chrominance B binary mapping. $n \times n$ corresponds to using only chrominance to embed bits using a binary mapping while keeping luminance set to 128. $n \times n$-Y embeds non-128 chrominance values along with chrominance. This plot also highlights the tension of using chrominance in tandem with luminance values. The diminished q,p-Recoverability may appear marginal when we are embedding binary data in the luminance space, but performance degrades significantly. As we explore the applicability of higher bit rates, we must carefully balance the interaction of luminance and chrominance.

think that the efficiency gain is so marginal as to be negligible. Thus we consider the of use $\left(Y^3_{1\times1}, C^1_{2\times2}\right)$ CPBs to gain an additional 0.5 bits per pixel. This gain with chrominance always requires error correction in order to attain q,p-Recoverability that is robust to OSN recompression.

On Decoding. To decode values from a Cryptagram JPEG, the decoder examines pixels in the same order as the encoder. Converting the RGB values to YC_bC_r, the decoding algorithm finds the nearest neighbor for the values in the co-domain of the B mapping for that protocol. The sequence of corresponding domain values is the decoded sequence of bits.

4.1.2 Balancing q,p-Recoverability and Efficiency with Error Correction

Since the nature of protocols that we investigate are probabilistically recoverable (the p in q,p-Recoverability), we consider the use of Error Correcting Codes (ECC) in order to improve the q,p-Recoverability of our protocols while maintaining efficiency of the embedding algorithm. Reed-Solomon codes are of interest to us for their well-understood ECC properties and space efficiency. In our case and especially in Section 6, we use $RS(255, 223)$ protocol, in which we use the $x^8 + x^7 + x^2 + x + 1$, or "0x187", field generator polynomial and 32 bytes for parity in order to recover up to 16 byte errors for a 255 byte transmission. The input to $RS(255, 223)$ is the encrypted output of the block cipher algorithm. With the application of $RS(255, 223)$ then, the q, p-Recoverable protocol directly embeds the output bit stream from $RS(255, 223)$.

From Figure 7, we note how the use of chrominance would always require ECC in order to recover from OSN recompression-induced errors for the CPB case we have highlighted: $\left(Y^3_{1\times1}, C^1_{2\times2}\right)$.

4.2 Lossless Compatibility

Our effort focuses on the JPEG format given its online prevalence, but it's worth noting that our approach is seamlessly compatible with lossless formats such as PNG [8].

In this lossless scenario, we trivially achieve recoverability. The PNG format has a maximum per pixel efficiency of 64 bits per pixel. Each of the four channels, red, green, blue, and alpha, can store 16-bit values. We take 64 bits of sequential data in m, split the 64 bits into four 16-bit segments, then write the respective 16-bit values into each of the four channels of a pixel.

4.3 Easy Key Management and Cryptography

Users have two options for managing access to their photos in Cryptagram: symmetric key cryptosystem or a hybrid (symmetric key and public key) cryptosystem.

In the case of the symmetric key cryptosystem, Cryptagram makes sharing keys easy. A single key can be used for an image, set of images, or an album, and shared amongst a group of friends. This makes key sharing easy and manageable by design, and our Cryptagram browser extension facilitates the use of a password across multiple images or an album by allowing users to store the password. Enabling a strong password to be applied across an entire album of photos means that Cryptagram makes key dissemination easy.

Employing a hybrid cryptosystem by following the principles of *key continuity management* [13] means that the Cryptagram design focuses on guiding the user to use a hybrid cryptosystem correctly. In particular, by (1) limiting the interface for the use of public keys for encryption and private keys only for decryption and (2) using strong defaults for the block cipher and public key cryptography algorithms, Cryptagram reduces the friction to secure and correct use of a hybrid cryptosystem.

For both schemes, users do not share the sensitive information through the social network. We advise users to use a separate channel (e.g., text messaging) to share sensitive credentials (e.g., an album password) so as to conform to the threat model in which Cryptagram is designed to protect users from a hosting OSN.

4.4 Usable Image Identifiers

While Cryptagram facilitates the creation of Cryptagram images, the question remains of how to identify and distinguish gray, fuzzy images for friends. We describe how we enable users to create images that are easier to identify for fellow human users.

Text Watermark: One challenge with the current format is that all output images look virtually identical. This is a problem when, for example, a user asks a friend for a Cryptagram password. Without a file name or album name, there is no codified way to refer to images. Using a simple extension to the encoding tool, we can enable the user to specify a text or image-based watermark to render underneath the Cryptagram image. A text watermark could specify useful identifiers, such as a URL or an email address for submitting password requests.

Chrominance Watermark: In cases that we do not use the chrominance channels for data transmission, we can use these channels for identification purposes. We modify the C_b and C_r channels to add chrominance to output Cryptagrams and do so without corrupting the luminance payload.

We embed images in these chrominance channels so long as luminance remains unaffected. This watermark is not suitable for embedding most natural images since, perceptually, we rely heavily on luminance, but the technique works well with high-saturation icons or logos.

4.5 Surviving Partial Failures

The current protocol has error correction and can withstand some degree of noise from JPEG but will fail with a cropping transformation. We can extend the basic design to provide cropping robustness by dividing a Cryptagram's payload into smaller units. We encrypt each block with the same password and decryption will involve individually decrypting and concatenating all blocks. If one block fails to decode correctly due to cropping, the integrity of other blocks and their sequence within the original JPEG remain unharmed. Such an approach, however, does not apply to storing arbitrary bit streams, but for images one can replace unrecoverable blocks with zeroes in order to display as much of the original image as possible as shown in Figure 8.

5. IMPLEMENTATION AND DEPLOYMENT

In this section we describe the current state of the applications deployed under the Cryptagram name, including several components and continuously evolving inner protocols. The code is open source and online:

`http://github.com/prglab/cryptagram`.

5.1 Microbenchmarks

As of the submission of this paper, Cryptagram has over 400 active users (installed and currently present on the user's system) of its Chrome extension, distributed through the Chrome webstore. We request user-consent for an IRB-approved study to gather non-identifying log reports and consent has been granted from 373 unique browser installations.

We built the Chrome Extension with the Closure framework [15], requiring approximately 4000 Source Lines of Code (SLOC) [7, 38], porting the core components to a Firefox add-on with some additional code.

Our benchmarking framework consists of an ECC implementation and benchmarking code, and relies on the Reed-Solomon kernel module code ported to userspace, libjpeg-turbo codec, and a corresponding image interface ported from the Google Chromium browser [35]) (3000 SLOC).

The iOS App uses WebKit's JavaScriptCore to leverage the same cryptographic library as our JavaScript extension whereas the Android App achieves JavaScript integration through a WebKitView (2300 SLOC). These applications enable local Cryptagram encoding and decoding – we do not currently integrate with OSNs. While we do not have

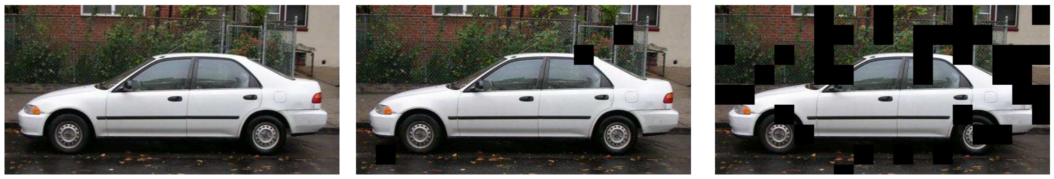

(a) Quality 78 **(b)** Quality 74 **(c)** Quality 70

Figure 8: Partial failure resiliant protocol.

user data for the mobile versions of Cryptagram at this time to present, we have challenges in engineering and usability to consider. With respect to engineering we have found that configuring native cryptographic libraries to be compatible across languages can be a difficult sea to navigate: our wrapping JavaScript libraries results in a performance penalty but simplifies the assurance of algorthmic parity across platforms. We also encounter usability challenges with respect to accessing user's OSN photos from a third-party application. The current aim is to seamlessly integrate with a user's existing social workflow rather than require users to use our product to access their OSNs. If Cryptagram were to be a self-sufficient entity, then we could foresee aiming to encourage user's to see Cryptagram as a portal to their OSNs. Of course, then one must balance the terms of service requirements of OSNs with the information our product reveals or does not reveal to those OSNs.

5.2 Browser Extension Decoder

We implemented the first version of the software as a browser extension, a framework supported by Chrome and Firefox browsers, which allows us to augment the user experience on any website by JavaScript injection.

For our first deployment of Cryptagram we adopted an embedding protocol with a $\left(Y^3_{2\times 2}, C^0\right)$ CPB. This protocol also embeds a checksum for verifying the integrity of the decoded encrypted bits. The checksum is not of the cleartext data; it is a checksum of the encrypted data and embedded adjacent to the encrypted data for data integrity purposes.

Decoding in place. Extensions can access pixel data of images on a website. The extensions perform image processing to produce new images to insert into the original image's container, as shown in figure 2. We add a contextual menu item so a user can right-click any image and attempt to decrypt it as a Cryptagram. With the correct credentials, the extension decrypts the original image, which pops into place.

5.3 Web-based Encoder

We wrote the web-based encoder in JavaScript with the Closure Framework, sharing much of the codebase with the decoder extensions. The encoder allows users to drag-and-drop cleartext images onto the encoder page. The drag-and-drop triggers an event to prompt users for a strong password (in the symmetric key case) as well as desired settings (e.g., the preferred tradeoff of a high-resolution, low-quality image or low-resolution, high-quality image). The encryption, encoding, and produced download zip requires no server interaction and thus allows for complete offline operation by end-users.

6. EVALUATION

We now explore the evaluation of the Cryptagram system. We begin with microbenchmarks as well as observations that serve as background for the subsequent evaluations. In particular, we will present the efficiency performance of protocols that we find to be the most useful for end-users and reason about the utility of the current deployment.

6.1 Efficiency Microbenchmarks

With microbenchmarks, we aim to establish a sense of the tangible weight that Cryptagram adds to the user experience of sharing photos as well as the system overhead.

On Browser Performance. We found that the input file size to the encoder and decoder correlated linearly with time to execute. The approximate ratio of time to complete the operation (milliseconds) to input filesize (KB) was 2.684 for the encoder and 1.989 for the decoder on an iMac with 2 x 2.26 Quad Core processors in the Chrome browser (one core for the browser process). While the noticeable human visual reaction time is in the range of 190 to 330 milliseconds [19], the results demonstrate that the overhead of using Cryptagram for viewing OSN photos is marginal.

On File Size. Since the high entropy of Cryptagram counteracts the compressive power of JPEG, the output file size depends entirely on the chosen embedding protocol and constraints imposed by the OSN. For Google+ and Facebook, uploading images have a cap based solely on image dimensions. The authors have found that the maximum upload dimensions in these OSNs is 2048×2048. This means that for a scheme that attains an efficiency of three bits per pixel, we can store at most 1.5 MB in the spatial domain of an uploaded JPEG image.

How does the size of the input data relate to the output Cryptagram image size? The nature of JPEG compression complicates this question. The output Cryptagram image may be saved at 100% quality, creating a large filesize footprint. While this may seem necessary given that we examine q, p-Recoverability with respect to the compression applied by an OSN, the composition of JPEG compression is neither idempotent nor cleanly-defined recursively. Instead, as we explore later in this section, we consider the observed error rates of compressing already-compressed Cryptagram images (simulating what an OSN would do).

Table 2 shows the expansion ratio from a given input size. For the case of a $(Y^3_{1\times 1}, C^0)$ CPB with an output Cryptagram image with JPEG compression 70, the filesize on disk inflation is $1.4\times$.

For the sake of minimizing upload bandwidth, users may opt to export Cryptagram images with less than 100% quality and Cryptagram will still guarantee q, p-Recoverability within a certain range.

6.2 Compressing the Compressed

Apropos to the question of file size expansion, we examine the implications of a recompressed Cryptagram JPEG on q, p-Recoverability. Figure 9 shows the effects of exporting

	$(Y^3_{1\times1}, C^0)$			$(Y^1_{1\times1}, C^0)$		
Quality:	90	80	70	90	70	50
Expansion:	2.25	1.75	1.40	7.82	5.29	4.39

Table 2: We present the tabular data that illustrates the file size expansion when using various protocol choices in the Cryptagram framework.

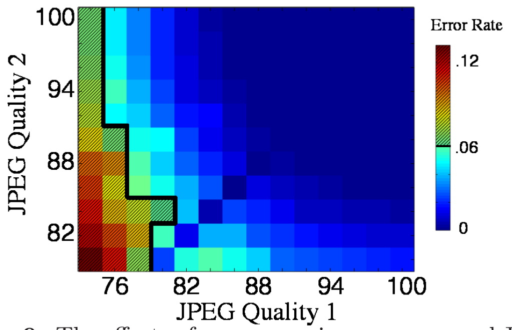


Figure 9: The effects of recompressing a compressed JPEG. The x-axis shows the quality of the original Cryptagram JPEG. The y-axis shows the recompressed quality of the JPEG. The line separating striped versus unstriped values is the q, p-Recoverability threshold we encounter with $RS(255, 223)$. Any values to the right of the 0.06 line show the successive recompressions that Cryptagram can tolerate for $\left(Y^3_{1\times1}, C^0\right)$. Error rates were determined by testing approximately 2,400 pseudorandom 8×8 images at each combination of quality levels.

a Cryptagram to a JPEG Quality 1 and then (as an OSN would do) recompressing the image at JPEG Quality 2. The error rate indicates the fraction of CPBs that were broken through successive recompression. This data indicates that we can export Cryptagram JPEGs to 82% quality and OSNs' recompression still permits recoverability, assuming that we leverage $RS(255, 223)$ ECC.

6.3 OSN Photo Quality

As much of our evaluation relates error rates to JPEG quality level, we want to know the JPEG settings employed by popular OSNs. To estimate these quality levels, we exported a variety of images as quality 95 JPEGs, uploaded those images to both Facebook and Google+, then re-downloaded the images for analysis.

On Google+, 30 such test images came back bitwise identical, meaning images were not recompressed.[4]

Facebook, on the other hand, applies JPEG compression to save disk space. After downloading images from Facebook, we looked for evidence of quality in the JPEG headers. Out of 30 natural images, 25 came back with a JPEG quantization matrix exactly equivalent to that of a quality 74 JPEG, the other five having matrices equivalent to JPEG qualities in the range of 76 to 86.

Fortunately for Cryptagram, high entropy images all appear similarly to the JPEG algorithm and are treated predictably when uploaded to Facebook. All test Cryptagrams uploaded then downloaded came back with the quantization matrix from a quality 85 or 87 JPEG, which we measured by explicitly examining the quantization tables of the downloaded JPEG file. This quality level puts us safely above the necessary threshold of our deployed embedding protocol.

[4]Google+ does recompress images for quicker display during album browsing but it is trivial to convert any such hyperlinks to their full-resolution equivalents.

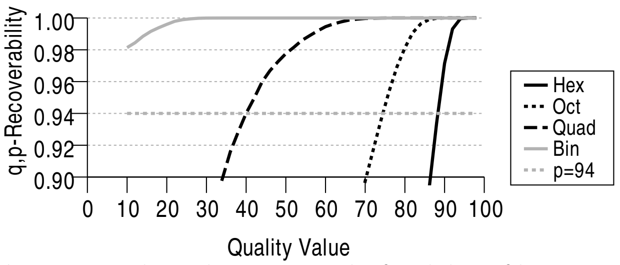


Figure 10: This indicates to us the feasibility of leveraging $RS(255, 223)$ to improve the q, p-Recoverability with various $\left(Y^B_{1\times1}, C^0\right)$ embedding protocols.

6.4 Embeddings with ECC

In this section, we examine the benefit of using ECC to reconcile the tradeoffs we must consider between efficiency and q, p-Recoverability. We presented in Section 4 the performance of the $\left(Y^B_{1\times1}, C^0\right)$ CPB for various B mappings. From that experience, we conclude that a protocol without error correction is limited to using quad or bin mapping strategies.

We examine the utility of applying our ECC algorithm of choice for embedding data to measure q, p-Recoverability in lower quality regimes of JPEG compression. With the use of $RS(255, 223)$ for ECC, we note that we embed 14% extra data for the recovery so our subsequent evaluation considers the effective efficiency of a system that adds this data overhead.

Figure 10 allows us to explore the design space of applying ECC to evaluate the q, p-Recoverability for given $\left(Y^B_{1\times1}, C^0\right)$ luminance-only embedding schemes. We see that the Bin, Quad and Oct embedding schemes perform above p=94 in the regime around 85%, thus enabling us to achieve q, p-Recoverability on Facebook.

Figure 11 illustrates the benefit of using luminance and chrominance embeddings in order to achieve 3.5 bits per pixel embedding efficiency for q, p-Recoverability that satisfies OSN recompression and ECC. In the interest of saving space, we do not show the q, p-Recoverability curves, but instead summarize the details relevant to the ECC discussion in Table 3. Given that ECC with $RS(255, 223)$ recovers up to 16 bytes ($\approx 6.27\%$) of damaged bytes for every 255 bytes of data, we can establish our target recoverability probability at $\approx 94\%$; in other words, if less than 6% of bytes break then applying ECC enables us to use that particular encoding scheme. We highlight in Table 3 the q, p-Recoverable protocol that we choose for Cryptagram.

This efficiency is superior to X-pire! [1], which had a capacity of two bits per pixel with ECC. We have 1.75× this capacity, significant considering the size and quality of images this enables users to upload to OSNs.

Comparison with Steganographic Efficiency. Though the goals of steganography and Cryptagram differ, both embed data in images, so we can compare the two in terms of bits/pixel efficiency.

Related work has expounded on the efficiency of steganographic embeddings [5, 37], reducing the approach to one embedding p message bits into $2^p - 1$ pixels, yielding a rela-

	Without ECC		With ECC ($RS(255,223)$)				
lum $\|B\|$	q, 100-Rec (quality)	Efficiency (bits/pix)	q, 94-Rec (quality)	Effective Efficiency	Lum+Chrom q, 94-Rec	Efficiency (bits/pix)	Effective Efficiency
Hex	-	-	90	3.5	90	4.5	3.94
Oct	90	3	76	2.62	77	3.5	3.06
Quad	80	2	44	1.75	66	2.5	2.19
Bin	< 20	1	< 10	0.87	38	1.5	1.31

Table 3: Summary of the results that inform how to proceed with applying $RS(255,223)$ FEC for embedding values in JPEGs that are recompressible.

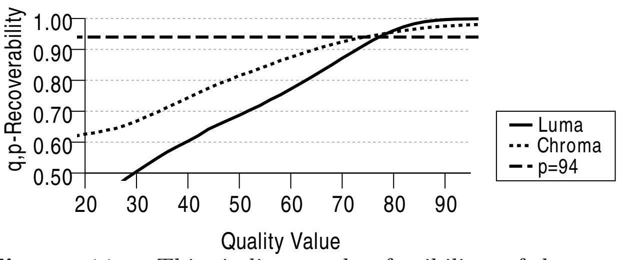


Figure 11: This indicates the feasibility of leveraging $RS(255,223)$ to improve the q,p-Recoverability of a $\left(Y^3_{1\times1}, C^1_{2\times2}\right)$ protocol.

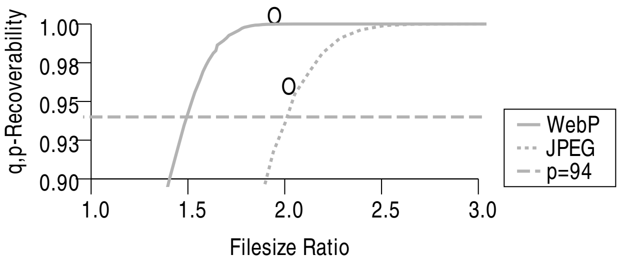


Figure 12: Showing the comparison of JPEG and lossy webp recoverability vs. filesize ratio. We draw the reader's attention to the $p = 94$ threshold as a point of comparison with the ECC studies in the rest of this paper. We acknowledge that JPEG and webp quality settings are not related and cannot be directly compared. However, this figure shows that for a similar notion of q,p-Recoverability, webp has a smaller filesize expansion than JPEG to achieve the same probability of recovery. To note the distinction in the meaning of "quality" between libjpeg and webp, we highlight the points along the curves where quality is 74 for each codec.

tive payload of $\alpha = p/(2^p - 1)$. While steganography choose slightly higher values of p a low value of p yields 0.42 bits per pixel for $p = 3$. In the highlighted row, our effective efficiency is 3.06 bits per pixel. In comparison, our approach represents a minimum 7.5× improvement.

6.5 File Format Embedding Comparison

In Figure 12, we show the q,p-Recoverability versus filesize ratio of JPEG versus webp image compression formats. By file ratio, we mean the on-disk size of the output image format for the same image canvas input. Notably, the embeddings are always three bits per pixel in the Figure. We see that for the same probability of recovery, p, webp has a much smaller filesize ratio than JPEG. As OSNs besides Google+ begin to experiment with webp deployment [32], the opportunity for lower bandwidth and storage requirements while maintaining q,p-Recoverability means that Cryptagram can be applied as improved media compression formats are adopted.

6.6 Deployment Usage Data

At the time of submission, Cryptagram has nearly 400 active installations, with 373 users agreeing to participate in our IRB-approved study. Through this study, we receive high-level data about the Cryptagram encryption and decryption habits of our users. The following data does not include the authors' own tests or images. We have had more than 3,300 Cryptagram image decryption events with more than 160 unique encrypted images generated. Of the decrypted images, we can confirm that 102 unique images have been decrypted from Facebook and 217 unique images from Google+.

7. DISCUSSION

Applicability of q,p-Recoverability to Lossy Formats. OSNs continue to use lossy image formats in order to reduce demands on storage infrastructure and reduce delivery latencies to end-users. Recently developed formats shoud be considered given these goals. We have begun to examine the webp [16] format for Cryptagram. The tool of q,p-Recoverability applies in the analysis of these formats given that the spatial domain pixel value is the key component of Cryptagram communication.

Transformations. We aim to handle a variety of transformations with the development of q,p-Recoverable protocols. In previous sections, we discussed the design and evaluated our protocols' q,p-Recoverability with respect to the JPEG transformation. We have begun prototyping our approach to cropping and noising transformations on images produced by Cryptagram as well, leveraging blocking algorithms coupled with ECC. While we do not address rotation explicitly, we do not consider such a transformation intractable as we apply techniques from QR Codes (two dimensional barcodes) by orienting corner features in future iterations.

Scaling transformations are of interest given the pervasiveness of lower resolution images (e.g., thumbnails) to partially depict images on a social networking website. We have considered the integration of pyramid representations [3, 6] in the design of future embedding protocols to meet this transformation request.

The Economics of Privacy. Our culture values greatly the power of images to document and record in ways that words simply cannot. We say *seeing is believing*. Images convey a range of human experience, and unfortunately, that includes images that can irrevocably damage a person's reputation.

OSNs offer privacy features and third parties have even developed commercial products to address photo privacy. McAfee Social Protection lets users store cleartext photos on their server while uploading blurred versions to Facebook, then facilitates access requests [30]. This superficially addresses photo privacy, but in the end, amounts to an escrow service that redirects trust from one third party to another.

Our optimistic vision for this project is that its adoption could articulate to OSNs that users desire increased ownership over personal data. We envision a scenario in which an OSN embraces the philosophy of Cryptagram and provides client-side tools to make end-to-end encryption feasible. Wide adoption of Cryptagram would require more storage for the encrypted files and may create less potential for targeted advertising.

8. RELATED WORK

P3 [29] examined the use of non-colluding services to store minimally-revealing cleartext images in one service and encrypted versions of DCT coefficients of JPEG images in another service. Their system experienced a 10-20% file size increase from the original compressed image when one follows their recommended privacy-preserving settings by setting the DCT-hiding threshold in the range $T \in [10, 20]$. The authors acknowledged their technique's vulnerability to face identification when $T \geq 35$. Cryptagram fundamentally differs from P3 in two ways. First, Cryptagram completely avoids the use of third parties. Secondly, Cryptagram works only in the encrypted bit space and does not expose any unencrypted data to the end-user. Unless users' keys are compromised, users cannot have their faces detected with any of our embedding protocols.

Steganography. Cryptagram is superficially reminiscent of various attempts to embed cryptographic data in JPEG through traditional steganographic techniques [11], but differs significantly from conventional JPEG steganography. Cryptagram makes obvious that it is hiding data to attain greater efficiency, and furthermore, does so in a way that is robust to image compression, which steganography generally is not. Attempts to achieve lossy compression tolerant steganography are early works with inefficient results [17].

One recent work attempted to embed data in JPEG DCT coefficients without the steganographic constraint of being hidden. The non-linearities of DCT quantization and rounding in standard compression and decompression required very conservative data embedding that resulted in efficiency significantly lower than what we were able to achieve [1].

Li et al. [20] address the concern of hiding data within DCT coefficients but shuffle the DCT coefficients between blocks that then are quantized during the JPEG compression algorithm. Likewise, Poller et al. demonstrate that DCT coefficient permutation and spatial domain permutation do provide some security features but do not address efficiency or prove the correctness of their security mechanism [26].

A formalization for the description of the embeddings that we use in Cryptagram have been explored by Galand and Kabatiansky [12] but the authors do not explore how to construct such protocols.

But Cheddad et al. [4] claim that spatial domain techniques are not robust against noise, only work in bitmap-formatted images, and are not robust against lossy compression and image filters. Cryptagram overcomes all of these drawbacks in spatial domain embedding and demonstrates the useful privacy and security properties that can be available for OSN photo sharing. Cryptagram achieves q, p-Recoverability in the face of the complete recompression of the JPEG image containing sensitive information.

9. CONCLUSIONS

The advent of popular online social networking has resulted in the compromise of traditional notions of privacy, especially in visual media. In order to facilitate convenient and principled protection of photo privacy online, we have presented the design, implementation, and evaluation of Cryptagram, a system that efficiently and correctly protects users photo privacy across popular OSNs. We have introduced q, p-Recoverability and demonstrated Cryptagram's ability to embed cryptographic primitives correctly to attain q, p-Recoverability through JPEG compression in our implementation.

Acknowledgments: We would like to thank Helen Nissenbaum, Alex Rubinsteyn, David Iserovich, the members of the NYU Networking and Wide-area Systems Group, the members of the NYU Privacy Research Group, our shepherd Steven Bellovin, and the anonymous COSN reviewers for their helpful comments. This work was supported in part by the NSF (grant 0966187) and in part by the ONR (contract N000141210327). The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of any of the sponsors.

10. REFERENCES

[1] Backes, J., Backes, M., Dürmuth, M., Gerling, S., and Lorenz, S. X-pire! - a digital expiration date for images in social networks. *CoRR abs/1112.2649* (2011).

[2] boyd, d. m., and Ellison, N. B. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication 13*, 1 (2007), 210–230.

[3] Burt, P. J. Fast filter transform for image processing. *Computer graphics and image processing 16*, 1 (1981), 20–51.

[4] Cheddad, A., Condell, J., Curran, K., and Mc Kevitt, P. Digital image steganography: Survey and analysis of current methods. *Signal Processing 90*, 3 (2010), 727–752.

[5] Crandall, R. Some notes on steganography. *Posted on steganography mailing list* (1998).

[6] Crowley, J. L. A representation for visual information. Tech. Rep. CMU-RI-TR-82-07, Robotics Institute, Pittsburgh, PA, November 1981.

[7] Daniel, A. CLOC: Count Lines of Code. http://cloc.sourceforge.net/.

[8] Duce, D., and Boutell, T. Portable network graphics (png) specification. *Information technology ISO/IEC 15948* (2003), 2003.

[9] Duell, M. Mark Zuckerberg's private Facebook photos revealed: Security 'glitch' allows web expert to access billionaire's personal pictures. *The Daily Mail (MailOnline)* (December 2011). http://www.dailymail.co.uk/news/article-2070749/Facebook-security-glitch-reveals-Mark-Zuckerbergs-private-photos.html.

[10] Finkenzeller, K. *Data Integrity.* Wiley Online Library, 2003.

[11] Fridrich, J. *Steganography in Digital Media: Principles, Algorithms, and Applications.* Cambridge University Press, 2009.

[12] Galand, F., and Kabatiansky, G. Information hiding by coverings. In *Information Theory Workshop,*

2003. Proceedings. 2003 IEEE (2003), IEEE, pp. 151–154.

[13] Garfinkel, S. L., and Miller, R. C. Johnny 2: a user test of key continuity management with s/mime and outlook express. In *Proceedings of the 2005 symposium on Usable privacy and security* (2005), ACM, pp. 13–24.

[14] Good, J. How many photos have ever been taken? http://blog.1000memories.com/94-number-of-photos-ever-taken-digital-and-analog-in-shoebox.

[15] Google. Closure Tools. https://developers.google.com/closure/.

[16] Google. webp: A new image format for the Web. https://developers.google.com/speed/webp/.

[17] Hwang, R.-J., Shih, T., Kao, C.-H., and Chang, T.-M. Lossy compression tolerant steganography. In *The Human Society and the Internet Internet-Related Socio-Economic Issues*, W. Kim, T.-W. Ling, Y.-J. Lee, and S.-S. Park, Eds., vol. 2105 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2001, pp. 427–435.

[18] Johnson, M., Egelman, S., and Bellovin, S. M. Facebook and privacy: it's complicated. In *SOUPS '12: Proceedings of the Eighth Symposium on Usable Privacy and Security* (2012).

[19] Kosinski, R. J. A literature review on reaction time. *Clemson University 10* (2008).

[20] Li, W., and Yu, N. A robust chaos-based image encryption scheme. In *Multimedia and Expo, 2009. ICME 2009. IEEE International Conference on* (2009), IEEE, pp. 1034–1037.

[21] Lin, K.-Y., and Lu, H.-P. Why people use social networking sites: An empirical study integrating network externalities and motivation theory. *Computers in Human Behavior 27*, 3 (2011), 1152–1161.

[22] Microsoft Safety Security Center. Create strong passwords, 2013. http://www.microsoft.com/security/online-privacy/passwords-create.aspx.

[23] Nissenbaum, H. *Privacy in context: Technology, policy, and the integrity of social life*. Stanford Law Books, 2009.

[24] Parr, B. Facebook by the Numbers. http://mashable.com/2011/10/21/facebook-infographic/.

[25] Pfeil, U., Arjan, R., and Zaphiris, P. Age differences in online social networking–a study of user profiles and the social capital divide among teenagers and older users in myspace. *Computers in Human Behavior 25*, 3 (2009), 643–654.

[26] Poller, A., Steinebach, M., and Liu, H. Robust image obfuscation for privacy protection in web 2.0 applications. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* (2012), vol. 8303, p. 1.

[27] Powell, J. *33 million people in the room: how to create, influence, and run a successful business with social networking*. Ft Press, 2008.

[28] Princeton University Office of Information Technology IT Security. Tips for creating strong, easy-to-remember passwords, 2013. http://www.princeton.edu/itsecurity/basics/passwords/.

[29] Ra, M.-R., Govindan, R., and Ortega, A. P3: Toward Privacy-Preserving Photo Sharing. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation* (2013), USENIX Association.

[30] Security, M. McAfee Social Protection. https://apps.facebook.com/socialprotection/.

[31] Sengupta, S., and OâĂŹBrien, K. J. Facebook Can ID Faces, but Using Them Grows Tricky. *The New York Times* (2012).

[32] Shankland, S. Facebook tries Google's WebP image format. `http://news.cnet.com/8301-1023_3-57580664-93/facebook-tries-googles-webp-image-format-users-squawk/`.

[33] Stone, Z., Zickler, T., and Darrell, T. Toward large-scale face recognition using social network context. *Proceedings of the IEEE 98*, 8 (2010), 1408–1415.

[34] Tapscott, D. *Grown Up Digital: How the Net Generation is Changing Your World HC*, 1 ed. Mcgraw-Hill, 2008.

[35] The Chromium Authors. libjpeg | The Chromium Projects. `http://src.chromium.org/viewvc/chrome/trunk/src/third_party/libjpeg/`.

[36] Union, I. T. Digital Compression and Coding of Continuous-tone Still images. CCITT Rec. T.81, 1992.

[37] Westfeld, A., and Pfitzmann, A. High capacity despite better steganalysis (f5–a steganographic algorithm). In *Information Hiding, 4th International Workshop* (2001), vol. 2137, Pittsburgh, PA, pp. 289–302.

[38] Wheeler, D. SLOCCount. http://www.dwheeler.com/sloccount/.

[39] Zimmermann, P. R. *The official PGP user's guide*. MIT press, 1995.

APPENDIX

A. APPENDIX

A.1 JPEG Review

JPEG is a lossy codec designed to provide reasonable tradeoffs between compression and recoverability of the original image [36]. We chose JPEG because of its prevalence on the web and the availability of efficient JPEG libraries. Here we review the core elements of JPEG that affect our protocol design.

Transformations.

The first step in compressing an input bitmap image, M to JPEG is a color space transformation. In particular, JPEG transforms every pixel in M from the RGB to the YC_bC_r color space through a linear transformation that converts red, green, and blue (RGB) pixel values to luminance (Y), chrominance blue (C_b), and chrominance red (C_r) values.

$$
\begin{aligned}
Y' &= (0.299 \cdot R'_D) + (0.587 \cdot G'_D) + (0.114 \cdot B'_D) \\
C_B &= 128 - (0.168736 \cdot R'_D) - (0.331264 \cdot G'_D) + (0.5 \cdot B'_D) \\
C_R &= 128 + (0.5 \cdot R'_D) - (0.418688 \cdot G'_D) - (0.081312 \cdot B'_D)
\end{aligned}
$$

After the color space transformation, the JPEG algorithm transforms the three color channels of YC_bC_r independently.

Subsampling.

Following the initial color space transformation, JPEG subsamples the color channels. In the default `libjpeg` codec settings, luminance is not subsampled while chrominance blue and chrominance red data undergo 2:1 subsampling vertically and horizontally, also known as "4:2:0" subsampling. This results in one-fourth the number of pixels that represent each of the original chrominance blue and chrominance red channels.

Discrete Cosine Transform.

The output matrices of subsampling are then transformed using the Discrete Cosine Transform. Since the DCT for JPEG operates only on 8×8 pixel blocks, JPEG breaks each of the subsampled spaces into non-overlapping 8×8 pixel blocks. This paper we will refer to these units as JPEG Pixel Blocks or JPBs.

The DCT is a well-understood transformation from the spatial to frequency domain. We present the two dimensional DCT here:

$$G_{u,v} = \sum_{x=0}^{7} \sum_{y=0}^{7} \alpha(u)\alpha(v) g_{x,y} \cos\left[\frac{\pi}{8}\left(x + \frac{1}{2}\right)u\right] \cos\left[\frac{\pi}{8}\left(y + \frac{1}{2}\right)v\right],$$

where $u \in \{0, 1, \ldots, 7\}$ is the horizontal spatial frequency; $v \in \{0, 1, \ldots, 7\}$ is the vertical spatial frequency;

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } u = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$$

is a normalizing scale factor to maintain orthonormality; $g_{x,y}$ is the pixel value at coordinates (x, y); and $G_{u,v}$ is the DCT coefficient at coordinates (u, v). The two dimensional DCT,

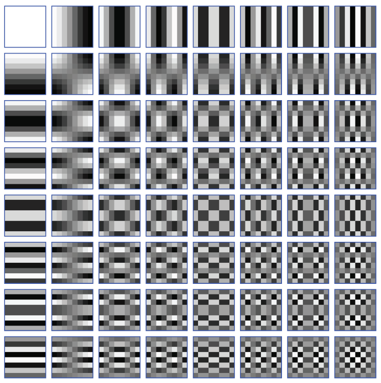


Figure 13: Visualization of the 64 DCT basis functions.

computed on each 8×8 block of pixels in an image, results in 64 coefficients per block; the visual representation of which is in Figure 13.

Quantization.

The DCT returns real values but we have to represent them on disk with limited precision. The JPEG codec stores each of the 64 coefficients not as a float but as a single 8-bit number that, combined with the quantization matrix, can approximate a real number.

Here is the standard luminance quantization table in JPEG.

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

And for chrominance:

$$\begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

This means, for example, that to approximate the value 99.0 for the 0^{th} (pure DC) component, we would write the integer $\lfloor \frac{99.0}{16} \rfloor = 6$. In short, the lower the value in the quantization matrix, the greater the precision for preservation of that particular DCT coefficient. By scaling all values in the quantization table, JPEG prioritizes lower frequencies (which are more obvious to the human eye) while allowing end-users to achieve a range of qualities.

Chrominance red and chrominance blue use a separate quantization matrix which is more heavily quantized, since the human eye is less sensitive to variations in chrominance than luminance [36].

Entropy Coding.

Once JPEG computes the quantized values, JPEG losslessly writes these values to disk. This deterministic operation provides the biggest savings for bytes on disk. In particular, values are zigzag encoded, meaning that the order of values written to disk follows a zigzag pattern. Any repeated values in the zigzag ordered list JPEG writes as a compressed value; e.g., the JPEG algorithm dictates writing 20 sequential 0's as 20{0}. The benefits of this process is a reduced amount of disk space required to represent the values.

A.1.1 Decoding JPEG

Dequantization.

Given the original JPEG quantization matrix in the JPEG file, the decompression algorithm multiplies the values read from disk with the corresponding quantization matrix entries.

Inverse DCT.

JPEG applies the inverse DCT function to each set of 64 dequantized coefficients to produce the lossy output 8×8 block of pixels.

Shifting and Upsampling Chrominance.

Values in all of color spaces are then shifted up by 128 to be within the valid display ranges again. JPEG upsamples chrominance accordingly.

Conversion from YC_bC_r to RGB.

Finally, JPEG executes the last linear transformation between YC_bC_r to RGB to present a human comprehensible RGB image.

$$R = Y + 1.402 \cdot (C_R - 128)$$
$$G = Y - 0.34414 \cdot (C_B - 128) - 0.71414 \cdot (C_R - 128)$$
$$B = Y + 1.772 \cdot (C_B - 128)$$