Secret Message Sharing Using Online Social Media

Jianxia Ning*, Indrajeet Singh*, Harsha V. Madhyastha*,
Srikanth V. Krishnamurthy*, Guohong Cao§, and Prasant Mohapatra‡
*University of California, Riverside §Penn State University ‡University of Calfiornia, Davis
{jning, singhi, harsha, krish}@cs.ucr.edu gcao@cse.psu.edu prasant@cs.ucdavis.edu

Abstract—**Recently, there have been proposals to evade censors by using steganography to embed secret messages in images shared on public photo-sharing sites. However, establishing a covert channel in this manner is not straightforward. First, photo-sharing sites often process uploaded images, thus destroying any embedded message. Second, prior work assumes the existence of an out-of-band channel, using which the communicating users can exchange metadata or secret keys a priori; establishing such out-of-band channels, not monitored by censors, is difficult.**

In this paper, we address these issues to facilitate private communications on photo-sharing sites. In doing so, first, we conduct an in-depth measurement study of the feasibility of hiding data on four popular photo-sharing sites. Second, based on the understanding derived, we propose a novel approach for embedding secret messages in uploaded photos while preserving the integrity of such messages. We demonstrate that, despite the processing on photo-sharing sites, our approach ensures reliable covert communication, without increasing the likelihood of being detected via steganalysis. Lastly, we design and implement a scheme for bootstrapping private communications without an out-of-band channel, i.e., by exchanging keys via uploaded images.

I. Introduction

The idea of hiding messages, using steganography, in user-generated content on photo-sharing sites has recently received increased attention; for example, Burnett et al. [19] suggest that the approach can be used to "chip away" at censorship firewalls. However, while the idea is conceptually attractive, there exist several challenges in creating a viable covert channel of this type. First, photo-sharing sites often process uploaded images [6]. While some of the processing functions are clearly specified on the photo-sharing sites [8], [5] (e.g., any photo exceeding a pre-specified size limit will be re-sized), not all such functions are publicly known. These (possibly unknown) processing functions often interfere with the use of steganography. Second, it is well known that steganography does not offer perfect secrecy. Censors can try to read the embedded message by applying a variety of extraction algorithms on a carrier image. Thus, to prevent exposure in the rare cases of interception, one will have to encrypt the secret information embedded in the shared photographs. Encryption requires the establishment of secret keys between communicating entities, for which prior work often assumes the existence of an out-of-band channel. However, in cases where people are trying to hide information from government-controlled censors, the creation of such an out-of-band channel is difficult because phone calls, e-mail exchanges, and Internet communication may be monitored [29].

Our goal is to address the above challenges and build a framework for private communication on public photo-sharing sites. Towards this, we make three key contributions.

First, to understand how secretly embedded messages are affected by processing done on photo-sharing sites, we perform an in-depth measurement study. We analyze photos uploaded on four popular sharing sites—Google+, Facebook, Twitter, and Flickr. We consider both photos wherein secret information is embedded and photos without any such embedding. We observe that, while the integrity of hidden messages is preserved on some sites (e.g., Google+), other sites (e.g., Facebook and Flickr) perform various processing functions on uploaded images and hence the extraction of secret messages from downloaded images fails. Our study sheds light on the processing performed on different sites and provides an understanding of why secret content is affected.

Second, based on the understanding obtained above, we propose simple changes to the steganographic encoding process which ensure that, unlike prior approaches, the embedded secret messages survive the image processing performed by photo-sharing sites. Though simple, our approach is not apparent without the detailed study on the different photo-sharing sites. Importantly, this improved reliability does not come at the expense of greater likelihood of detection of hidden messages. We evaluate our approach by applying two state-of-the-art steganalysis tools and observe that, for a fixed amount of secret data, the likelihood of detecting secret information embedded with our approach is similar (or even lower in some cases) to the probability of detection when prior approaches for steganographic embedding are applied (while surviving the processing done on the site).

Finally, as discussed above, encrypting the secretly embedded messages is a must. Therefore, to enable recipients of the shared photo to extract the raw data, a key exchange between the sender and recipients is essential. Towards this, we propose a protocol for bootstrapping the private communication without any out-of-band channel (unlike what is assumed in prior work [19]). Our bootstrapping phase uses the very same channel, i.e., uploaded images, to exchange keys.

II. Background and Related Work

In this section, we first present relevant background and subsequently discuss related prior work.

JPEG image steganography: Steganographic techniques are typically developed to exploit the structure of JPEG (the most common image format used on photo-sharing sites), and hence, we focus on this format here. The image in which the message is hidden is called the *cover* or the *carrier*. The JPEG encoding process consists of several steps including applying lossy compression, the division of the image into blocks, application of the Discrete Consine Transform (DCT) on each block, and the quantization of the DCT coefficients. More details on JPEG encoding and decoding are found in [24].

Structure-based steganography exploits certain, usually optional, markers in the JPEG format to embed secret data. Examples include embedding the message using the *Exchangeable Image File (EXIF)* (e.g., as in [18]) or the *Comment* markers (e.g., as in [1], [11]). The decoder tool simply examines marker locations to extract the message.

Spatial domain techniques typically modify the Least Significant Bit (LSB) of the pixel values to embed the secret information [31]. These techniques exploit the fact that human perception is not sensitive to subtle changes in pixels. Informa-

Tool	Details on approach	Facebook	Twitter	Flickr	Google+
GhostHost [43]	Embedding after the EOI marker	$\times$	$\times$	$\times$	$\surd$
Steghide [15]	Changing Pixel values	$\times$	$\surd$	$\times$	$\surd$
OutGuess [13]	Changing DCT coefficients (pseudo-random)	$\times$	$\surd$	$\times$	$\surd$
F5 [4]	Changing DCT coefficients (non-zero)	$\times$	$\surd$	$\times$	$\surd$
YASS [41]	Changing DCT coefficients (error correcting)	$\surd^*$	$\surd$	$\surd^*$	$\surd$

TABLE I
EVALUATION OF STEGO TOOLS ($\times$ = FAILURE; $\surd$ = SUCCESS; $\surd^*$ = CONDITIONAL SUCCESS)

tion hiding in pixel values is however not reliable, especially when used with lossy image compression schemes such as JPEG. Steghide [15] is a stego implementation in this category.

Frequency domain based methods replace the LSBs in the quantized discrete cosine transform (DCT) coefficients [22]. To avoid visual distortion, embedding of secret messages is avoided for DCT coefficients whose value is zero; these coefficients typically correspond to high frequency components. JSteg [12], OutGuess [13] (which uses a pseudo-random number generator to select DCT coefficients), and F5 [4] (which decreases the absolute value of non-zero DCT coefficients by one) are examples in this category.

Distortion-resistant schemes are more robust to image processing. To lower the bit error rate (BER), these schemes perform transformations in other domains (like with the Discrete Wavelet Transform) or use redundancy and/or masking techniques. For example, YASS [41] uses a redundancy parameter to control the number of times an information bit is repeated inside an image.

Use of steganography on images shared online: Photos upoloaded onto online sites provide a means of sharing secret messages. However, it is known that photo-sharing sites process such uploaded images [28], [17], [8], [16]; while these sites explicitly indicate that they process images, the specifics are not made known (no documentation is readily available). There have been studies on whether messages are hidden in images posted on the Internet [39]. However, Provos et al. [39] analyzed two million images downloaded from eBay for hidden messages but not a single such message was found. Because of the specific detection approach applied, and their source for the images, the insights gained from their attempt are limited.

The use of social media and steganography to build a covert channel is recognized as promising in [23]. Zeljko et al. [42] implement SecretTwit, a Twitter client that hides secrets in tweets and images. An anti-censorship system proposed in [19] is based on two parties exchanging messages in images on Flickr. While the idea of a botnet performing private communication using images on Facebook is suggested in [35], the authors do not examine issues relating to image processing or detection likelihood of hidden messages as we do here.

Despite this attention, the feasibility of private communication on OSNs or photo-sharing sites has not been fully explored in prior work. From their first hand experience, some Internet users have already realized that certain steganography techniques do not work with images uploaded on Facebook [30]. However, the reasons for this are not well understood. Three characteristics of images published on some OSNs, namely image format, metadata and pixel resolutions of digital images have been analyzed by Castiglione et al. [20]. However, the use of steganography has not been examined. Realizing that user images are usually processed on OSNs, Castiglione *et al.* [21] propose the use of the name and tags of the images as an alternative means of hiding information and thereby establishing a secret communication channel.

Redundancy	2	6	10	14	18
Facebook	0.3442	0.1498	0.0411	0.0000	0.0000
Flickr	0.3491	0.1592	0.0456	0.0000	0.0000

TABLE II
AVERAGE BER WITH YASS WITH DIFFERENT REDUNDANCY LEVELS

In summary, thus far there is no thorough investigation on how the processing of images on public photo-sharing sites impacts different information hiding techniques. To the best of our knowledge, our work is the first to fill this gap. We also believe that we are the first to propose an approach to ensure that secret messages can indeed be reliably communicated via the photos uploaded onto these sites.

III. Secret Embedding Feasibility

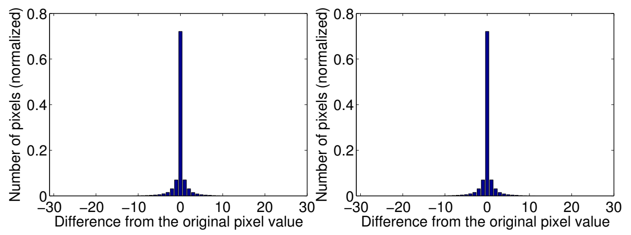
In this section, we present our in-depth measurement study on understanding the feasibility of embedding secrets in images uploaded on online photo sharing sites.

A. Hiding information on different photo-sharing sites

We use multiple representative steganography tools from each category described in Section II to hide messages in images. We upload these images on to various sites and then attempt to retrieve the hidden messages from the downloaded images. We use 100 images from a database made available by CMU [25]. We believe that this diverse set of images is representative of the kinds of photos that people share online. The sizes of these images range from a few KB to thousands of KB. The minimum pixel resolution from among all the images is 192 x 261 and the maximum is 4288 x 2848.

Steganography tools: The steganography tools used are listed in Table I. These tools are chosen as they are widely used and are publicly accessible. GhostHost simply appends the hidden message after the End-of-Image marker. Steghide, F5, and OutGuess are the most widely used tools for benchmarking in academia. Yet Another Steganographic Scheme (YASS) embeds data at randomized locations within an image and repeats an information bit multiple times inside the image. The redundancy (the number of times that a bit is repeated) is a tunable parameter. Table I shows, for each site, whether we were able to retrieve the hidden messages embedded with each tool.

Definition of terms: For ease of discussion, we define the following terms. (a) *Success* implies that the extracted message is equivalent to the hidden message. (b) *Failure* means that the retrieval effort does not yield a meaningful output. For example, a failure causes the output of Steghide to be: "extracting data...could not extract any data". A failure is experienced even if only a part of the message is corrupted; a checksum may fail or metadata could yield mismatches. In all of such cases, one cannot retrieve the original hidden message. (c) *Conditional success* only applies to YASS which



(a) First dimension (b) Second dimension

Fig. 1. Distribution of the differences in the pixel values in two color dimensions between original images and after they are uploaded on Facebook.

uses redundancy to control the decoding BER. The decoding BER (the ratio of message bits in error to the total number of message bits in an image) of YASS depends on the redundancy parameter in use. We experiment with different redundancy parameters and the results are presented in Table II. We observe that when the redundancy parameter is larger than 10, the BERs significantly decrease and approach 0. Similar results are reported in [35]. We observe that the BERs experienced for a given redundancy are similar on Facebook and Flickr.

Summary of the results: At first glance, we see that most of the tools (except YASS) fail on Facebook and Flickr but succeed on Google+ and Twitter. Google+ is the most generous platform and accommodates all the steganography tools. Twitter is the next best; GhostHost fails on Twitter but the other tools are able to successfully exchange hidden content. Facebook and Flickr show the least compatibility with steganography in that all the tools except YASS (with high redundancy) fail in successfully exchanging secret content.

To understand the above results, we next examine the processing changes at the bitstream level done at each site on the uploaded images.

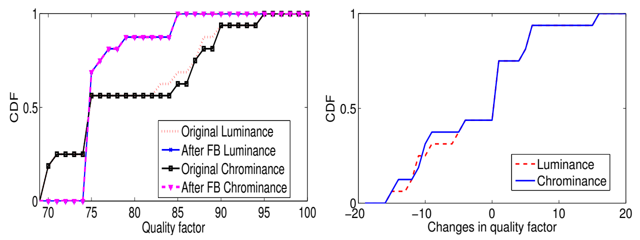
B. Impact of processing on hidden messages

Google+: ***Image integrity is preserved.*** Experimenting with our sample data set, we observe that Google+ preserves the original images, when their sizes are within 2048 pixels by 2048 pixels (Table I). Since the integrity of an image is preserved as long as the image adheres to the permitted resolution, any steganographic tool will work on Google+.

Twitter: ***Metadata fields are cleaned up.*** Some of the fields for storing metadata within the JPEG image, (e.g., the COM and the APP fields [24]), are rewritten by Twitter with its own data. In addition, anything that appears after the EOI (End-of-Image) marker is stripped off. The consequence is that tools that exploit metadata markers for embedding messages (e.g., GhostHost) will not work. Except for this 'clean up' of the metadata fields, Twitter preserves the image integrity as long as the image size is no larger than 1024 pixels by 768 pixels. Exceeding this limit will cause a loss of integrity. Hence, as seen in Table I, Twitter accommodates most steganographic tools as long as the image size is within the limit.

Facebook: Similar to Twitter, Facebook removes the content in some of the metadata fields. In addition, we find that Facebook applies the following processing functions.

Changes in pixel values. We find that, for a fraction of pixels, the RGB ($\in [0, 255]$) values are changed on Facebook after image upload. Across the examined set of images, Fig. 1 shows the distribution (probability density function) of the deviation of the pixel values from the original values for two color (RGB) dimensions. The distribution for the third dimension is similar and is not shown. We see that, while most pixels ($> 60\%$) remain unchanged, some are modified. The maximum deviation of the pixel value from the original value can be up to 30. The distribution of the deviation in general, seems to follow a Gaussian distribution.



(a) Quality factor (b) Quality factor change

Fig. 2. CDF of quality factor and the change before and after upload

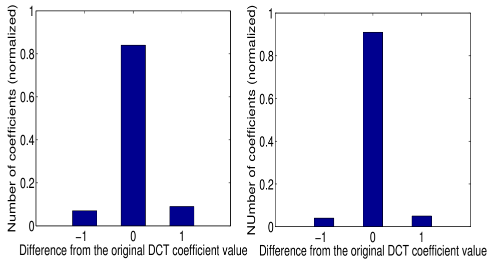
Pixel value changes can be due to JPEG's lossy compression (adopted by Facebook) and/or other manipulations (discussed later). In either case, the steganography tools that rely on embedding the messages into the pixel values (e.g., StegHide) do not work. Images larger than 2048 pixels in either the length or width dimensions get resized on Facebook. Resizing causes the pixels to shift from their original locations or even to be lost, thus destroying the integrity of embedded messages.

Changes in compression ratio. From the quantization tables for luminance and chrominance associated with the original images and the downloaded ones, we note that Facebook adjusts the compression ratios for many images.

For color JPEG images, distinct luminance and chrominance quantization tables are part of the JPEG structure and are stored in the JPEG file. With the utility JPEGsnoop [27], we can access these quantization tables. By comparing the quantization tables of an image before and after upload, with the tables from the International JPEG Group standard (libjpeg [10]), we get the approximate quality factors before and after upload. Note that, though the quality factor of a JPEG image is usually represented by an integer in [0, 100], the calculated values are not necessarily so due to the fact that custom tables may be used for each particular image. We round each calculated number to the nearest integer. The results are shown in Fig. III-A.

We observe that while the quality factors for both the luminance and chrominance of the original images vary from 70 to 95, Facebook adjusts them to 75 in about 70% of all cases. This matches the observation in [35] that the quality factor of Facebook compression is approximately 75. Fig. III-A shows the cumulative distribution function (CDF) of the quality factor change for all images. We observe that, for about half of the images, Facebook uses a higher compression ratio (resulting in lower quality factors) than the original.

When a lower quality factor is used, it is likely that the pixel values will change due to compression. Interestingly, we observe that even for those images whose quality factors are unchanged, the pixel values are modified. We conjecture that it may be the case that Facebook applies other image processing besides just compression. There is no official statement about what the exact processing is, but some online discussions suggest that a low pass filter is used (e.g., see bit.ly/UBlbI5).



(a) Facebook DCT changes (b) Flickr DCT changes

Fig. 3. Variations in DCT coefficients with Facebook and Flickr

Method	BER	FEC overhead	Detection likelihood (ensemble classifier)	Detection likelihood (StegAlyzerAS)
LSB	0.15239	0.0	0.44	0.69
LSB + 2-LSB	0.08144	0.0	0.47	0.68
2-LSB	0.00968	0.0	0.50	0.63
LSB+FEC *[15,13]*	0.09375	0.1333	0.45	0.69
LSB+FEC *[15,11]*	0.01125	0.2667	0.48	0.69
LSB+FEC *[7,3]*	0.0	0.5714	0.53	0.72
LSB+2-LSB+FEC *[15,13]*	0.02993	0.1333	0.50	0.68
LSB+2-LSB+FEC *[15,11]*	0.0	0.2667	0.51	0.69
2-LSB+FEC *[15,13]*	0.0	0.1333	0.51	0.63

TABLE III
COMPARISON BETWEEN LSB, 2-LSB AND MIXED LSB+2LSB STEGO METHODS

Changes in DCT coefficients. We access the DCT coefficients using a JPEG dump utility [9] based on the libjpeg library. Fig. III-B shows the number of DCT coefficients having a specific normalized change as compared to the original value. We notice that, while the majority of coefficients remain the same (with no difference from the original value), about 20% are decreased or increased by one. It seems that the change is small (only one) and appears to occur in the least significant bits (LSBs); as one might recall, the least significant bits of the DCTs are exactly where the bits corresponding to the hidden message reside when many of the stego embedding tools are employed. A careful examination reveals that DCT coefficient changes are evenly (uniform) distributed all over the image (result not plotted due to space constraints).

A careful inspection suggests that there are two potential reasons for the above changes in the DCT coefficients. First, Facebook uses a different set of quantization tables from that in the standard JPEG libraries. Second, it also changes the pixel values themselves, thereby compounding the effect. We were unsuccessful in preserving the DCT coefficients even when uploading images with the same quality factors seen in the images downloaded from Facebook. It is also possible that Facebook applies a watermark to the uploaded images; however, we were unable to verify this.

The above findings indicate that embedding messages in the DCT coefficients (e.g., with F5 and Outguess) runs the risk of extraction failures when the images are uploaded on to Facebook and subsequently downloaded. Tools with error correcting capabilities (e.g., YASS) can lower the decoding BER and even eliminate errors in some cases. However, the BER depends on the message itself and where it is encoded within the image. Thus, it may not be possible to extract the message in all cases.

Finally, we wish to point out that the default DCT encoding in JPEG images is done with baseline encoding; however, Facebook uses progressive encoding (which enables the user to see a blurred version of the image while it is being downloaded). However, we verified that this does not affect the values of DCT coefficients. The encoding scheme only determines if a specified band (i.e., a lower or higher part of the frequency spectrum) is encoded first, and if the most significant (and the number of) bits of the coefficients are encoded first. This results in changes in the way that the coefficients are represented in the JPEG format, but not in their values. We verified this by changing the encoding scheme on the original JPEG images.

Flickr: Flickr cleans up the metadata fields as other sites do. Unlike Facebook, it uses a constant quality ratio of 96 for both the luminance and the chrominance while re-compressing images. We have done extensive experiments that show that the changes in the pixel and DCT coefficients in Flickr is almost identical to that with Facebook for the set of images considered. Specifically, the majority of the DCT coefficients remain unchanged and less than 15 % are either increased or decreased by one (Fig III-B). *The fact that the processing on Facebook and Flickr are almost identical suggests that the modifications to DCT coefficients are not site specific.* We speculate that they are mostly likely due to watermarks inserted by the sites and only affect the LSB of the DCT coefficients. Typically, watermarks are inserted by modifying the LSBs to ensure that the image quality does not degrade significantly. Thus, one might expect that these processing functions on Facebook and Flickr are unlikely to change (if at all) over time.

Summary: (a) Some sites (Google+, Twitter) preserve the integrity of images to a large extent. Common steganography tools can be used directly on images uploaded on these sites. (b) Other sites (Facebook, Flickr) process uploaded images, thus making it difficult to use these tools directly. Specifically, metadata fields, pixels or DCT coefficients are exposed to the manipulation by these sites. Our key observation is that the modifications to the DCT coefficients are most likely due to watermarks and affect only the LSBs of these coefficients.

IV. Reliable Embedding on Facebook and Flickr

While secret content can be reliably exchanged via images on Google+ and Twitter, Facebook is today the most popular OSN. Similarly, today, Flickr is considered as the top photo-sharing site [38]. Thus, we ask the question: in spite of the processing that is performed on Facebook and Flickr, can we enable users to secretly communicate on these sites while simultaneously ensuring that the detection probabilities with steganalysis tools remain similar to that with common steganographic embedding approaches?

Our answer to this question is based on our observation that the failures with common steganography tools are because the messages are embedded in the LSBs of either the DCT coefficients or the pixel values (which are more prone to processing changes). Thus, to preserve secret content embedded in the LSBs one must use robust forward error correction (FEC) codes as with YASS. However, since these bits are often subject to processing changes, the overhead incurred will be high, thereby reducing the secret carrying capacity (shown later). Furthermore, as we also show later, the use of high degrees of redundancy is one factor that increases the chances of detecting the presence of a secret message via steganalysis.

Therefore, we ask the question: "Are there locations within an image that remain relatively unaffected after processing on Facebook or Flickr?" If there are, we could then embed secrets in such locations, possibly with much weaker FEC.

Recall that the maximum change in the pixel values is about 30 for almost all images (Fig. 1), and the maximum change observed in the DCT coefficients is 1 (Fig. 3). Intuitively this suggests that embedding the message in the higher significant bits of a DCT coefficient, as opposed to embedding it using the LSB, could protect it during the processing operations on the photo-sharing site. However, this approach poses a potential pitfall. To evade the detection using steganalysis of a message hidden within an image, there is an inherent tradeoff between preserving integrity by changing higher-order values and keeping the detection likelihood low.

Consider an example with a given color image, wherein a pixel is represented by 3 bytes, one each for the RGB dimensions. If two bytes only differ in the LSB, the represented colors are virtually indistinguishable to the human eye. A variation at the start of each byte causes more drastic color differences. In addition, to detect hidden messages, a steganalysis tool could examine the colors of adjacent "pixel pairs" and determine how close they are to each other. It could examine the rare occurrences of abrupt color changes within the image and flag the image if such occurrences are observed. In general, changing the higher order bits of pixels causes more drastic color changes resulting in easier detection.

In fact, several sophisticated steganalysis tools have been developed to detect steganographic embedding. As an example, one modern image steganography detection tool [34] adopts a machine learning approach trained to distinguish between the original and "stego-ed" images. The algorithm is sensitive to steganographic embedding changes, but is insensitive to the original image content. It also captures many dependencies among individual DCT coefficients; there is an increased likelihood that at least some of these dependencies will be disturbed by embedding. Because of this, common steganographic embedding tools typically hide data using the LSBs. Needless to say, there is a race between the development of new steganographic embedding solutions, and steganalysis tools to combat such approaches.

Our approach: Given this, we propose the embedding of secret information in the 2^{nd} least significant bit (2-LSB) in the DCT coefficients; this provides (as shown later) the best tradeoff between detection evasion and preserving the hidden messages on Facebook and Flickr [1]. In order to decrease the likelihood of detection via steganalysis, one can envision using a combination of LSB and 2-LSB embedding. For example, whether the changed bit of a DCT coefficient is its LSB or 2-LSB can depend on whether the chosen coefficient index in the image is an odd number or even. This mixed approach is referred hereafter as the LSB+2-LSB method.

We modify the open source stego tool F5 [4], which embeds the secret message bits in the LSBs of pseudo-randomly chosen *non-zero* DCT coefficients. We embed the message bits in the 2-LSB of these coefficients instead. In typical images, with both a length and width of about 1000 pixels, there are about 10,000 non-zero coefficients. The number of these usable coefficients is called the "image capacity". As an example, when the image capacity is 10,000 bits, by using 10% of the capacity to embed secret information, we can embed 125 bytes or characters. This translates to approximately 25 words (based on published statistics that show that there are about 4.5 characters per word on average [37]).

We choose an arbitrary image and compare it with its stego-ed version obtained with the LSB and 2-LSB methods, when a reasonable amount of data is embedded (10% of the image capacity). We use the peak signal to noise ratio or PSNR metric (typically used to quantify the difference between an image and its processed (noisy) version) to compare the stego-ed and original images. A high PSNR indicates that the quality of the original image is preserved well in the stego-ed image. The PSNR of the image with LSB and 2-LSB embedding, with respect to the original image, are 57.21dB and 56.82dB, respectively. The differences in the PSNRs with LSB and 2-LSB embedding for all other images in our candidate set were also very low (almost insignificant). This demonstrates that there is not a significant hit in the image quality with 2-LSB embedding as compared to LSB embedding.

Using the above three methods (the message length is 10% of the image capacity), we upload three sets of stego-ed images to Facebook and then download the images. We calculate the bit error rates (BER) from the retrieved messages.

BER behaviors: From Table III (see Column 1, Row 3), we see that embedding information in the 2-LSB of the DCT coefficients encounters *much fewer* bit errors ($\approx 1\%$) as compared to using the LSB ($\approx 15\%$). This is because, when the DCT coefficients are changed by 1 (recall Fig. 3), the LSBs are altered and so are the embedded data bits (if the embedding is done in the LSB). Note that, with a unit change in the LSB, the 2-LSB may be sometimes altered due to a carrier overflow or a borrowing from the LSB. While using the 2-LSB does not provide perfect error-free decoding, it comes really close. Partially inheriting the merits of 2-LSB, the mixed LSB+2-LSB method incurs a BER of about 8%.

Applying forward error correcting codes (FEC): Next, we seek to eliminate errors by applying FECs to the hidden message. Considering the small BER induced by the 2-LSB method, we expect the overhead to be minor. We experiment with Reed-Solomon codes [40] with different error-correcting abilities. A Reed-Solomon code is a linear block code represented in the form $[n,\ k]$; n is the length of the code word and k is the length of the message. The redundancy is (n - k) and, in general, up to $(n-k)/2$ errors can be corrected.

We experiment with three settings—with *[15,13]*, *[15,11]*, and *[7,3]* codes—with the LSB, 2-LSB, and the mixed LSB+2-LSB methods. The results are in rows 4 to 9 in Table III. We note that using the weakest code *([15,13])* protects the 2-LSB method from bit errors, while the LSB method needs the strongest code of all—*[7,3]*—to achieve the same result. The LSB+2-LSB method needs the code with medium strength *[15,11]* to eliminate bit errors. In terms of delivering the same amount of error-free secret data, the FEC overhead is about 13% for the 2-LSB method, about 58% for the LSB method, and about 27% for the LSB+2-LSB method.

Summary: By comparing the BERs between the conventional LSB stego method, the mixed LSB+2-LSB method, and our 2-LSB approach on Facebook, we find that embedding

[1]We do not pursue embedding in 3-LSB and above because 2-LSB embedding suffices for preserving secret messages with a low detection likelihood.

secret information in the 2-LSB jointly with a weak FEC is sufficient. It outperforms the LSB and LSB+2-LSB methods in terms of FEC overhead for a given message length.

A. Evaluation with steganalysis

Next, we use state-of-the-art steganalysis techniques to evaluate the likelihood of detection with our 2-LSB and LSB+2-LSB approaches. We compare this with the detection likelihood in cases where traditional steganography tools, which embed information in the LSBs of the DCT coefficients, are used. Our goal here is to show that our approaches do not significantly increase the detection likelihood when the same message capacity is delivered.

Steganalysis techniques in use: Steganalysis seeks to detect the presence of embedded data in an image (it does not attempt to extract the embedded message itself). To date, the most advanced steganalysis methods do supervised classification using machine learning tools like SVM or ensemble classifiers [33]. We use the ensemble classifier from [33], implemented in Matlab [3] along with the 548-dimensional CC-PEV features [7] [2]. As to commercial steganalysis products, Steganography Analyzer Artifact Scanner (StegAlyzerAS) developed by Steganography Analysis and Research Center (SARC) [14] is probably the best available steganalysis software in the market today. We use a limited time trial version with full functionality from their site.

Methodology: When using the ensemble classifier, we use a training set of 100 (each) normal and stego-ed images, respectively. The stego-ed images are produced by the traditional F5 tool which uses (i) the LSB method, (ii) our modified 2-LSB method, and (iii) the mixed LSB+2-LSB method. An equal number from both sources are considered. For both the normal and stego-ed images, we uploaded and then downloaded the images from Facebook.

Next, we apply the trained steganalysis tool on a test data set consisting of 100 different normal and stego-ed images (each). The false alarms on normal images contribute to the computed false positive rate, and missed detections on stego-ed images contribute to the computed false negative rates. When using StegAlyzerAS, we simply scan the folders containing normal and stego-ed images. We experiment on different sets of stego-ed images with embedded message lengths that consume 10%–50% of the image capacity. The embedded messages include the secret data augmented by error correcting codes. The FECs are *[7,3]* for LSB, *[15,13]* for 2-LSB and *[15,11]* for LSB+2-LSB methods, respectively. These FECs are chosen to ensure that the same BER (zero) is achieved with the three schemes. The DCT coefficients to be modified are pseudo-randomly chosen, and spread out evenly in the image.

Detection accuracy is defined to be the fraction of images that are successfully flagged by the steganalysis tools in use from among all the stego-ed images. We use terms detection accuracy and detection likelihood interchangeably.

Results and interpretation: Tables IV and V present the detection accuracy on stego-ed images produced by the academic and commercial tools we use. From Table IV, we see that the ensemble classifier can detect more stego-ed images constructed with the 2-LSB method than with the LSB method;

[2]CC-PEV was first proposed in [36] and its analysis is based on the use of an extensive set of DCT coefficients and other features.

Capacity used	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*
LSB	0.44	0.62	0.75	0.87	1.00
LSB+2-LSB	0.47	0.66	0.81	0.92	1.00
2-LSB	0.50	0.66	0.81	0.94	1.00

TABLE IV
DETECTION RATE OF ENSEMBLE CLASSIFIER (FALSE POSITIVE RATE 0.24)

Capacity used	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*
LSB	0.69	0.77	0.83	1.00	1.00
LSB +2-LSB	0.66	0.75	0.80	0.94	1.00
2-LSB	0.63	0.75	0.79	0.81	0.81

TABLE V
DETECTION RATE OF STEGALYZERAS (FALSE POSITIVE RATE 0.20)

however, the difference is insignificant. The detection rates with the mixed LSB+2-LSB method are higher than those with the LSB method, but lower than those with the 2-LSB method. This is to be expected since the mixed method distributes the changes over the LSBs and the 2-LSBs in images.

Thus, in typical regimes of interest (when the used image capacity is 10% or so), steganalysis on the images created with all methods exhibits very similar detection rates. As we aggressively embed more data into the images, the detection likelihood increases with all methods.

Surprisingly, the commercial tool StegAlyzerAS correctly categorizes a larger number of stego-ed images with the LSB method than with the 2-LSB method (see Table V). This is probably because the ensemble classifier uses machine learning while StegAlyzerAS relies on known stego signatures and identifiable patterns of specific steganography tools.

Most importantly, from columns 4 and 5 in Table III, we observe that when delivering the same amount of secret data,the 2-LSB approach with a weak FEC (2LSB+FEC [15,13]*) is less likely to be detected by* both *steganalysis tools than the traditional LSB approach with a strong FEC (LSB+FEC* [7,3]*) or the mixed LSB+2-LSB approach with a medium strength FEC (LSB+2-LSB+FEC* [15,11]*).* While embedding information in the 2-LSB can increase detection likelihood, so can increased redundancy; the latter results in a higher number of changes to an image to deliver the same amount of secret data. We observe here that, due to the reduction in the level of redundancy needed, the 2-LSB scheme can in fact out-perform the traditional embedding method and the mixed LSB+2-LSB method when used on photo-sharing sites. In fact, due to the low redundancy required by 2-LSB, the detection likelihood with StegAlyzerAS when using this approach is practically identical with and without FEC.

Finally, we observe that the two steganalysis tools have a false-negative rate of 40–50% in cases where only 10% of the image capacity is used for secret embedding. Furthermore, as seen in Tables IV and V, the false positive rates are also fairly high ($\approx$ 20%); given the millions of images uploaded on to Facebook and Flickr daily, the number of false positives will far outweigh the number of images correctly detected to have secret content. The detection accuracy will be even lower if lesser image capacity (say 5%) is used for secret embedding. Thus, if users take care to not embed secret information in the majority of the photographs that they upload, this suggests that the likelihood of detection is very low.

Summary: In summary, our experiments show that 2-LSB embedding decreases the observed BER in secret messages hidden in images uploaded to Facebook. While embedding in a higher order bit inherently increases the likelihood of detection with steganalysis, the reduced BER with 2-LSB decreases the level of redundancy required as compared to that required with LSB or mixed LSB+2-LSB embedding; this in turn decreases the likelihood of detection via steganalysis. Finally, we observe that the state-of-the art steganalysis tools only offer 40%–50% likelihood of detection in the common case wherein 10% of the image capacity is used for steganographic embedding; moreover, the false positive rates are about 20%. This suggests that 2-LSB embedding is a practical method for embedding secret content on images uploaded onto Facebook. We have done several experiments on Flickr that show similar results showcasing the efficacy of 2-LSB embedding. However, we do not show those results here due to space limitations.

V. Enabling Private Communication

Besides enabling successful secret message recovery, there is another fundamental requirement with regards to reliable private communications: the secret messages must *not* be recoverable by any party other than the legitimate recipients. Towards fulfilling this, we use encryption in conjunction with steganography for the pre-processing of the secret messages to avoid the exposure of the secret. Encrypting secret messages requires the establishment of secret keys between the communicating entities. In prior work, the existence of an out-of-band channel is assumed [19], via which such keys are established. However, in cases where censor authorities have pervasive access to information, the availability of such an out-of-band channel may be difficult (e.g., email, voice calls and Internet-based communication may be monitored). In this section, we first describe the censor's capabilities as part of our threat model; we then propose an approach for bootstrapping the private communication without any out-of-band channel, i.e., the covert channel is established by uploading images to exchange keys. Subsequently, we discuss the security properties provided by our approach.

A. Threat model

Censor abilities: A censor's characteristics depend on factors like motivation, resource and time in different contexts. The capabilities of a government-sponsored censor are influenced by the laws and policies. The threat model we present is similar to that in [19] and is one that we believe is able to capture the current capabilities of a censor. Nevertheless, censorship is an arms race; thus, as the censor's capabilities become more advanced and sophisticated, efforts towards evading censorship will also need to evolve in response.

We assume that the censor by default allows OSN access to users, but can: a) monitor all network traffic, b) can inspect all publicly available content on the OSN, and c) can access privately shared content on the OSN (e.g., via subpoenas). We believe that this assumption reflects the current state of censor behaviors. If the censor blocks users from accessing a particular OSN, users can communicate covertly on a different unblocked OSN. Even if the censor is capable of altering OSN content, it will be hard for it to determine what content to modify, since it cannot reliably detect steganographic embedding in images. Given this, we assume that the integrity of uploaded content is preserved and is not manipulated by the censor; the only cause of integrity loss is the processing on images done by the OSN for orthogonal reasons (e.g., to save storage and bandwidth). We assume that the OSN has no incentive to modify its image processing with the sole aim of disrupting potential use of steganography. Beyond these basic assumptions, the specific capabilities of a censor depend on the effort it expends in capturing and analyzing OSN content. We assume that the censor's abilities in doing so will ultimately be constrained by cost (given the scale of OSN content). In other words, we expect that it will be infeasible for a censor to analyze all uploaded images on OSNs.

We also assume that the censor has unlimited access to any steganalysis tool that is available and may develop its own tool targeting specific steganographic schemes, including our 2-LSB method. Note that we have shown in Section IV that a steganalysis technique trained to detect 2-LSB based embedding could have higher detection accuracy than common tools, but the increase is not significant.

Users and OSN accounts: We assume the users are who they claim to be; $Alice$'s account indeed belongs to Alice and not to some malicious third party posing as $Alice$. Protection against fake accounts is beyond the scope of this paper.

Finally, we assume that users do not post large volumes of secret content i.e., they do not embed messages in a majority of the images that they upload; this is unlikely in practice and in turn would make their data susceptible to batched or pooled steganalysis. Limits on the extents of secret embedding without being vulnerable to steganalysis is discussed in [32].

B. Covert channel to circumvent the censor

Next, we seek to build a covert channel which can be used by users to send/receive secret messages such that a censor can neither determine the existence of such messages nor intercept their content. To achieve this goal, we use cryptography in conjunction with our proposed 2-LSB steganographic scheme to embed secret messages in images uploaded to OSNs. As shown in Section IV, the stego-ed images using our 2-LSB scheme are detected only with low probability by the state of the art steganalysis tools; this offers users a certain level of deniability in the face of a censor's accusations. Encryption of the secret messages ensures that no one, except those who have the requisite key, can read the content. The communicating entities need to establish the secret keys before exchanging the secret. In what follows, we propose a protocol for bootstrapping the private communication with an in-band channel (i.e., via the images posted on OSNs).

C. Bootstrapping the covert channel

In order to establish covert communication channels via photo sharing sites, we propose that a user first embed her public key (using steganography) in her profile photograph (without loss of generality, we assume an OSN such as Facebook for this discussion). By uploading new profile photos, the public key can be changed.

Now, let us consider a scenario wherein users $Alice$ and Bob are friends on Facebook and have embedded their public keys on their profile pictures. $Alice$ and Bob also install our common bootstrapping software. Now let us assume that Bob wishes to initiate the establishment of a covert channel with $Alice$. In what follows, we first describe the protocol in brief and then describe the steps in greater detail.

Protocol overview: Given that $Alice$'s public key may or may not be embedded in her profile photo, and $Alice$ may or may not realize Bob's intent to build the covert channel, a handshake is necessary here. Consequently, our key establishment is an interactive procedure. First, Bob will need to alert $Alice$ of his intent to communicate. $Alice$ will need to send a response to let Bob know that she received his message. A session key for future communication can be exchanged at this time as well. Bob will need to send another confirmation to let $Alice$ know that the session key is agreed upon. All of these communications are carried out covertly by embedding messages in uploaded images. Once established, a secret conversation can last as long as needed.

Protocol details: The bootstrapping phase consists of the following steps.

1. First, the software on Bob's device fetches $Alice$'s profile photo and extracts the first L_k 2-LSB bits from the DCT coefficients, where L_k corresponds to the length of the key (the length is configured in the software). At this point, Bob does not know if what he has is $Alice$'s public key or is simply some arbitrary string, since $Alice$ may not have embedded a public key in her photo. The string of length L_k that is extracted is called K_A^{pu}.
2. Bob next encrypts a signal ("request") with K_A^{pu} and embeds it in an uploaded image. The request also contains metadata that indicates the length of the message (the 2-LSB bits it consumes), and a nonce.
3. If K_A^{pu} is a random string, and not $Alice$'s public key as assumed, $Alice$ does not respond to Bob's request. If $Alice$ has indeed embedded her public key in her profile photo, she may extract the hidden message (depending on when she views the image). At this point, she uses her private key K_A^{pr} to decrypt the request in Bob's image, thereby learning of Bob's intent to communicate.
4. If $Alice$ trusts Bob (e.g., that he is not a user controlled by the censor), $Alice$ then retrieves Bob's profile photo and obtains his public key K_B^{pu}. Note that, since Bob sent her a request, at this moment, $Alice$ knows for certain that Bob has included a valid key (K_B^{pu}) in his profile image and that she has not extracted a random string.
5. $Alice$ then creates a signal ("ack"), attaches a symmetric key k_s and the nonce associated with Bob's request. She encrypts this content with K_B^{pu}. The (secret) encrypted message is then embedded in an uploaded image.
6. Bob extracts the encrypted message from $Alice$'s image, decrypts it using his private key K_B^{pr}, checks whether the message is an ack, verifies the nonce, and extracts k_s. Note that a decryption failure at this stage indicates that $Alice$ did not respond to his request, and that this step together with step 5 is necessary for Bob to confirm that $Alice$ received his message and is aware of his intention of establishing the secret key.
7. Bob then encrypts a new signal ("ack2") with k_s and embeds this in a new image, which he then uploads.
8. $Alice$ extracts the secret from Bob's image, decrypts the message using k_s, and obtains the signal "ack2". At this point, $Alice$ has validated that Bob has the secret key k_s, and thus, the covert channel is established.

At the end of these steps, all the secret messages exchanged between $Alice$ and Bob are encrypted with k_s. k_s can also be used as the seed to generate the pseudo-random series of DCT coefficients chosen to carry the secret message. For instance, the sequence of DCT coefficients that are changed, can be generated by a pseudo-random number generator that uses a concatenation of k_s and the photo ID as the seed. This ensures that the DCT coefficient sequence changed differs across photos; one can expect this to lower the detection probability.

Key selection: Recall that $Alice$'s public key K_A^{pu} is embedded in her profile photo. For some cryptographic schemes, the public key is only divisible by large primes (small prime numbers such as 2, 3 or 5 are not factors of the key). This may allow Bob, or even an adversary, to *suspect* that a public key is embedded in a profile photograph. However, with many state of the art cryptographic techniques (such as Elliptic Curve Cryptography (ECC) [2]), the keys are divisible by small prime numbers and thus, this problem does not arise. Nevertheless, as identified in prior work [26], one should avoid using bad public keys generated by key generation implementations that do not use sufficient randomness.

D. Security properties

A ***Man-in-the-middle (MIM) attack*** could occur during a key exchange, wherein an eavesdropper, say $Chloe$, somehow intercepts the communication between $Alice$ and Bob. $Chloe$ could send her own public key and mislead Bob into believing that he has $Alice$'s public key (and similarly deceive $Alice$ into believing she has Bob's public key). However, since we assume that photo-sharing sites preserve the integrity of uploaded content, the censor can launch a MIM attack only in one of two ways: (i) it can compromise a user's account and replace the user's profile photo, or (ii) it can intercept a user's network traffic when the user is uploading/downloading photos from the photo-sharing site and modify photos on the fly. Preventing compromises of user accounts is beyond the scope of this paper. On the other hand, users can prevent the second type of MIM attack by using HTTPS to upload/download content to/from the photo-sharing sites. In fact, Facebook, Twitter, and Google+ use HTTPS by default. When users use HTTPS, the censor will have to subvert users' lookup of SSL certificates in order to perform a MIM attack. Again, making HTTPS resilient to such attacks is beyond the scope of this paper.

Detection of the existence of embedded key: As discussed earlier, steganalysis tools are far from perfect. As seen in Tables IV and V, the likelihood of false positives and detection misses are high. In fact, if only 10% of the image capacity is used for embedding secret information, our study suggests that the tools yield a detection miss rate of about 50%. This, combined with the high false positive rate, makes steganalysis difficult, if not impossible, if encrypted content is hidden.

Forward and backward secrecy: Since the key (k_s) is per-conversation rather than per-user (i.e., a fresh key is established for each conversation), our protocol ensures that undesired users (and censors) are unable to uncover any additional information from past or future conversations with k_s. In contrast, consider a Diffie Hellman-style key exchange, where every user includes her public key component in her profile photo, and either party can derive a shared key without any interactions. In this case, the shared key between a pair of users would be fixed across all of their conversations, hence putting their past and future communications at risk if the shared key is leaked. Also note that any user A's public and private keys

(K_A^{pu} and K_A^{pr}) can be updated periodically by changing A's profile photo. This will limit the effectiveness of a brute force attack in discovering A's private key (K_A^{pr}).

Access and sharing patterns. For step 3 to work reliably, $Alice$ needs to check for the request signal in a sizeable fraction of photos shared with her, but not necessarily all; Bob can retry step 2 if he sees no response from $Alice$ for a timeout period. When $Alice$ does access Bob's photo containing the request signal, this will not leak much information since Bob can share that photo with all of his friends on the OSN, who will all also access the photo. On the other hand, in response to receiving a request from Bob, $Alice$ does not need to immediately share a photo in order to respond; she can embed an ack message whenever she uploads a photo next.

E. Discussion

Finally, we discuss two issues related to making the bootstrapping phase efficient.

Key length: In our implementation of the above key exchange process, we use a publicly available implementation of the RSA algorithm to generate the public key (of length 1024 bits). It is widely known that generating a public key component with Elliptic Curve Cryptography (ECC) may be a better alternative to RSA [2]; with ECC, the key length is much shorter for providing similar security. For example, a 160 bit key generated with ECC provides equivalent security as a 1024 bit RSA key. The reduction in the key size directly translates to the embedding of a shorter secret message in the profile picture. Since shorter messages are harder to detect, this in turn will lead to a further decrease in the detection likelihood with steganalysis tools.

Embedding multiple messages in the same image: If Bob wants to send a "request" to other users besides $Alice$ (step 2), he can encrypt copies of the "request" using these users' public keys and embed all the ciphers back to back in the same image. For instance, with a 160 bit key generated with ECC, if 10% of the image capacity is to be used, 6 requests can be packed in an image. Since the ciphers will have the same length, the entire secret message is composed of segments of the same length. After a recipient extracts the composite message (consisting of the segments) from an image, he can decrypt each of these segments using his private key. It does not matter that there are segments which are not meant for him. As long as he sees the signal "request" in one of these segments, he would know that he is one of the intended recipients. In this way, Bob can bootstrap the communication with multiple users at the same time. A similar approach can be used for responding to a request or an ack (in steps 5 and 7). We will consider such implementations in future work.

VI. Conclusions

In this work, we build a covert communication channel using uploaded photos on popular public photo-sharing sites. While using steganography for this purpose had previously received some attention, many nuances were ignored. Our in-depth measurement study shows that the processing performed by online sites on the uploaded photos destroys the secret message in many cases. Our study also reveals the reasons for this loss. Based on the understanding developed with our study, we propose a new approach to ensure the integrity of a hidden message, while at the same time maintaining a low likelihood of detection via steganalysis. Finally, we also propose and implement a protocol wherein users can establish keys to encrypt the messages, via an in-band channel on the photo sharing site.

References

[1] Camouflage. http://bit.ly/bANwWp.
[2] The case for elliptic curve cryptography. http://1.usa.gov/WTeCRn.
[3] Ensemble classifier. http://bit.ly/W77ux4.
[4] F5. http://bit.ly/UBnnQ1.
[5] Facebook Help Center. http://on.fb.me/zS3eBc.
[6] Facebook photo upload compression. http://bit.ly/cBUcZR.
[7] Feature Extractors for Steganalysis. http://bit.ly/YCViV2.
[8] Flickr-Help-Photos. http://www.flickr.com/help/photos/.
[9] JPEG analysis utilities. http://bit.ly/XlFBl8.
[10] JPEG Group. libjpeg. http://www.ijg.org/.
[11] JpegX. http://bit.ly/15rcLXd.
[12] JSteg. http://bit.ly/XhxJEt.
[13] OutGuess. http://www.outguess.org/.
[14] Steganography analysis and research center. http://www.sarc-wv.com/.
[15] StegHide. http://bit.ly/Y4UWYu.
[16] The Facebook blog. http://on.fb.me/aVLPVz.
[17] Twitter Help Center. http://bit.ly/q9DJb3.
[18] P. Alvarez. Using extended file information (EXIF) file headers in digital evidence analysis. In *Intl. Journal of Digital Evidence, Economic Crime Institute*, 2004.
[19] S. Burnett, N. Feamster, and S. Vempala. Chipping away at censorship firewalls with user-generated content. In *USENIX Security*, 2010.
[20] A. Castiglione, G. Cattaneo, and A. De Santis. A forensic analysis of images on online social networks. In *International Conf. on Intelligent Networking and Collaborative Systems*, 2011.
[21] A. Castiglione, B. D'Alessio, and A. De Santis. Steganography and secure communication on online social networks and online photo sharing. In *International Conference on Broadband and Wireless Computing, Communication and Applications*, 2011.
[22] C. C. Chang, T. S. Chen, and L. Z. Chung. A steganographic method based upon JPEG and quantization table modification. In *Information Sciences*, 2002.
[23] Steganographic Command and Control: Building a communication channel that withstands hostile scrutiny. http://bit.ly/bU6dsG.
[24] The JPEG committee home page. http://www.jpeg.org/.
[25] CMU database. http://www.cs.cmu.edu/ cil/v-images.html.
[26] S. Farrell. Public Key Checking Protocol. Internet draft. 2012. http://bit.ly/12FzaQV.
[27] C. Hass. JPEGsnoop 1.5.2. http://bit.ly/12j0yy.
[28] Google+ help. http://bit.ly/rp5No9.
[29] Internet Censorship in China. http://nyti.ms/130r4Ce.
[30] Steganography in Social Media. http://bit.ly/W76qcl.
[31] Neil F. Johnson and S. Jajodia. Exploring steganography: Seeing the unseen. In *IEEE Computer*, 1998.
[32] Andrew D. Ker. Batch steganography and pooled steganalysis. In *Information Hiding Workshop, Springer LNCS*, 2006.
[33] J. Kodovsky and J. Fridrich. Steganalysis of JPEG images using rich models. *SPIE, Electronic Imaging, Media Watermarking, Security, and Forensics*, 2012.
[34] J. Kodovsky, J. Fridrich, and V. Holub. Ensemble classifiers for steganalysis of digital media. *IEEE Transactions on Information Forensics and Security*, 2012.
[35] S. Nagaraja, A. Houmansadr, P. Piyawongwisal, V. Singh, P. Agarwal, and N. Borisov. Stegobot: a covert social network botnet. In *Intl. Conf. on Information hiding*, 2011.
[36] T. Pevny and J. Fridrich. Merging markov and DCT features for multiclass JPEG steganalysis. *SPIE, Electronic Imaging, Security, Steganography, and Watermarking of Multimedia Contents*, 2007.
[37] J. R. Pierce. *An Introduction to Information Theory: Symbols, Signals and Noise*. Dover Publications, 1980.
[38] Flickr Popularity. http://bit.ly/bwqwRq.
[39] N. Provos and P. Honeyman. Detecting steganographic content on the internet. In *NDSS*, 2002.
[40] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 1960.
[41] K. Solanki, A. Sarkar, and B. S. Manjunath. YASS: Yet another steganographic scheme that resists blind steganalysis. In *Intl. Workshop on Information Hiding*, 2007.
[42] SecreTwit: Social Steganography. http://bit.ly/W0NnWB.
[43] K. Wilson. Ghosthost: http://bit.ly/Xhycqb.