StegaGram

Alex Renger (arenger@uccs.edu) and David Stites (dstites@uccs.edu)

ABSTRACT

The use of cryptography for securing private communication can raise suspicion among those who take note of the communication. A third party may not be able to decrypt the sensitive information, but the mere awareness of its existence can sometimes be too costly of a contingency for those attempting to communicate secretly. Steganography is used these cases, but it should still be used in combination with cryptography. Additionally, an environment in which people are motivated to use steganography poses new challenges for key management and distribution.

By way of familiarizing ourselves with these methodologies and their related challenges, we created a mobile application that employs asymmetric cryptography, effective steganography, and a novel approach to key distribution.

I. Introduction

THE idea of communicating information using secret code words that are understood only by the sender and receiver has been around for thousands of years. Cryptography has taken many forms and had many different uses throughout history, but never has it been used as widely or as frequently as it is today. With computers now involved, it is constantly used for many different applications. These applications include securing financial transactions, enforcing copyright laws, validating message authenticity, and protecting private communication. When used as a means to protect personal communication between two private parties, however, cryptography used by itself has one primary disadvantage: suspicion.

Packet analyzers and intrusion detection systems are becoming increasingly common, as is the number of people who are comfortable dealing with raw streams of digital data. These tools allow for an organization to monitor all forms of TCP/IP traffic that pass in and out of its systems. For example, a free IDS suite, known as Snort [9], can be easily configured to notify a system administrator if email traffic between two given IP addresses contains certain keywords and/or PGP signatures. With this in mind, the use of cryptography alone to protect private communication is only sufficient when those who are communicating have no concern that others are aware of the fact that they are sharing secret information. This is probably true for the majority of private communication, but it is not true in all cases. Addressing this issue is the primary motivation of a technique known as steganography, which attempts to hide the very existence of private communication.

While the application of steganography is narrow, it will never be irrelevant. There will always be people who, for good and bad reasons, want to conceal the existence of their communication. On May 1st, 2012, CNN released a report [1] about a man who was apparently in the possession of over 100 al Qaeda documents that were hidden inside video files. The man had been stopped and questioned by police in Berlin, one year earlier. He was detained, and German cryptologists later discovered the documents while examining the contents of memory cards found on his person.

Other examples in which individuals might employ steganography include journalists in non-friendly territories, government intelligence agents, and those trafficking in industrial espionage. Of these, governments and large companies are the most likely users. They must have an interest in at least detecting hidden communication, and possibly in employing it themselves. Of course, it is also an active field of academic research [2][3][7].

II. Motivation

Neither cryptography nor steganography are new concepts, but their methods are always changing. Our primary goal was to become familiar with the current methods used in these areas, by way of creating a mobile application that employs them. We built an iPhone application, named StegaGram, which allows two people to easily exchange messages that are both encrypted and hidden during their transport between devices. It is important that the application be intuitive and easy to use, since usability can have a direct impact on security [8].

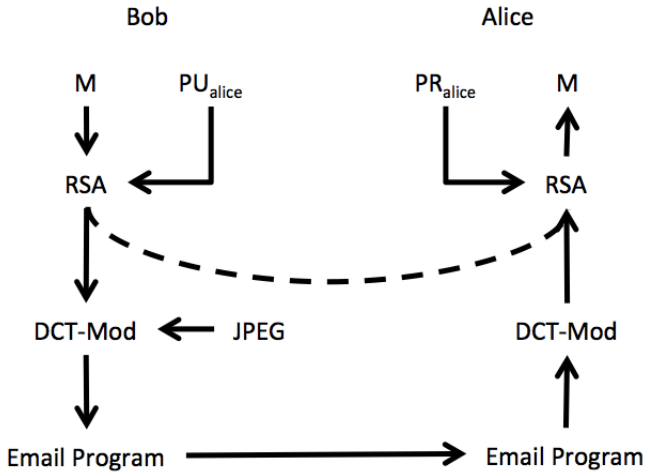
Ideally, whenever steganography is used, it should be combined with cryptography. This way, even if the methods of hiding the communication are compromised (i.e., the existence of covert communication is detected), the actual contents of the message will be computationally infeasible to obtain. Figure 1 gives an overview of these two main concepts used by StegaGram. A message from Bob is first encrypted using the RSA algorithm. Details about the encryption will be discussed in section 3. The encrypted message is then hidden within an image using DCT modulation, and then passed to the Mail program. Details about DCT modulation will be discussed in section 4. When Alice receives the image, she saves it to the device and then opens it within StegaGram. StegaGram extracts the encrypted message and then decrypts it to obtain the original message.

III. Cryptography

There are many different ways to encrypt and decrypt data. This section describes why we chose to use the RSA algorithm as the encryption method within StegaGram, and how we came to devise an uncommon method of exchanging RSA keys.

Passwords are frequently used as a means to protect confidential information. As we all know, short passwords are not very secure, and long passwords can be difficult to remember. We chose to avoid these problems by using long (2048-bit), randomly generated keys and storing them within the application itself. This provides excellent security without the need to remember a password, but adds a new problem of how to transfer these keys between phones. We further chose to use asymmetric keys, rather than symmetric keys, for reasons we will describe shortly.

Figure 1

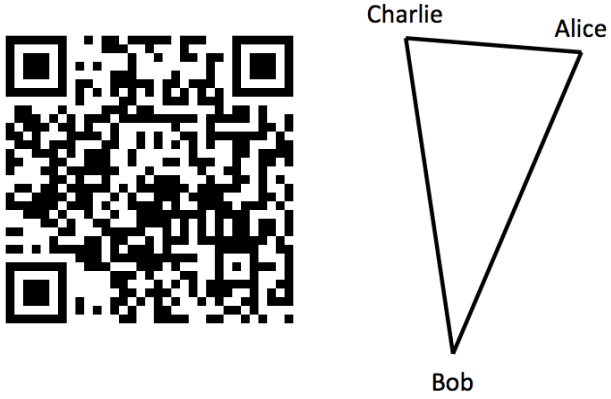


The problem of how to securely exchange public keys between two devices is obviously not new, but there are added complications when the detection of cryptographic communication is a concern. We assume that an application like StegaGram would be used in such an environment (one we will to refer as a Cryptographically Sensitive Environment, or CSE), and have therefore considered its restrictions in our design. The public keys should not be simply transmitted (e.g., via a local wireless network) because this exchange would clearly raise suspicion within a CSE.

As an alternative, the keys (like the messages) could first be hidden in an image before being sent across a network. This would avoid the problem of suspicion, but would be inconvenient and technically susceptible to a Man-in-the-Middle (MIM) attack, if the steganographic methods were compromised. The MIM attack is normally solved via involvement of a Certificate Authority (CA), but again, we considered such involvement to be infeasible within a CSE. Therefore, we decided to exchange the keys via Quick Response (QR) Codes. An example of a QR Code is shown in Figure 2a.

The use of QR Codes allows for easy transfer of the keys in a manner that cannot be detected via network observations in a CSE (since no network connection is even involved) and that is obviously not susceptible to a MIM attack. It does require that some of the keys be exchanged in-person, but not all of them. This leads to our choice of asymmetric keys over symmetric keys.

Asymmetric keys allow for digital signatures, which can in turn be used to build a "Web of Trust", similar to that used within the PGP community. Consider a simple example in



Figures 2a and 2b

which Bob lives in Colorado and Alice and Charlie live in Wyoming (Figure 2b). Alice visits Bob in Colorado and exchanges public keys via QR Code transfer and then returns to Wyoming. Charlie would then like to send a secure message to Bob. In order to do so, Alice passes Bob's public key to Charlie via QR Code transfer. At this point, Charlie can trust that he has received a genuine copy of Bob's public key, based on his trust in Alice. Alice acts as a sort of CA in this instance. However, only half of the problem is solved. Charlie could send his public key remotely to Bob after encrypting it with Bob's public key, but Bob is not assured that Charlie really created the message. If Bob's public key is truly public (as is generally a good assumption), then someone else could be pretending to be Charlie. The solution, of course, is for Charlie to first have his public key signed by Alice. Charlie then encrypts the combination of his public key and Alice's signature of his public key and sends this to Bob. To clarify, Charlie would send the following to Bob:

$$E(PU_{bob}, PU_{charlie} \,||\, E(PR_{alice}, H(PU_{charlie})))$$

In this equation, H is a cryptographic hash function, PU represents a public key, and PR represents a private key. Bob can then confirm that only Alice could have signed Charlie's public key, and he can accept it as genuine based on his trust in Alice. This is a simplification of the key-introduction methodology proposed by Phil Zimmermann in his manual for Pretty Good Privacy (PGP) [4].

The above approach can be used by StegaGram to allow for secure key exchange when a QR Code transfer is not possible, provided at least one trusted person has traveled between the

geographically separated locations. The initial version of StegaGram does not fully implement the signing of public keys or their transmission via steganography between remote locations, but it does allow for the public keys of others to be passed via QR Code. In context of the above example, it allows for Alice to pass Bob's key to Charlie, but Charlie cannot yet send his key to Bob. Full support of the Web of Trust is part of the StegaGram roadmap.

IV. Steganography

There are many ways to hide information within information. Like cryptography, steganography has been around for a couple millennia, and it probably has as many variants as it does people who desire to employ them. Just as needles are harder to find in larger haystacks than they are in smaller ones, it is easier to hide information when there is a large amount of "cover" information in which to hide it. This is why the applications of steganography in the digital information realm tend to focus on media files, namely audio, sound, images, and video. StegaGram uses images as the cover media. This provides enough space to hide information but not so much as to prevent it from being commonly sent by email.

When a photo is taken with the camera on the iPhone, it is stored in the JPEG image format. It could therefore be considered the "native" image format for the phone, and is likely the most common format used for the images that are transmitted between iPhones. For this reason, we chose to use preserve the JPEG format for the images that contain hidden information. Figure 3 gives a high-level overview of the types of steganography that are related to digital image formats.

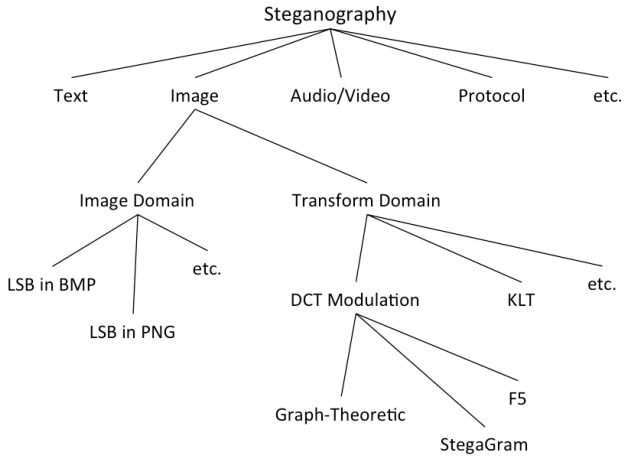


Figure 3

There are two primary ways of embedding data within an image; these are known as Image Domain and Transform Domain. The Image Domain methods are used with formats that are lossless, like BMP, GIP, or PNG. Among these methods, the most common is simply the use the least significant bit (LSB) of a particular pixel color for the message data while preserving the rest for the image. In most cases, the color depth is enough to make small changes like this visually imperceptible. This approach cannot be used with JPEG images, because the JPEG format itself does not always preserve the LSB of a given pixel color. In contrast to the lossless image formats, JPEG is among the formats known as "lossy", because it makes a compromise between image quality and the disk space needed to store it. (There are versions of the JPEG format that are lossless, but its lossy version is the most common.) It transforms the image into a format that lends itself to efficient compression for storage. Steganographic methods used with formats that involve an intermediate transformation are grouped into the Transform Domain. Methods within the Transform Domain are more robust (resistant to minor changes to the cover image after the data has been embedded) than those within the Image Domain because by nature they spread the message data across more of the cover image [2].

At the heart of the JPEG format is a process called discrete cosine transformation (DCT). Its details are beyond the scope of this report, but it is sufficient to know that the process yields a series of numbers (coefficients) that are used to represent an image. The most common method of embedding data within a JPEG image is to modulate these DCT coefficients. In the next section, we explore two naïve methods of DCT modulation, by way of introducing more advanced methods in the section thereafter.

A. Rudimentary Steganalysis

Of course, the goal of any steganographic method is to keep the cover data (in this case, an image) as close to original as possible. It is trivial for a given method to produce a cover image that is visually identical to the original. As this section will show, it is more difficult to find a method that is not detectible by statistical analysis of its digital representation.

1) Method 1

The most basic approach to DCT modulation is simply to adjust the coefficient up or down by 1 (i.e., use the LSB of the DCT coefficient), as needed to represent one bit of the message data. We refer to this approach as Method 1. To describe the problems with Method 1, we must first introduce the concepts of first- and second-order statistics. An excellent summary of these terms is found in a paper related to medical imaging [5], quoted below:

"First-order statistics are calculated from the probability of observing a particular pixel value at a randomly chosen location in the image. They depend only on individual pixel values and not on interaction [with] neighboring pixel values. The mean of image gray intensity is an example of the 1st-order statistic. Second-order statistics are calculated from the probability of observing a pair of pixel values in the image that are some vector D apart. Note that second-order statistics become first-order if D = (0,0)."

This quote refers to pixels, but the same concepts can be applied to DCT coefficients. An example of a first-order statistic of a JPEG image is the number of DCT coefficients

that have a value of zero. A histogram of DCT coefficients, then, is a collection of first order statistics. We will use such a histogram in our evaluation of Method 2. An example of a second-order statistic of a JPEG image is easier to visualize using Figure 4. This figure shows an example of DCT coefficients laid out in a matrix format. Matrix transformations are essential to the JPEG compression algorithm, as is the placement of a given DCT coefficient within a matrix. A possible second-order statistic would be the likelihood of finding a non-zero value in the farthest right column of the bottom row.

Given these concepts and examples, we can now describe why Method 1 is easy to detect, and is therefore ineffective steganography. Due to the nature of the JPEG compression algorithm, the number of zero-valued DCT coefficients is highly disproportional to the number of non-zero-valued coefficients. Furthermore, the non-zero-valued coefficients are concentrated in the upper-left portion of the matrix, as exemplified in Figure 4. The reasons for this are again beyond the scope of this report, but it is necessary to point out that it is statistically unlikely for a coefficient in the lower-right portion of the matrix to have a non-zero value. However, it is quite likely that an image produced by Method 1 would have values of 1 or -1 located in the lower-right region of the matrix. Therefore, Method 1 would be easily detectible using basic second-order statistics.

$$\begin{bmatrix} -27 & -3 & -7 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -3 & 1 & 1 & 0 & 0 & 0 \\ -2 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -2 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4

2) Method 2

A refined approach would be to modulate only non-zero values. Method 2 is essentially the same as Method 1 except that it adds this discrimination. This avoids the problem previously described, but it is still easy to detect via first-order statistics. The diamonds in Figure 5 show a histogram of DCT coefficients from an original (unmodified) image. The series marked with X's is the histogram after a message has been embedded in the image. The vertical axis is logarithmic, in order to ease visual comparison of the values. Notice that the number of even-valued coefficients has increased and the number of odd-valued coefficients has decreased. This is intuitive, assuming that the message data has an approximately equal number of one-bits as it does zero-bits. Because zero-valued coefficients are left unmodified, any coefficient with a value of 1 or -1 that needs modulation must become a 2 or -2, respectively -- but there are far less 2's to work with at the start. There is a similar imbalance between 3 and its neighbors, and so on across the histogram, adding weight to the even values and taking it from the odd values.

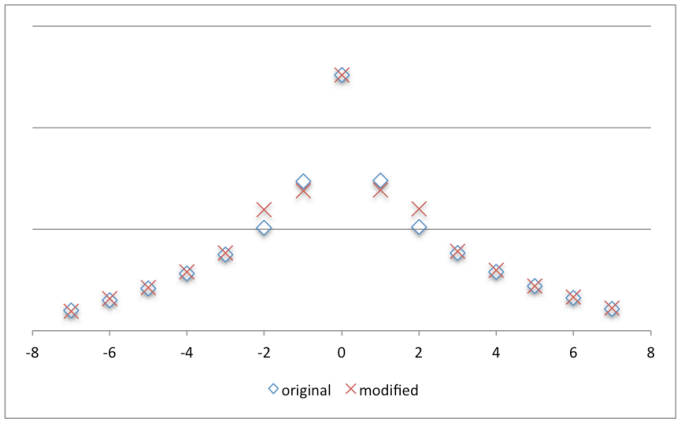


Figure 5

To prove how this observation can be leveraged to detect hidden data, we implemented Method 2 and tested it using 100 JPEG images and 100 randomly generated binary data files. We then measured the ratio of even-valued coefficients over odd-valued coefficients, excluding zero, before and after the images were modified. We call this the even-over-odd (EOO) ratio, or the EOOR, pronounced like the donkey's name in Winnie-the-Pooh books. A plot of the EOOR values across the 100 original images is shown in Figure 6. The six obvious outliers were removed, and the mean and standard deviation was calculated over the remaining set. The sum of this mean and standard deviation can then be used as a threshold when analyzing the modified images. Fifteen of the original images exceeded this threshold (0.534), and ninety-four of the modified images exceeded it. Therefore, this simple test can predict with 79% accuracy whether data is hidden in a given image. Of course, there are various ways to refine this approach. One simple improvement is to only calculate the EOOR over the band of coefficients from -4 to 4, inclusive. Using that approach, and removing the same six outliers, the accuracy was increased to 93%.

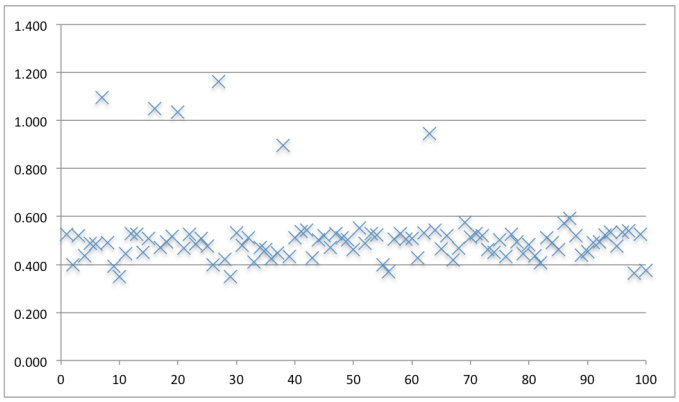


Figure 6

Note that if it were necessary to automate the above tests, the outliers could be removed by calculating the three quartiles, the inter-quartile range (IQR), and the upper and lower fence values across the ratios from the original set. The upper fence is typically defined to be Q3 + 1.5(IQR), and the lower fence

Q1 - 1.5(IQR). The outliers, then, are those that lie below the lower fence or above the upper fence. This approach identifies the same six outliers that were manually removed during our tests. It turns out that these outliers correspond with images that were rotated to the portrait view, as opposed to the more-common landscape view. We theorize that the EOOR for those images is thrown off by the "dummy data" that is used to fill incomplete blocks used in the JPEG compression algorithm. If this were the case, the test could be improved further by identifying such data and excluding it from the calculation of the ratio.

B. Preservation of First Order Statistics

The problem shown to exist with Method 2 and others like it can be solved, or at least mitigated. Stefan Hetzl and Petra Mutzel devised a way to preserve the first order statistics by using a method they called Graph-Theoretic Steganography [3]. We refer to this method as GTS. The key to their approach lies in their goal of *swapping* sample values when possible, rather than simply overwriting them. For example, if a given DCT coefficient has a value of 1, but the string of message bits requires it to be adjusted, then a 2 from elsewhere in the array of coefficients is swapped with the current coefficient. If this were possible for every coefficient that needed modulation, then the first-order statistics would be perfectly preserved. They implemented their algorithm in a tool they called steghide [6], which they then released to the public as open source.

Code Block A

```
void StegaJpg::c_char( unsigned char c,
std::vector<unsigned long>::iterator& it )
{
   for ( int i = 0; i < 8; i++ ) {
      short d = Array1[*it];
      if ( ( c & 1 ) ^ ( d & 1 ) ) {
         short away   = ( d < 0 ) ? -1 : 1;
         short toward = away * -1;
         if ( (abs(d) == 1) ||
              (dhist.get( d + away ) <=
               dhist.get( d + toward )) )
         {
            Array1[*it] += away;
            dhist.inc( d + away );
         } else {
            Array1[*it] += toward;
            dhist.inc( d + toward );
         }
         dhist.dec( d );
      }
      it++;
      c >>= 1;
   }
}
```

We downloaded the source code for steghide, compiled it, and ran tests with it. Indeed, their algorithm does an excellent job, but it might be overly complex. They build a graph in which vertices correspond with the necessity of making a change to the cover data and candidates for an exchange are connected by an edge. The goal of first order statistical preservation during the embedding process then becomes synonymous with the combinatorial problem of finding a maximal matching in a graph. While searching for an effective steganographic algorithm to include with StegaGram, we had additional interest in its memory and CPU utilization, since it will be running on a mobile platform. The GTS approach, though effective, made us curious as to whether there was a simpler means to achieve the same goal. With this in mind, we created a simpler algorithm that turns out to yield comparable results using fewer CPU cycles and less memory. We refer to this algorithm as Differential Histogram Steganography, or DHS.

The key data structure behind the DHS method is a histogram that can contain negative values. It is used to represent the difference between the original histogram and the current histogram, as the algorithm progresses. While the message is being embedded, the differential histogram is used to decide whether to modulate up or down. Then, after the last bit of message data has been embedded, the remaining portion of the image is be modified to offset (or rebalance) the changes that were made while the message was being embedded.

Differential Histogram Steganography:

1) Create an array of all the DCT coefficients.
2) Create a second array that contains indexes to the first array. Only indices to values that are non-zero should be stored in the second array.
3) Scramble the second array by looping L/2 times, where L is the length of the second array. During each iteration, randomly choose two different numbers in the second array and swap them. A secure PRNG should be used with a known seed value so that the same sequence can be obtained later to extract the message.
4) Initialize the differential histogram. This is simply a map between two numbers. In C++, for example, a `std::map<short,int>` object would suffice, but we found that a simpler, custom implementation performed faster.
5) While embedding the message data, if modulation is needed for a given DCT, then decrease the count in the differential histogram for that DCT (to denote a deficit), and increase the count for the value to which the DCT was modulated (denoting a surplus). When a modulation is needed, modulate to the neighbor that has the greater deficit. Of course, a DCT of +-1 is a special case in which the choice should always be +-2, respectively. Shown in Code Block A is the C++ code we used to hide one byte of message data. The function receives as input the octet to be hidden and an iterator over the array created in step 2.
6) After the data has been embedded, the differential histogram will typically contain deficits for odd-valued DCT coefficients and surpluses for even-valued DCT coefficients. To correct this, continue to modify if the image, even though no data is being embedded. If the count for a given DCT has a surplus, then check its neighbors to look for a deficit. If both have a deficit, choose the one with the greater deficit. Decrement the count for this DCT and increment the count for the chosen

neighbor. If neither neighbor has a deficit, make no change to the DCT or the differential histogram. Code Block B shows an example of this step. The `getMag()` method of the `dhist` class returns the cumulative distance from zero. If `getMag()` returns zero, then the image is balanced. It is a minor optimization that can be safely ignored.

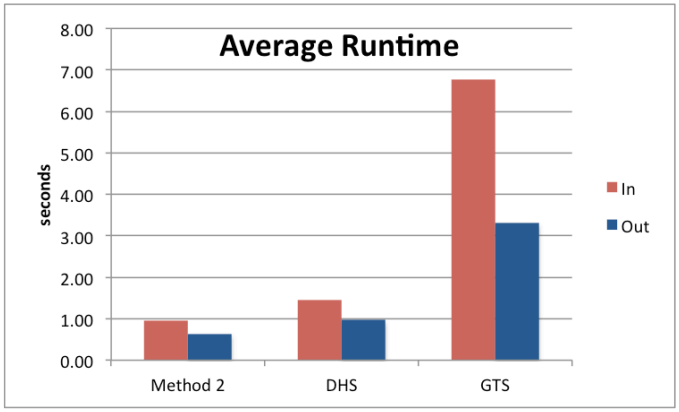
Code Block B

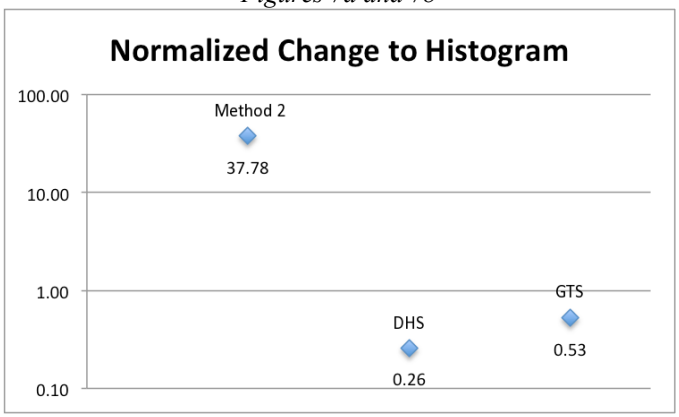
```
void StegaJpg::balance(
   std::vector<unsigned long>::iterator& it )
{
   while ( ( dhist.getMag() != 0 ) &&
           ( it < Array2.end() ) )
   {
      short d = Array1[*it];
      if ( dhist.get( d ) > 0 ) {
         // surplus - so determine which neighbor
         // is lacking the most (if any)
         short x = ( dhist.get( d - 1 ) <
                     dhist.get( d + 1 ) ) ? -1 : 1;
         if ( dhist.get( d + x ) < 0 ) {
            Array1[*it] += x;
            dhist.dec( d );
            dhist.inc( d + x );
         }
      }
      it++;
   }
}
```

Note that if step 3 were not performed, the first order statistics would still be preserved but the latter half of Array1 would, on average, have a lower EOOR than the first half. This could be detected, of course, which would weaken or compromise the steganographic algorithm. Also note that the preservation of first order statistics decreases the size of the data that can be hidden in a given image, space must be left rebalance the histogram. Both GTS and DHS require this extra space.

To compare DHS with GTS, we used the same 100 images with which we tested Method 2. For each image / data file pair, we used Perl's `Time::HiRes` module to measure how long it took to embed the data file in the image using each method. We then extracted the data from the image and compared the result to the original via CRC checksum, so as to confirm correctness of each implementation. To quantify the impact on first order statistics, we created a simple tool that compares the histograms of the images. It outputs the total difference in number of coefficients between the original and modified histograms. For example, if the modified histogram showed 16,539 occurrences of the coefficients with the value of 7, but the original histogram had 16,541, then 2 would be added to the total for index 7. The same is done for every index across the histogram and the sum is reported. We normalized this sum against the number of bits in the message data. Figures 7A and 7b show the results of our comparison. We also included Method 2 in our tests as an interesting reference.

We were surprised to find that DHS actually showed a lower overall impact on first order statistics. We expected GTS to be the most effective but to have a higher average runtime. It did have the highest average runtime, but it was slightly less





Figures 7a and 7b

effective, on average, than DHS. It is worth noting that the histogram comparison tool showed that GTS frequently made fewer than 20 changes and sometimes yielded an exact match of the histogram. The number of changes made by DHS never fell below 160 and averaged over 800, but the average for DHS was raised due to about two-dozen high points. In other words, even though DHS shows a better mean, GTS has a better median. This could be due to a bug in the steghide implementation, as the GTS algorithm itself appears to be very sound. We chose to use the DHS algorithm in StegaGram.

V. Conclusion

We set out to create a mobile application that uses both cryptography and steganography to provide secure communication. We reached this goal and learned a great deal along the way. The final product incorporates state-of-the-art cryptography with a novel approach to key distribution and additionally employs effective steganography. We created our own steganographic method that yields comparable results to the Graph-Theoretic approach and yet is simpler to implement and has lower runtime. In a world where cryptographic applications are ubiquitous and the use of steganography is on the rise, we are thankful to have had the opportunity to familiarize ourselves with these concepts.

Acknowledgment

We thank the authors of and contributors to the following open source projects, used by StegaGram: libjpeg [10], ZebraXing [11], OpenFlow [12], libqrencode [13].

REFERENCES

[1] N. Robertson, P. Cruickshank, T. Lister. "Documents reveal al Qaeda's plans for seizing cruise ships, carnage in Europe". CNN, Tuesday May 1, 2012

[2] T. Morkel, J.H.P. Eloff, M.S. Olivier, "An Overview of Image Steganography", Proceedings of the Fifth Annual Information Security South Africa Conference (ISSA2005), Sandton, South Africa, 2005.

[3] S. Hetzl and P. Mutzel. A graph-theoretic approach to steganography. In J. Dittmann et al., editor, Communications and Multimedia Security. 9th IFIP TC-6 TC-11 International Conference, volume 3677 of Lecture Notes in Computer Science, pages 119–128, 2005.

[4] Zimmermann, P. "PGP User's Guide" 1990-1994 http://www.pa.msu.edu/reference/pgpdoc1.html

[5] Bevk, M., Kononenko, I., 2002. A statistical approach to texture description of medical images: a preliminary study. In: Proceedings of the IEEE Symposium on Computer-Based Medical Systems, June, pp. 239–244.

[6] http://steghide.sourceforge.net

[7] Xu, E., Liu, B., Xu, L., Wei, Z., Zhao, B., Su, J., Adaptive VoIP Steganography for Information Hiding within Network Audio Streams, In Proc. of 2011 International Conference on Network-Based Information Systems, Tirana, Albania, September 2011, pp. 612 - 617

[8] Whitten, A. and J.D. Tygar. Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. In 8th USENIX Security Symposium. pp. 1999.

[9] http://www.snort.org/

[10] http://libjpeg.sourceforge.net/

[11] http://code.google.com/p/zxing/

[12] http://apparentlogic.com/openflow/

[13] http://fukuchi.org/works/qrencode/index.html.en