Exploring Covert Communication on Google Hangouts

Subash Kumar Saladi[1], Andrew H. Sung[2] and Qingzhong Liu[1]
[1]Department of Computer Science, Sam Houston State University
Huntsville, TX 77341, USA,
Email: sks042@shsu.edu; liu@shsu.edu
[2]School of Computing, University of Southern Mississippi
Hattiesburg, MS 39406-0001, USA
Email: andrew.sung@usm.edu

Abstract - *Google+ hangouts is one of the most social media where people share information using the digital data like texts, images, video and audio files. While sharing the sensitive data in social sites, through illegal means these sensitive data may be intercepted. Hence there is a need of secret sharing for secure transmission of these data. Taking these parameters into consideration, we develop a Google hangouts application to implement image steganography for the users joining hangouts by hiding data in digital images, providing a real-life application for covert communication on Google Hangouts, one popular social media platform.*

Keywords: Steganography; Google Hangouts; covert communication; image; social media

1 Introduction

In the world of digital technologies, it is very much imaginable that the secret data carrier, was not necessarily an image or Web page source code, but may have been any other file type or organizational unit of data for example, a packet or a frame which occurs in computer networks. However, the process of embedding secret information into an innocent-looking carrier is not some recent invention but was being practiced since ages. This process is called steganography and its origins can be traced back to ancient times. Moreover, its importance has not decreased since its birth.

To protect the security of concerned message data, the information sharing method is developed. A secret message is logically constructed into several shares and they can be distributed to different participants to keep and are rearranged at the receiver to form a meaningful data. To have different participants, these carriers for the shares can be of any digital components like image, video or audio files.

Steganography's applications provide means for conducting clandestine communication. The purpose of establishing such information can fall into the category of legal or illicit activity. Frequently, the illegal aspect of steganography include the criminal communication, through information leakage from guarded systems, cyber weapon exchange. On the other side of the spectrum like legitimate uses, which include surveillance, F5 computer forensics.

The inverse of steganography—steganalysis, detects the covert communication which is started to surface fairly recently, Programs for the embedding of data considerably outnumber those dedicated to the detection and extraction of embedded content.

Images are the most popular cover objects for steganography for given proliferation of digital images in the Internet, and large amount of redundant bits present in the digital representation of an image. This project will focus on hiding information in images. Image steganography techniques can be divided into two groups: Image Domain and Transform Domain. Image domains embed messages in the intensity of the pixels, while for transform domains images are first transformed and then the message is embedded in the image which is more robust making the embedded message survive conversion between loss and lossless compression.

Google Hangouts [3] is a communication platform developed by Google which includes instant messaging, video chat, SMS and VoIP features. It replaces three messaging products that Google had implemented concurrently within its services, including Google Talk, Google+ Messenger, and Hangouts, a video chat system present within Google+, which previously used XMPP protocols that are now replaced by Google's proprietary protocols. Hangouts run natively into many browsers making it evolve into a standards- based cloud video conferencing implemented in client server model to support more than 10 clients to chat at one instance unlike in Skype, which is based on peer to peer network connections. Hangouts use WebRTC interfaces for browser integrations and use VIDYO to facilitate its video chats implemented in H.264/SVC as the primary codec for transmission.

Hangouts provide interface for users to jump into conversations between two or more users with a maximum number of people connecting in a hangout being 10. The service can be accessed online through the Gmail or Google+ websites or through mobile apps available for Android and iOS. Hangouts are distributed as a successor to their existing Google Talk apps by Google. Most third-party applications which had access to Google Talk do not have access to

Google+ Hangouts because these hangouts use proprietary protocol instead of the XMPP open standard protocol previously used in Google Talk.

On the other side, even after support for XMPP has been terminated by Google, the GVJackApp for magicJack and the GVMate Phone Adapter both of which are signaling independent will continue to work for users as normal using the Google Hangouts platform. As Google switched away from the XMPP protocol it used, brought in new challenges like android SMS support in Hangouts doesn't fully integrate with Google Voice for calls or texts. Hangouts work the same way in desktops, mobiles (android and iOS) and so the chat histories are saved online, allowing them to be synced among the multiple devices. Photos can be shared, use color emoji symbols in their messages during conversations. Photos are later uploaded into Google private album. In this paper, we explore the covert communication on Google Hangouts by hiding data in digital images.

2 Proposed method

We explore a new embeddable space with better quality of resulting stego-image and more data hiding capacity, we are using JPEG image's alpha channel plane as cover medium which embeds the secret data to be sent, by combining the advantages of JPEG's high compression rate with the flexibility of having alpha channels like in PNG.

1. JPEG image, which is used as a cover image having set the alpha channel value of each pixel to 255 initially making it a transparent colored one in the beginning of embedding process.

2. Message or the binary data string is transformed into shares using bundles per character based on Shamir's secret sharing method [6], which are then embedded into alpha-channel of JPEG cover image using the HTML5 canvas element.

3. Co-efficient parameters involved in the sharing of data strings are carriers of the hidden data.

4. Prime number used in this method will be taking care of resulting image visual quality and data hiding capacity for the stego-image. Hence the selection of appropriate prime number is very important for quality of transmission.

5. Mapping function is designed in such a way that the alpha-channel values create a uniform transparency for the resulting stego-image.

6. Original image pixels or DCT coefficients are untouched so as to maintain the original image appearance.

A JavaScript coding is used to implement the above technique and embed inside an XML file to be deployed as a Google + hangout application.

2.1 Algorithm for Shamir secret sharing

Input: a secret *d* in the form of an integer, the number *n* of participants, and a threshold *k* not larger than n.

Output: *n* shares in the form of integers for the *n* participants to keep.

Steps:

1. Choose a prime number *p* randomly.

2. Select k - 1 integer values $c_1, c_2, \ldots, c_{k-1}$ within the range of 0 through *p* - 1.

3. Select *n* distinct real values $x_1, x_2, \ldots, x_n$.

4. Use the following (k-1)-degree polynomial to generate *n* equations to compute n function values $f(x_i)$:

$$f(x_i) = (d + c_1 x_i + c_2 x_i^2 + \ldots + c_{k-1} x_i^{k-1}) \bmod p \qquad (1)$$

where i = 1, 2, …, n.

5. Deliver the 2-tuple $(x_i, f(x_i))$ as a share to the i-th participant where i = 1, 2, …, n.

2.2 Algorithm for Shamir secret recovery

Input: *m* shares in the form of $(x_j, f(x_j))$ collected from the *n* participants where 1<= j <= n, k <= m <= n, and *k* is the threshold mentioned in above algorithm.

Output: the secret *d* hidden in the shares.

Steps:

1. Collect any k of the *m* shares, say, $(x_{i1}, f(x_{i1}))$, $(x_{i2}, f(x_{i2}))$, …, $(x_{ik}, f(x_{ik}))$ and use them to set up the following equations:

$$f(x_{ij}) = (d + c_1 x_{ij} + c_2 x_{ij}^2 + \ldots + c_{k-1} x_{ij}^{k-1}) \bmod p \qquad (2)$$

where j = 1, 2, ..., k and 1 <= ij<= n.

$$d = \left(\sum_{j=1}^{k} (-1)^{k-1} \left[f(x_{i_j}) \prod_{m=1(m \neq j)}^{k} \frac{x_{i_m}}{(x_{i_j} - x_{i_m})} \right] \right)_{\bmod p} \qquad (3)$$

2.3 Data embedding algorithm

We utilized the methods in the references [1, 2], described below.

Input: a cover JPEG image *I* and a secret message *M* in the form of a binary data string.

Output: a stego-image *I′* in the JPEG format.

Steps:

1. (Initialization) Divide *M* into t-bit segments with t = 3 and transform each segment into a decimal number, resulting in a decimal-number sequence $M' = d_1\ d_2\ d_3 \ldots$ where $0 <= d_i <= 7$.

2. (Beginning of Looping) Take the first four elements from M' as m_1, m_2, m_3, and m_4, starting from the beginning of M'.

3. (Partial share creation) Set p, c_i, and x_i in Eqs. (1) of Shamir's Algorithm to be the following values:

(a) $p = 11$ (the smallest prime number larger than 7);

(b) $d = m_1$, $c_1 = m_2$, $c_2 = m_3$, and $c_3 = m_4$;

(c) $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, and $x_4 = 4$, resulting in the following equations:

$$q_1 = f(x_1) = (m_1 + m_2x_1 + m_3x_1^2 + m_4x_1^3) \bmod p,$$

$$q_2 = f(x_2) = (m_1 + m_2x_2 + m_3x_2^2 + m_4x_2^3) \bmod p,$$

$$q_3 = f(x_3) = (m_1 + m_2x_3 + m_3x_3^2 + m_4x_3^3) \bmod p,$$

$$q_4 = f(x_4) = (m_1 + m_2x_4 + m_3x_4^2 + m_4x_4^3) \bmod p.$$

4. (Mapping of partial share values) Add 245 to each of q_1 through q_4 to form q_1', q_2', q_3', and q_4', respectively.

5. (Data embedding) Embed q_1' through q_4' into the alpha-channel plane of I in the following way.

i) Take in a raster-scan order four unprocessed pixels of I and set their alpha-channel values to be q_1' through q_4', respectively.

ii) Remove m_i through m_{i+3} from M'.

6. (End of looping) If M' is not empty, then go to Step 2 to process the next four decimal numbers in M′; otherwise, take the final I as the desired stego-image I'.

The above algorithm is a (4, 4)-threshold secret sharing method. As the prime value *p* is 11, the possible values of q_1 through q_4 in Step 3 of the above algorithm are between 0 and 10 which are inserted in the alpha channels of I. The values of q1' through q4' form a small range of integers from 245 to 255 which are then embedded into the alpha channels of the cover image I. As the distribution of alpha channel pixels of the image are mostly the similar values, which makes a nearly uniform transparent image, making the intruder not to suspect the image transmission in the network.

To calculate the hiding capacity of algorithm, we know every four 3-bit segments of the secret data string are embedded into the alpha-channel values of four pixels of the cover image I to yield the stego-image I' which means that if the size of the cover image is S, then the data hiding capacity is

$$R = (4*t)*(S/4) = ts \text{ bits}.$$

Above relation shows that hiding capacity is directly proportional to the value t, larger values of t produce larger q1', q2', q3', q4' values which are beyond 255, thereby reducing the picture quality and creating a suspicion to the intruders.

2.4 Data extraction algorithm

Input: a stego-image I' created by embedding algorithm in the JPEG format.

Output: the binary data string M hidden in I'.

Steps:

1. (Initialization) Create an empty string M.

2. (Beginning of looping) Take in a raster-scan order four alpha-channel values q_1', q_2', q_3', and q_4', from I'.

3. Subtract 245 from each of q_1' through q_4' to obtain q_1 through q_4, respectively. Perform the secret recovery process described by Shamir's data extraction algorithm to extract the values m_1 through m_4 of the decimal format hidden in q_1 through q_4.

4. Transform the extracted values of m_1 through m_4 into binary bits and append in order each of them to the end of M.

5. (End of looping) If all shares embedded in I′ are processed, then take the final M as output; otherwise, go to Step 2.

3 Covert communication on Google Hangouts

3.1 Create a steganography Hangout App

The Hangouts API enables us to develop collaborative applications that run inside of a Google+ Hangout [3].

The following steps show how a hangouts host and run a pre-built steganography application.

1) Add the stegoApp.xml(XML file of any application) file on a server so that it is publicly available. The server should have no firewalls and require no login authentication to access this file.

2) Login to you gmail account and go to the Google Developers Console .

3) Create a new project by clicking Create Project - steganoApp:

4) In the Project name field, type in a name for our project -steganoApp.

5) In the Project ID field, optionally type in a project ID for your project or use the one that the console has created for us. This ID must be unique world-wide.

6) Click the Create button and wait for the project to be created.

7) Click on the new project name in the list to start editing the project.

8) In the left sidebar, select the APIs item below "APIs & auth". A list of Google web services appears.

9) Find the Google+ Hangouts API service and set its status to ON.

10) In the sidebar under "APIs & auth", select Consent screen.

11) Choose an Email Address and specify a Product Name.

To the right of the Google+ Hangouts API service name, click on the gear icon. In the Application URL field, enter the URL where you published your Hangout gadget XML file. Click Save.

3.2 Developing application in Google Hangouts

To develop a steganography application which can be used by participants in a googlehangout can share messages only through Google Server. So there should be API's which google should support the communication between the user application and the google server. Since our application is a browser deployed application, we used JavaScript client library to make API requests to interact with Google Services [4].

The general operations to make an API request using the JavaScript client library are described below.

1. The SteganoApp loads the JavaScript client library.

2. The SteganoApp references its API key, which authenticates the SteganoApp with Google services.

3. If the SteganoApp opens a session with a Google auth server. The auth server opens a dialog box which prompts the user to authorize the use of personal information.

4. The SteganoApp loads the API for the Google service.

5. The SteganoApp initializes a request object (also called a service object) that specifies the data to be returned by the API.

6. The SteganoApp executes the request and processes the data returned by the API.

As a client side application, whole application is developed using Javascript. We can use java script to add Google service with a part of code shown below.

```
<script src="//plus.google.com/hangouts/_/api/v1/hangout.js">
</script>
<script>
function showParticipants()
{
        var participants = gapi.hangout.getParticipants();
        var retVal = '<p>Participants: </p><ul>';
        for (var index in participants)
        {
          var participant = participants[index];
          if (!participant.person)
          {
            retVal += '<li>A participant not running this app</li>';
          }
          retVal += '<li>' + participant.person.displayName + '</li>';
        }
        retVal += '</ul>';
        var div = document.getElementById('participantsDiv');
        div.innerHTML = retVal;
}
</script>
```

Here "hangout.js" is the file that imports all the required Google Services API methods in our application. In the above code, we can see gapi.hangout.getParticipants() is used to show different participants who joined the call.

Accessing Google API's is through the Google APIs Console. The steps are listed below:

- Login and visit the Google APIs Console.
- Select Services from the menu for the steganoApp project. The list of accessible Google services appears, enable Google Hangouts' API ON.
- The API access pane appears. For authorized access as we need to run the application between different participants, we should continue as below.
- Click Create an OAuth 2.0 client ID.
- The Create Client ID dialog appears.
- Click the Web application radio button to create a client ID. The Authorized API Access section now displays our steganoApp's OAuth 2.0 credentials [5].

This client ID created is used in the application as follows:

```
gapi.hangout.onApiReady.add(
function(eventObj) {
if (eventObj.isApiReady) {
document.getElementById('showParticipants')
.style.visibility = 'visible';
}
});

var clientId = '989443731264-
dvm1t58ousjfr080sjvksj5jah3jcs4q.apps.googleusercontent.com';
var apiKey = 'AIzaSyDxVGV0-
KTxjNFQ3cASXPeVmC7gP6dyI98';
var scopes =
'https://www.googleapis.com/auth/hangout.participants';
function handleClientLoad() {
gapi.client.setApiKey(apiKey);
window.setTimeout(checkAuth,1);
}

function checkAuth() {
gapi.auth.authorize({client_id: clientId, scope: scopes, immediate:
true}, handleAuthResult);
}

function handleAuthResult(authResult) {
```

}
}

3.3 Sending and receiving

When participants run steganoApp in a Hangout, they are each running the app in their own separate in-stance of the Hangout client. Hangouts have two major data channels for sharing application-specific data be-tween these instances. They are shared state and sendMessage.

sendMessage( ): When used sendMessage( ), there were problems at the receiver end and then started using Shared State among the participants because sendMessage( ) sends a message to the other application participants. Messages are not retained or stored, and should have lower latency than objects stored via sub-mitDelta method in the API. These messages might be lost, so this method should only be used to send things that can be dropped.

Shared State( ): There is only one shared state object per Hangout. The shared state object contains data that is kept up-to-date with every instance of the Hangout client that is running our steganoApp. The object is a regular JavaScript object with paired key/value strings. This object is our stegano-message sliced into 7000 characters per message with a sequence number. This key/value pair of sequence number and stegano-message string will be distributed to every instance of the Hangout client that is running our application, i.e., all participants of hangout call.

3.4 Encoding and decoding

To decrease the number of HTTP requests to server by decreasing the resources and increasing the overall performance of the website, this application uses base64 data stream of image within the code.

Base64 converts binary data into ASCII string format by translating it into a radix-64 representation .The base64 technique generates ASCII representation for binary data of image file and browser can parse these ASCII values and render an image on the page which makes the application lighter than the cost of an additional external resource link to an image.

4 Experiments

The results of applying the data embedding meth-od used to embed a long sequence of message data into the two images are shown in Figure 1. The original image and the steganogram are visually almost identical.

(a) Original image (b) steganogram

Figure 1. An example of original image and steganogram

Table 1. Data hiding capacity and PSNR values

t value	DHC (bits) [R= t*size]	PSNR
1	62217	78.80
2	124434	70.15
3	186651	67.48

5 Conclusions

A covert communication on Google Hangouts has been established via image steganography carrying the secret information with the use of Shamir's secret sharing method. The alpha-channel plane of a cover JPEG image is created and utilized to embed the partial shares, resulting in a stego-image with undesirable white noise. The white noise is then eliminated by choosing a small prime number, dividing the input data string into 3-bit segments, and mapping computed share values into a range of alpha-channel values near their maximum value of 255. Deploying this in Google + hangout with two participants were tested, which can be extended to a maximum of 10 people. It is a novel way of sending sensitive information using steganography methods in social media.

This project gives a large scope to develop into a large scale commercial software product, thereby enhancing the UI design for more interactive interface for user to do dynamic selection of pictures and also test for other safe possible areas in an image file for data hiding.

Steganography can also be implemented in audio, video sharing files within the hangout but might post few challenges on real time data but on research we might find Google API's that support real time data at much ease. A further study may be focused on more secure steganographic systems [7, 8] and the covert communication may be expanded to other social media platforms, such as Reddits, Facebook, etc. Additionally, the advanced steganalysis methods [9, 10, 11] will be conducted to examine the undetectable ability.

6 Acknowledgements

Part support for this study under the NSF award No. 1318688 and the support from the SHSU Office of Research and Sponsored Programs are highly appreciated.

7 References

1. C. W. Lee and W. H. Tsai, A secret-sharing-based method for authentication of grayscale document images via the use of the PNG image with a data repair capability, IEEE Trans. on Image Process. 21 (1) (2012) 207-218.
2. C-W. Lee and W-H Tsai, A data hiding method based on information sharing via PNG images for applications of color image authentication and metadata embedding, Signal Processing, 93(7) (2013) 2010-2025.
3. https://developers.google.com/+/hangouts/getting-started
4. https://developers.google.com/api-client-library/javascript/start/start-js
5. https://developers.google.com/api-client-library/javascript/features/authentication
6. A. Shamir, How to share a secret, *Communication of the ACM*, vol. 22, pp. 612-613, 1979.
7. Q. Liu, A. H. Sung, Z. Chen and X. Huang. A JPEG-based statistically invisible steganography. *Proc. 3rd International Conference on Internet Multimedia Computing and Service*, pages 78-81, 2011.
8. J. Fridrich and J. Kodovsky. Multivariate Gaussian model for designing additive distortion for steganography. *Proceedings of 2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (ICASSP), pages 2949-2953.
9. M. Goljan and J. Fridrich. CFA-aware features for steganalysis of color images, *Proc. SPIE 9409*, Media Watermarking, Security, and Forensics 2015, 94090V (March 4, 2015); doi:10.1117/12.207839
10. Q. Liu, A. H., Sung and M. Qiao. Neighboring joint density-based JPEG steganalysis. *ACM Trans. Intelligent Systems and Technology*, 2(2):16. 2011.
11. Q. Liu and Z. Chen, Improved approaches with calibrated neighboring joint density to steganalysis and seam-carved forgery detection in JPEG images. *ACM Trans. Intelligent Systems and Technology*, 5(4):63, 2014.