Android-Stego: A Novel Service Provider Imperceptible MMS Steganography Technique Robust to Message Loss

Avinash Srinivasan[*]
Computer and Information Sciences
Temple University
Philadelphia, PA
U.S.A.
avinash@temple.edu

Jie Wu[†]
Computer and Information Sciences
Temple University
Philadelphia, PA
U.S.A.
jiewu@temple.edu

Justin Shi
Computer and Information Sciences
Temple University
Philadelphia, PA
U.S.A.
shi@temple.edu

ABSTRACT
Information hiding techniques, specifically steganography, have been extensively researched for over two decades. However, steganography on smartphones over cellular carrier networks is yet to be fully explored. Today, smartphones which are at the epitome of ubiquitous and pervasive computing, make steganography an easily accessible covert communication channel. In this paper, we propose Android-Stego – a framework for steganography employing smartphones. Android-Stego has been evaluated and confirmed to achieve covert communication over real world cellular service providers' communication networks such as Verizon and Sprint. A key contribution of our research presented in this paper is the benchmark results we have provided by analyzing real world cellular carriers' network restrictions on MMS message size. We have also analyzed the actions the carriers take – such as compression and/or format conversion – on MMS messages that fall outside the established MMS communication norm, which varies for each service provider. Finally, we have used these benchmark results in implementing Android-Stego such that it is sensitive to carrier restrictions and robust to message loss.

Categories and Subject Descriptors
D.2.11 [**Software Architectures**]: Information hiding; D.4.6 [**Security and Protection**]; E.3 [**General**]: Security and protection

General Terms
Algorithms, Design, Security, Steganography.

[*]Also affiliated with the *Centre for Security and Forensics Education & Research (C-SaFER)*, CIS, Temple University.
[†]Is the director of the Center for Networked Computing. CIS, Temple University

Keywords
Android, Information hiding, Multimedia Messaging Service (MMS), Security, Steganography.

1. INTRODUCTION
Steganography, the art and science of hiding communication [12], and other forms of information hiding have been extensively researched over the last two decades. Although steganography has been known for centuries, only recently has it proliferated new grounds with the all-pervasive and ubiquitous telecommunications services as the most lucrative.

The methods for embedding secret data have constantly evolved, and are more sophisticated today than ever before. Nonetheless, the basic principles of hiding information using steganography techniques remain unchanged with a limited number of possibilities and corresponding algorithms. Over the years, one very popular type of steganography has been the digital image steganography. Digital image steganography is a method of concealing data within a digital image.

Within digital image steganography, numerous algorithms have been proposed to embed the secret data using various image formats as cover files. Through all the advances that steganography has witnessed, the *Least Significant Bit* (LSB) [19] method of embedding secret messages remains to be the most popular and the simplest of all to implement. In this technique, the LSB of some random bytes or perhaps all of the bytes of the cover (aka carrier) image file are changed appropriately to encode the secret message.

Steganographic data are often hidden through the use of mathematical techniques that add information content to digital objects such as images, video files, and audio files, including various other digital objects such as executable code [3]. However, when sophisticated embedding techniques are used, the degradation in quality or increase in payload is perceptible. Additionally, a steganography message can be encrypted prior to hiding, making it substantially harder to detect, extract, and finally recover the message. In particular, encryption makes it harder to use statistical analysis, particularly entropy-based techniques, since encrypted data has very high entropy – between 7.5-8.0 bits-per-byte.

MOBIMEDIA 2015, May 25-27, Chengdu, People's Republic of China
Copyright © 2015 ICST
DOI 10.4108/icst.mobimedia.2015.259071

Furthermore, there is no universally applicable methodology for detecting steganographic embeddings, and the few general principles that exist tend to be rather ad-hoc. Today, steganography has slowly gained momentum and become very popular within the underground hackers' community. This should cause serious concerns, as steganography is now a new weapon in the already-sophisticated arsenal of the Black-Hat hackers. It has, over the years, become the adversary's preferred mode for delivering malware and exploit payloads. Underground hacking communities are quickly embracing steganography techniques for data exfiltration.

1.1 Assumptions and Threat Model

In our discussions of the proposed Android-Stego framework, we assume that Alice (sender) and Bob (receiver) are the end users, and they use a PKI-based digital certificate for mutual authentication. Also, with the digital certificate, Alice and Bob are equipped to negotiate a shared session key spontaneously on an unsecured communication channel. In our scenario, an unsecured channel is one that the service provider can monitor for the presence of cover channels.

1.2 Android-Stego – Our Contributions

We acknowledge that there are numerous existing works that are similar to our proposed Android-Stego – in their approach as well as capabilities [7]. However, we do believe our work has novelty and some very unique and important contributions. Android-Stego, in its implementation and application, it markedly differs from being a mere LSB technique. It independently segments the secret message file into chunks, encodes them, and finally embeds them in a series of PNG images - all of which is done with minimum user interaction. Most importantly, the prototype implementation relies on native image and MMS functionality common to most Android devices, and does not necessarily depend on direct Internet connectivity or carrier limitations.

Android-Stego is a segmented distributed multipart MMS Steganography with detailed cryptographic processes to provide core security requirements - *confidentiality*, *integrity*, and *source authentication*, which has been discussed in detail in section 4. Our paper presents the first and perhaps the only work to study and analyze the restriction on MMS message sizes and its impact on MMS-based Steganography over four real world cellular service providers – Verizon, T-Mobile, Sprint, and ATT. Furthermore, our paper also presents results summarizing the actions that each of the aforementioned carrier perform on MMS messages whose size exceeds the carrier-permissible maximum size.

We have custom-implemented a real world working prototype due to the lack of libraries for the Android platform. This was also one of the reasons we deviated from the popular F5 algorithm. In section 5, we have presented the benchmark results for carrier-imposed size restriction and the corresponding actions on oversized MMS messages. Most importantly, the proposed framework and its prototype implementation relies on native image and MMS functionality common to most Android devices, and does not necessarily depend on direct Internet connectivity or carrier limitations.

1.3 Road Map

The rest of the paper is organized as follows. In section 2, we present some of the more relevant mobile device steganography related works followed by a quick overview of the Android mobile platform in section 3. Later, in section 4, we discuss the Android-Stego architecture highlighting details of the covert channel communication accomplished with MMS steganography. This section also provides details on the embedding process, sender side operations, as well as the receiver side operations, with relevant cryptographic operations. Section 5 presents discussions on security properties of the proposed framework, carrier restrictions on MMS size and the actions carriers take when MMS size exceeds the imposed limit, and some of the unique features of the proposed framework. Finally, we conclude the paper with directions for future research in section 6.

2. RELATED WORK

"Operation Twins," culminated in 2002 with the capture of criminals associated with the "Shadowz Brotherhood," a pedophile organization responsible for the distribution of child pornography with the aid of steganography. Steganographic methods have also proven to be useful tools for data exfiltration, which was evidenced in the 2008 incident in which someone at the U.S. Department of Justice smuggled sensitive financial data out of the agency by embedding it in several image files. In 2010, the FBI arrested members of a Russian spy ring called "illegals". These members were allegedly sending classified US government documents through clandestine messages to Moscow through digital image steganography over publicly-available websites [9, 15].

A new worm called Duqu was discovered sometime during September 2011 with glaring similarities to Stuxnet [17]. Both Stuxnet and Duqu had similar general malware structures and characteristics. However, upon closer examination, Duqu revealed a marked difference in comparison to Stuxnet - Duqu was written to gather information on the infected system and transferred the gathered information back to the command and control center (C&C) using a backdoor. The information transmitted via the backdoor to the C&C was hidden in seemingly innocent pictures, without raising any suspicion. A similar functioning mechanism was coincidentally discovered in a new variant of the Alureon malware [1] and around the same time as Duqu.

Rosziati Ibrahim et al. [10] proposed a steganography algorithm for hiding secret messages inside a bitmap (BMP) image. Although the paper does not fully disclose the algorithm utilized, it is suspected that an approach similar to LSB is used for hiding the message in a carrier BMP image. In this algorithm, a secret message is first encrypted with a key. The encrypted secret message is then compressed into a zip file. Subsequently, the encrypted and compressed secret message is then converted into a binary file. The encryption key is also zipped and converted to binary. To hide the message and key into the cover BMP file, binary codes from the series are encoded two bits at a time into an image pixel, until all the binary codes have been exhausted. The process is then reversed at the receiver's end to retrieve the secret message from the BMP carrier image. Rather than using a single bit per byte to hide a message, as in a traditional LSB substitution approach, their algorithm uses two bits per byte to maximize the amount of secret data that can be embedded in the cover image and transmitted. Based on this algorithm, Rosziati Ibrahim et al. [11] further developed a Google Android-based application called MoBiSiS (Mobile Steganography Imaging System) to send steganographic images through MMS or email. They extend an algorithm to support additional image formats such as JPEG, GIF, and PNG, in addition to merely BMP in their original algorithm.

Finally, in [6], authors have considered speech signal as the cover media to hide secret data. This work steeped away from the conventional image-based steganography venturing into audio files as a potential carrier medium. Similar message condensing approaches have been proposed that have adaptively segmented the cover image based on the key, have selectively chosen pixels based on certain characteristics, and that have used left nibble changes (4 bits from the left end of a byte) in a byte [8]. These changes have minimally and imperceptibly changed the cover images, and have made them less susceptible to steganalysis.

Mazurczyk et al. proposed SkyDe that utilizes encrypted Skype voice packets as a hidden data carrier. Their proposed SkyDe achieves a steganographic bandwidth of about 2 Kbit/s. Mazurczyk et al. also proposed StegTorrent [13], which is a network steganographic method for BitTorrent - the popular P2P file transfer service. StegTorrent takes advantage of the many-to-one transmissions in BitTorrent with the μTP protocol header providing a mechanism for numbering packets and retrieving their original sequence. This facilitates clandestine data transfer at a rate of about 270 b/s. Motivated readers are encouraged to read the paper by Zielnska et. al. [21], which is literature review of the state-of-the-art in steganography. There have been other works that also address steganography and other information hiding techniques in smartphones. In [14], authors have provided a summary of such efforts, along with mitigation techniques. Additionally, in [16], authors have presented a very detailed survey of information hiding techniques.

3. BACKGROUND

In this section, we will provide some basic discussions necessary for the reader, who is not a smartphone enthusiast, to appreciate the work.

3.1 Cryptography vs Steganography

While cryptography and steganography have a lot of similarities as well as differences, a key difference lies in their fundamental objective. Cryptography operates with the fundamental objective of securing the communication from eavesdroppers; it does not hide the fact that a secret message exists. This is because, the encrypted information can be seen by anyone. Hence, cryptographic communications fall well within the bounds of overt communication channels. Steganography, on the other hand, operates with the fundamental objective of concealing the very fact that a secret message is hidden. It hides information by concealing it within another innocuous-looking medium, and no third-person will ever know that a secret message exists. Therefore, steganography communications constitute a covert communication channel. In particular, due to the vastness of cyberspace, steganography provides an unprecedented capability for true adversaries to transmit information that can easily evade detection mechanisms.

3.2 Smartphones & Steganography

The information technology industry has undergone unprecedented advances in the last few years, particularly in the realm of mobile devices and their Operating Systems (OSs). Contemporary mobile devices, in particular smartphones, have become extremely integrative in nature, combining all computing features and functionalities that a user needs, into a single portable device. Today, smartphones have risen to the epitome of ubiquitous and pervasive computing, and on such powerful devices that are true miniatures of personal computers, steganography is an easily-accessible covert communication channel. With the growing size of mobile networks, even in developing countries, and the growing ubiquity of camera-equipped smartphones, it may be important to revisit image steganography as a powerful and potent covert communication channel between mobile devices.

In this paper, we propose and discuss Android-Stego - a novel, robust steganography framework for Android-based smartphones. Android-Stego is an extension of image-based steganography into an MMS-based steganography for Android smartphones. It is aware of cellular carriers' restrictions on MMS message sizes and can traverse the network from sender to receiver surviving compressions and format conversions. We demonstrate our framework through the implementation of a real world working prototype. The prototype implementation relies on native image and MMS functionality common to most Android devices, and does not necessarily depend on direct Internet connectivity or carrier limitations.

Our prototype is a segmented and distributed multipart implementation that support MMS-based steganography on both sender and receiver devices with the following capabilities:

1. Splitting and encoding a secret message (can be multi-part depending on message size and service provider restrictions) on the sender side.
2. Encoded secret message successfully traverses the cellular networks, transparent to service provider restrictions.
3. Decoding the received secret message, and reassembling if it is a multi-part message, on the receiver side.

The two key features expected from steganography carriers, as noted in [21] and listed below are both satisfied by the cover file in our proposed Android Stego:

1. The cover file should be popular such that its usage should not in itself be considered an anomaly. Our proposed Android-Stego technique meets this requirement since MMS messages, which serve as the cover file, are very popular, and do not account for anomalies.
2. Modifications to the cover file resulting from insertion of the secret message should be imperceptible to a third-party, who is not aware of the covert communication channel. So, insertion of a secret message

into a single instance of the carrier should be upper bound by the imperceptibility threshold to modifications. Consequently, additional instances of the cover file should be used to accommodate the leftover part of the secret message. Our proposed Android-Stego technique meets this requirement by incorporating multipart, segmented, and distributed capabilities into the LSB encoding algorithm.

3.3 Android Mobile Platform Overview

Android is Google's Linux-based open source mobile platform and has evolved into a dominant and popular smartphone OS. It has a rich application programming interface for software developers, and Android applications are written in Java using the Software Development Kit (SDK). While robust steganography libraries do exist, few have been ported to support the Android OS. All Android applications are comprised of the following: *Component Activities*, *Services*, *Content Providers*, and *Broadcast Receivers.*

4. ANDROID-STEGO FRAMEWORK

In this section, with the help of schematic diagram, we describe the operations of various subsystems within our Android-Stego application framework. Recall that Android-Stego operates on both sides of the communication channel enabling the end users - the sender and the receiver in exchanging MMS-based steganography messages. For our discussion here, we assume that *Alice (sender)* and *Bob (receiver)* have established a covert channel, using the supporting PKI infrastructure, which they will use during MMS message exchanges.

In our discussions, we make generic references to PKI infrastructure, digital certificates, and asymmetric algorithms. However, the *Elliptic Cuver Cryptography* (ECC) encryption and digital certificate scheme is the most ideal for mobile devices [18], since it is computationally cheap and yet secure.

ECC is a public key encryption technique that is based on elliptic curve theory and can be used to create faster, smaller, and more efficient cryptographic keys. A 160-bit ECC encryption key provides the same security as a 1024-bit RSA encryption key [5]. Additionally, it can be up to 15 times faster, depending on the specific platform on which it is implemented.

Secnario: In our discussion, we assume that Alice and Bob are friends communicating over an insecure channel. In reality, however, they could be total strangers and equally easily establish mutual authentication using the PKI infrastructure. We will not discuss further details about the PKI infrastructure and the process of key negotiation due to page length restrictions.

4.1 Process Overview

In this section, we discuss the details of the process of MMS-based secret message exchange between Alice and Bob under the proposed Android-Stego framework. With regards to key management, Alice and Bob make use of a secure shared key that they can either be negotiated offline, or can make use of asymmetric algorithms to negotiate the key online and in realtime. Below are the steps involved in the encrypted

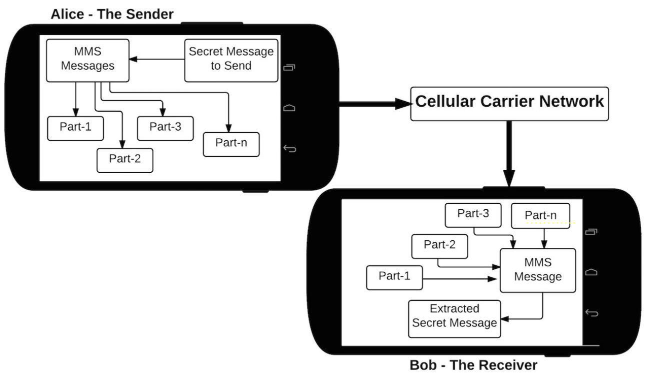

Figure 1: Generating an MMS Stego message using our Android-Stego Framework

Image Type	Size	Pixel Value (x, y)
Cover Image C	MxM	(0, M-1)
Steganized Image S	NxN	(0, N-1)

Table 1: Summary of size and pixel range for the cover image and the steganographic image.

and authenticated steganographic message exchange, assuming a shared key has been established previously.

Stage-1: Alice selects a secret file Msec that she wants to send to Bob using the user interface on the prototype application. Subsequently, she also selects the cover image files she wants to use, through the user interface. Alice then invokes the encoding process that will embed a secret file into the cover image bitmap files. The secret file is serialized by the Android-Stego prototype application and is broken into segments that are combined with the bitmap carrier images.

Stage-2: The prototype application uses the *PngStegoImage* class to combine the bitmap files and the serialized secret file segments, and then converts them into steganographic bitmap PNG images as shown in Figure 1. For each pixel holding some part of the secret data, the integers representing red, green, and blue are evaluated and modified, when necessary, such that even and odd values represent the zeros and ones of the binary encoded data.

The secret file images are then shepherded to the MMS (or Email service) of the Android application, which then uses the cellular carrier services to send the images to Bob, the recipient.

On the security and integrity of the secret message itself, the framework uses a hybrid cryptographic system, a combination of asymmetric and symmetric encryption algorithms as delineated below.

Operations on Alice's (sender's) side:

Step-1: Alice is in possession of M_{sec}, the secret message that

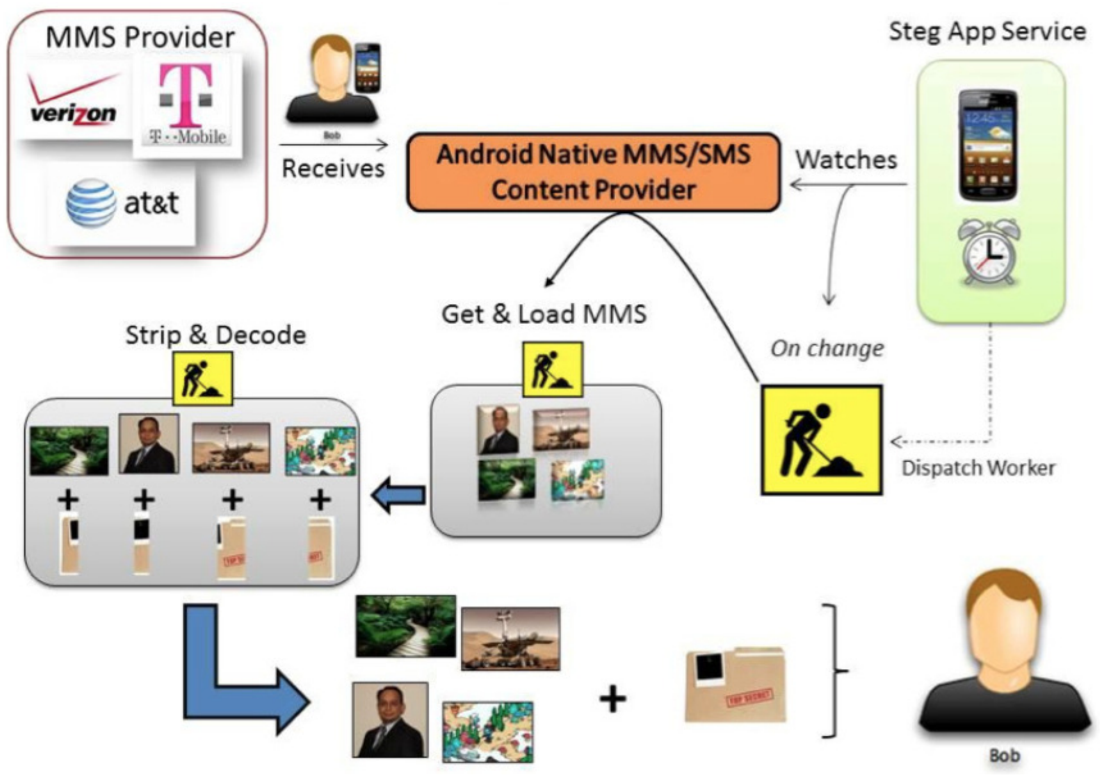

Figure 2: The process of embedding the secret message in a Steganographic MMS message.

she wishes to share with Bob.

Step-2: Alice generates a random sessions key: K_{rand}^{Alice}. The session key is unique for every secret message.

Step-3: Alice computes a hash of the secret message: $h([M_{sec}])$.

Step-4: If the size of M_{sec} is $\geq$ a predetermined permissible message size, Alice splits the message into multiple parts. This permissible size varies from carrier to carrier, a summary of which is presented in Table 2 for four major carriers in the USA.

Step-5: Alice encrypts the secret message (or a chunk of the message split into multiple parts) with the session key: $[M_{sec}]_{K_{rand}^{Alice}}$
Note: Each chunk of a multi-part secret message is encrypted with the same session key K_{rand}^{Alice}.

Step-6: Alice computes a hash of the encrypted message from Step-5: $h\left[[M_{sec}]_{K_{rand}^{Alice}}\right]$

Step-7: Alice encrypts the session key with her private key: $[K_{rand}^{Alice}]_{K_{prv}^{Alice}}$

Step-8: Alice appends the output from Steps-5, 6, and 7:

$$\left[[M_{sec}]_{K_{rand}^{Alice}}||[K_{rand}^{Alice}]_{K_{prv}^{Alice}}||h\left[[M_{sec}]_{K_{rand}^{Alice}}\right]\right]$$

This message provides the following security requirements: *confidentiality*, *message integrity*, and *source authentication*.

Step-9: Alice encrypts the output from Step-8 with Bob's public key:

$$\left[[M_{sec}]_{K_{rand}^{Alice}}||[K_{rand}^{Alice}]_{K_{prv}^{Alice}}||h(M_{sec}]_{K_{rand}^{Alice}})\right]_{K_{pub}^{Bob}}$$

For a multipart message, Alice includes K_{rand}^{Alice} only if she had to change the session key for any security reasons. Otherwise, only the very first message transmitted will include $[K_{rand}^{Alice}]$

Step-10: Alice repeats Steps-5 to 9, until all pieces of the secret message are transmitted.

Step-11: Finally, Alice transmits the hash of the original unencrypted message that she computed in Step-3.

Stage-3: On the receiver side, the Android platform's native SMS/MMS content provider handles the receipt of messages, as shown in Figure 3. On Bob's device, the prototype application service watches for changes upon a receipt of a message (MMS/Email). On change, the steganography application service dispatches a worker to execute the steps required to load and decode the images.

Operations on Bob's (receiver's) side:

Step-1: Bob decrypts the received message with his private key and extracts the following:
$[M_{sec}]_{K_{rand}^{Alice}}$, $[K_{rand}^{Alice}]_{K_{prv}^{Alice}}$, and $h([M_{sec}]_{K_{rand}^{Alice}})$

Step-2: Bob verifies the integrity of the received secret message by computing the hash of the session key encrypted

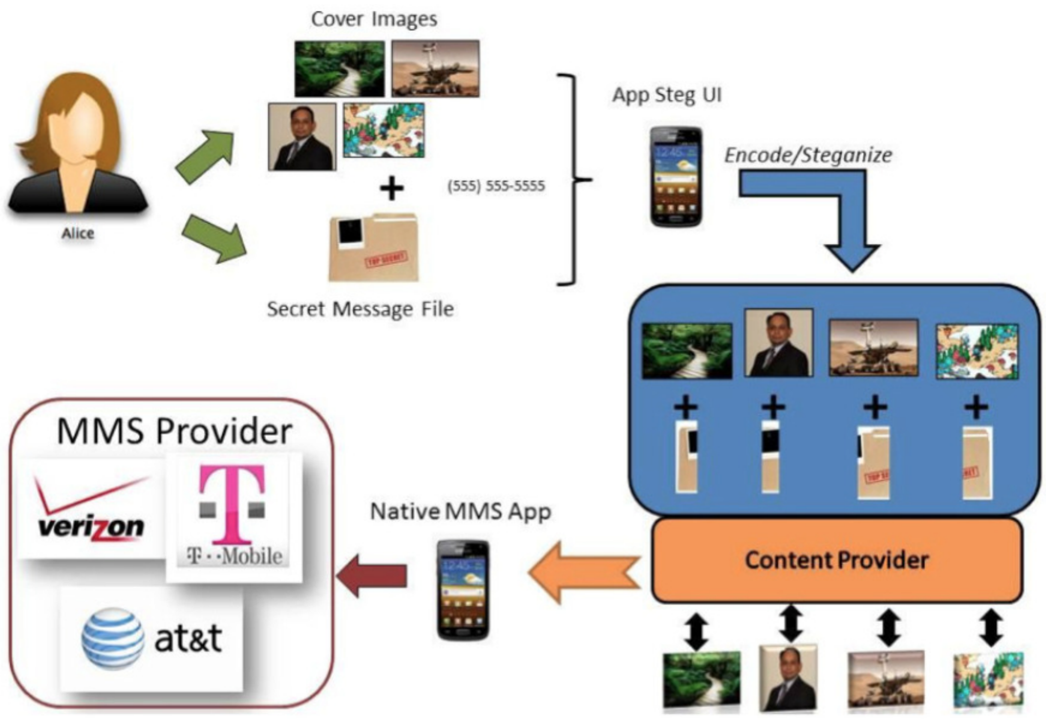

Figure 3: The process of extracting the secret message from a Steganographic MMS message.

message $[M_{sec}]_{K_{rand}^{Alice}}$ and verifies integrity by comparing it to the hash received $h([M_{sec}]_{K_{rand}^{Alice}})$. If the two match, then he continues to step-3; otherwise, he stops processing.

Step-3: Bob extracts the sessions key K_{rand}^{Alice} from $[K_{rand}^{Alice}]_{K_{prv}^{Alice}}$ using Alice's public key K_{pub}^{Alice}

Step-4: Bob decrypts the secret message using the session key extracted in step-4 above.

Step-5: Bob repeats the above steps (steps-1 to 4) until all pieces of the secret message are received, verified and decrypted.

Step-6: Bob, in the case of a multi-part secret message, reconstructs the original secret message from the pieces. He then computes the hash of this reconstructed original message and compares it to the hash of the original unencrypted message received from Alice in Step-11.

The prototype implementation and the framework are, in general, designed to be modular, so that additional features can be easily incorporated into the framework. This even facilitates using encryption schemes of the user's choice.

4.2 Implementation Challenges

Implementing an MMS-based steganography system on the Android platform presents certain very unique challenges. Any such implementation is dependent on systems both within the Android environment, as well as carrier-specific systems external to and outside the control of the Android environment. Several of these systems' requirements and specifications are neither very consistent nor well documented. So, it is imperative that we test these external systems, in order to understand and predict their behavior.

Furthermore, while some free and open source steganography libraries do exist for the Java platform, they rely on existing graphics and GUI libraries - such as Oracle's Abstract Window Toolkit - that are not implemented for the Android platform [2, 4, 20]. Therefore, we found it necessary to specify, design, and implement a custom application under our proposed framework using the Android SDK, which provides API libraries and developer tools necessary to build, test, and debug applications for Android devices.

The proposed Android-Stego framework encodes binary data into a bitmap image using the very popular LSB technique by making slight modifications to the pixel data if and only if, and where necessary. Specifically, the encoded binary data will be represented by the parity of the red, green, and blue values in a group of pixels. This will allow us to encode three bits of secret data per pixel, or equivalently one byte of secret message per 2.66 pixels while making only slight changes to those which are imperceptible to the original image data.

Because our algorithm relies on bitmap data, which are lossy-compressed image formats, such as GIF or JPEG are not very suitable as cover images. However, compression is beneficial to data throughput. Therefore, we chose to use the Portable Network Graphic (PNG) image format. PNG image format employs a lossless-compression technique that preserves pixel fidelity.

Part of any good digital steganographic method is the ability to validate and measure the quality of the algorithm in obfuscating the message in the carrier image. Changes in the carrier images must not only be imperceptible to the casual observer, but the image must also be able to withstand statistical and steganalytical scrutiny. A standard measurement used in steganography, to test the quality of the steganographic images, is called Peak Signal-to-Noise Ratio [11]. The higher the value of PSNR, the higher the quality of the steganographic image will be.

$$PSNR = 10log_{10} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [C(x,y) - S(x,y)]^2$$

As noted in [11], if the cover image C has a size (MM) and the steganographic image S has a size (NN), then each C and S will have pixel value (x, y) ranging from [0 to $(M-1)$] and [0 to $(N-1)$] respectively. The MAX value is the maximum number of pixels for the image. This information has been summarized in Table 1.

5. DISCUSSIONS

5.1 Security Properties of Android-Stego

In this section, we discuss the security requirements satisfied by our proposed Android-Stego framework.

- **Confidentiality of the secret message:** Since the secret message M_{sec} is encrypted with a session key K_{rand}^{Alice}, which is generated by Alice, no one will be able to discern the message without the key. Further more, the session key is encrypted with Bob's public key when it is transmitted. Consequently, an attacker will not be able to decipher the message without Bob's private key.
- **Integrity of the secret message:** The secret message is first encrypted with the session key and then is hashed. The resultant hash is included along with the encrypted message. The receiver computes the hash of the received encrypted message (or that particular chunk of a larger message) and compares it to the hash included with the message. If the two hash values match, message integrity is verified and the receiver proceeds to decrypt the message. Else, the receiver discards the message (or that chunk).
- **Sender authentication:** A sender is authenticated in our proposed framework by having the sender encrypt the session key with the sender's private key. On the receiver side, the session key is extracted using the sender's public key. Hence, the session key could not have been generated by anyone else.

5.2 Carrier Restrictions on MMS Size

Network operators play an active role in MMS communication, when compared to standard voice, data or SMS communication. Carriers store uploaded MMS data on their own servers, and then forward the data to capable handsets, or grant access to subscribers to access the uploaded data on their server. This implementation, while primitive, is necessary to ensure backward compatibility so that subscribers using older mobile devices can still view the MMS content, or through an alternative medium, like a carrier's website. Because of this, it is possible that content sent via MMS may be rejected outright by the carrier if they sense any malicious content.

A rejection may occur, for example, if a MIME type is unknown or if a file is deemed too large. These restrictions may not be documented or consistent between carriers. Our implementation relies on PNG-compressed bitmaps, so we tested the behavior of this kind of data on four major U.S. carriers. We were interested in data size limits and data integrity (compression or conversion, for example) measures employed by these carriers that would prevent MMS-based steganographic communication.

To test for any such limitations, we attempted transmission of incrementally larger steganographic PNGs via MMS, and then verified their integrity on the receiving end by comparing the SHA-256 hash value of the message with that on the sender's side. With this information in hand, we worked to develop a working prototype of the framework that would work within the confines of MMS and any carrier-imposed limitations. In the prototype, we have established a binary specification, and an application that can both encode and decode this data from a bitmap image.

6. CONCLUSIONS

Considering the integrative nature of contemporary smartphones, steganography is most definitely an easily accessible alternative for covert communication. In this paper, we have proposed Android-Stego, a new framework that implements of a multipart MMS-based steganography on Android smartphones. For implementing a real world working prototype of Android-Stego, we have written a custom implementation of the LSB algorithm, which can hide arbitrary binary data. Our implementation is a segmented and distributed implementation built on the LSB encoding technique.

We have also analyzed MMS handling behavior by various cellular service providers to ensure that our implementation would work on most domestic networks, and it is robust to message loss resulting from cellular operator manipulations of steganographic MMS messages. We have presented the restrictions placed on user MMS message size by four major carriers and the actions - compression and/or format conversion - the carriers perform on the MMS messages once they exceed the imposed limit.

Our prototype implementation is modular, and built with existing Android APIs. Therefore, new features can be easily introduced, making it more capable in hiding, as well as robust to detection. One specific item of inetrest on our agenda of future research is building fault tolerance to message exchanges. If a particular message chunk, in case of a multi-part message, fails the integrity check, then that chunk can be discarded without the need for retransmission, and the original message can still be recovered using techniques similar to (k, n) thershold secret sharing.

7. ACKNOWLEDGMENTS

The authors would like to thank Chris Brahms, Stephan Reimers, and Thomas Valadez for their contributions towards the implementation and validation of Android-Stego, as well as the draft of this paper.

Receiving Carrier	Receive Status	File Integrity
Verizon	True	Partial. All images of size $\geq$ 1MB were compressed (and converted to JPEG) by the native MMS application. Smaller images remained intact.
T-Mobile	True	No. All images of size $\geq$ 1MB were compressed (and converted to JPEG) by the native MMS application. Files of size 500KB and 750KB were compressed (as PNGs) by the carrier.
Sprint	True	Partial. All images of size $\geq$ 1MB were compressed (and converted to JPEG) by the native MMS application. Smaller images remained intact.
AT& T	True	Partial. All images of size $\geq$ 1MB were compressed (and converted to JPEG) by the native MMS application. Smaller images remained intact.

Table 2: Summary of carrier restriction on in-coming MMS message size for the big-four cellular service providers in North America.

8. REFERENCES

[1] Alureon trojan uses steganography to receive commands.

[2] F5- steganography in java.

[3] Federal plan for cyber security and information assurance research and development.

[4] Openstego: The free steganography solution.

[5] J. W. Bos, M. E. Kaihara, T. Kleinjung, A. K. Lenstra, and P. L. Montgomery. On the security of 1024-bit rsa and 160-bit elliptic curve cryptography.

[6] S. K. Dastoor and V. Patel. A novel android based mobile application as a virtue of covert communication for concealing information in the speech signal. In *Emerging Technology Trends in Electronics, Communication and Networking (ET2ECN), 2012 1st International Conference on*, pages 1–6. IEEE, 2012.

[7] D. Dhobale Dhanashri, S. Patil Babaso, and H. Patil Shubhangi. Mms steganography for smartphone devices. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, volume 4, pages V4–513. IEEE, 2010.

[8] N. N. EL-Emam. Hiding a large amount of data with high security using steganography algorithm. *Journal of Computer Science*, 3(4):223, 2007.

[9] K. J. Higgins. Busted alleged russian spies used steganography to conceal communications. *Dark Readings*, June 29 2010.

[10] R. Ibrahim and T. S. Kuan. Steganography algorithm to hide secret message inside an image. *arXiv preprint arXiv:1112.2809*, 2011.

[11] R. Ibrahim and C. K. Law. Mobisis: An android-based application for sending stego image through mms. In *ICCGI 2012, The Seventh International Multi-Conference on Computing in the Global Information Technology*, pages 115–120, 2012.

[12] N. F. Johnson and S. Jajodia. Exploring steganography: Seeing the unseen. *Computer*, 31(2):26–34, 1998.

[13] P. Kopiczko, W. Mazurczyk, and K. Szczypiorski. Stegtorrent: a steganographic method for the p2p file sharing service. In *Security and Privacy Workshops (SPW), 2013 IEEE*, pages 151–157. IEEE, 2013.

[14] W. Mazurczyk and L. Caviglione. Steganography in modern smartphones and mitigation techniques. 2014.

[15] U. D. of Justice. Criminal complaint, united states vs. christopher r. metsos et al. *FBI Documents*, 2010.

[16] F. A. Petitcolas, R. J. Anderson, and M. G. Kuhn. Information hiding-a survey. *Proceedings of the IEEE*, 87(7):1062–1078, 1999.

[17] Symantec. W32.duqu - the precursor to the next stuxnet (version 1.4). *Symantec Security Response*, Oct. 14 2011.

[18] M. Toorani and A. Beheshti. Lpki-a lightweight public key infrastructure for the mobile environments. In *Communication Systems, 2008. ICCS 2008. 11th IEEE Singapore International Conference on*, pages 162–166. IEEE, 2008.

[19] R. G. Van Schyndel, A. Z. Tirkel, and C. F. Osborne. A digital watermark. In *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference*, volume 2, pages 86–90. IEEE, 1994.

[20] A. Westfeld. F5 - a steganographic algorithm. In *Information hiding*, pages 289–302. Springer, 2001.

[21] E. Zielińska, W. Mazurczyk, and K. Szczypiorski. Trends in steganography. *Communications of the ACM*, 57(3):86–95, 2014.