Secure android covert channel with robust survivability to service provider restrictions

Avinash Srinivasan*, Jie Wu and Justin Shi

Department of Computer and Information Sciences,
Temple University,
Philadelphia, PA 19122, USA
Email: avinash@temple.edu
Email: jiewu@temple.edu
Email: shi@temple.edu
*Corresponding author

Abstract: The overarching field of information hiding has been extensively researched. However, the potential for use of smartphones – communicating over cellular service provider networks – has yet to be fully explored. In this paper, we propose Android-Stego – a covert communication framework for Android smartphones. We have presented results analysing real-world cellular service-providers' restrictions on MMS messages and the corrective actions they take using a prototype implementation of Android-Stego. With the prototype implementation, we have also analysed the messages at both the sending and the receiving ends to determine the service providers' actions on MMS messages – such as compression and/or format conversion.

Keywords: covert communication; information hiding; mobile security; robust security; information secrecy; steganography.

Reference to this paper should be made as follows: Srinivasan, A., Wu, J. and Shi, J. (2017) 'Secure android covert channel with robust survivability to service provider restrictions', *Int. J. Security and Networks*, Vol. 12, No. 1, pp.27–39.

Biographical notes: Avinash Srinivasan received his PhD in Computer Science from Florida Atlantic University in August 2008 and is currently a Faculty Member in the Department of Computer and Information Sciences at Temple University. His research interests include network security and forensics, forensic analysis of file systems, forensic file carving, and privacy and anonymity. He has published 38 papers in refereed conferences and journals including IEEE INFOCOM and ACM SAC. He is also a *Certified Ethical Hacker* and has trained both law enforcement officers and civilians in security and forensics. Since January 2016, he is a Fellow at the National Cybersecurity Institute, Washington DC, USA.

Jie Wu is the Chair and a Laura H. Carnell Professor in the Department of Computer and Information Sciences at Temple University, USA. Prior to joining Temple University, he was a Program Director at the National Science Foundation and Distinguished Professor at Florida Atlantic University. His research interests include wireless networks, mobile computing, routing protocols, fault-tolerant computing, and interconnection networks. His publications include over 600 papers in scholarly journals, conference proceedings, and books. He has served on several editorial boards, including IEEE Transactions on Computers and Journal of Parallel and Distributed Computing. He is a Fellow of the IEEE.

Justin Shi is an Associate Professor in the Department of Computer and Information Sciences at Temple University. His research interests include mobile programming, cloud construction, and high performance computing architectures and applications. He received his PhD and MS degrees from the Computer Science Department of the University of Pennsylvania and his BS degree in Computer Engineering from Shanghai Jiaotong University. He has contributed extensively in the cloud and parallel computing fields.

This paper is a revised and expanded version of a paper entitled 'Android-Stego: a novel service provider imperceptible MMS steganography technique robust to message loss' presented at *The 8th ACM International Conference on Mobile Multimedia Communications (MobiMedia)*, Chengdu, China, 25–27 May, 2015.

1 Introduction

The overarching field of *information hiding* techniques has been extensively researched. However, the potential for use of smartphones over cellular carrier networks has yet to be fully explored. *Steganography* – the art and science of hiding communication (Johnson and Jajodia, 1998) – is one form of information hiding that has been extensively researched over the last two decades. Additionally, while steganography has been known for centuries, dating back to the Roman Empire, only recently has it proliferated newer grounds – the all-pervasive and ubiquitous mobile cellular telecommunications services. The mobile cellular services are the most easily, readily, and economically accessible communication means to the masses. The mobile cellular penetration density was approximately 7.1 billion global subscriber identity module (SIM) subscriptions worldwide at the end of 2014,[1] and the total world population of 7.4 billion in Mid. 2016.[2]

With such ubiquity, they become the most lucrative attack weapons as well as targets. The methods for embedding secret data have been constantly evolving, 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.

Definition 1: Steganography is the art and science of hiding information within other innocent looking files.

Definition 2: Digital image steganography is one form of steganography in which data is hidden within a digital image file (a.k.a. graphics file) – including but not limited to – *.bmp*, *.gif*, and *.png*.

Definition 3: Entropy is a measure of randomness in a given message which increases when the message is closer to random and decreases otherwise.

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) (El-Seoud and Taj-Eddin, 2013; Patel et al., 2013a; Van Schyndel et al., 1994) method of embedding secret messages remains the most popular and the simplest of all to implement. Furthermore, LSB is also one of the main techniques in spatial domain image steganography. In this technique, the LSBs of some random bytes or perhaps all of the bytes of the cover image file (aka the carrier file) are changed appropriately to encode the secret message. The LSB embedding technique is explained in greater detail in Section 5.

Steganographic data is often hidden through the use of mathematical techniques that add information content to digital objects such as images, audio, and video files (Sinha et al., 2015), including other various digital objects such as executable codes (https://www.nitrd.gov/pubs/csia/csia_federal_plan.pdf). However, when sophisticated embedding techniques are used, the degradation in quality or increase in payload is perceptible. One of the first works on the feasibility of MMS-based steganography on smartphones is presented in Amoroso and Masotti (2006).

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. 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 has become very popular within the underground hacker community. This should cause serious concern, as steganography is now a new weapon in the already-sophisticated arsenal of the Black-Hat hackers. Steganography, over the years, has become the adversary's preferred mode for delivering malware and exploit payloads. Underground hacking communities are quickly embracing steganography techniques for data exfiltration. Our work combines cryptography and steganography, like the one presented in Dominic and Crina (2013), to achieve a secure communication channel on the Android platform.

Assumptions and threat model. For simplicity and clarity, we assume that Alice and Bob are the principals involved in covert communication over an unsecured channel.[3] They intend to exchange steganographic messages exploiting MMS services on Android Devices. We assume they have not established a pre-shared secret key. Hence, they will authenticate each other and spontaneously negotiate a session key using the standard public key infrastructure (PKI) and digital certificates. We will not discuss additional details pertaining to PKI infrastructure and the process of key negotiation due to page length restrictions. The communication channel used by Alice and Bob is subject to monitoring and it will be shut down if detected. Finally, Alice and Bob have no control over the communication channel once the message leaves the device. In our implementation, we have assumed and implemented a unique session key for each new message. Finally, this paper is an extension -version of our previous work (Srinivasan et al., 2015).

Our contributions. While we acknowledge numerous existing works similar to our proposed Android-Stego, including Dhobale Dhanashri et al. (2010), we do believe our work has its novelty, offers some very unique features, and makes important contributions. Some of the salient features of Android-Stego can be summarised as follows. Android-Stego – in its implementation and application – differs markedly from its contemporary methods that are mere LSB techniques. Android-Stego independently segments the secret message file into chunks, encodes them, and finally embeds them in a series of PNG (or other user supplied image format) images – all of which is done with minimum user interaction.

Furthermore, Android-Stego is a segmented, distributed, multi-part MMS Steganography with detailed cryptographic processes to provide core security requirements – *confidentiality*, *integrity*, and *source authentication*, which have been discussed in detail in later sections. Our paper presents the first and perhaps the only work to study and analyse 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 summarising the actions that each of the aforementioned carriers perform on MMS messages whose sizes exceed 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 this paper, we have presented the benchmark results for carrier-imposed size restrictions and the corresponding actions on oversized MMS messages. Most importantly, the prototype implementation relies on native images and MMS functionality common to most Android devices and does not necessarily depend on direct internet connectivity or carrier limitations.

Road map. The rest of the paper is organised as follows. In Section 2, we present some of the more relevant mobile device steganography related works, followed by a quick overview of background information related to the proposed work 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. A summary of all notations used throughout this paper is presented in Table 1.

Table 1 Table summarising the notations used

Notation	*Description*
M_{sec}	Secret message or file to be exchanged
$K_{\text{pub}}^{\text{Alice}}$	Alice's public key
$K_{\text{prv}}^{\text{Alice}}$	Alice's private key
$K_{\text{pub}}^{\text{Bob}}$	Bob's public key
$K_{\text{prv}}^{\text{Bob}}$	Bob's private key
$K_{\text{rand}}^{\text{Alice}}$	Session key randomly generated by Alice

2 Related work

'Operation twins', culminated in 2002 with the capture of criminals associated with the 'Shadowz Brotherhood', a pedophile organisation responsible for the distribution of child pornography, with the aid of steganography. Steganographic methods have also proven to be useful tools for data exfiltration, evidenced in the 2008 incident in which someone at the US 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 (Higgins, 2010; US Department of Justice, 2010).

A new worm called 'Duqu' was discovered sometime during September 2011, with glaring similarities to Stuxnet (stuxnet, 2011). 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 centre (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 (https://www.virusbtn.com/blog/2011/09_26.xml) and around the same time as Duqu.

Ibrahim and Kuan (2011) proposed a steganography algorithm for hiding secret messages inside a bitmap (BMP) image. Although the paper does not fully disclose the algorithm utilised, 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 in 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 maximise the amount of secret data that can be embedded in the cover image and transmitted. Based on the above algorithm, Ibrahim and Law (2012) 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 from their original algorithm.

Bucerzan et al. (2013) have presented SmartSteg, an application that works on Android platform and is able to hide and quickly encrypt files using digital images of MB dimension as a cover. LSB steganography is combined with a random function and symmetric key cryptography to transfer digital information, in a secure manner between smartphones that run under Android. However, their work does not talk about multi-part MMS messages, or secret message integrity, and also assume a pre-shared symmetric key between the principals. Zhou et al. (2012) have proposed a new MMS-based steganography method for an iOS, specifically iPhone, platform.

Another new approach securing data while transmitting or storing it in smartphones has been proposed in Vivek Amruth and Amrita (2014). The proposed method employs Multi-Level Steganography with a defense-in-depth mechanism for enhanced protection of embedded data and a compression method to effectively compress the data. Multi-level steganography has many levels of steganography employed, which makes it hard to detect without knowing the level number and the methods used in each level. The 'In-painting' method is used for compressing secret images and YASS steganography, which works on JPEG cover images, and resists blind steganalysis on both the levels. To enhance the security in the inner level, *one time padding* (XOR) is used.

Dastoor and Patel (2012) have considered speech signals 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, 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 (EL-Emam, 2007). These changes have minimally and imperceptibly changed the cover images and have made them less susceptible to steganalysis.

Mazurczyk et al. proposed SkyDe that utilises 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 (Kopiczko et al., 2013), 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 Zieliñska et al. (2014), which presents a literature review of state-of-the-art steganography. Another paper which interested readers may refer to is (Wendzel et al., 2014), in which authors have presented potential techniques for countering Network steganography. Mazurczyk and Caviglione (2014) have provided a summary of such efforts, along with mitigation techniques. Additionally, in Petitcolas et al. (1999), authors have presented a very detailed survey of information hiding techniques. Another survey on information hiding can be found in Patel et al. (2013b). Finally, there have been numerous other works that address steganography and other information hiding techniques in general including Thakre and Chitaliya (2014) and Gupta et al. (2013).

Sutherland et al. (2011) have presented and discussed two key areas – the potential for the drive operation to be impacted by malicious software and the possibility for the drive firmware to be manipulated to enable a form of steganography. Furthermore, in Wang et al. (2013), authors introduce information hiding technique for Flash memory that hides data within an analogue characteristic of Flash – the program time of individual bits. They argue and show that their technique uses analogue behaviours and has no impact on normal Flash memory operations, and hidden information is invisible in the data stored in the memory.

Suarez-Tangil et al. (2014) discuss a class of smartphone malware that uses steganographic techniques to hide malicious executable components within their assets, such as documents, databases, or multimedia files. Authors claim that they have introduced various types of stegomalware and demonstrate its feasibility with a prototype implementation of a stegomalware app that they say has remained undetected in Google Play. Finally, they have also introduced a detection system for stegomalware and use it to analyse around 55,000 apps retrieved from both malware sources and alternative app markets, but results are inconclusive.

3 Background

In this section, we will provide some basic discussions necessary for the reader 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 and 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, smartphones in particular, 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 become 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 images 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 multi-part implementation that supports MMS-based steganography on both sender and receiver devices with the following capabilities:

- splitting and encoding a secret message (can be multi-part depending on message size and service provider restrictions) on the sender side
- encoded message successfully traverses the networks surviving service provider restrictions
- 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 Zieliňska et al. (2014) and listed below, are both satisfied by the cover file in our proposed Android-Stego:

- 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.
- 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 multi-part, segmented, and distributed capabilities into the LSB encoding algorithm.

3.3 Android 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*.

3.4 Fundamentals of image file steganography

All image files employ some form of compression and all such compressions algorithms can be categorised into two fundamental types.^{4} –

- *lossless compression algorithms* – PNG, GIF
- *lossy compression algorithms* – JPG.

Definition 4: *Lossless compression algorithms* – these are algorithms that reduce file size while preserving a perfect copy of the original uncompressed image. Lossless compression are often known to, though not always, results in larger files than lossy compression. Lossless compression image formats should be the preferred choice when using image file as the cover file for generating steganographic object.

Definition 5: *Lossy compression algorithms* – these are algorithms that preserve a representation of the original uncompressed image that may appear to be a perfect copy, but it is not a perfect copy. Often lossy compression is able to achieve smaller file sizes than lossless compression. Most lossy compression algorithms allow for variable compression that trades image quality for file size.

There are numerous image file types. Especially when considering proprietary types, we look at hundreds of them. However, the most popular image file types happen to be – PNG, JPEG, and GIF – particularly on the internet. Also, some of the major operating systems like Macintosh OS X has an inbuilt screen capture utility,^{5} which by default saves the captured screen image as a .png image.

Another dimension along which images can be categorised, which has an impact on the proposed Android-Stego framework, is their representation, which again is categorised into two primary types – *Raster Images* (e.g., PNG) and *Vector Images* (e.g., JPG).

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 illustration, 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 Curve Cryptography* (ECC) encryption and digital certificate scheme is the most ideal for mobile devices (Toorani and Beheshti, 2008), 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 (Bos et al., 2009). Additionally, it can be up to 15 times faster, depending on the specific platform on which it is implemented.

4.1 Definitions

Definition 6: Asymmetric key cryptographic (a.k.a Public key cryptography) is one in which a unique pair of mathematically related keys – a public key and a private key – are assigned to each principal. The key pair is generated such that what one key does, the other key can reverse it. This property makes it very popular for achieving core security requirements of confidentiality and authentication.

Definition 7: Symmetric key cryptographic (a.k.a Private key cryptography) is one in which a single key is assigned to two principals involved in the communication. While symmetric key cryptography can provide very fast and strong security against disclosure, i.e., providing strong confidentiality, it fails to provide authentication and subsequently non-repudiation.

Definition 8: Symmetric key assigned to two principals is commonly known as a pairwise shared key. When a fresh symmetric key is generated for each session, then it is known as a pairwise session key.

Definition 9: Symmetric key assigned to more than two principals is commonly known as a group key. When a fresh key is generated for each session, then it is known as a group session key.

4.2 LSB embedding

Least significant bit simply refers to the rightmost bit of a byte of data in the big-endian representation. LSB embedding is a general steganographic technique employed to embed secret data into the LSBs of the carrier/cover file. While LSB based secret data embedding can be used with any file type, the most popular being digital media, specifically for hiding secret data inside an image file. LSB embedding derives its popularity primarily from their simple design and significantly high effectiveness. Furthermore, the LSB embedding technique is simple and easy to implement, with the only variation stemming from the file that is being used as the carrier/cover file.

Let us consider a simple example to understand LSB embedding in action. In Figure 1, we have presented four possible cases considering a scenario with a cover file of 8 bytes and a secret message of 1 byte. We chose such small size for simplicity and intuitive communication of our work. The stego-object is generated when the secret message is embedded into the cover file, one bit at a time, into the LSB of a previously unused byte of the cover file. The secret data bit that needs to be embedded is compared with the LSB of the byte under consideration for embedding. If the two match, then the corresponding cover file byte remains unchanged. Otherwise, the cover file byte's bit is flipped to match that of the secret data bit in question. With that, we have the following possible cases:

- *Best case*: All secret data bits exactly match the LSBs of the corresponding cover file's bytes. This case is presented in Figure 1(a).
- *Average case*: Only half of the secret data bits exactly match the corresponding cover file bytes' LSBs. This means, 50% of the cover file bytes used will have their lest significant bits flipped. This case is presented in Figure 1(b).
- *Worst case*: None of the secret data bits match the corresponding cover file bytes' LSBs. Therefore, 100% of the cover file bytes used for embedding the secret data bits will have their lest significant bits flipped. This case is presented in Figure 1(c).

Finally, in Figure 1(d), we present the scenario of representing a random byte of secret data in the given 8-byte cover file, and as can be seen, 6 bytes of the cover file have their LSBs flipped while two remain unaltered. This particular scenario falls between the average and worst case scenarios.

4.3 Android-Stego: 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, which can be negotiated offline, or negotiated online, in real time, using asymmetric algorithms. Below are the details pertaining to the encrypted and authenticated steganographic message exchange, assuming a shared key has been previously negotiated either in-band or out-of-band.

Stage-1: Alice selects a secret file M_{sec} 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 serialised 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 serialised secret file segments, and then converts them into steganographic bitmap PNG images, as shown in Figure 2.

For each pixel holding some part of the secret data, the integers representing red, green, and blue are evaluated and

Figure 1 Illustration of LSB embedding performance cases: (a) best case; (b) average case; (c) worst case and (d) random case (see online version for colours)

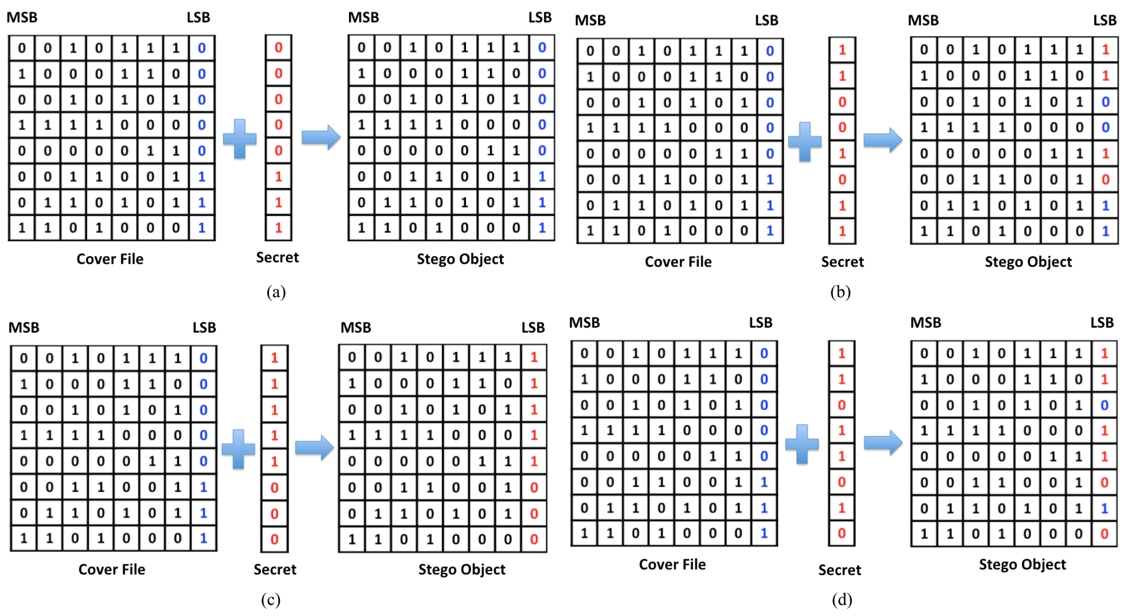
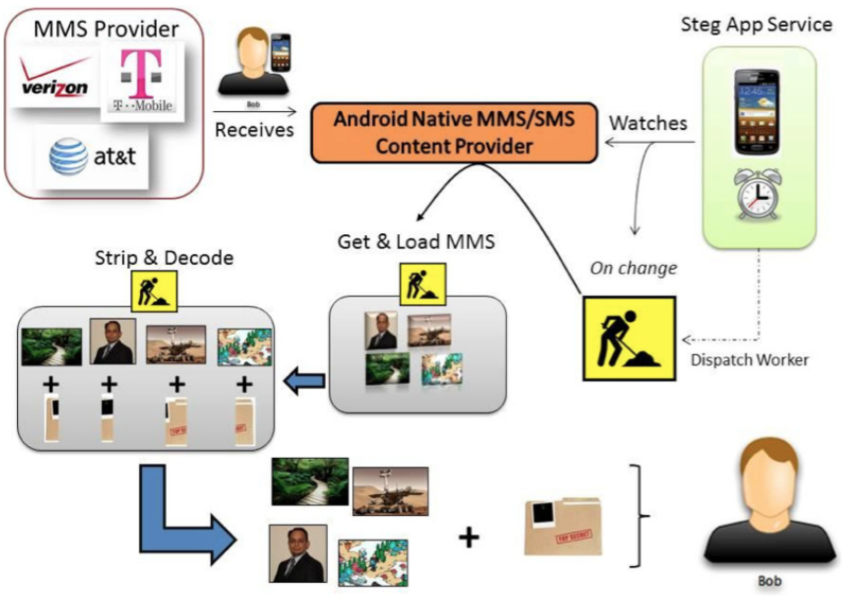


Figure 2 The process of embedding the secret message in a steganographic MMS message (see online version for colours)



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.

1 Alice is in possession of M_{sec}, the secret message that she wishes to share with Bob. She is also in possession of her public key $K^{\text{Alice}}_{\text{pub}}$, her private key $K^{\text{Alice}}_{\text{prv}}$, and Bob's public key $K^{\text{Bob}}_{\text{pub}}$.

2 Alice generates a random sessions key: $K^{\text{Alice}}_{\text{rand}}$, which she will use to encrypt M_{sec}.

3 If the size of M_{sec} is greater or equal to a predetermined permissible message size, Alice splits the message M_{sec} into multiple parts as follows – $\left[M^{\text{part}-1}_{\text{sec}}, M^{\text{part}-2}_{\text{sec}}, \cdot M^{\text{part}-\text{n}}_{\text{sec}}\right]$. The permissible size for these parts varies among service providers, a summary of which is presented for four major North American carriers in Table 2.

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

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

4 Alice computes a hash of the message M_{sec}, and its parts if any: $h(M_{\text{sec}})$. If the situation warrants that the message be split into multiple parts, as discussed in the above step, then Alice computes a hash value for each part, in addition to the hash of the complete original message.

5 Alice encrypts the secret message (or each individual part of a multi-part message) with the session key:

$$[M_{\text{sec}}]_{K^{\text{Alice}}_{\text{rand}}}. \tag{1}$$

Here, note that each part of a multi-part message is encrypted with the same session key $K^{\text{Alice}}_{\text{rand}}$. However, in implementation, different session keys can be used for added security.

6 Alice then encrypts the session key, which she generated in step 2 above, with her private key:

$$[K^{\text{Alice}}_{\text{rand}}]_{K^{\text{Alice}}_{\text{prv}}}. \tag{2}$$

7 Now, Alice appends the output from steps 4–6 as follows:

$$\left[[M_{\text{sec}}]_{K^{\text{Alice}}_{\text{rand}}} || [K^{\text{Alice}}_{\text{rand}}]_{K^{\text{Alice}}_{\text{prv}}} || [h(M_{\text{sec}})]_{K^{\text{Alice}}_{\text{rand}}}\right]. \tag{3}$$

8 Alice encrypts the output from step 7 with Bob's public key $K^{\text{Bob}}_{\text{pub}}$, as shown in Figure 3:

$$\left[[M_{\text{sec}}]_{K^{\text{Alice}}_{\text{rand}}} || [K^{\text{Alice}}_{\text{rand}}]_{K^{\text{Alice}}_{\text{prv}}} || [h(M_{\text{sec}})]_{K^{\text{Alice}}_{\text{rand}}}\right]_{K^{\text{Bob}}_{\text{pub}}}. \tag{4}$$

For a multi-part message, Alice includes $K^{\text{Alice}}_{\text{rand}}$ only if she had to change the session key for any security reasons. Otherwise, only the very first message transmitted will include $[K^{\text{Alice}}_{\text{rand}}]$.

9 Alice repeats steps 4–8, until all pieces of the secret message are transmitted.

10 In the case of a multi-part message, Alice transmits the hash of the original complete 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 4. On Bob's device, the prototype application service watches for changes upon the receipt of a message (MMS/Email). If a change is detected, the steganography application service dispatches a worker to execute the steps required to load and decode the images.

1 Bob decrypts the received message, shown in equation (4), with his private key $K^{\text{Bob}}_{\text{prv}}$ and extracts the following three components:

$$[M_{\text{sec}}]_{K^{\text{Alice}}_{\text{rand}}} \tag{5}$$

$$[K^{\text{Alice}}_{\text{rand}}]_{K^{\text{Alice}}_{\text{prv}}} \tag{6}$$

$$[h(M_{\text{sec}})]_{K^{\text{Alice}}_{\text{rand}}}. \tag{7}$$

2 Then, he uses Alice's public key $K^{\text{Alice}}_{\text{pub}}$ on the extracted component shown in equation (6), and extracts $K^{\text{Alice}}_{\text{rand}}$.

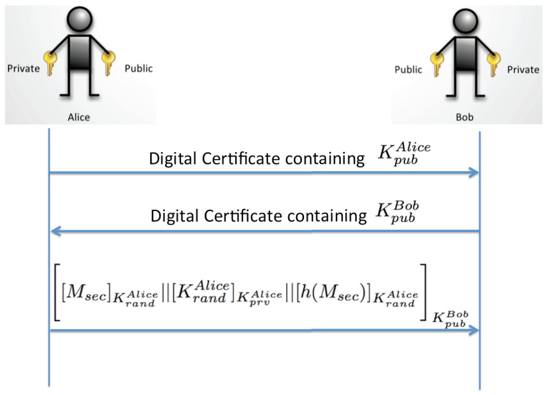
3 Later, using the session key $K^{\text{Alice}}_{\text{rand}}$, Bob decrypts the secret message and its hash shown in equations (5) and (7), respectively.

4 Bob then computes the hash of the decrypted message and verifies message integrity by comparing it to the hash extracted in the previous step. If the hash computed by Bob matches the hash included in Alice's original message, then Bob continues with the steps below; otherwise, he stops processing the message. In case of a multi-part message, he will discard that specific chunk and continues with the rest starting from step 1 above.

5 In the case of a multi-part secret message, Bob repeats the above steps (steps-1–4) until all pieces of the secret message are received, verified, and decrypted. He then reconstructs the original secret message from the pieces, computes the hash of this reconstructed original message, and compares it to the hash of the original unencrypted message received from Alice in step 10 for integrity verification of the overall message.

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.

Figure 3 Exchange of information between Alice and Bob (see online version for colours)


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

4.4 *Android-Stego implementation*

The 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. 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. The Android-Stego framework is presented in Figure 5.

Our implementation is a segmented and distributed implementation built on the LSB embedding technique. For the sake of completeness and enable readers better appreciate the work, we present a brief discussion on how LSB embedding technique works.

Here, we would like to draw the readers' attention to an important fact. Android-Stego is a framework that can be further customised to support formats other than PNG images, which is what we use to demonstrate its performance. For the curious reader, we have not attempted testing the Android-Stego framework on image types such as JPEG since they employ lossy compression algorithms.

More importantly, lossy compression can interfere making it hard to clearly attribute the lost stego messages to the appropriate cause and source, such as – channel loss, service provider filtering, reversal of lossy compressed cover image, etc. However, other formats that employ lossless compression (e.g., GIF) or apply no compression at all (e.g., BMP) can be used as cover files.

Since, we are focusing on a multi-part-based segmented implementation, we avoid utilising every single bytes' LSB of the cover file. Also, since the number of MMS messages required to send the complete secret message is controlled by the user (which is the client side of the app), there is latitude for the user to choose the level of compaction in each cover file. Lower the ratio of secret message size to cover file size – $\left[\frac{M_{\text{sec}}^{\text{size}}}{C_{\text{size}}}\right]$ – the higher the probability of survivability.

For illustration, let us consider that the size of the secret message $M_{\text{sec}}^{\text{size}} = 10\,\text{KB}^6$. and the size of the cover file denoted as $C_{\text{size}} = 2\,\text{KB}$. The framework can be tuned to use varying rates of embedding such that each cover file has only a certain percentage of secret message in it payload. The results of overhead and throughput are presented below. Only messages that pass message integrity check are counted towards the framework's throughput.

At payload ratio of 25%, only the first 25% of the cover file bytes are used for embedding secret message bits using the LSB technique. In the above scenario, at 25% payload ratio, we need 20 cover files of size 2 KB to embed and transfer a secret message of size 10 KB. If we increase the payload ratio to 50%, then we only need 10 cover file to transfer the secret message.

Therefore, the number of cover files required is impacted more by the embedding density, which we call the payload ratio, more than the size of the actual secret message. To this aim, consider the cover files to be of size 1 KB. Now, with a payload ratio of 50%, we can still successfully transfer the 10 KB secret message with 20 cover files.

4.5 *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 behaviour. 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 (https://code.google.com/p/f5-steganography/; http://www.openstego.info; Westfeld, 2001).

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, 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.

Our algorithm relies on bitmap data, which are lossy-compressed image formats, such as GIF or JPEG and they 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. The 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

Figure 4 The process of extracting the secret message from a steganographic MMS message (see online version for colours)

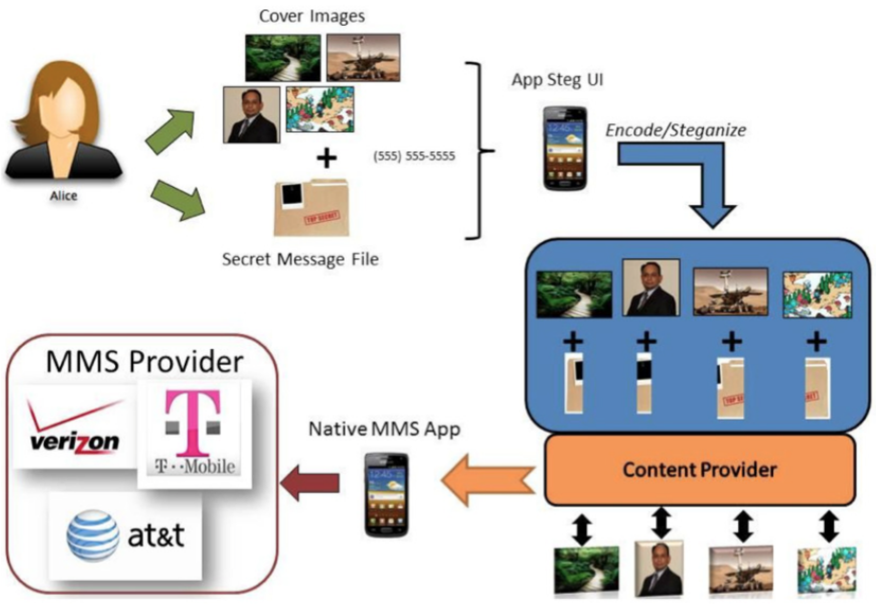
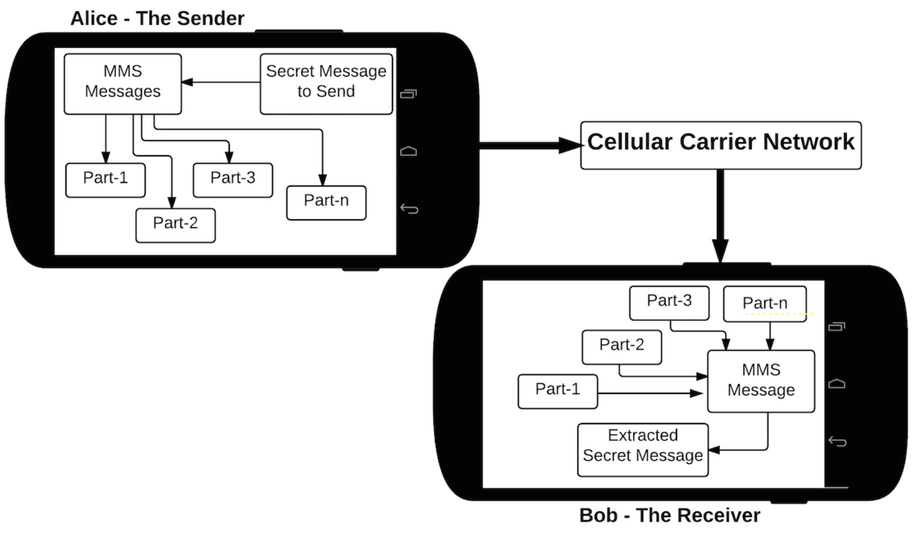


Figure 5 Generating an MMS stego message using our Android-Stego framework



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 (Ibrahim and Law, 2012). The higher the value of PSNR, the higher the quality of the steganographic image will be.

$$\text{PSNR} = 10\log_{10} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [C(x,y) - S(x,y)]^2. \qquad (8)$$

As noted in Ibrahim and Law (2012), if the cover image C has a size $(M \times M)$ and the steganographic image S has a size $(N \times N)$, then each C and S will have a pixel value of (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 summarised in Table 3.

Table 3 Summary of size and pixel range for the cover image and the steganographic image

Image type	*Size*	*Pixel value (x, y)*
Cover image C	$M \times M$	(0, $M - 1$)
Steganised image S	$N \times N$	(0, $N - 1$)

5 Discussions

5.1 *Android-Stego framework: security robustness*

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 $K_{\text{rand}}^{\text{Alice}}$, which is generated by Alice, no one will be able to discern the message without the key. Furthermore, 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. Otherwise, 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 communications. 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 very much necessary to ensure backward compatibility. This enables subscribers using older mobile devices to still view contents through MMS, 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 behaviour of this kind of data on four major US 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

Today, smartphones have risen to the pinnacle of pervasive and ubiquitous computing. Smartphones have created an unprecedented dependence thereby becoming the locus of our lives. Considering the ubiquity combined with the integrative nature of contemporary smartphones, steganography is undoubtedly the most easily accessible point for covert communication.

In this paper, we have proposed Android-Stego, a new framework that implements a multi-part steganography on Android smartphones exploiting the native MMS capabilities of the platform. We have implemented a working prototype of the proposed Android-Stego, and have verified its security robustness. Through actual message exchanges, we have confirmed the survivability of covert messages created using our Android-Stego.

We have also analysed MMS handling behaviour 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.

Part of our future research agenda is building fault tolerance into the proposed Android-Stego using techniques similar to threshold secret sharing. With such fault-tolerance, if a particular message chunk, in case of a multi-part message, fails the integrity check, then that chunk can be discarded without needing retransmission. Nevertheless, the original message can still be recovered as long as loss is under the threshold.

Acknowledgements

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.

References

Amoroso, A. and Masotti, M. (2006) 'Lightweight steganography on smartphones', *Consumer Communications and Networking Conference, 2006. CCNC 2006*, 3rd IEEE, IEEE, Vol. 2, pp.1158–1162.

Bos, J.W., Kaihara, M.E., Kleinjung, T., Lenstra, A.K. and Montgomery, P.L. (2009) *On the Security of 1024-bit RSA and 160-bit Elliptic Curve Cryptography*, No. EPFL-REPORT-164549.

Bucerzan, D., Raþiu, C. and Manolescu, M-J. (2013) 'Smartsteg: a new android based steganography application', *International Journal of Computers Communications and Control*, Vol. 8, No. 5, pp.681–688.

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

Dhobale Dhanashri, D., Patil Babaso, S. and Patil Shubhangi, H. (2010) 'Mms steganography for smartphone devices', *2nd International Conference on Computer Engineering and Technology (ICCET), 2010*, IEEE, Vol. 4, pp.V4–513.

Dominic, B. and Crina, R. (2013) 'Steganography and cryptography on mobile platforms', *Universitatii Maritime Constanta. Analele*, Vol. 14, No. 20, p.121.

EL-Emam, N.N. (2007) 'Hiding a large amount of data with high security using steganography algorithm', *Journal of Computer Science*, Vol. 3, No. 4, p.223.

El-Seoud, S. and Taj-Eddin, I. (2013) 'On the information hiding technique using least significant bits steganography', *International Journal of Computer Science and Information Security*, Vol. 11, No. 11, p.34.

Gupta, S., Bhushan, B., Singhania, S. and Gulani, J. (2013) 'A hybrid approach for ensuring security in data communication', *CCSIT, 2013*, 18–20 February, Bangalore, India.

Higgins, K.J. (2010) *Busted Alleged Russian Spies used Steganography to Conceal Communications*, Dark Readings, 29 June.

Ibrahim, R. and Kuan, T.S. (2011) *Steganography Algorithm to Hide Secret Message Inside an Image*, arXiv preprint arXiv:1112.2809.

Ibrahim, R. and Law, C.K. (2012) 'Mobisis: an android-based application for sending stego image through mms', *ICCGI 2012, The Seventh International Multi-Conference on Computing in the Global Information Technology*, 24–29 June, Venice, Italy, pp.115–120.

Johnson, N.F. and Jajodia, S. (1998) 'Exploring steganography: seeing the unseen', *Computer*, Vol. 31, No. 2, pp.26–34.

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

Mazurczyk, W. and Caviglione, L. (2014) 'Steganography in modern smartphones and mitigation techniques', *IEEE Communications Surveys and Tutorials*, Vol. 17, No. 1, pp.334–357.

Patel, K., Utareja, S. and Gupta, H. (2013a) 'Information hiding using least significant bit steganography and blowfish algorithm', *International Journal of Computer Applications*, Vol. 63, No. 13.

Patel, K., Utareja, S. and Gupta, H. (2013b) 'A survey of information hiding techniques', *International Journal of Emerging Technology and Advanced Engineering*, Vol. 3, No. 1, pp.347–350.

Petitcolas, F.A., Anderson, R.J. and Kuhn, M.G. (1999) 'Information hiding-a survey', *Proceedings of the IEEE*, Vol. 87, No. 7, pp.1062–1078.

Sinha, N., Bhowmick, A. and Kishore, B. (2015) 'Encrypted information hiding using audio steganography and audio cryptography', *International Journal of Computer Applications*, Vol. 112, No. 5.

Srinivasan, A., Wu, J. and Shi, J. (2015) *Android-Stego: A Novel Service Provider Imperceptible MMS Steganography Technique Robust to Message Loss*, ACM, Vol. 8.

Suarez-Tangil, G., Tapiador, J.E. and Peris-Lopez, P. (2014) 'Stegomalware: playing hide and seek with malicious components in smartphone apps', *Information Security and Cryptology*, Springer, pp.496–515.

Sutherland, I., Davies, G. and Blyth, A. (2011) 'Malware and steganography in hard disk firmware', *Journal in Computer Virology*, Vol. 7, No. 3, pp.215–219.

stuxnet (2011) *Symantec. W32.duqu – The Precursor to the next stuxnet (version 1.4)*, Symantec Security Response, 14 October.

Thakre, K. and Chitaliya, N. (2014) 'Dual image steganography for communicating high security information', *International Journal of Soft Computing and Engineering (IJSCE)*, Vol. 4, No. 3.

Toorani, M. and Beheshti, A. (2008) 'Lpki-a lightweight public key infrastructure for the mobile environments', *11th IEEE Singapore International Conference on Communication Systems, 2008. ICCS 2008*, IEEE, Singapore, pp.162–166.

US Department of Justice (2010) *Criminal Complaint, United States vs. Christopher r. metsos et al. FBI Documents*.

Van Schyndel, R.G., Tirkel, A.Z. and Osborne, C.F. (1994) 'A digital watermark', *Proceedings of the IEEE International Conference on Image Processing. ICIP-94*, IEEE, Vol. 2, pp.86–90.

Vivek Amruth, C. and Amrita, P. (2014) 'Multi-level steganography for smart phones', *2014 First International Conference on Networks and Soft Computing (ICNSC)*, IEEE, Tokyo, Japan, pp.81–84.

Wang, Y., Yu, W-k., Xu, S.Q., Kan, E. and Suh, G.E. (2013) 'Hiding information in flash memory', *IEEE Symposium on Security and Privacy (SP), 2013*, IEEE, San Francisco, USA, pp.271–285.

Wendzel, S., Mazurczyk, W., Caviglione, L. and Meier, M. (2014) 'Hidden and uncontrolled-on the emergence of network steganographic threats', *ISSE 2014 Securing Electronic Business Processes*, Springer, Brussels, Belgium, pp.123–133.

Westfeld, A. (2001) *F5 – A Steganographic Algorithm*, Information Hiding, Springer, pp.289–302.

Zhou, F., Yang, R., Zheng, Z. and He, J. (2012) 'Steganography in multimedia messaging service of mobile intelligent terminal', *5th International Congress on Image and Signal Processing (CISP), 2012*, IEEE, Chongqing, Sichuan, China, pp.1340–1343.

Zieliñska, E., Mazurczyk, W. and Szczypiorski, K. (2014) 'Trends in steganography', *Communications of the ACM*, Vol. 57, No. 3, pp.86–95.

Notes

[1] http://www.gsmamobileeconomy.com

[2] https://en.wikipedia.org/wiki/World_population

[3] In our scenario, an unsecured channel is one that the service provider can monitor for the presence of cover channels.

[4] Some image formats do not make use of compression at all. One such image format is the BMP image.

[5] The screen capture utility is enabled from the keyboard with the following key combinations ⌘ + ⇧ + 4.

[6] 1 KB = 1000 Bytes.

Websites

Alureon malware, *Alureon Trojan uses Steganography to Receive Commands*, https://www.virusbtn.com/blog/2011/09_26.xml (Retrieved 23 January, 2015).

F5-Steganography in Java, https://code.google.com/p/f5-steganography/ (Retrieved 23 January, 2015).

Federal Plan for Cyber Security and Information Assurance Research and Development, https://www.nitrd.gov/pubs/csia/csia_federal_plan.pdf (Retrieved 23 January, 2015).

Openstego: The Free Steganography Solution, http://www.openstego.info (Retrieved 23 January, 2015).