Mobile Steganography Embedder

Thomas F. M. White¹, Jean E. Martina¹,²†

¹ Computer Laboratory
University of Cambridge
15 JJ Thomson Avenue
Cambridge - UK - CB3 0FD

²Universidade Federal de Santa Catarina
Departamento de Informática e de Estatística
Laboratório de Segurança em Computação
Campus Universitário - Florianópolis - SC - Brazil
fredley@gmail.com, Jean.Martina@cl.cam.ac.uk

Abstract. Despite the capabilities of Android mobile phones, compared in specification to personal computers a few years ago, few programs for applied steganography has been written for such devices. Our objective is to produce an application that uses echo steganography to hide a short text message in an audio message recorded by the user, and then share that message. Someone with the same application on their device who receives such a message will be able to extract the hidden data from the audio message. We show our development strategy as well the evaluation process for our application.

1. Introduction

SMS messages, and MMS messages are easy to screen, especially for simple keywords. GSM itself has also been compromised [Paget 2011], so sending sensitive messages could be dangerous. Merely encrypting messages sometimes cannot be enough, as the vast majority of traffic over such systems is unencrypted (aside from GSM). Malicious parties could detect high levels of encrypted traffic as a signal of unwanted activity. By tracking who a person is in communication with, oppressive governments can identify and track social groups using social network analysis [Scott 2000].

Our project will build on the work done by Jenkins’s Steganography in Audio [Jenkins 2009], and will focus on implementing the steganographic methods used on the Android platform. We build an application in which a user can perform steganography without any complex setup or configuration, or specialist knowledge.

A user will be able to use our application to communicate securely and secretly with others in a way that seems innocuous to an observer with complete access to all data. Should there be a danger of the device being inspected, the application can be set up to erase all traces of usage from the device. It will also multicast messages, so as to obscure the target of a particular message.

*Project Supervisor
†Supported by CAPES Foundation/Brazil on grant #4226-05-4
1.1. Relevant Material

Most existing steganography applications only make use of least-significant-bit encoding, and there are freely available tools which can be modified relatively simply to detect hidden messages reliably in this case [Steg Secret 2011]. Apart from MP3Stego [MP3 Stego 2011] all use uncompressed audio to hold the hidden data, and most have command-line interfaces.

The steganography techniques explored in Jenkins [Jenkins 2009] go beyond, providing methods which resistant to steganalysis, at least going as far as making the problem computationally infeasible on a large scale [Meghanathan and Nayak 2010]. We will take the implementation of echo steganography to our application.

First proposed in 1996 by Gruhl et al. [Gruhl et al. 1996], the idea behind echo steganography is to introduce tiny echoes into the audio, similar to the echoes present in a room. The brain ignores these, making the changes in the audio almost imperceptible. Since the data is encoded in the actual audio itself rather than the bits of the file, this method is resistant to format changes. This is ideal for a mobile application, where data connectivity is often slow and/or expensive.

2. Design Decisions

The audio manipulation package javax.sound is not present in the stripped down version of Java available in Android, and the SDK provides no alternative. There is also no provision for recording uncompressed audio or transcoding.

We used several libraries to provide the easiest way of performing each task, and to improve the readability of the code. There is a small cost in terms of the size of the application, but this is not too large. To overcome these problems we have used the following libraries, and 3rd-party code:

- Rehearsal Assistant [Rehearsal Assistant Source 2011] - we used some classes in order to record uncompressed audio data and store as a wave file.
- javax.sound.sampled - This package is present in the full JDK, and is used for exactly the kind of low-level audio manipulation required in this project

All other functionality, such as encryption with AES and sharing mechanisms were available in the Android SDK.

2.1. Methodology and Planning

We chose to use an Evolutionary Development Model while developing the application, as it allows it to be built up in sections, writing and testing each module as it is added to the project. Formal testing is done after each evolutionary cycle with unit tests to check each class behaves according to specification. Later on in the development cycle versions of the application were released onto the Android Market.
2.1.1. Requirements Analysis

With the goal of the project is to create a usable application, the target audience must be considered. Recently there has been a period of unrest in various countries. One notable method used by governments to control their people in times of unrest has been to severely clamp down on communication networks. This user would likely have the following characteristics and requirements:

- Potentially low-powered device
- Potentially older versions of Android installed
- Ability to send via different mediums (e.g. Bluetooth)
- Possible requirement for support of non-English character-sets
- Plausible deniability highly desirable

Police agents working under cover are under a huge amount of pressure, not just from the stress of their job, but from lack of communication with the outside world. This user would likely have the following characteristics and requirements:

- Ability to multicast, preferably publicly (broadcast) is essential
- Plausible deniability is essential
- Ability to receive messages from a variety of public sources

It is worth noting at this point that there are a number of potential misuses of this technology. This is of course true for any security application. Phil Zimmerman, the creator of PGP points out in an article [Zimmerman 2011] after the 9/11 attacks on the US that he has ‘No regrets about developing PGP’, and that ‘...strong cryptography does more good for a democratic society than harm...’.

Considering the user personas, the application will need to perform:

- Record audio from the microphone, and embed a short text message into it using echo steganography.
- Compress this audio file into a file which is of an appropriate size for sharing.
- Share, and multicast via all available methods on the device being used.
- Open audio messages received by any method and extract hidden information.
- Operate the application in a ‘paranoid mode’, in which all usage data is wiped from the device, to ensure plausible deniability.

2.2. Application Design

Writing applications for Android encourages applications to be designed within certain constraints, and everything is centred on the Activity class currently in focus [Android 2011b]. This has the advantage of making it easy to design applications in the Model-View-Controller (MVC) design pattern [Reenskaug 2011].

UI layout is declared in xml files, and interaction with the UI is handled in Activities. The actual work of the application is handled in other classes and calls to these are made from the Activity.
When designing a security-centric application, attention must be paid to the lifecycle of the application. On Android, the lifecycle of an application is governed by the life-cycles of the Activities, and has several idiosyncrasies, most notably there is no concept of ‘quitting’. Activities have four states:

- Running - The application is in the foreground and the user is able to interact.
- Paused - The application is not in the foreground, but is partially obscured.
- Stopped - The application is not visible (‘minimised’), but still alive: it is retained in memory and maintains state.
- Finished - The application is not active.

2.2.1. Model - Steganography

There are two classes which handle the bulk of the work, EchoStegFile and BitStream. The series of operations that need to be performed for embedding and extracting are shown here on Figures 1 and 2.

The structure of the application is simple, as the process of encoding and decoding is a 2-way pipeline. All views are managed by the StegDroid Activity, except in the case of Settings and Multicast. All input from the user is managed by the active Activity, in the cases of Settings and Multicast, this is done automatically.

2.2.2. View - User Interface

The application needs to perform two basic tasks, encoding and decoding. Of these the more challenging from a UI design perspective is encoding, since it requires stepping through a sequence of actions. Settings for paranoid mode and cryptographic keys are accessible via the menu key, and pressing the back key takes the user to the previous step, or exits the application from the first step.

2.2.3. Controller - Device Interaction

As previously stated, all interaction between the user and the rest of the application happens via the Activity classes. Functionality can be delegated from the Activity, but
it must pass the instance of itself to every class it wishes to delegate to for use as a context.

3. Implementation
In this section we will go through each stage of the steganography process and explain how we implemented it. Screenshots are provided in this document are of the finished application. StegDroid is available on the Android Market, a link\(^1\) and QR code are provided. At time of writing, StegDroid has been downloaded almost 2000 times, and has an overall rating of 4.1/5 stars on the Android Market rating system.

3.1. Class Organisation
BitStream class deals with taking a String and returning a stream of bits, or vice-versa. It has the option of being passed a key-phrase and returning/decoding an encrypted stream of bits. EchoStegFile class deals with the steganographic process of inserting and retrieving bits from an audio file. It deals only with audio files in wave format.

3.2. Audio Manipulation
Android built in MediaRecorder class does provide access to the raw PCM data from the microphone, but provides no built in mechanism for creating usable Wave files. Android again provides no way to manipulate Wave files at a sample level.

Transcoding is handled by Jcraft's Jorbis library \cite{Jcraft Jorbis Project 2011}. This library provides methods to transcode between Ogg Vorbis audio files and uncompressed Wave files. Ogg Vorbis files are used by Android as ringtone files, so it is a relatively innocuous data type to share.

3.3. Steganography
The processes of embedding and extracting data are very different. Embedding requires adding echoes to the audio, which is relatively straightforward. Extracting the data again requires performing Fourier Analysis on each sample in order to work out which echo was used.

The process of Echo Hiding convolves the raw audio signal with one of two echo kernels, with different delays. These echo kernels correspond to 1 and 0. A double back-forwards echo kernel is used, described by the equation
\[ y[n] = x[n] + \alpha \cdot x[n-d] + \alpha \cdot x[n+d] + \alpha \cdot x[n-de] + \alpha \cdot x[n+e], \]
where \(x\) is the original signal, \(y\) the output signal, \(\alpha\) is the amplitude of the echo and \(d\) and \(e\) are the two delays used.

The audio sample is split up into windows and the appropriate echo kernel is applied to each window. In order to prevent audible artefacts when switching between signals, a smoothing period is applied between each window, when the signal from the previous bit is faded out and the signal for the next bit is faded in. This is shown in Figure 3.\(^3\)

\(^1\)https://market.android.com/details?id=uk.ac.cam.tfmw2.stegdroid
3.3.1. Extraction

Using Fourier transforms, the cepstrum is calculated for each segment and the cepstrum for the echoes for 1 is compared against the cepstrum for the echoes for 0. The larger value determines the bit sent to the output bit stream.

The cepstrum of a signal is calculated by taking the complex logarithm of the Fourier transform of the signal and performing an inverse Fourier transform. The resulting data will show peaks corresponding to the echoes in the original signal. As can be seen by examining the convolution of the equations being employed. First take two input signals, $x_1[n]$ and $x_2[n]$. Their convolution, $y[n] = x_1[n] * x_2[n]$ is transformed into the Fourier domain:

$$Y(e^{j\omega}) = X_1(e^{j\omega})X_2(e^{j\omega})$$

The complex log of $Y(e^{j\omega})$ is then:

$$\log Y(e^{j\omega}) = \log(X_1(e^{j\omega})X_2(e^{j\omega})) = \log X_1(e^{j\omega}) + \log X_2(e^{j\omega})$$

The cepstrum of a signal $x[n]$ is defined to be $\tilde{x}[n]$, and the above equation is equivalent to:

$$\tilde{y}[n] = \tilde{x}_1[n] + \tilde{x}_2[n]$$

By comparing the cepstrum signal at the values for each of the 4 echoes, the echo kernel used on that segment of data should have a higher values its two echoes than the echo kernel not used. A Hamming window is applied to the signal in the time domain, before it is transformed. This is done by the function:

$$timeDomain[i] = 0.53836 - 0.46164 \cos(\frac{2\pi i}{N-1})$$

The Hamming window as transforming to the Fourier domain implies an infinitely repeating signal. Since the start and end of the signal are very unlikely to be continuous this will result in a lot of high-frequency noise in the result, which is undesirable. Windowing makes sure that the ends of the signal are continuous and prevents this spectral leakage.

Encryption of messages is provided, using the AES/ECB/PKCS7Padding cipher suite with a pre-shared key. The user can enter their passphrase in the settings.
page of the application, and a SHA-256 hash of the passphrase is used as the key.

3.4. User Interface
Care has been taken to create a user interface that guides the user through the process of encoding and decoding messages. An example screen for the system is shown on Figure 4.

Across all of the screens there are status indicators at the top of the screen, displaying whether encryption or paranoid mode are enabled. Below that is a progress indicator, displaying the step the user is currently on. Below that are brief instructions for the page. The Back and Next buttons are always at the very bottom. While playing or recording is active, other buttons are disabled. In the first and final steps, Back and Next buttons are displayed but are inactive.

For the application to actually be useful to users, it must be able to interact with other applications in order to send and receive messages. Luckily, with one notable exception, this is all handled by Android by means of a mechanism call Intents. Sadly, sharing data via MMS is not possible because the application does not register to handle audio data. If this is fixed in the future, the application will then be able to share data in such a way.

3.5. Paranoid Mode
This attempts to provide plausible deniability by removing all data created by its use from the device. When it is turned on, whenever the application is ‘Paused’ [Android 2011a], that is to say minimised or ‘closed’ by the user, if paranoid mode is enabled, all files created by the application are removed from the filesystem.

3.6. Multicast
We investigated a number of different ways of implementing multicast with the application. We chose to use contact groups built into the Google contact manager as a convenient way to send a message to a group of recipients at once.

4. Evaluation
Evaluation data has been collected from a variety of sources, including Android Market feedback. All these sources indicate the application fulfils its stated purpose, and is reliable and easy to use. During implementation, unit tests were used to confirm that parts of the application were working correctly. These were written in a separate test class which performed checks for the functionality it was testing. No unit tests were written for the steganographic process.
4.1. Steganography Testing

A variety of tests were done to optimise the steganography process. There are three factors to optimise: bit-rate, reliability (bit-error rate) and ease of detection. Bit-rate can be calculated from the parameters chosen, reliability can be calculated by measuring the bit-rate with a series of trials, but ease of detection is subjective, so a rough estimate will have to suffice.

Reliability was measured by creating a BitStream, passing this BitStream through the steganographic process and measuring the number of bits that were wrong in the output. A percentage error-rate was then calculated. The parameters that can be modified to optimise these factors are Segmentation Length, Windows Size, Volume Reducer and the four echo parameters. Each variable was altered in turn keeping the others constant. Three trials were done with three different recordings.

As shown on Figure 5, 1024 as the Segmentation Length seems to be the optimum from this data, performing as well as 2048 and above but with a higher bit-rate, but during transcoding testing, a Segmentation Length of 2048 proved much more reliable, and the longer file size required at this stage is compensated for to some extent by the fact that better compression can be used.

4.2. Transcoding Testing

Tests were conducted to find the optimum quality to encode the Ogg Vorbis files to get good reliability and minimise size. Three files were taken that had scored 100% on the previous test. These were encoded and decoded through the Vorbis decoder and the bit-error rate was calculated in the same manner as the previous test. The error rate is recorded, as is the compression rate as a percentage of the original size.

From these results, (Figure 6) it seems that a value of 0.4 is good. At this level there is still the possibility of the occasional bit error. Given the purpose of the application, small errors are not a problem, especially as users have the chance to test extraction of the message before sending it.

The overhead required to add an error-correction layer into BitStream would be too great, although if the application were adapted to send other kinds of data this would be necessary.
4.3. Threat Model Analysis

Given that the application is designed for sharing potentially sensitive information, an analysis of potential attacks is critical. Detection of the application is the main threat to be considered, as the whole point of using steganography on top of encryption is to prevent an attacker from even detecting potentially secretive communication.

4.4. User Survey

Most of the testing so far has focussed on the functionality of the application. As part of the specification was to create a usable application, evaluation of the usability was conducted with a survey of users. The survey was conducted with the participant using a Google Nexus S. They were given a brief tour of the Android operating system, and shown how to open and interact with applications. They were given a list of tasks to complete with the application. Once they had completed the tasks they were given a questionnaire to complete, in which there were asked to rate their experience of the application. Twenty participants were surveyed (University of Cambridge undergraduate students), with the mean results for the first 7 questions shown on Figure 7. The scores range from 0-5. The lower the better.

These are positive results, which show that the interface we have created allows people with relatively little knowledge of Android phones to be able to use the application easily, putting complex steganography within reach of many more people as a result.

![Figure 7. User Survey Result](image)

5. Conclusions

The project proposal set out the following success criteria:

- To create an Android application that makes use of audio steganography. This criterion has been fully realised, with steganography classes that not only perform the function of the application, but can be extended to act as a container for any data.
- To use audio steganography to embed a user’s text message in a voice message recorded on the device. This criteria has also been realised, the application guides users through the process of recording a voice message, embedding a text message and sharing that message.
- To make the application leave as little evidence of its use as possible. This criterion has been fulfilled, to a reasonable extent. In paranoid mode, the application removes all data from the disk and memory that could show use of the application whenever it is closed.
Having implemented these steganographic tools on the Android platform, they could potentially be used in a number of different ways. One interesting use could be to use steganography on audio during a phone call. This would allow for a data channel between two people during a phone call. This could be used to exchange covert text messages as in our application, or with other protocols to exchange any type of data.

References


