StegoTweet

MSc ASE Project Report

Terence McCartan

A thesis submitted in part fulfilment of the degree of MSc Advanced Software Engineering with the supervision of Dr. Gúenole Silvestre

School of Computer Science and Informatics

University College Dublin

22 April 2016

Abstract

This Thesis is based on generating a secure means of communication in a public forum using steganography and the Twitter API. It does this by embedding and deciphering secret text within images posted to Twitter API in the form of tweets. The embedding and deciphering stages uses strong steganography to securely and reliably store information within the image.

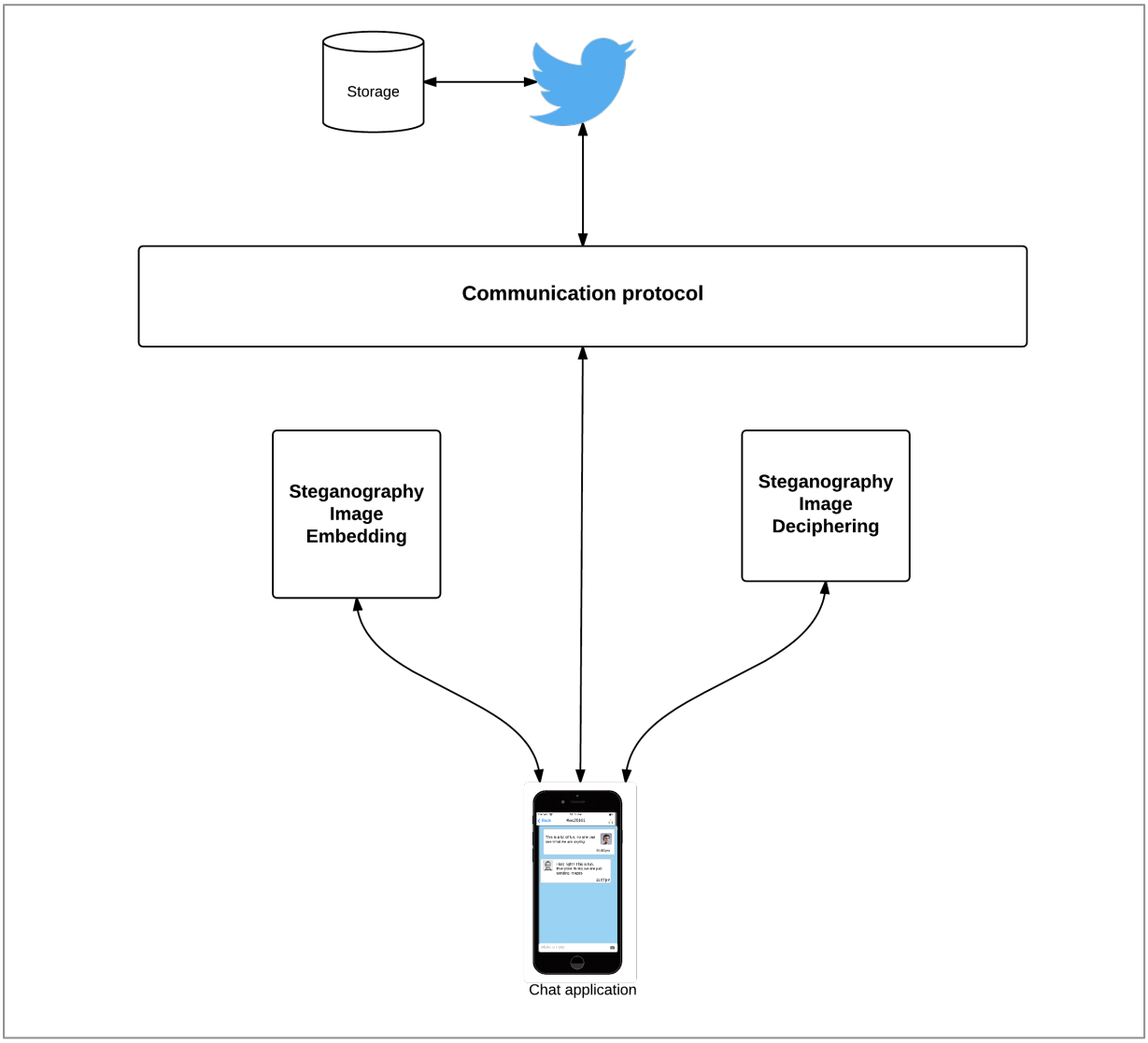

Fig 1 - Overview Diagram

By employing strong steganography and a communication protocol for creating and continuing conversations, it is possible to have a secure form of communication across a public forum without the entire Twitter audience being aware of the conversation.

Introduction

StegoTweet is an application that leverages on the Twitter API and steganography to create a secure means of communication between interested parties. Social media has played a critical role in the development of Modern communication, but it primarily provides a public forum for information sharing. The idea of obscuring or hiding secret messages in a seemingly public forum however has been used for hundreds of years. Some of the earliest forms of these secret messages can be found in ancient cryptography used in hieroglyphs that date back to the 1900's.

History of hiding messages

Steganography or the practice of hiding information to the naked eye can be traced back to 440 BC, where secret messages were encoded into seemingly normal wax tablets. Steganography provides benefits over other forms of ancient cryptography such as Caesar ciphers, scytale transposition ciphers since in these forms messages are obviously unreadable or incomprehensible but are easy to detect that there is a hidden meaning. Steganography hides information in such a form that is undetectable to the naked eye. As communication methods improved with time, so did the application of steganography. During the French revolution, messengers would use invisible ink to hide messages within seemingly normal communication, that if intercepted would not leak any critical information. This method became known as Sympathetic Messaging. Without knowing about the application of the ink, the message would be secure. This method while effective was brittle since once the secret of the ink was known the practice became useless.

Modern applications

Modern steganography can take on many forms with examples such as digital text, images, network and even more advanced printed forms of communication. Social media provides ample space for the application of steganography, since it provides multiple methods of modern communication such as text based statuses (or tweets), image based communication and even video based communication. With these advances, steganography algorithms have improved to become more robust. Early applications of steganography would cause some distortions to the image or creating simple patterns in the images bytes that allowed it to be easily detectable. The advances of detection algorithms forced user's of

steganography to come up with more robust applications that not only prevented detection by the naked eye but can also be used without knowing that there was any manipulation done to the raw image.

A digital image can be described a just a 2-D matrix containing the colour values at each grid point. Colour images utilize more colours that grey pixels as the describe combinations of Red, Green and Blue values (RGB) and can display colours at different intensities. The variance of a single digit in these colours is difficult for the human eye to pick up and it's the ability to create slight variances in the colour of a pixel that allows steganographic messages to be encoded.

Related Work

In this section, some of the more influential works in the area of steganography are discussed. Steganography, or its more modern equivalent Digital Watermarking is a rich area of study with some great studies in [1,2, 3,5,6,13]. This research forms the base for which the App is rooted on. Alongside these papers, other methods of steganography are shown in [4,8, 12] which provide different methods that can be used to embed data within a signal. An important area of research is also the analysis of possible steganography known as steganalysis. In [1, 6, 18] we learn about popular methods and how different steganography methods measure up in terms of steganalysis.

The area of steganography was the primary source of research used but secondary sources such as [14, 15] were helpful in designing a communication protocol that would serve the purpose needed. Chat Apps are widely available on the iPhone App store and these Apps provide the inspiration for the UI design used in StegoTweet. On App store there are other Apps that provide steganography functionality to embed data into images. For example SpyPix and StegoSec both allows users to embed data into an image although the form of steganography is not released. The do not include the ability to create conversations between parties, as they only allow the output of embedded images. Researching Apps that provide both strong steganography methods and ability to create conversations with the images turned up no results. This means that there could be a niche available in the market if there was such a desire.

Steganography methods

This section will introduce some of the more popular methods of steganography. Each algorithm has its benefits and disadvantages. These methods provide some insight into the various techniques used to create secret messages within an image.

LSB

LSB or least significant bit steganography is one of the main techniques used in spatial domain image steganography. It involves analysing the lowest significant bits in an image. Once these bits are identified, the secret message is embedded into the byte value of pixel.

By using the least significant bit, the algorithm is relying on the fact the level of precision of the average human eye, is far less than the precision of many image formats. The image that becomes altered looks the same to a normal user, but will contain slight changes to the colour of the original image. In most LSB techniques a certain number of bytes are required before a single byte can be used to “hide” the message in. For example, a conventional LSB algorithm will require 8 bytes that contain normal pixel information, with 1 byte then used to embed a different pixel value. This allows the image to be distorted in such a way that it is indistinguishable to the naked eye. Improvements have been made that require less bytes for hiding to occur and more advanced ways to embed the message to better secure the embedding process.

The primary issue with LSB is that it is not robust enough to handle modern digital image processing. Any modifications to the image such as resizing can cause bits to be removed or changed. This would cause parts of the message to be missing or lost entirely. Another issue is to do with detectability. Since it is quite easy to identify the lowest significant bits in an image, it is also quite easy to spot patterns of encoded messages making it easy for people to find images that follow the pattern. Although various methods have been suggested to improve the basic LSB algorithm (StegoTweet, BattleSteg) they still suffer from the simplicity of their approach.

Bit Plane Complexity Steganography

The bit plane complexity algorithm improves on the LSB algorithm by using image transformation and analysis to identify areas that messages can be embedded in. The process involves converting the base image from pure binary code (PBC) into a canonical grey code system (also known as CGC). Once it's converted into this format, the arrays of integer values are broken up into “bit planes”. These planes are then separated into two sets; an informative set and a noisy set using a complexity threshold value (typically a low value such as 0.3). The secret message is then encoded into a series of byte blocks. The secret blocks are then processed by comparing the complexity of the block to the selected threshold. If the block is less complex than the threshold, the block is then processed to become more complex than the threshold value. The block is then embedded into the noisy section of the image recorded into a map. The map itself is encoded at the end of the algorithm in the noisy sections as well, which is used for deciphering the message.

Bit plane improves on the detectability of steganography applied to the image because of the pre-processing and image manipulation that occurs. Since the image is broken up into bit planes by using the threshold value, without knowing the exact threshold value the secret message cannot be deciphered. If an incorrect threshold value is selected, the informative and noisy segments will not match up so any secret message bits that were originally embed in the noisy segments would be contained within the informative and vice versa. This includes the translation map, which holds the processing information that occurs for less complex bits then the selected threshold value.

Another benefit of the BCPS algorithm is an increased capacity for storing messages within an image. Studies have shown that for a true colour 32-bit image, storage capacity is increased by almost 50%. This is partly gained by altering the complexity threshold, which allows for a greater number of noisy segments. There is a trade off however; a reduction in the complexity threshold can create deterioration of the original image to a point that it becomes noticeable that processing has occurred. A simple way of increasing the capacity of the image is to first pre-process it by sharpening the image. This reduces the in between pixel values that handle the blur between colours. This reduction increases the complexity in the image, which in turn increases the noisy segments when the image bytes are passed through the algorithm.

A drawback to the BCPS is its robustness. Any modifications to the image after embedding can cause the information to be lost. This includes sharpening, clipping or general resizing. This is due to the image processing removing or modifying key information that is required to decipher the image. The inventors of the algorithm see this as a benefit because any modification of the image after processing automatically destroys the hidden messages. This means that only parties that have the correct customisation parameters can correctly decipher the message. This restriction makes the algorithm unsuitable for use within the scope of this application, since image processing can occur on social media websites.

Quantization Index Modulation Steganography

Quantization Index Modulation (QIM) algorithm as defined by B.Chen and G.Wornell in [1], is a method of embedding information within a source signal. Digital watermark and steganography share a lot of similarities and algorithms in one area are synonymous in the other. The primary application of QIM as described in the paper is digital watermarking for copyright material, which involves embedding a hidden signal within the source signal without degrading the original source signal. This allows for a digital fingerprint to added copyrighted material, which can be used to trace back illicit copies back to the original copy. Another interesting application of QIM as suggested in the paper is the upgrading of an existing communication system. This allows for dual band radio signals to broadcast without disrupting the original signal, by embedding the new signal within the old one.

Quantization is the reduction in a large set of input values into a smaller and more countable result set. A simple example of quantization is rounding of values. In image processing, quantization is a lossy compressions technique used to create a compression of a range of values into a single quantum value. To explain how QIM can be used for steganography, we must first define the problem model. When we want to embed a secret message we are slightly distorting the source image in a pattern that allows us to reverse the distortion to decode the message. To do this distortion, we need to use a set of quantizers that can distort a stream of data. The distortion must not cause the image to be detectable to the human eye and must also be resilient to detection of steganography.

The first step in applying the QIM method is to consider the problem model. The secret message can be considered as m and the host signal as a vector of $x \in < R^n$. We want to embed the bit from m in the host signal at rate of bits per host signal sample. The rate of embedded bits per host signal sample can be considered as R_m. Therefore we can consider m to be

$$m \in \{1, 2, \ldots . 2^{nR^m}\}$$

The next stage is to identify an embedding function that will embed information from m to the new signal (i.e. the image with the hidden message) $s \in R^n$. This function however must not distort the image past a certain boundary, otherwise the image would be too distorted and be easily detectable. To prevent this we introduce a distortion measure $D(s, x)$ that prevents the two signals from deviating too much. In [1] B.Chen and G.Wornell describe how to measure different expected distortions by creating scenarios in which new composite signals are processed using standard image processing manipulations. They continue on this path to inspect deliberate attempts to remove parts of the embedded signal to inspect the algorithms robustness. They introduce 3 channels that represent the outcome of these image manipulations namely "Bounded Perturbation", "Bounded Host-Distortion" and "Addition Noise" channels. They identified that the goals of having a high rate of embedding, a low distortion of the original image and high robustness as conflicting goals, which must be traded off from each other.

The embedding functions for signal processing have been widely researched but Quantization Index Modulation provides a good trade off for rate distortions versus robustness. If we treat the embedding function of QIM as a group a functions on x denoted by m, then the functions can described as $s(x: m)$. Since we are targeting the image to have the least amount of distortion as possible, then each function in the range must conform to or at least be close to an identity function so that $s(x: m) \approx x, \forall m$.

If all of these identify functions are quantizers then we can identify the embedding method as QIM. In short, the QIM method is to first modulate the indices of a signal then reduce it with a series of quantizers.

We can represent the message we want to encode as a series of bits which only have 2 possible values therefore we can define the message as $m \in \{0, 1\}$. To correctly embed the message within the source signal we need two quantizers, one to handle when the bit is 0 and the other to handle when it is 1. When processing the bit of the source image, we can embed the message by modulating the indices using the quantizers to increase or decrease each secret bit to the nearest index.

Dither Modulation Steganography

Building on the QIM algorithm, the Dither Modulation algorithm aims to embed a source signal within a host signal by using quantizers that are offset with a dither value. As defined in the QIM algorithm, two quantizers are used for the embedding of bits when they are 1, 0. The dither value is a pseudo random scalar value that offsets the quantizers so that the quantization of the bit to the nearest index is different to a standard quantizer without a dither. The purpose of the dither is to increase the resistance to the image being decoded by an unwanted party. For example, if an Alice and Bob are using QIM to embed their secret messages, it is possible but unlikely that another party named Charlie could analyse enough images to discover how the quantizers are setup and decode Alice and Bob's images. Initially described in B. Chen, G. Wornells paper [1] dither modulation was suggested as a way to increase the robustness of the QIM algorithm while still maintaining the benefits of QIM.

Basic Dither Modulation realization also known as Coded Binary Dither Modulation systems are constructed by creating a coded bit sequence of the secret message. By representing the embedded message as a vector of information bits NR_m as $NR_m\{ b_1, b_2 \ldots. b_{NR_m} \}$ and error correcting each bit using a rate K_u / $K_c$code, a coded bit sequence is obtained $\{z_1, z_2, \ldots. z_{N/L}\}$ where

$$L = \frac{1}{Rm}(k_u/k_c)$$

The host signal *x* is then divided into *N/L* non-overlapping blocks of length *L*. The available coded bits z_i are then embedded into the available blocks of *L*. The embedding of these bits requires that two length *L* dither sequences are created $d[k, 0]$, $d[k, 1]$and one

length L sequence of uniform, scalar quantizers. The step sizes of each of these quantizers are generated with the following constraint.

$$d[k,1] = \begin{cases} d[k,0] + \Delta_k/2, & d[k,0] < 0 \\ d[k,0] - \Delta_k/2, & d[k,0] \geq 0 \end{cases}, \quad k = 1, \ldots, L,$$

The purpose of this constraint is to maintain that each of the dither quantizers do not collide when embedding data. To embed the data, each i-th block of x (the host signal) is quantized with the dithered quantizer using the dither sequence of $d[k, z_i]$. In the diagram below, FEC stands for forward error correction, which is described later in the section.

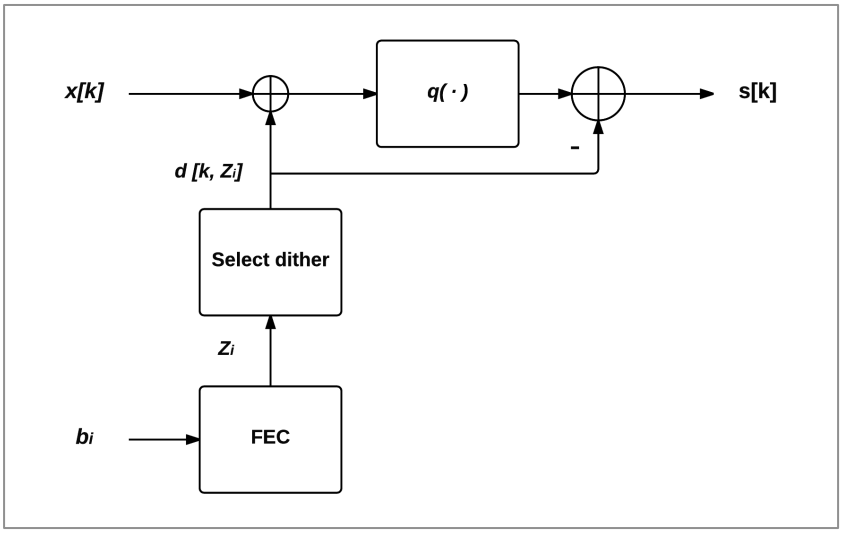

Fig 2 - Dither Modulation Embedding

Spread Transform Dither Modulation (STDM) is a form of coded binary dither modulation where only certain vectors of a host signal are quantized. Quantizing these vectors and combining them with non-quantized vectors offer performance improvements such as less distortion of the source image. The quantizers in STDM are similar to the ones used in QIM, with the addition that each quantizer is modified with a dither. If we take the quantizer $q(\cdot)$ we can describe it as a DM quantizer when the generated ranges are shifted by the value of the pseudo randomly chosen dither. With DM the dither vector is tied to the information that is being embedded; therefore each possible value of the secret message m is given a dither value $d(m)$. This allows us to rewrite the embedding function of QIM to be defined as STDM such that

$$s(x:m) = q\big(x + d(m) - d(m)\big)$$

A simple example is that the signal is broken up into a number of Δ where each Δ is given a quantizer of 1, or 0. The Δ is found by breaking the signal into smaller ranges similar to QIM but after creating the ranges, they are offset with the dither. If we want to encode a bit of 1 into the signal we move the current signal to the centroid of the nearest Q1 sections and vice versa for a 0 bit.

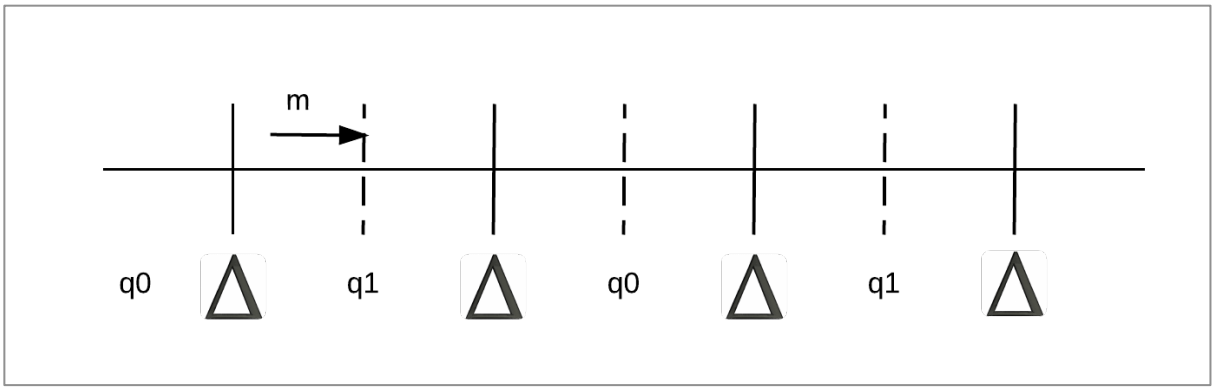

Fig 3 - Quantizer Example

To decode an image that had a message embedded using STDM, it follows the same principles of decoding embedded information in base DM systems. The host signal y is sampled so that $y[k]$ is used to calculate the distance between itself and the nearest reconstructed quantizer *s[k,1]* , *s[k,0]*. To decipher whether the bit is a 1 or a 0, we calculate the minimum quantization error by looking at the where the current signal is within the range. The following block diagram specifies the steps involved in the decipher an image with STDM

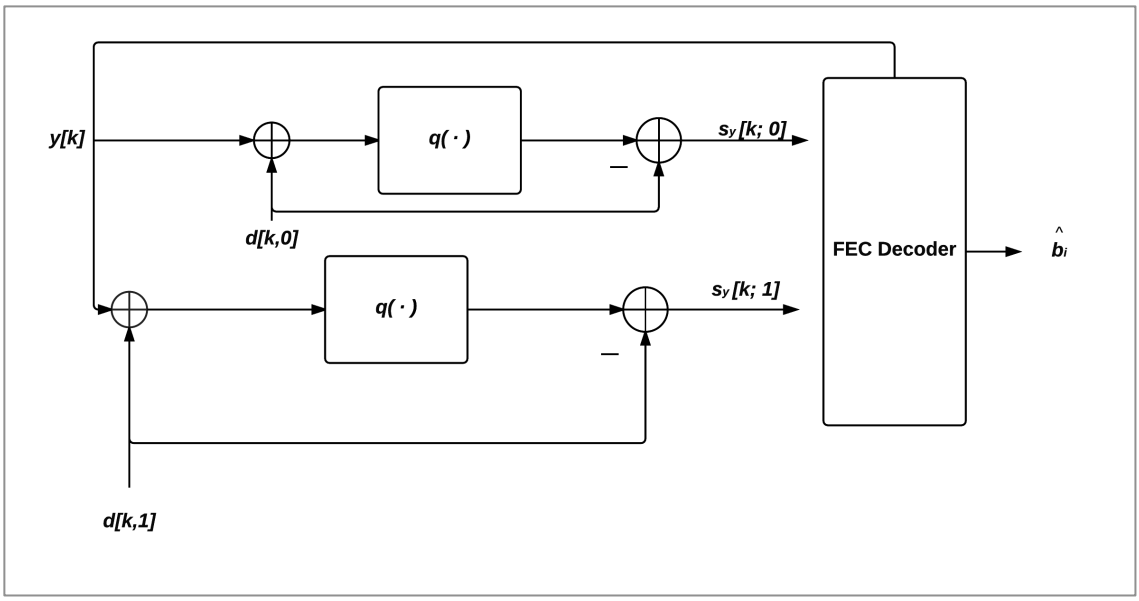

Fig 4 - Dither Modulation Deciphering

To decipher of an image, the formula can therefore be described as

$$\hat{z}_i = \arg\min_{l\in\{0,1\}} \sum_{k=(i-1)L+1}^{iL} (y[k] - s_y[k;l])^2, \qquad i = 1, \ldots, N/L.$$

Fig 5 - Deciphering Formula for Dither Modulation

A simple definition of this is that the host signal y[k] will be used to calculate the minimum distance to the nearest quantizer region. This allows for the removal of the error distortion introduced during the embedding process. The validity of this adjustment greatly depends on the values chosen for the steps in the quantizer. In the embedding stage we move the host signal into the centre of the quantizer region that matches the bit value, when decipher we only need to reconstruct the dithered quantized regions to determine if the current value is in a 1 or 0 region.

In describing the performance improvements of the DM algorithm versus the other algorithms, we must take into account the problem space. The algorithm must be resilient to detection otherwise can be targeted for decoding by another party, must be able to embedded a certain amount of a payload message, must be robust enough to be decoded without loss of

payload and must not distort the image to a point that it is detectable. QIM performs quite well in all of these categories with the exception of being resilient to detection. The inclusion of the dither increases the security of the embedded image as it changes the quantized steps, which makes it hard to detect the values of the quantizers. The overhead in implementing QIM and DM versus the other algorithms are the increased computational and image processing required to implement them. This increase however is deemed acceptable within the scope of this thesis, since the gains of security and robustness offset the additional image processing.

Summary

The purpose of this section was to provide the reader an overview of steganography algorithms and the techniques employed. Each method has both its advantages and disadvantages, which depends on the environment that the algorithm is being used. For the purpose of this thesis, DM steganography was chosen as the best algorithm to use. This was due to both its robustness for handling image modification and its effectiveness for avoiding detections. The drawback of using the algorithm is the computational performance in both embedding the message and deciphering it. As discussed in later chapters this caused a design decision in how the application was created, which resulted in trying to find the most performant way in which images can be manipulated.

The Dither in DM steganography also resulted in the requirement that the seed used to generate the pseudo random dither values be exchanged between users. These dither values are used in the embedding and decoding stages. With the correct dither values the image will be decoded successfully, without it all that is returned will be noise. Transfer of this number between users required consideration when developing the communication protocol.

Twitter API forensics

The purpose of this chapter is to outline the features and functionality that the app will consume from the Twitter API. The API is a REST service that provides consumers with endpoints that can be used to post and receive tweets. The API's consumes and returns JSON data that describes information the user is receiving/ sending.

Authentication

The primary method of Authenticating with the API is by way of OAuth. Developed in 2006 by engineers at twitter, OAuth or Open Authentication provides a mechanism where user can authenticate without revealing their passwords. OAuth works by authenticating a users request via a 3rd party authentication method. Each authentication request passes through an authentication server where both parties can agree to the communication. The first OAuth protocol version was released for comments in 2010 in RFC 5849 and since August 2010 all twitter APIs have used OAuth.

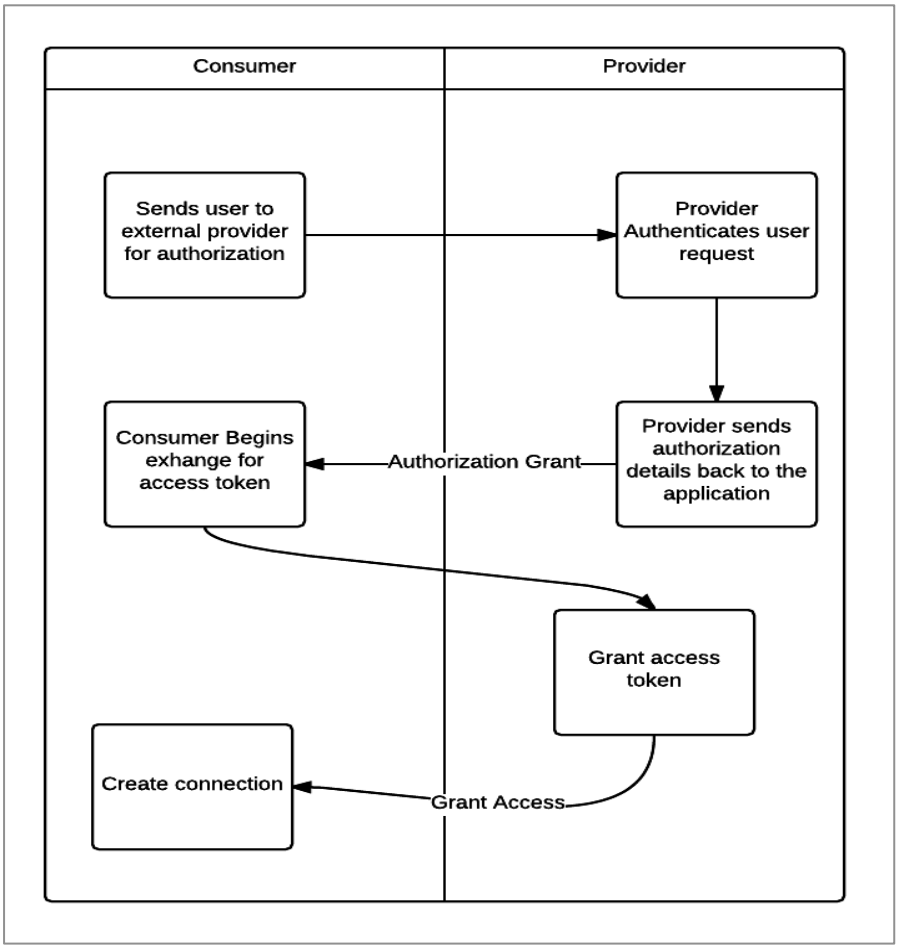

Fig 6 - OAuth Example

Twitter allows for various methods of authenticating applications to use the API. The StegoTweet application requires the use of the ACAccount ability that was first made available in iOS 5. ACAccounts uses OAuth 2.0 authentication to authenticate users. Once the account is setup on the device the iOS framework handles the authentication of the requests to and from the API. The account object is serialised with the request to the API, which twitter uses to authenticate the user. This means that the user must only create their account on the device and the authentication happens behind the scene meaning a more seamless process for both users and developers.

Twitter API search

One of the features the application uses is Twitter's powerful search API. Search has been a feature since version 1.1 of the API. Instead of precision, the search API focuses on relevance of results. This means that the results may not ne complete and some user's tweets may be excluded from the results. It is assumed that this will be sufficient enough to support the communication protocol that is being suggested later in the document however if it is not sufficient, Twitter provides a much more precise API at the cost of increased complexity. The search API uses query string syntax to filter tweets relating to the data that is passed.

An example query is shown below, for the purpose of simplicity OAuth 1 is used with the request.

```
GET /1.1/search/tweets.json?q=%40tinyter28%26%23funny
HTTP/1.1
Authorization:
OAuth
oauth_consumer_key="DC0sePOBbQ8bYdC8r4Smg",oauth_signature_method="HMAC-SHA1",oauth_time
stamp="1458993718",oauth_nonce="3463939284",oauth_version="1.0",oauth_token="22506660-uapqY5U0
Sc32KuH7Hf3D2nOzfl4y5ipJVf7eT9Qu7",oauth_signature="aGpIZKeq%2BzKgUvDwD6Hjzwc8aFU%3D
"
Host:
api.twitter.com
X-Target-URI:
https://api.twitter.com
Connection:
Keep-Alive
```

Fig 7 - Twitter API Search Example

The syntax used in this query is relatively simple. The q query parameter allows the user to specify a search text that is used to query against the twitter API data model. The request contains html escape characters '#' or '%23' for searching for hashtags, '@' or '%40' for search for tweets mentioning this user and finally '&' or '%26' for joining them together. The exhaustive list of possible search functionality is available at [14]

Twitter API post

For uploading tweets, the API provides various methods of sending information to the API to create tweets. The application needs to leverage both the ability to add text-based tweets and the ability to upload images. These images are images that include secret messages that are embedded by the app. Twitter have certain guidelines and restrictions for uploading media. For example the maximum image size that can be attached to an image is 3mb. The image data can be sent to twitter in two formats, raw binary of the image or a base 64 representations. For the purpose of this application single images are uploaded but twitter provides an ability to upload multiple images at a time, which could be leveraged as future work. The recommend method in which twitter asks developers to upload media for status is to first upload the image then link the response with the status update.

For uploading images the following API call is used

```
POST /1.1/media/upload.json HTTP/1.1?media_data=...
Authorization:
OAuth
oauth_consumer_key="DC0sePOBbQ8bYdC8r4Smg",oauth_signature_method="HMAC-SHA1",oauth_time
stamp="1458995334",oauth_nonce="3024352760",oauth_version="1.0",oauth_token="22506660-uapqY5U0
Sc32KuH7Hf3D2nOzfl4y5ipJVf7eT9Qu7",oauth_signature="EqBb7CdXYSrhhXpqeCF%2BpLeEtjU%3D"
Host:
upload.twitter.com
X-Target-URI:
https://upload.twitter.com
Connection:
Keep-Alive
```

Fig 8- Twitter API Post Media Example

Once the media is uploaded, the next call to the API is to upload the status and tie the tweet back to the image. This is achieved by using the following request

```
POST
/1.1/statuses/update.json?status=%40tinyter28%20testing%20o
ut%20the%20status%20uploads%20with%20%23twitterAPI&media_id
=123456 HTTP/1.1
Authorization:
OAuth
oauth_consumer_key="DC0sePOBbQ8bYdC8r4Smg",oauth_signature_method="HMAC-SHA1",oauth_time
stamp="1458996021",oauth_nonce="247681534",oauth_version="1.0",oauth_token="22506660-uapqY5U0S
c32KuH7Hf3D2nOzfl4y5ipJVf7eT9Qu7",oauth_signature="A7HIOL%2BirAbeoUo5h5cx9KUP3zg%3D"
Host:
api.twitter.com
Content-Length:
0
X-Target-URI:
https://api.twitter.com
Connection:
Keep-Alive
```

Fig 9 – Twitter API Status Post

An important aspect of the image upload is the restriction twitter place on daily image uploads. Twitter has validation in place that prevents users from exceeding a daily upload limit, once this limit is reached the user needs to wait a period of time before they can upload more media. These limits are specified in [14] but for the purpose of this application the limits aren't envisaged to be an issue. Twitter also places a restriction on the number of tweets that can be posted in a day, but this limit is far more generous and since the application requires images to be uploaded for conversations it's not likely that this limit would be reached

Swift forensics

Overview

Apple first started developing a replacement for their programming language of choice in 2009. The primary language at the time was Objective C, which had close ties to C and SmallTalk. Although successful in its own merit, Apple wanted to introduce new features of the language and decided that a fresh approach was needed. Swift not only retains the features that made Objective C a successful language but builds upon them. Automatic Reference Counting (ARC) has been improved alongside internal frameworks and Cocoa library upgrades. The compiler for Swift is optimised for development which allows new developer friendly tools to be created such as the Swift playground. The ability to develop in both Objective C and Swift alongside each other allows developers to adjust to the new language while maintaining compatibility existing code.

Swift Features

This section provides some information on the improvements of Swift over Objective C. During the construction of the App, these new features of the language were researched and used within the code.

Type Inference

One of the major changes in Swift is the compiler's ability to infer types. With Objective C the developer must define which type the variable will be, so that compile time type checking can occur. Swifts compiler can infer the type of the object by what object is first passed to the variable.

```
let coordinates3D: (x: Int, y: Int, z: Int) = (2, 3, 1)
let (x, y, z) = coordinates3D
_ = x + 1;
_ = y + 1;
_ = z + 1;
```

Fig 10 Swift Type Inference

This is in contrast to Objective-C, which only fully resolves types at compile time. The ability to resolve types before compilation also allows Swift's compiler to create optimisations by reducing the dynamic look-ups for methods.

Containers

Swift introduces new containers that allow developers to strongly type objects that contain variables. In Objective-C, any container could contain variables of different types unless it was specifically designed to accept a singular type. The swift collections can now detect when objects of an incorrect type are added to a collection before compilation

Generics

Generics have been a core part in most OO languages for a number of years. C# introduced then in version 2.0 and Java introduced them in JSE 5.0. There addition to the iOS framework moves the environment closer to a strongly typed OO framework. An example into how generics work within Swift is shown below which allows for both doubles and int values to be passed, without explicitly stating the type

```
func simpleMax<T: Comparable>(x: T, _ y: T) -> T {
    if x < y {
        return y
    }
    return x
}
```

Fig 11 - Swift Generic Types

Enumerations

Enumerations improvements are one of the additions Swift brings to iOS developers. Enumerations can represent values within code without having to share values around your codebase. Swift goes one step further and allows for Enumerations to contain values within the Enumeration. This allows for developers to assign values to the Enumeration. An example is shown below of how useful this feature can be.

```
public enum SearchResultType : String {
    case Mixed = "The results are mixed"
    case Recent  = "Only recent results"
    case Popular = "The most popular results"

    var description: String {
        return self.rawValue
    }

}
```

Fig 12 - Swift Enumerations

Extensions

The final new feature of Swift that will be discussed is the extensions Keyword. This feature allows developers to mutate existing types in a multitude of ways to provide new functionality. This feature doesn't only apply to code the developer wrote, it applies to framework level classes as well. This feature allows developers to customize how framework classes function, which allows a flexibility not seen in most other languages.

```
private extension NSString {

    var asTwitterDate: NSDate? {
        get {
            let dateFormatter = NSDateFormatter()
            dateFormatter.locale = NSLocale(localeIdentifier: "en_US")
            dateFormatter.dateFormat = "EEE MMM dd HH:mm:ss Z yyyy"
            return dateFormatter.dateFromString(self as String)
        }
    }
}
```

Fig 13 - Swift Extensions

Apple Core Image and Apple Metal Framework

Core Image (CI) is Apple's framework for image processing and analysis. It provides almost real time analysis and processing of images. It processes images from a combination of Apple's Core images, Core Image, I/O frameworks and a combination of either GPU or CPU rendering. Core Image creates an API that allows the developer to leverage the full power of the GPU without having in depth knowledge of GPU Programming. Although it abstracts that knowledge, it does not provide the same performance that can be leverage with accessing the GPU. It also helps with multithreading on the device, abstracting away any knowledge of Apple's Grand Central Dispatch to provide highly performant multi-threading.

Metal is a framework that Apple developed to allow developers to fully harness the power of the GPU on certain devices. Metal was debuted on iOS 8 with devices that use the A7 processor and above. Metal provides a highly performant and sophisticated graphic rendering and computational power by providing low-overhead access to the GPU. When compared to direct OpenGL, it has been shown that it can provide better performance due to reasons such as pre-computed shaders, explicit synchronization and shared memory between CPU and GPU and as mentioned low level access to drivers on the device. Although OpenGL is more widely support on other mobile devices, Metal pushes the limits of what's possible on the device by targeting specific processors and drivers that are only available on iOS (although support has been announced for OSX). Shaders are programmed using a language that is based on C++11 and the framework allows for explicit setup of thread that allow for highly parallel processing capabilities.

The two frameworks are different in their goals, Core Image is Apple's general approach to image processing that has a high number of compatible devices and is very developer friendly due to abstraction of low level image processing methods. It's designed primarily to be used on the CPU of the machine, which causes a reduction in its performance. Metal on the other hand is its polar opposite. It targets specific devices and knowledge of GPU programming is required before a developer can fully grasp its full functionality.

Communication Protocol

Overview

The communication protocol is the method in which users of the application can communicate between each other. Since the application uses Twitter as a means of communication, the protocol is heavily based on the methods available in Twitter. These include direct mentions (using the @ symbol e.g. @user_name), hash-tags or trends (e.g. #funny, #you_need_to_see_this) and the ability of uploading media with text. The difficulty with creating this protocol is trying to maintain the secretive concept of the app in a public forum while also providing enough functionality to have conversations. This requires the functionality to create threads of messages, the ability response to threads and to look over existing messages.

The communication protocol must be effective for the App to be a success; therefore a good method in evaluating the effectiveness is to compare the values of security, functionality and the ease of use for the users. Before having a conversation, it is necessary for two users to exchange a secure passphrase. It allows for users to ensure that only the interested parties are able to decipher messages. The addition of this exchange to the communication protocol raises some interesting problems such as how to make the exchange look like normal tweets and how does it prevent unwanted parties joining the conversation. Some additional areas of the communication protocol are the inclusion of data relating to the protocol. As the protocol progresses, there could be a need to include extra information such as multi party threads, different steganography methods or different authentication methods. These are discussed and possible solutions are suggested.

Creating a conversation

Creating a conversation requires the user to create a tweet that mentions the other user that they want to have a conversation with. There are some required fields that need to be added to the tweet for the conversation to take place. These fields each contain information that the app will use to sort and combine tweets into threads. The mention also has the benefit of causing a notification from twitter to the other user, if they are setup to receive these notifications Firstly, the hashtag that the first tweet is sent with acts a message key. Each tweet that is associated with this hashtag from the mentioned user will be treated as a reply to first tweet, thus creating a thread of messages. Without the hashtag, any tweets that have a mention and an image with a different tag will result in a new thread being created. These hashtags are unique to the thread and therefore create a 1 to 1 relationship.

The first tweet in creating a conversation must include a public key that the other parties can use to decipher messages from this user. The key exchange is discussed at a created detail in a later section. Without this key, images that have embedded data cannot be deciphered. If the first tweet does not contain a key or contains content that when interpreted does not conform to a key, the tweet should be ignored and should not create a conversation.

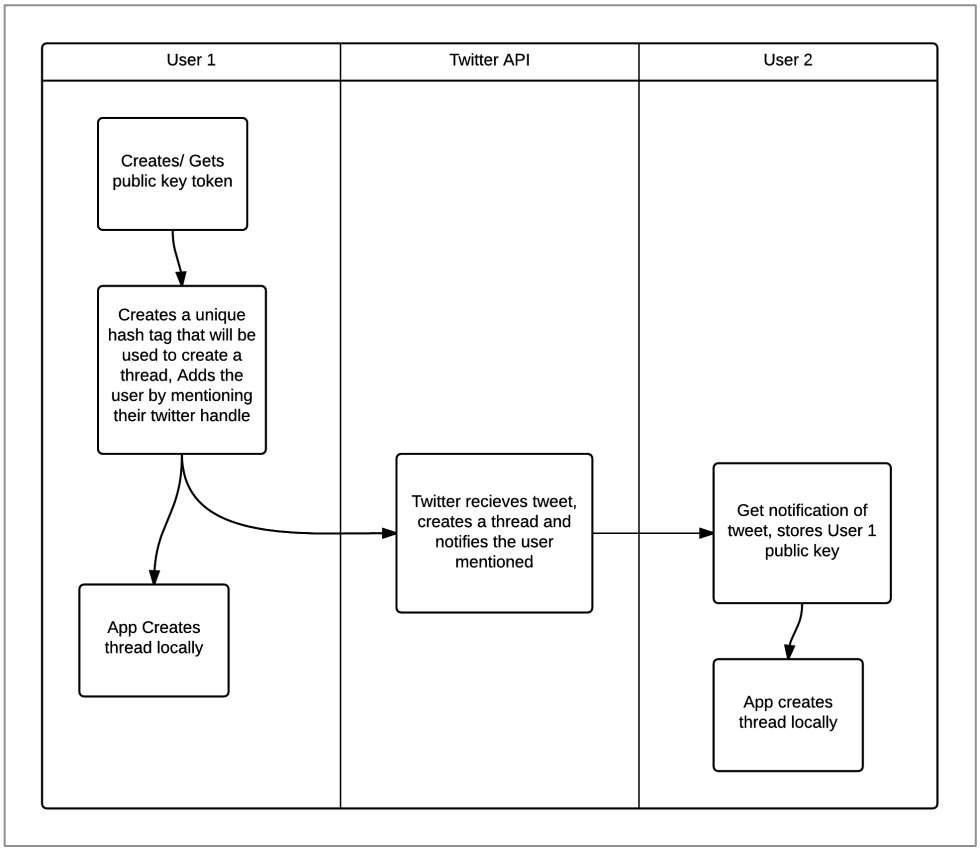

Fig 14 - Creating a Conversation

Any mention of the current user that is tweeted from a user that they are currently not involved in a conversation with will create a new thread within the app. The creation of a new conversation should record the key that is sent with the first tweet since this will be used to decode information later.

Continuing a conversation

The hashtag is key to continuing the conversation. The user must include the hashtag on each tweet that is part of the thread. The secret information however is held within the image. The user that is wishing to continue the conversation must also exchange their public key so that other parties can decipher their message. The user does this with the first message they send to the thread. Each time a user wants to respond to a conversation, an image must be selected for the message to be hidden in.

Within the app, there will be functionality that displays to users a list of conversations that they are part of. This allows the user to quickly respond to messages that have been exchanged before. The ability to reply to a conversation will be built into this section, which will include the twitter specific methods of communications such as mentions and hashtags.

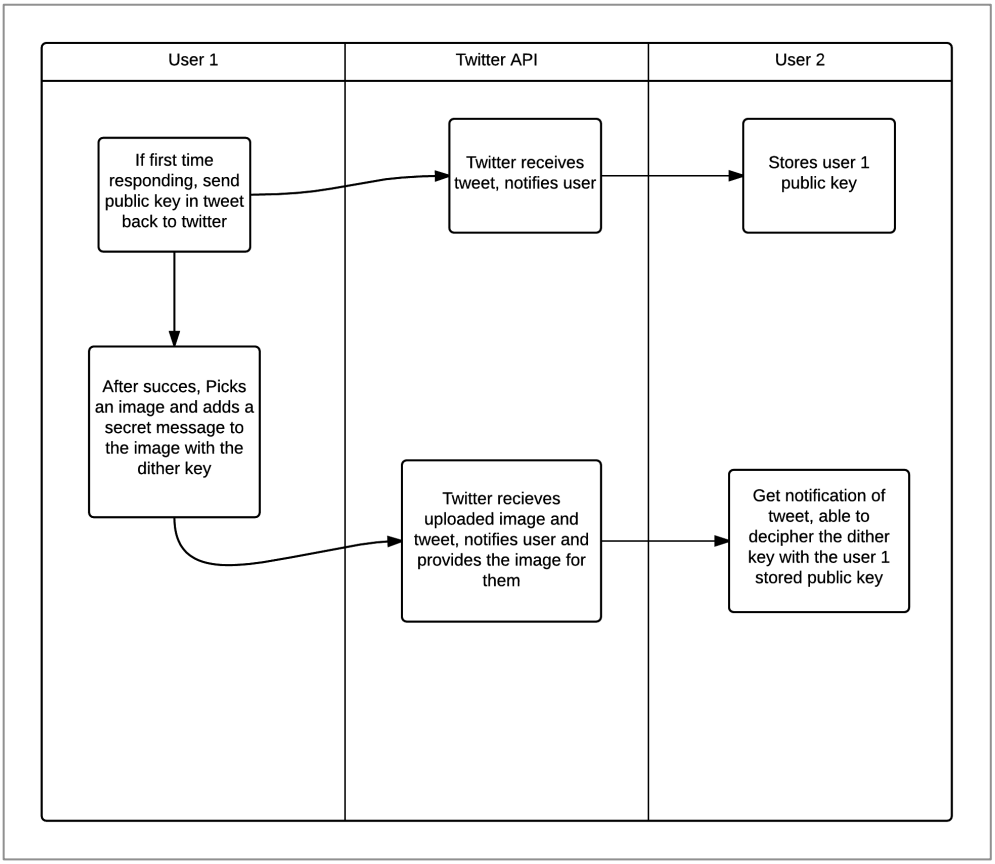

Fig 15 - Continuing a Conversation

When there is a reply to a conversation that the user is part of, they are shown an icon identifying that there are new replies. This allows the user to easily see new messages within threads. The user will have no immediate functionality to stop a conversation apart from not replying to the thread. Functionality can be built into the app that will use the Twitter API to block specific users, but that is not covered within the protocol.

Public Key exchange

As mentioned in the creating and continuing conversation sections, the users of the app must exchange a public key that the other parties can use to decipher the message. Public key encryption, which is also known as asymmetric key encryption, is a form of cryptographical algorithms that requires the use of two keys. The two keys are known as the public key and the private key.

The suggested encryption algorithm used to encrypt the messages and generate the keys is the Rivest-Shamir-Adleman Algorithm (a.k.a RSA). RSA is an industry acceptable method of creating a cryptosystem that allows for secure public private key encryption across network traffic. First described in 1977 by the authors Ron Rivest, Adi Shamir and Leonard Adleman it quickly became the foundation for a many network based communication exchanges such as SSH, OpenPGP and SSL/TLS. The reason for RSA being so popular was its simple design, which derived from the computation of large prime numbers. It functions by factoring large integers that are the product of two large numbers. The computation complexity in trying to calculate the original prime numbers is considered infeasible even by modern day computation power so long as they are sufficiently big.

A simple example of the RSA algorithm is the exchange of information between two people named Alice and Bob. Alice selects two prime numbers p = 17, q = 19. She then uses these numbers to calculate the modulus $n = p \; x \; q$ which results in 306. The totient of n is $\phi(n)=(p-1)x(q-1)=96$. She then picks her public key e, which is 7. Using the Extended Euclidean Algorithm she calculates her private key to be 55. Bob tries to send Alice an encrypted message so firsts obtains her public key (n, e). For example if Bob wanted to send the number 9 encrypted into the cipher text C the formula would be $M^e \; mod \; n \; = \; 9^7 mod \; 306 \; = \; 189 \; = \; C$. When Alice receives Bob's message she decrypts it by using her private key (d, n) as follows $C^d \; mod \; n \; = \; 189^{55} mod \; 306 \; = \; 9$. This is a

simple example of how the RSA algorithm is used to encrypt messages but in practice the numbers select from p and q are much larger and are usually generated by the Rabin-Miller primality test algorithm.

For the purpose of this communication protocol, RSA is used to encrypt the dither random number seed that is used for the embedding of data within the image. The dither seed is necessary for the deciphering information in the image because it generates the sequence of random numbers that introduce noise into the image that represents the secret message. If a third party tried to decipher the message without the dither seed all that would be returned would be noise, since the dither randomly changes the amount of noise that is added when embedding the data.

Additional security can be added to the protocol to help obfuscate the public key. The purpose of this is to make the tweet look like a normal tweet for any third party. For example if two users were having a conversation it wouldn't look like an organic conversation. This would arouse suspicion and would invite unwanted attention to the users. A simple mechanism that could be used to prevent this would be to introduce a simple cipher that would change characters of the public key into emoticons. Since the use of emoticons is so widespread in social media, this would mitigate the amount of attention tweets that contain public keys attract. This does not provide a high level of security but rather provides security by obfuscation.

Alongside the obfuscation, other means of security can be introduced. For example, there could be a requirement for users to change their public key after a given amount of time. The purpose of this key timeout is to prevent the ability of a third party gaining enough data to try and decipher what the key that is being used is.

Evolution of the protocol

For a protocol to be successful it must be able to evolve and adopt new features as it matures. The protocol allows for this by providing areas that metadata about the protocol can be added to. For example if the protocol was to allow for multiple encryption methods to be used due to a number of factors, it must be flexible enough to provide this functionality. Some of the additional functionality will happen at different levels. For the given example about the form of encryption that is being used, it must be able to be able to be understood by unauthenticated users while information regarding the steganography methods should only be provided to fully authenticated users.

For the first level of data regarding the encryption protocol used, the tweet content that is sent with the image can be leveraged upon. This data is not encrypted with the encryption protocol so can therefore be used to send information regarding the encryption method used. As mentioned earlier, the tweet content that contains the public key can be obfuscated to look like a normal tweet so all that is required to understand the security method is to know how to de-obfuscate this information. This approach does have its limitations and concerns. The maximum length of tweet content is set to 140 characters so methods will be needed to ensure that any additional unsecured information fits alongside any required information, such as public keys. A concern of this approach is leaving clues regarding the encryption protocols that are being used since the data is unsecured. This approach requires that the obfuscation step be complex enough to provide enough some resilience to this. In general this information should not be enough to break the encryption given the assumption that only strong forms of encryption such as RSA is employed.

The second level is the secured information, which needs to be contained behind some form of encryption. An example of data that may be used at this level is information regarding the steganography algorithm that is used with the image. For example if the protocol is expanded to include multiple steganography algorithms such as Bit Plane or QIM without dithers, the protocol must provide functionality around this. One possible solution to this is to implement a data structure instead of the embedded message. This data structure can be embedded into the image using DM first. The data structure can then point to the steganography algorithm that is used to decipher the message. This allows for the actual

message to be embedded with different algorithms and can contain sensitive information that should not be easily available.

Another evolution of the protocol is the ability to have multiple users in a conversation, which is also known as a group conversation. Group conversations require the exchange of each member's public key so that all members in the group can decipher all the images. Within the protocol this is capable, with the requirement that any user that wishes to join in the conversation must send their public key with the first tweet in response to the hash tag thread. Replying to the hashtag thread and mentioning the other users in the conversation will cause the users to be alerted of the new group member, which means they can record the new public key.

User Experience and Interface Design

Overview

Modern mobile applications face a tough competition to gain users. The number of apps that focus on the same or similar style of functionality of an app is high and therefore User experience (UX) and User Interface (UI) is key in winning users. For chat style applications, the battle is to gain enough users so that the users of the app drive adoption. It is therefore key that any chat style application presents a UI that feels familiar and simple while providing a UX that s engaging and easy to understand.

To compete, these Apps have taken inspiration from each other to provide a style of chat that is homogenous across all the applications. Even Apps that provide novel means of conversation such as SnapChat, where user converse by sending temporary images back and forth, follow the same guidelines or adopt some of them. These guidelines consist of a conversation view where the user can see the conversations they are involved in, a chat view, where they can see past messages and a view that allows them to create new conversations. These views differ across each application in the way they present themselves to the user (UI) and how the user can interact with them (UX) and it's these differences that play a key part in whether the App will be a success or not.

The following section outlines these views and provides information regarding their UX and UI design. These views have taken inspiration from some of the most popular chatting style applications while trying to keep the familiarity of the iOS framework design. The reason for this is so that the users feel the chat functionality is familiar and provides the same or similar functionality to its competitors.

Application views

Conversation Views

The first view the user is shown is the conversation view. This view aims to display all current conversations that user is involved with, alongside the ability to create new conversations.

Fig 16 - Conversation Screen

Each conversation row involves the twitter avatar of the user and the hashtag of the conversation. The user can see updates to conversations that they are involved in, denoted by the 1 in the diagrams. This number is updated when a tweet is available on the hash tag thread that they have not read yet. The view will scroll if the number of rows exceeds the device screen dimensions. The user can search using the magnifying glass that will search against the hash tags.

Clicking on a row will bring the user to the chat view that will load the messages within the chat. The user can delete a chat swiping right on the row, similar to normal iOS option rows. To create a new conversation the user needs to click on the '+' icon that will bring the user to the new chat screen. The reason to have this on the top bar is to maximise the space available for the conversation rows, while keeping the new conversation button visible at all times.

Chat View

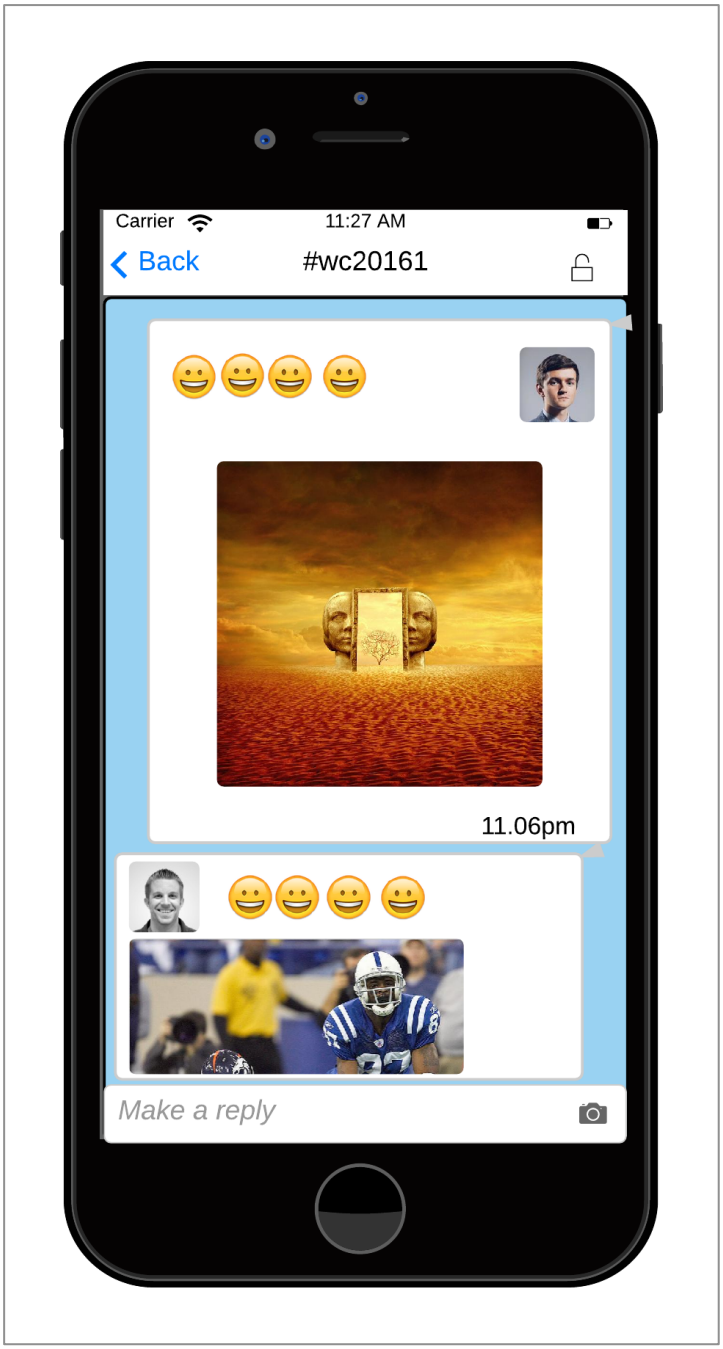

Fig 17 Locked conversation Screen

Locked mode

The chat view contains two modes, a locked mode where the tweets are shown in their unprocessed and an unlocked mode where the tweets are processed and their secret messages are shown like a typical chat conversations.

The purpose of the locked mode is two fold. It acts as a placeholder where pin/password authentication can be used in future before deciphering the message and provides an additional level of authentication.

Its primary use would be to show the raw tweets from twitter and shows the transformation between the raw tweets and the deciphered images. In a production application that doesn't need a pin/password authentication, this view is probably unnecessary.

Messages are sorted by date descending and are each stamped with a timestamp of when they were authored. The messages also include the twitter avatar for the user as well as the embedded image and the tweet content (their dither key encrypted) From this view the user can navigate back to the conversation view or author a new tweet. By clicking on the make a reply button they are shown the screen in fig 19. The user can unlock the tweets by simply clicking on the unlock button on the top right of the screen.

Unlocked mode

The unlocked mode allows for the user so see the tweets as a normal thread of chats with replies. It functions the same as the locked mode where messages are arranged by date descending and are each time stamped. The primary difference in the unlocked mode is that each tweet has the image and tweet content replaced with the secret message content.

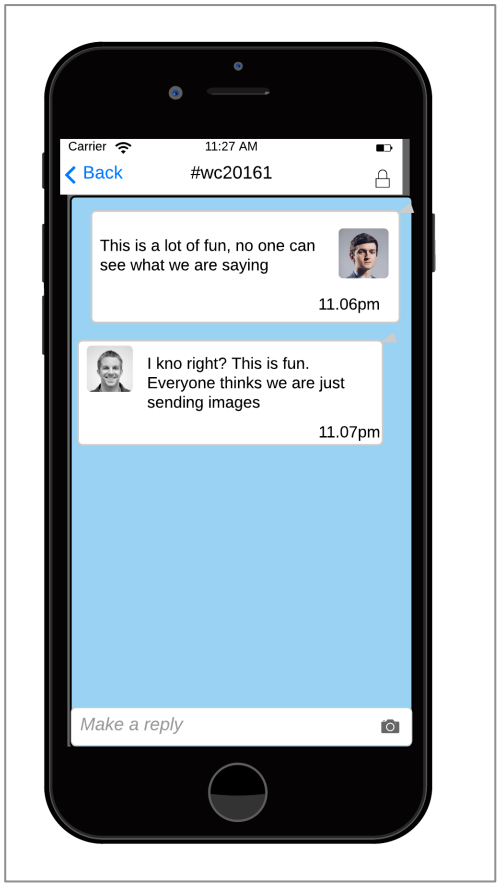

Fig 18 - Unlocked Conversation screen

The user can reply in this mode but it is still necessary to include an image with the message as this is what will be sent to twitter. While in unlocked mode even the new message that is post should be shown as just the secret message. It is envisaged that the unlocked mode would be the only mode provided if the App was made for production.

The user can attach an image by touching the camera icon. This will load the native framework image selector that will show available images on the device. The requirement of having an image on the device is an area of improvement, where instead of user supplied images, random images from the web can be used. When the user attempts to send the image (i.e. presses the return button) a pop up will be show that prompts the user for a dither number. This number is send with the tweet so that the other user can correctly decipher its content. Some improvements to this are that the App randomly picks a sufficiently large dither value automatically, which provides the user with a better experience since the chat becomes more fluid.

Fig 19 - New reply screen

New Conversation View

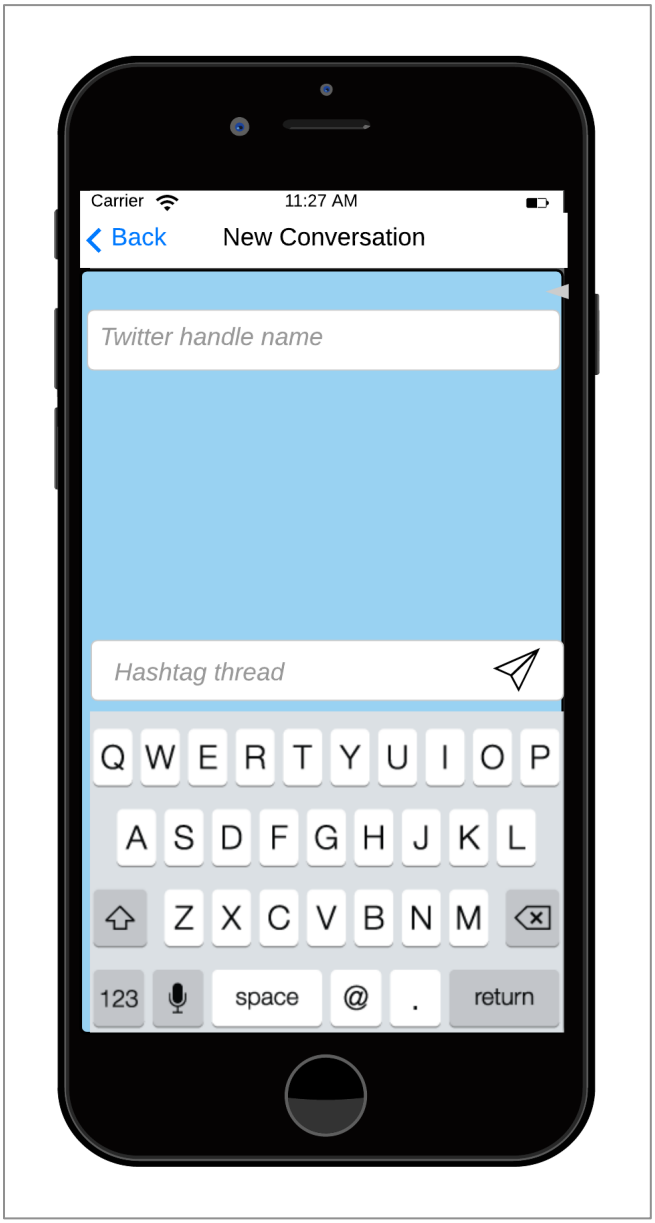

Fig 20 - New Conversation Screen

The new conversation view is available from the conversation screen. It allows users to create new conversations with users on twitter. To do this, the user must enter in the twitter handle for the user and provide a hashtag that will be used to index the conversation.

The new conversation app will begin the communication protocol by sending the user's public key as the first tweet. This tweet will not be made aware to the sending user as it serves no information to them. It will however be stored on the device to be used to decipher images.

On receiving a new conversation, the receiving user will see a new conversation row appear in their conversation row. If the user response in the app, the App will first complete the protocol by sending back the current user’s public key and then sending the response. This allows for the communication protocol to be completed before conversation begins. The sending of the private keys will happen without the user being aware of it in the application.

Conclusion

This section highlighted some of the UI of the App and how users interact with these UI's. There is some UI's that were left out since they are not key to the user experience of the App (such as settings). There are some interactions that are not covered by this section since they happen behind the scenes. The notification method of the App is reliant on the twitter data model, there is a requirement that the twitter API is polled at a frequency so that it can receive and process new updates. This allows the App to seem as if it is a normal chat application while relying on the Twitter API for its data storage.

Some of the suggested improvements listed in the previous sections show that the UI design can be an iterative process, where designs can influence the overall function of the App and the communication protocol that powers it. These improvements show that there is a push-pull model in play where the need to improve UI comes at a cost of security. For example the suggestion of the App providing random dithers creates a weakness in the protocol, wherein if the App is reversed engineered the process for creating these dither is found and the protocol becomes less secure. This highlights only one area that such a weakness can be exploited but shows that design and both positively impact usability while jeopardizing correctness and that a trade off between the two must be reached.

Implementing the Steganography algorithm

Overview

Implementation of the algorithm in iOS framework required understanding not only how the algorithm behaves but also how to translate the steps in the algorithm into steps that could be programmed. In this section, two implementations of the algorithm are considered one at the CPU level using the iOS Core Image framework and one at the GPU level leveraging on the Metal Framework. There are benefits and drawbacks in each approach from the perspective of performance based on speed, ease of use and scalability. In general however, it has been shown that the move toward GPU versus CPU is much more performant but at the cost of ease of use due the complexity of GPU programming.

The two suggested implementations are not the only available implementation methods. There is the possibility of using raw OpenGL shaders amongst others.

Defining the algorithm

Regardless of the framework the steps required to implement DM steganography are the same. These steps are based on the learning's from the formal specification in the steganography section. They provide a general overview of what each implementation aims to do. The specific implementation will differ depending on the framework. These differences are discussed as contrasted in the implementation sections.

Embedding

1. Translate an Identity string e.g. "StegoTweet" the byte length of the message and the message string as bytes into an array of byte values.
2. Seed a random number generator that will be used to generate the dither values
3. For each byte in the array
   3.1. Break the byte into its bit value
      3.1.1. For each bit in the byte value and dither in the random number array
      3.1.2. Break the range 0-255 – the current dither value into a small parts of alternation regions being Q1 and Q0.

3.1.2.1. If the bit is 1 move the bit to the centre of the nearest Q1 region

3.1.2.2. If the bit is 0 move the pixel value towards to the centre of the nearest Q0 region

4. Repeat 4 if necessary or required

Deciphering

1. Read all the pixels from the image

2. With the dither seed from the communication protocol, reconstruct the random number generator with the seed

3. Recreate the quantizers by using the step value used in the embedding process

4. For each pixel

4.1. Use the random dither to offset the pixel value

4.2. Find whether the offset pixel value is in a Q1 region or a Q0 region

4.2.1. If in Q1, read the bit as 1

4.2.2. Else read as 0

5. Reconstruct the bits read from the image back into bytes

6. Convert the bytes back to their string equivalent

Implementation in Metal

The purpose of Metal is to allow developers' access to low level device drivers to provide high performance GPU processing. The supported devices for Metal are any devices with a processor chip greater than Apple's A7 chip and the majority of the framework classes used in the implementation of the algorithm require iOS 9 (some have support from iOS 8 but not all). This significantly reduces the devices that can run an App that relies on Metal and also prevents normal debugging on the iOS simulator.

To use Metal, a developer is required to create a Shader that will be run across the texture that the developer provides. The shader is programmed using a language that is based on C++11. The first stage in getting a device to run a shader is to get a reference to the Metal framework device (1). The next stage is to tell the device to compile the Shader, which will execute on the device (2). Since commands are executed on the GPU rather than the CPU, the developer must encode commands that are used within the shader, such as textures, buffers and samplers (3). Once the commands are encoded they are sent to the GPU for processing at the next available cycle (4).

```
// (1) - Get Reference to the device
let device: MTLDevice = MTLCreateSystemDefaultDevice()!
// Create a default library
let defaultLibrary: MTLLibrary = device.newDefaultLibrary()!

// (2) reference to the function and create the pipeline state
let kernelFunction: MTLFunction = defaultLibrary.newFunctionWithName("stego_embded_image")!
let pipelineState: MTLComputePipelineState =
    try! device.newComputePipelineStateWithFunction(kernelFunction)

// (3) Create a command queue, a command buffer that will pass information to the shader
let commandQueue: MTLCommandQueue = device.newCommandQueue()
let commandBuffer: MTLCommandBuffer = commandQueue.commandBuffer()

//Define the command encoder
let commandEncoder: MTLComputeCommandEncoder = commandBuffer.computeCommandEncoder()
commandEncoder.setComputePipelineState(pipelineState)

//..... Code that creates the information passed to the metal shader

// (4) End the encoding and pass the shader on for processing
commandEncoder.endEncoding()
commandBuffer.commit();
```

Fig 21 - Getting a Metal Device

The code for the Shader is relatively simple, since this kernel is run over every pixel in the image, the code must be efficient. The approach is to pass in all information to the shader and allow the shader to only do the pixel change by deciding on what information it is passed for this pixel.

```
kernel void stego_embded_image( texture2d<half, access::sample> source_texture [[texture(0)]],
                                texture2d<half, access::write> dest_texture [[texture(1)]],
                                constant int &message_byte [[buffer(0)]],
                                constant int &dither [[buffer(1)]],
                                uint2 gid [[thread_position_in_grid]])

{
    const float delta = 15;
    half4 colorAtPixel = source_texture.read(gid);
    float ditheredDelta = delta - dither;

    float quantum = ditheredDelta;
    bool cover = message_byte == 1  ? true : false;

    half r = quantize(colorAtPixel.r, quantum, cover);
    half b = quantize(colorAtPixel.b, quantum, cover);
    half g = quantize(colorAtPixel.g, quantum, cover);
    half a = colorAtPixel.a;
    half4 newColour = half4(r, g, b, a);
    return dest_texture.write(newColour, gid);
}
```

Fig 22 Metal Implementation

For deciphering, the source image texture is passed to a deciphering shader that will extract information from each pixel. Alongside this, the dither seed buffer used in the image creation is reconstructed and passed into the shader so that each pixel can be correctly quantized. This extracted information is passed back to the CPU where the message information can be parsed and displayed on screen.

Implementation in Core Graphics

Unlike Metal, the implementation in core graphics takes place purely on the CPU. This allows the developer to mutate the pixels directly. The first stage in the implementation it create objects that will contain the information that is read from the source image. This data structure allows the developer to quickly move over the pixels in the image on a one by one basis. The ability to take control allows for the developer to optimise the use of the number of loops when processing image pixels with the trade off of taking up CPU cycles.

The next step is breaking each pixel into its red, green, blue and alpha components. Since there are four possible channels for information to be stored on, each channel is used for storing information in. Each channel will store a bit value by quantizing its value towards the centre of its nearest quantization point. This is the maximum storage capacity but also causes the most distortion in the image. This is alleviated by adjusting the amount that is stored per pixel and spread the data over more that one pixel but in general the approach is the same. These improvements and more are referenced in the evaluations section.

The deciphering step is started by loading the pixels into the same data source that was used for embedding. The pixels are then processed for their Identifier text i.e. “StegoTweet”, the byte length of the string and its stored data. If the identity text is not found, the developer can opt out of processing the remaining pixels due to the additional control at the CPU level. The byte length also allows the developer to stop processing pixels once all the stored data is read.

```
srand(UInt32(seed));
let buf = createSecretMessageBytes(width * height);
for byte in buf {
    let strBits = String(UInt(byte), radix: 2);
    let bits = strBits.characters.map { Int(String($0)) }
    for bit in bits {

        let index = y * width + x

        let dither = rand() % (maxNumber + 1 - minNumber) + minNumber
        let ditheredRange = Int(delta) - Int(dither)

        var pixel = rgba.pixels[index]
        let cover = bit == 1 ? true : false

        //if bit is 1 move the pixel toward the center of the nearest postive interval, 0
        // move it away
        pixel.red = UInt8(quantize(Int(pixel.red), quantum: ditheredRange, cover: cover))
        pixel.green = UInt8(quantize(Int(pixel.green), quantum: ditheredRange, cover: cover))
        pixel.blue = UInt8(quantize(Int(pixel.blue), quantum: ditheredRange, cover: cover))

        if( (x + 1) != width) {
            x += 1
        }
        else {
            x = 0;
            y += 1
        }

    }
}
let finalImage =  rgba.toUIImage()!
```

Fig 23 Core Image Implementation

Evaluations

Communication protocol

Overview

The purpose of this section is to evaluate the communication protocol and to provide areas of future work. Two areas that the protocol will be evaluated against is its effectiveness in being a communication protocol and the security that the protocol provides. These areas are used to provide an impartial view of the protocol, how well it currently functions and highlights the gaps where improvements can be made.

As discussed earlier, the communication is heavily based on Twitter and Twitter's current version of the API therefore certain assumptions are made with regards to the functionality it provides. For example, if the API were to revoke the ability to upload images, the communication protocol would suffer since this ability is at its core. It is therefore important to identify this as a possible issue however this section is aimed at discussing the communication protocol given the assumption that the Twitter API continues to support features that the protocol employs.

Effectiveness

To evaluate effectiveness of the communication protocol certain questions must be asked. These questions apply to the effectiveness of any communication protocol and should help highlight both gaps in the protocol and areas in which the protocol performs well.

Does the protocol have a well-defined and comprehensive flow of communication ?

As discussed in the protocol section, the protocol contains a clear method of creating a conversation and a clear method in continuing it. The flow of communication back and forth between the users involved in the conversations is clear, where each user replies to a tweet with an encrypted key and an image. It also caters for incorrect messages or noise being retrieve due to the use of the filters that are applied to the Twitter API. The ability to

end a conversation however is not well defined and as such there is no specified end state for the flow. This is intentional in the way that normal chat applications do not provide a way to end a conversation, apart from heavy-handed measures such as blocking all communication from the user in future.

Can the protocol recover from a loss of information or an interruption of information?

The proposed protocol is not fully tolerant to a loss in messaging. The storage of the complete message information such as a properly formed tweet relies on the Twitter API. It is assumed that the API is reliable but the protocol relies on the initial tweet that contains the user's public key and it is a critical piece of information. The loss of this tweet for a user means that the current thread of images that the other user has sent can no longer be deciphered. This again relies on the Twitter API to provide the data integrity that is needed so that tweets that are sent by each user and stored and can be accessible at any given time. There is no method specified for retrying a message that was not received by Twitter apart from the user resending the entire message.

Can the protocol evolve to match required changes?

The protocol has shown that it can evolve to provide additional functionality with regards to the tweet content and metadata. An important ability in any protocol is its ability to react to new requirements for example if there is new requirement such as the ability to change steganography algorithms or different public key encryption method was added the protocol could be improved to include these. This ability is not endless as there are limitations on the amount of data that can be contained within one tweet. Another area in which the communication protocol can evolve is the ability to include more that one user in a conversations. This functionality is possible but is not catered for in the initial iteration.

Security

In evaluating the security of the protocol, we must look at the protocol from an impartial view and try to find weaknesses that other users could exploit to decipher information. With any form of secure communication, the ability to identify its weaknesses and improve upon them is key to the future of the protocol.

The use of Public Key encryption is well known and used widely on the web. Since its inception, public key encryption has been used in communication protocols such as SLL, SSH and even to encrypt PDF files across the wire. Its secureness does not need to be proven, but how the protocol implements it does. The first level of security is the obfuscation of the public key on the first tweet. This allows the tweet to appear normal on first glance but is easily susceptible to deciphering. This allows people then to detect tweets that are being used for the protocol and identify the users involved. The key itself is secure by way of the algorithm used (RSA given a large enough key). The transfer of information is susceptible for eavesdropping but since the data of the message is embedded in the image there is not much that can be deciphered from the tweet apart from the Public Key, the encrypted dither and the image. This is however assuming that the current computation power is taken into effect of modern computers with regards to factoring the RSA algorithm [16] [17].

A security concern is that a person could take hold of the device that the user is using StegoTweet from and therefore have access to conversations they shouldn't have. This can be alleviated by implementing a form of device authentication but is outside the scope of the communication protocol. Also, the ability for someone to trap the requests from a device and pretend to act like the Twitter API is a security concern since there is no current way to authenticate that the responses are from Twitter and not from a third party impersonating it. This is mitigated by the use of OAuth but it remains a possibility so must be treated as a security concern.

Conclusion

This section aimed to critically analyse the communication protocol from both a security and effectiveness point of view. It showed that although the communication protocol is secure with the inclusion of Public Key encryption it does have some security flaws that

could be exploited. As for the protocols effectiveness, it provides a comprehensive flow of communication that can be expanded upon and grow into a more complete protocol but does have some drawbacks.

Steganography

Dither modulation steganography is a key component of the App and its implementation is key to the Apps usability and performance. The two suggested implementations for the embedding and decipher stages of the steganography algorithm provide the same functionality of embedding data into an image but provide different levels of performance and ease of use for a developer. This section aims to compare and contrast the algorithms use within the project and its performance. The performance is based on the different implementations of the algorithm in both the Metal framework and Apple's Core Image framework.

Alongside evaluating the performance of the algorithm in the two frameworks, we must also look at how secure the algorithm is from a steganalysis point of view. The communication protocol aims to provide a secure way to create and continue a conversation therefore the images cannot be a weak link since they contain the secret information. It is also important to evaluate the reliability of the embedding and deciphering of information from the images due to the fact that a loss of information within the image can cause a point of weakness in the App.

Performance

When analysing the performance of the two implementations there are certain factors that affect the outcome of the test. The first factor is the capacity of how much information can be stored within an image and how fast can it be embedded. This is not taking into account the distortion amount since if every pixel changed in an image it would become obvious and easily detectable. This allows for the test to be fair in such a way that if the image was pushed to the maximum it could contain, then the quickest framework would be the best decision. The next factor is to do with possible image sizes and how fast each implementation can handle images of different sizes and different amount of images in succession. As the Twitter API restricts the media

size, the majority of the images will be within a similar range of sizes but to evaluate performance additional image sizes must be considered. The final factor in performance is to look at the amount of data that is to be encoded and how that affects each of the implementations.

The chart below plots the performance of each implementation when pushed it maximum storage capability. The implementations are tested against an image of small size (450 x 600) medium sized (1024 × 1024) and large size (2048 × 1366) with each embedding 500 chars of text over all available pixels.

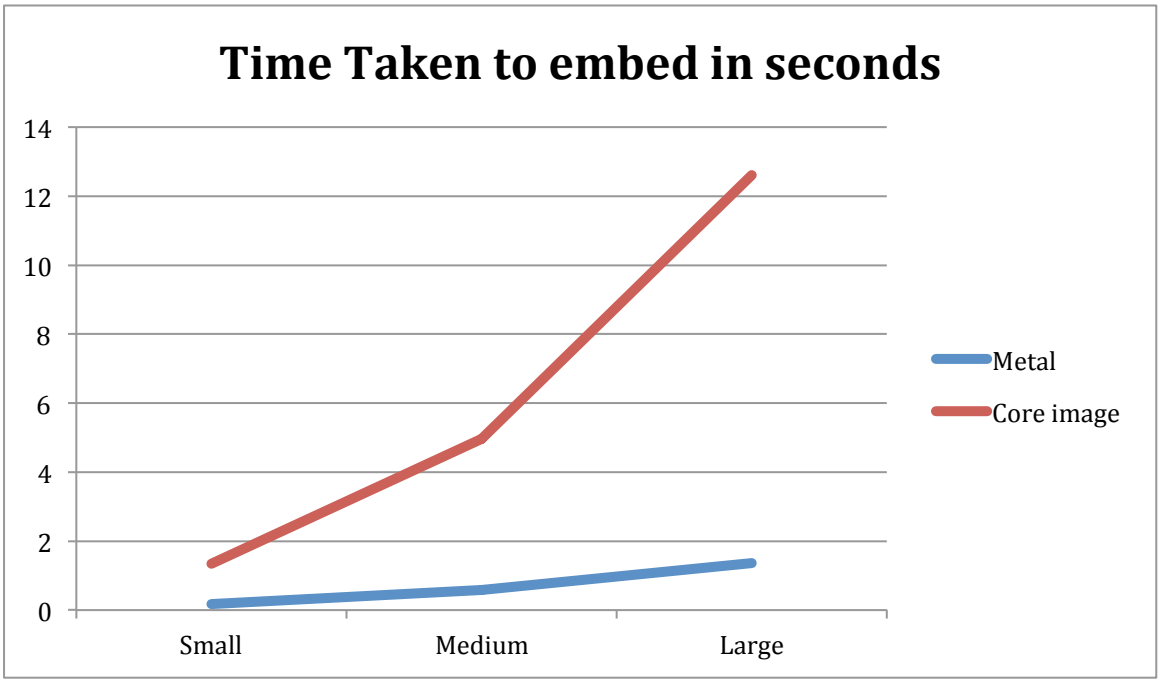

Fig 24 Performance of Core Graphics V Metal

From the results, its clear to see that Metal provides the best performance at all levels of image sizes and even shows that as image sizes grows, the length of time required remains low. The number of characters encoded did not cause much difference due to the test embedding a character in each pixel. It was possible to tune Core Image to provide results that are similar to Metal by focusing on just embedding a message once on a small image. This however is not a sustainable model since it directly affects the possibility of introducing reliability measures that are discussed in the next section.

Reliability

The reliability of the steganography algorithm is also known as the probability of bit errors when deciphering information from an embedded image. Bit errors occur when the algorithm for deciphering information miscalculates the embedded bit as the wrong value or the image has been modified so that bit information is missing. Studies have shown the bit error for Dither modulation is low given certain scenarios with [22] showing that bit errors can be as low as 5×10^{-6}

This number shows that the algorithm itself is reliable given the scenario when the image is not modified in between the embedding and deciphering stages. For example when a user uploads an image to social media site Instagram they are given the option to crop and modify the image with filters. With Twitter, the user is not provided these options so the only noise or disruption that can be introduced is by the storage and serving of the image.

To alleviate this risk, the embedded data can be repeated across multiple areas of the image. This repetition of data across a noisy channel is a basic error correcting code is known as Repetition coding. This allows for a correction by majority system where the data is deciphered and compared against each other and a consensus is created on what is the correct value. Repetition coding is a basic form of error correction but there are more advanced forms of error correction such as Forward error correction using Hamming code [20] and soft error correction using the Viterbi Algorithm [21]

To implement repetition coding in the implementations is rather simple and would not overly impact the performance. The image would be broken up into parts and each part having the full message embedded in it. The only drawbacks with using repetition coding are that the overall capacity of data stored is reduced, since you are working with a smaller image and the distortion introduced can be above a detectability threshold.

Detectability/Security

Detectability in this scenario can be defined in two ways. The first way being traditional steganalysis methods of analysing images for possible modifications and deciphering information from the image. The second and more important for this App is the detectability of a typical user identifying that the image has been modified.

Evaluating the algorithm against steganography is discussed heavily in [14] and their results gives insight as to how robust the implementation is against steganography. The algorithm used in the Paper is Quantization Index Modulation, which is the starting point for Dither Modulation. In the paper the authors found that the ability to detect information within images is difficult because of the varying nature of images but with supervised learning they proved that standard implementations of QIM could be detected reliably. The also continue to note that the results do not take into account any attempts of increasing its robustness. Dither modulation is one of the steps taken to increase QIM robustness, which prevents the attacker for looking for similarities within the embedded information. Since the dither constantly moves the quantization regions analysing the information becomes much more difficult to characterise.

Since the selected algorithm is resistant to steganalysis, the next evaluation topic is the detectability by the average user. In [19] a model is created to analysis the human perception to alterations in an image. It found that there is a threshold where humans will more noticeably notice a change, which includes colour channels. This raises concerns when embedding data into colour channels and therefore steps are required to reduce this risk. As discussed earlier by embedding every pixel with bit information cause the image to be clearly identified as modified, therefore a trade off of capacity versus detectability is needed. This noise threshold needs more investigation, as the trade off from image, capacity and detectability is an evolving process due the complexity of an image selected. For example, a simple image with plenty of white space would easily highlight that an image is modified if the algorithm embeds too much information into the image.

Conclusion

This thesis provides a proof of concept that secure communications using a public forum such as a social network is possible using steganography. The App provides an intuitive instant message based form of communication where users can send and receive information that is hosted on Twitter. During the development of the design certain factors needed to be addressed and solved.

- The performance of embedding and deciphering data is key when trying to provide an instant message like application. The results from analysing two popular frameworks available in iOS show that Metal is the best framework for providing High performance image processing with the overhead of limited device support.
- The communication protocol is key in communicating between parties in a reliable and secure manner. Using public key encryption for the exchange of the dither key provides some protection around the secure flow of information. The protocol also provides areas for its evolution by outlining places where additional functionality can be added.
- The UI Design provides an abstracted view of the communication flow by providing an instant message style UI and UX. This allows the user to feel like they are using a standard chat App while the complexities of the communication is handled without their knowledge. This abstract also drove the need for a highly performant image processing as delays would significantly reduce this experience.

The application provides a reasonably sound method of creating a secure method of communication but can be improved upon. For example incorporating the ability to include multiple steganography methods would improve the difficulty for attackers to decipher information, incorporate higher key sizes with RSA to prevent attackers from factoring the keys and finally improving the protocol to provide more functionality such as group chats or the ability to spread the data across other social media sites

References

1. Chen, B. and Wornell, G. (2001). Quantization index modulation: a class of provably good methods for digital watermarking and information embedding. *IEEE Trans. Inform. Theory*, 47(4), pp.1423-1443.
2. Chen, B., 2000. *Design and analysis of digital watermarking, information embedding, and data hiding systems* (Doctoral dissertation, University of Michigan).
3. Hosam, O. (2013). Side-Informed Image Watermarking Scheme Based on Dither Modulation in the Frequency Domain. *TOSIGPJ*, 5(1), pp.1-6.
4. Kawaguchi, E. and Eason, R. (1999). Principles and applications of BPCS steganography. *In Photonics East (ISAM, VVDC, IEMB)*, pp.(pp. 464-473).
5. Liu, J., Li, X., Sun, J., Yang, X. and Liu, W. (2016). Multiple Watermarking Algorithm Based on Spread Transform Dither Modulation. [online] Available at: http://arxiv.org/abs/1601.04522.
6. Malik, H., Subbalakshmi, K. and Chandramouli, R. (2013). Nonparametric Steganalysis of QIM Steganography Using Approximate Entropy. *IEEE Transactions on Information Forensics and Security (Volume: , Issue: 2)*, 7(2), pp.418 - 431.
7. Noda, H., Niimi, M. and Kawaguchi, E. (2006). High-performance JPEG steganography using quantization index modulation in DCT domain. *Pattern Recognition Letters*, 27(5), pp.455-461.
8. Swanson, M., Kobayashi, M. and Tewfik, A. (1998). Multimedia data-embedding and watermarking technologies. *Proceedings of the IEEE*, 86(6), pp.1064-1087.
9. Developer.apple.com, (2016). *Mac Developer Library*. [online] Available at: https://developer.apple.com/library/mac/documentation/ [Accessed 1 Mar. 2016].
10. Zhang, Q. and Boston, N., 2003, October. Quantization Index Modulation using E~ 8 Lattice. *In Proceedings of the Annual Allerton Conference on Communication Control and Computing* (Vol. 41, No. 1, pp. 488-489). The University; 1998.
11. Brunk, H., 2001, August. Quantizer characteristics important for quantization index modulation. In *Photonics West 2001-Electronic Imaging* (pp. 686-694). International Society for Optics and Photonics.
12. Davern, P., 1997. *The application of steganography to fractal image compression* (Doctoral dissertation, Dublin City University).

13. Hogan, M.T., Hurley, N.J., Silvestre, G.C., Balado, F. and Whelan, K.M., 2005, March. ML detection of steganography. In *Electronic Imaging 2005*(pp. 16-27). International Society for Optics and Photonics
14. Dev.twitter.com. (n.d.). *REST APIs | Twitter Developers*. [online] Available at: https://dev.twitter.com/rest/public [Accessed 26 Mar. 2016].
15. Sunshine, C, (1979). Formal Techniques for Protocol Specification and Verification. *Computer*, 12(9), pp.20-27
16. Kleinjung, T., Aoki, K., Franke, J., Lenstra, A.K., Thomé, E., Bos, J.W., Gaudry, P., Kruppa, A., Montgomery, P.L., Osvik, D.A. and Te Riele, H., 2010. Factorization of a 768-bit RSA modulus. In *Advances in Cryptology–CRYPTO 2010* (pp. 333-350). Springer Berlin Heidelberg.
17. Valenta, L., Cohney, S., Liao, A., Fried, J., Bodduluri, S. and Heninger, N., *Factoring as a Service*.
18. Sullivan, K., Bi, Z., Madhow, U., Chandrasekaran, S. and Manjunath, B.S., 2004, October. *Steganalysis of quantization index modulation data hiding*. InImage Processing, 2004. ICIP'04. 2004 International Conference on (Vol. 2, pp. 1165-1168). IEEE.
19. Chou, C.H. and Liu, K.C., 2002. *A visual model for estimating perceptual redundancy inherent in color image. In Advances in Multimedia Information Processing*—PCM 2002 (pp. 353-360). Springer Berlin Heidelberg.
20. Mitchell, G., 2009. Investigation of hamming, reed-solomon, and turbo forward error correcting codes (No. ARL-TR-4901). ARMY RESEARCH LAB ADELPHI MD.
21. Hagenauer, J. and Hoeher, P., 1989, November. A Viterbi algorithm with soft-decision outputs and its applications. In Global Telecommunications Conference and Exhibition'Communications Technology for the 1990s and Beyond'(GLOBECOM), 1989. IEEE (pp. 1680-1686). IEEE.
22. Chen, B. and Wornell, G.W., 1999, July. Achievable performance of digital watermarking systems. In Multimedia Computing and Systems, 1999. IEEE International Conference on (Vol. 1, pp. 13-18). IEEE.