Steganography in Live Video

Student Name: Jonathan Browning

Supervisor Name: Dr. Ioannis Ivrissimtzis

Submitted as part of the degree of BSc Computer Science to the

Board of Examiners in the School of Engineering and Computing Sciences, Durham University

Abstract —

Context/Background: Steganography is the practice of embedding secret information inside of a cover object, such as an image, or in our case a live video stream, for the purpose of covert communication. Unlike cryptography, where an eavesdropper is easily able to detect that covert communication is taking place, steganography is designed to keep the covert communication undetected, with the eavesdropper believing that the innocuous cover object is the only communication that has taken place. In recent years, the use of live video streaming has increased rapidly, in the forms of one-to-one video calling (such as Skype) and live video broadcast (such as Facebook Live) to name a couple. This presents a new media format in which to hide secret information.

Aims: This project aims to investigate the feasibility of using streaming live video as a cover object for covert communication using LSB steganography on consumer grade hardware. The key research question to be answered is whether LSB embedding is an appropriate technique for use with live video streaming, including whether it is possible to achieve realtime encoding of a secret message into a live video stream using LSB, and whether the technique can be developed to provide resilience against lossy video compression.

Method: Python-based software has been developed for use on consumer hardware implementing LSB steganography using a live video feed (from a local webcam) as the cover object. The robustness of this technique has been evaluated under lossy MJPEG compression, and improvements to the technique have been proposed, implemented and their robustness evaluated.

Results: A prototype app was successfully implemented, allowing the one-way communication of multiple arbitrary ASCII text messages using a lossless video stream as a cover object through the use of the simple LSB technique, however the video resolution is severely limited. When subjected to MJPEG compression, the percentage of message bits correctly received by the client was only just over 50%. Improvements to the technique increased this to slightly below 70%, but did not increase it any further, therefore deviating from the predictions made using the model of the binary symmetric channel.

Conclusion: LSB steganography in live video streaming was found to be achievable using lossless video transmission, but subject to severe limitations on resolution due to bandwidth and encoding time required. LSB steganography was not found to be a feasible means of communication when used with lossy MJPEG compression, as while the transmission error was successfully reduced by the use of the repetition code, this was not enough to achieve reliable text-based communication.

Keywords — steganography, LSB, data-hiding, live video, streaming, MJPEG

I INTRODUCTION

A Steganography

Steganography is the process of hiding secret information inside other seemingly innocuous media, such as an image or a video file known as the *cover object*. The modified cover object, known as the *steganogram*, can then be sent over an insecure channel to another party, but it may be intercepted by an eavesdropper on the way. To the eavesdropper, the communication looks to be nothing more than the contents of the cover object.

Steganography can be seen as an alternative to cryptography for the purposes of covert communication. While the contents of an encrypted message are not visible to an eavesdropper, the presence of covert communication is obvious. Using steganography, however, the presence of covert communication is hidden.

Digital image and video files are often used as cover objects, as they contain a lot of visually redundant information that can be replaced with the contents of the secret message, meaning that a significant size of secret message can be embedded before the effect is visible. Due to the widespread use of lossy compression on the Web (Fridrich et al. 2002*b*), small but noticeable visual defects in digital images and video files are commonplace and therefore unsuspicious, so the capacity of such cover objects can be increased, despite visible defects, without arousing suspicion. A key advantage of the use of live video as a cover object is the lack of evidence left behind: unlike images or complete video files, live video streams, such as video conferences, are not typically retained after they have been displayed to the recipient. Therefore, unless an attacker suspects the use of steganography in advance, they will miss the opportunity to undertake steganalysis of the video stream.

B Steganalysis

Steganalysis encompasses the efforts of a third party to try to detect the use of steganography. Attempts to 'break' a steganographic system are known as *attacks*. There are two broad categories of attacks: visual and statistical attacks. Visual attacks involve using the naked eye to detect the use of steganography. This may simply be by directly viewing the image, if the steganographic technique introduces visible distortion; or the distortions may become visible after image processing. Statistical attacks require some form of statistical analysis of the steganogram to determine the presence of a steganographic message.

However, steganographic techniques that are vulnerable to attack have a distinct advantage over equivalently weak cryptographic systems: the use of steganography is usually not obvious without specifically looking for it, whereas the use of a cryptographic system is immediately obvious to anyone who can see the network traffic. The majority of users of steganography rely on this fact to avoid detection; this is demonstrated by the fact that LSB encoding (detailed in the next section) remains the most popular steganographic method for digital media despite its well-known vulnerabilities (Fredrich 2009, p.59). Since digital images and video files are now so prevalent on the World Wide Web, their transmission is unlikely to arouse suspicion, so this project will be less focused on the security of the steganographic techniques, and more on the feasibility of steganography in live video.

C LSB

Digital images (and by extension, individual frames of a video) are represented in a computer by multi-dimensional arrays (typically of the shape *width of image* $\times$ *height of image* $\times$ *number of colour channels*), containing a pixel value for each colour channel of each pixel in the image. In *least-significant bit (LSB)* steganography for images and video, only the least-significant bits of the pixel values are replaced with data representing a secret message. By using the least-significant bits, only the finest details of the carrier image/cover object are affected. The *embedding depth* of the encoding scheme is the number of bits per pixel of the carrier image that are replaced with data representing the secret message. This creates a trade-off between capacity and imperceptibility: the more bits that are used to embed the secret message, the higher the capacity, but the more obvious the distortion of the carrier image.

D Considerations for live video streaming

A key issue arising from using a live video stream as a cover object for steganography is the need for the embedding to be performed in real time. In order for the video stream to continue to be transmitted at the intended frame rate, each frame must be processed in time for the next frame.

Bandwidth is a major concern for live video streaming. Unlike streaming a complete video, the client is unable to compensate for low bandwidth through buffering, since the server is of course unable to transmit a frame before it has been created. Streaming uncompressed high-resolution video requires huge bandwidth. It is possible to reduce the bandwidth requirements without loss of information through the use of lossless compression. In addition to generic lossless data compression methods such as Huffman coding (Huffman 1952), lossless video compression techniques can take advantage of the nature of typical video sources, in particular:

- *Spacial correlation* - in a typical scene, neighbouring pixels tend to have similar pixel values, so we can reduce entropy by not storing each individual pixel value. Instead, we can sometimes record only the *differences* between neighbouring pixels in the same frame, in the form of a *corrective vector*.
- *Temporal correlation* - the same pixels tend to have similar pixel values between frames, so we can reduce entropy by not storing every pixel value for every frame. Instead, we can sometimes record only the *differences* between the same pixels in adjacent frames, in the form of a *corrective vector*.

For most applications, such as video calling, lossy compression (resulting in loss of information) is used to further reduce the bandwidth requirements. Digital video is very suitable for lossy compression as it contains a lot of redundant information that can be removed with minimal deterioration of the visual quality of the video (Bhaskaran & Konstantinides 1997, pp.2-3).

While lossy compression is very effective at reducing the size of the data to be transmitted, it creates difficulties for steganography. By removing redundant information, it removes many of the 'hiding places' that steganography relies upon to store secret data. In particular, simple LSB embedding in pixel values is generally ineffective when the video data is later compressed, as changes to the LSBs during compression corrupt the secret data.

E MJPEG encoding

MJPEG encoding is a form of lossy video encoding where each frame of the video is encoded separately as a JPEG image, and the frames are transmitted in sequence (Lieverse et al. 2001). Unlike some other codecs, it does not exploit any temporal correlation in the source video, as each frame is independently transmitted. The JPEG encoding of each frame consists of a number of steps, some of which are lossy, some of which are lossless.

Each 8x8 block of pixels (for each colour channel) is transformed into a frequency-domain representation via a discrete cosine transformation (DCT). This representation is then subjected to quantization, whereby the coefficients of the lowest frequencies are represented to a higher precision, as they have the biggest visual effect on the image. Finally, the quantized matrix is losslessly compressed using a form of Huffman encoding. The process to decode a JPEG image is simply the reverse of this process (Wallace 1991).

MJPEG encoding was chosen as it contains the key lossy DCT step that is featured in many modern codecs such as MPEG-4 and H.264, so will be suitable for testing the robustness of the embedding schemes used. However, it is much simpler to implement than other such codecs.

F MSSIM

The mean structural similarity (MSSIM) index is used to estimate the perceived quality of an image compared to a reference image. It is a better metric for perceived quality than other methods, such as MSE (mean-squared error) and PSNR (peak signal-to-noise ratio), as it is more representative of the Human Vision System (HVS). In particular, the HVS is much more sensitive to differences in structural information (the interdependencies between pixels, particularly between pixels close to each other) than luminance and contrast (Wang et al. 2004).

G Aims and Objectives

The purpose of this project is to evaluate the suitability of LSB embedding in live video streams as a steganographic system through the implementation of a prototype system on consumer grade hardware. By evaluating the robustness of the implementation, we will be able to determine whether LSB is an appropriate technique for use with live video streaming. The project objectives are detailed below.

Minimum Objectives:

1. Implementation of a simple method for embedding data in a live video stream.
2. Extraction of the data.
3. GUI supporting message preparation, embedding and extraction.

Intermediate Deliverables:

1. Multiple messages preparation, embedding and extraction in the same stream.
2. Analysis of robustness when subject to MJPEG encoding.

Advanced Deliverables:

1. Implement a more robust embedding method based on increased redundancy.
2. Analysis of robustness when subject to MJPEG encoding.

II RELATED WORK

A Image Steganography

A.1 Spacial domain steganography

The spacial domain represents images as numeric values arranged in a grid representing the brightness at that (spacial) point in the image. While LSB is the most popular steganographic technique for use in the spacial domain, a number of other techniques exist.

Several *cover escrow* techniques exist that require the unmodified cover object, as well as the steganogram, to be able to extract the message (Cox et al. 1996) (Podilchuk & Zeng 1997) (Swanson et al. 1996). However, this approach is not feasible in the context of live video streaming, as it is not possible to send the cover object in advance because the embedding is done in real time, frame by frame, as the video is captured from the camera. We are instead interested in *blind* techniques that do not require the original cover object to be able to extract the message.

Spread-spectrum image steganography (SSIS) (Marvel et al. 1999) modulates a signal over a carrier signal that constantly varies (in a manner determined by a secret key known to the recipient) to spread the signal over a wide frequency spectrum, decreasing its density so that it is less noticeable than natural noise. It has reasonably high capacity (although not a high as LSB), low visibility, robustness against noise and similar attacks. While it is not trivial to detect the use of SSIS, steganalytic techniques do exist (Sullivan et al. 2005).

Palette-based file formats, such as GIF (Graphics Interchange Format), do not store the pixel intensities, but rather represent the image using a fixed number (typically 256) of colours, using a dictionary to define the colours used in the image as RGB (Red, Green, Blue) values. Steganographic techniques exist to hide information in the palette. Gifshuffle uses the *ordering* of the palette to hide the secret message without affecting the appearance of the image (Fridrich & Du 1999). However, since many applications sort the palette by luminance by default, having a differently ordered palette may be suspicious to a third party. A number of methods decrease the colour depth (number of colours used in the image) before the embedding starts (Fridrich & Du 1999). In this way, subtly different 'duplicate' colours can be used to embed informaton with minimal visual impact on the image. However, the new palettes will then have groups of unnaturally close colours, which can be easily detected.

The EZStego tool for GIF images first orders the palette by luminance, then uses LSB steganography in the indices of the pixels to embed hidden data (Fridrich & Du 1999). This relies on the tendency for neighbouring colours in a palette sorted by luminance to *also* be close in colour space. However, exceptions to this are not uncommon, which can cause significant visible distortions in the image. Some improvements exist to avoid this by only embedding in colours that are close in colour space as well as luminance (Fridrich & Du 1999).

A.2 JPEG steganography

JPEG compression involves the conversion from the *spatial* domain (i.e. pixel brightness values) to the *frequency* or *transform* domain via the discrete cosine transformation (DCT) operation. Due to the lossy nature of this conversion, performing steganography using JPEG images as cover objects is more difficult than using images in the spatial domain, as small changes made to pixel intensities are easily corrupted during the compression process.

The first known algorithm developed to perform steganography in JPEG images was *JSteg* (Upman 1997). After the quantization stage of JPEG compression, the least-significant bit (LSB) of each DCT coefficient is replaced with a bit from the secret message. Coefficients of value zero or one are skipped (Westfeld 2001).

The JSteg algorithm is very vulnerable to statistical attacks. Assuming the message has a similar number of 0s and 1s (a reasonable assumption), JSteg equalises pairs of coefficients i.e. there will be a similar number of coefficients of value 2 as of value 3 etc. Since JPEG coefficients follow a standard distribution, the use of JSteg can easily be detected by examining the histogram of JPEG coefficients.

The OutGuess algorithm (Provos 2001) similarly embeds the message using LSB embedding into a random sample of the JPEG coefficients. However, it then performs a second pass over the JPEG coefficients to restore the histogram by tweaking the coefficients that have not been used to embed the message. Whilst resistant to chi-squared attacks, the OutGuess algorithm is detectable using more advanced statistical techniques (Fridrich et al. 2002*a*).

The F3 algorithm (Westfeld 2001) also alters the LSB of the JPEG coefficients. However, F3 does not overwrite bits, but rather decrements the value of the coefficient if the LSB does not match the bit of the secret message. Clearly, this is not possible if the coefficient is zero, so coefficients of value zero are not used. This causes a phenomenon called *shrinkage* when coefficients of values 1 and -1 are changed to 0 (i.e. to embed a zero message bit). The decoder does not know whether this bit should be skipped or interpreted as a zero message bit. The sender therefore repeats the zero message bit so that it is embedded in a coefficient with value other than 0, 1 or -1. This produces a greater number of even coefficients than odd coefficients in the histogram of JPEG coefficients, making it easily statistically detectable.

The F4 algorithm (Westfeld 2001), an improvement of F3, eliminates this weakness by making negative coefficients hold the *inverse* steganography/data value of the positive coefficients. That is, an even positive coefficient represents a zero message bit and an odd positive coefficient represents a one message bit (as before), but an even negative coefficient represents a one, while an odd negative coefficient represents a zero. Therefore the histogram of JPEG images after F4 embedding is much more similar to the expected histogram of an unaltered JPEG image..

The F5 algorithm (Westfeld 2001) provides a further improvement on F4 by using *permutative straddling* to scatter the secret message across the image, hindering attacks that exploit differences between image regions containing steganographic information and regions that don't. Nonetheless, the F5 algorithm has been broken through the use of more advanced statistical techniques (Fridrich et al. 2002*b*)

B Video Steganography

(Xu et al. 2006) propose an algorithm for embedding and extraction of data directly into the compressed bitstream of a video. This manipulates the motion vectors used by MPEG encoding, in particular choosing macroblocks with larger moving speed, both to resist further video processing, and to minimise the visible effect on the carrier video.

(Stanescu et al. 2007) similarly embed data directly into the MPEG bitstream, however they choose only to embed information in the I-frames using Least-Significant-Bit encoding. This has the advantage of simplicity, while still maintaining most of the capacity as the I-frames are able to store more hidden information with minimal visible effect.

(Cheddad et al. 2008) present a video steganography mechanism which exploits the YCbCr

colour space to minimise the visible effects on the carrier video due to human colour-response characteristics. Converting images from RGB to YCbCr removes the correlation between the RGB channels in an image, and therefore leads to less noticeable distortion when LSB encoding is applied to different colour channels. Only regions of the image that will be used to embed data are converted to the YCbCr colour space, in order to increase the overall Peak-Signal-to-Noise Ratio (PSNR). This is due to the lossy nature of the RGB to YCbCr conversion.

(Eltahir et al. 2009) describe a simpler method using LSB encoding and a 3-3-2 approach (3 least-significant bits from the Red and Green channels, and 2 from the Blue channel) to make the resultant image visually similar to the original, since the Human Vision System is more sensitive to the colour blue than red or green. This 3-3-2 approach allows for a greater capacity for hidden data before the visual defects become noticeable to the human eye.

(Westfeld & Wolf 1998) describe a method of steganography in a video conferencing application. The data is embedded in the DCT coefficients of the H.261 video feeds, allowing for real-time two-way text communication alongside the video conference, with no perceptible distortion to the video signal. However, the technique is vulnerable to statistical attacks, but as discussed earlier, the use of live video streaming makes the capture of the stream for the purposes of steganalysis much less likely without prior suspicion.

(Shirali-Shahreza 2006) outlines a method of real-time video steganography without requiring storage of the carrier message. This has the advantage of eliminating any evidence that could later reveal that covert communication had taken place through steganalysis, and the message is not recoverable by a third party. This is achieved by embedding data into a display (e.g. a billboard) in real-time, taking a photograph of the display and processing this image to retrieve the hidden data. However, while the data is embedded into multiple frames of the video stream, the client only retrieves the data from a single frame. This limits the capacity of this method to that of image steganography, defeating one of the key advantages of video steganography over image steganography.

C Protocol-based steganography

As well as techniques for embedding a secret message in the *content* of the digital media, there exist a number of techniques for hiding data inside the networking protocols used to transmit the media. A few of these techniques are described here. (Rowland 1997) describes a method of steganography in IP (Internet Protocol) headers using the packet IDs to encode ASCII characters, while (Xu et al. 2007) describe a method for hiding encrypted information in the 'identification' field of IP headers also using ASCII encoding. (Mazurczyk & Szczypiorski 2008) describe a technique to hide data inside a VoIP (Voice over Internet Protocol) stream achieving a data rate of over 1.3Mbps in one direction, and work (partly) by the same authors (Frączek et al. 2012) describes a technique for hiding information in STCP (Stream Transmission Control Protocol) streams for use in internet telephony applications.

III SOLUTION

A Software tools

The Python language was chosen for the software built as part of this project, due to its cross-platform compatibility and excellent third-party libraries, including the open-source OpenCV library which is used heavily for the image-processing aspects of the software. The graphical user interfaces were developed using Tkinter, the most commonly used Python package for GUI development (*Tkinter - Python Wiki* n.d.). This allowed for rapid prototyping and for easy integration with the rest of the Python-based system.

B System Architecture

The system is based on a client-server model. The *server* is the transmitting/encoding device, and provides the source video feed from a local webcam, while the *client* is the device that extracts the hidden message. The client connects to the server, starting the video stream. In order to *send* a message/video stream, the user must use the 'server' application, but to *receive* a message/video stream, the user must use the 'client' application. This method has the advantage of not requiring a central server to assist with establishing connections as in e.g. Skype (Baset & Schulzrinne 2004), but has the disadvantage of requiring the client user to have knowledge of the server user's network address (hostname/IP address and port).

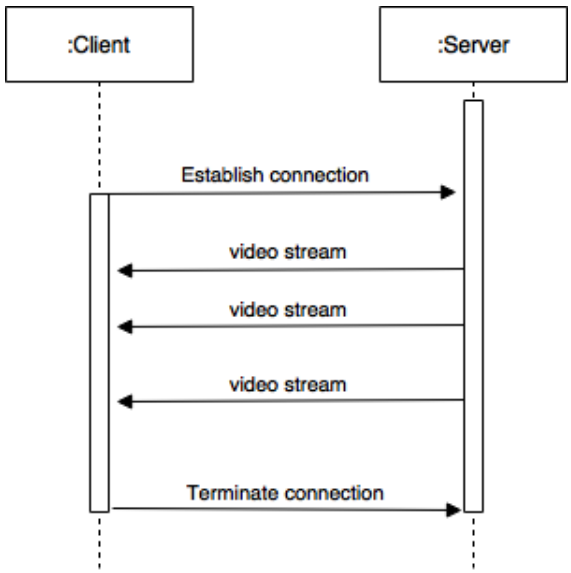


Figure 1: A sequence diagram demonstrating the system architecture. Note that the connection can be terminated by either the client or the server.

C Encoding Schemes

C.1 Simple LSB and N-th LSB

The first encoding scheme implemented was the simplest form of LSB, where the least-significant bit of each pixel value of the video frame is swapped for a single bit of the bitstring to be encoded. This bitstring can be formed from an arbitrary text-based message using ASCII encoding and terminated with an ASCII EOT (end of transmission) character, or can be randomly generated (for testing purposes).

The simplest way to implement this would be to loop over all the pixels in the array, modifying pixel values one at a time. However, this was found to be too slow for realtime video processing, due to the speed that Python code runs at. The same effect was achieved in real time using the numpy library's bitwise operators, *bitwise_or* and *bitwise_and*. Since the library function itself runs (compiled) C code, the embedding was able to execute at a much quicker pace.

A simple variant of LSB embedding is to use the N-th least-significant bit, rather than the least-significant bit. For a 24-bit image, N can be between 1 and 8 (3 colour channels $\times$ 8 bits per colour channel = 24 bits). The greater the value of N, the greater the change in the pixel value, and therefore the more visible the effect on the image.

$$\text{Maximum change in pixel value} = \pm 2^{N-1} \tag{1}$$

This technique shall be referred to as *N-th least-significant bit encoding*, or *N-th LSB encoding*.

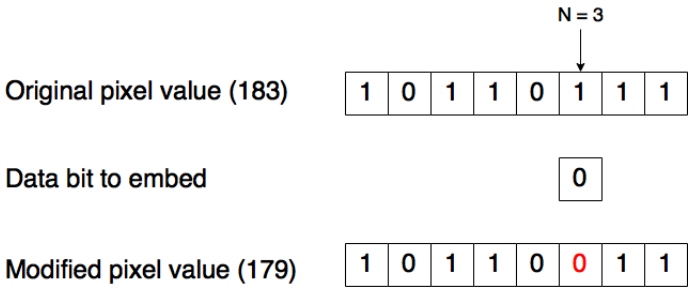


Figure 2: A simple diagram showing the embedding of a bit from the secret message into the 3rd least-significant bit of a pixel value (N-th least-significant bit encoding).

C.2 N least-significant bits (N LSBs)

A further variant on N-th LSB encoding is to alter not just in the N-th least-significant bit of the pixel values, but also each less-significant bit. For example, if we chose to employ 3rd least-significant bit embedding, we would also use alter least-significant and 2nd least-significant bits.

The data embedded in the lower bits could be different from the data embedded in the N-th bit (i.e. to achieve greater capacity), but here they are set to maximise the gap between the pixel value and the nearest pixel value that would cause the N-th least-significant bit to flip. This is designed to increase the robustness of the system against MJPEG compression by minimizing the chance that the N-th least-significant bit is flipped.

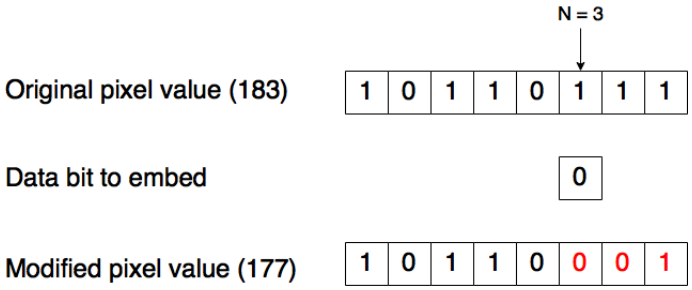


Figure 3: A simple diagram showing the embedding of a bit from the secret message into the 3 least-significant bits of a pixel value (N least-significant bits encoding). The modified pixel value would need to be changed by either at least +3 or -2 for the 3rd LSB to flip. Alternatively a pixel value of 178 would require either a +2 or a -3 change for the 3rd LSB to flip. The choice between these two options is done randomly.

C.3 Simple repetition code

In order to improve the robustness of the system when MJPEG video compression is introduced, a simple repetition code has been implemented. The number of repetitions that can fit inside a frame is calculated by taking the dimensions of the frame, multiplying by the number of colour channels (3) and dividing by the chosen maximum length of a message (in bits). Since this may not result in an integer, the result is rounded *down* to the nearest integer. If the result is an even integer, 1 is subtracted to make it an odd number to ensure a majority vote. The message is repeated this number of times, and the least-significant bits of the remaining pixels in the frame are filled randomly.

At the client, the message is extracted by taking a 'vote'. Each repetition of the message 'votes' for each bit of the message with what it has received (0 or 1) for that bit of the message. The majority result is then chosen for the decoded message. The probability of an error for a repetition code is

$$P_{error} = \sum_{k=\frac{n+1}{2}}^{n} \binom{n}{k} \epsilon^k (1-\epsilon)^{n-k} \tag{2}$$

(Hanzo et al. 2007, p.372) where n is the repetition factor (number of repetitions) and ϵ is the transmission error rate. This equation assumes that the transmission errors are limited to bits flipping (i.e. a 1 is flipped to a 0 or a 0 is flipped to a 1, and no bits are 'lost'), are independent of each other, and a $0 \rightarrow 1$ bit flip has the same probability as a $1 \rightarrow 0$ bit flip. This model is known as a *binary symmetric channel*. Repetition codes can detect up to $\lfloor n/2 \rfloor$ transmission errors (per message bit) on a binary symmetric channel without decoding error. An example of a repetition code is shown in Figure 4.

Figure 4: A bitstring of length 16 is encoded with a simple repetition code (5 repetitions), transmitted over a noisy channel (probability of bit flip = 0.2) and correctly decoded.

C.4 Alternating repetition code

The alternating repetition code is very similar to the simple repetition code, however instead of repeating the message as-is, every other repetition is inverted. For example, a message of 1001000110111111 would be repeated as

1001000110111111 0110111001000000 1001000110111111 0110111001000000 etc.

This variation aims to counteract the effect of any difference between the percentages of message 0s and 1s being correctly decoded.

D Transmission of stream

D.1 Lossless transmission

Lossless transmission between separate server and client applications was achieved by using TCP (Transmission Control Protocol). The server listens for connections on the chosen port, and once a connection has been established, the video streaming can begin. The array representing the pixel values of a video frame is serialized, or *pickled*, and the frames are transmitted sequentially from the server to the client.

At the client, each frame is deserialized, or *unpickled*, displayed to the user, and the message is extracted.

D.2 MJPEG streaming

For MJPEG streaming, TCP is also used in the form of a HTTP (Hyper-Text Transfer Protocol) server. When the HTTP server receives a GET request for a URL with a '.mjpg' file extension, the necessary headers for a MJPEG stream are sent from the server to the client.

Each frame has the secret message embedded into it, before being converted into JPEG format. Headers are sent for the individual JPEG frame, followed by the JPEG data of the frame itself. At the client, the stream of data is separated into individual frames by finding boundary 'markers' in the stream. The individual JPEG frames are then converted back to arrays of pixel values, from which the message is then extracted.

By using the MJPEG format, we allow for the video stream to be accessible by many modern web browsers, that are able to view the video without being able to extract the message. By using a common format for the video transmission, we are able to reduce the risk of arousing suspicion by allowing third parties to easily view the video stream without the special client program that can also extract the hidden messages.

E Measurement

To evaluate the robustness of the different embedding techniques against MJPEG compression, a random bitstring was generated, saved to disk, and embedded into each frame of the video feed. Each original frame captured from the webcam was saved to disk. For the N-th LSB and N LSBs methods, the length of the bitstring message was equal to the number of pixels in each frame (i.e. the video resolution multiplied by 3 colour channels), so that every pixel in the frame contained a message bit per colour channel . For the repetition codes, the bitstring was a fixed length of 1120 bits (140 bytes, equivalent to 140 characters), and the N-th LSB of each pixel that was not affected by the repetition code was set randomly.

The video feed was streamed to the client for 300 frames, where each frame was saved to disk, as was the bitstring message decoded from each frame. A separate program was then run to compare the original message generated by the server to each of the 300 versions of the received message, and to compare the original and received frames. The following statistics were calculated:

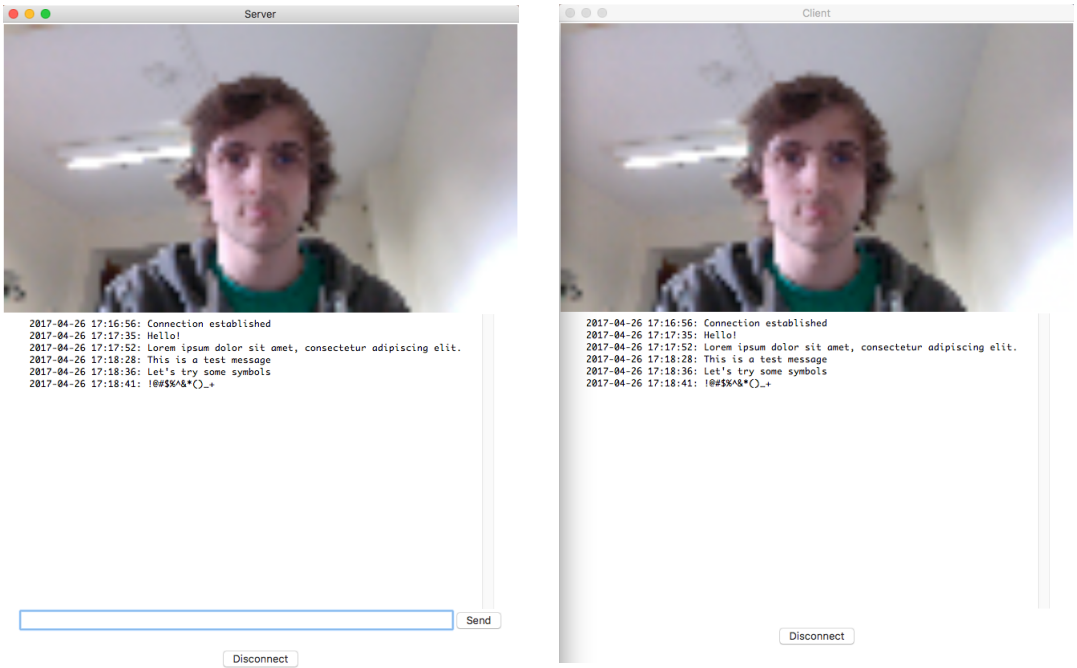
- The percentages of 0s and 1s in the generated bitstring
- The percentages of 0s and 1s of each frame correctly decoded at the client
- MSSIM of each received frame
- The average pixel value of each unmodified frame (as captured from the webcam)

These results were then averaged over the 300 frames for comparison with other methods. For the purposes of testing, the server and client applications were run on the same device (Mid-2012 15" Apple MacBook Pro with Intel i7 processor and 16GB RAM running OS X 10.12) using the loopback network interface.

IV RESULTS

A *Lossless Transmission*

A prototype app was successfully implemented (screenshots in Figure 5), allowing one-way communication of multiple arbitrary ASCII text messages embedded into the video feed using the simple LSB technique, sent losslessly to the client, and correctly extracted. The highest resolution achieved for server and client on the same machine (i.e. using the loopback network interface) was 128×72, compared to the 1280×720 native resolution of the webcam.



(a) Server call view **(b)** Client call view

Figure 5: Screenshots of the prototype app during a video stream.

Additionally, a randomly generated bitstring message was embedded and extracted successfully, and the results are detailed in Table 1, averaged over 300 frames of video.

Table 1: AVERAGE STATISTICS FOR LOSSLESS TRANSMISSION

Statistic	Value
Percentage of 0s in generated bitstring	50.43%
Percentage of 1s in generated bitstring	49.57%
% of bits correctly decoded	100.00%
MSSIM	0.9983

Since there was no room for improvement on the accuracy of the decoded message, the other techniques (N-th LSB embedding, N LSBs embedding, repetition code) were not implemented for lossless transmission.

B MJPEG streaming

When the simple LSB method was used with MJPEG compression, the text-based messages were not successfully received by the client, but garbage text was received instead. Using a randomly generated bitstring, the N-th LSB and N LSBs techniques were evaluated by comparing the received bitstrings to the originals, varying N between 1 and 8. These results are presented in Figure 6. There are noticeable differences between the percentages of 0s and 1s that are correctly received, and by comparing (a) and (b) we can see that this skew is dependent on the original pixel values in the scene. The N LSBs technique eliminates this phenomenon almost entirely.

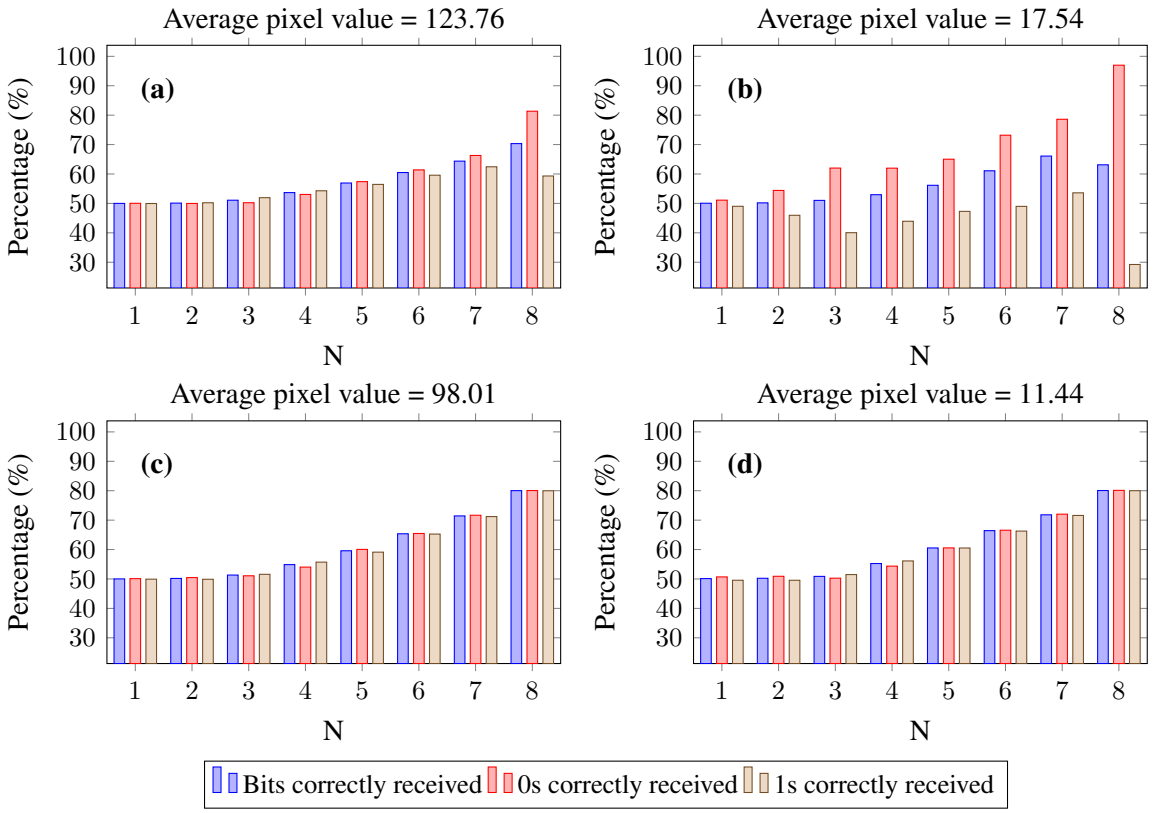


Figure 6: (a) and (b) show the success rates for N-th LSB encoding, while (c) and (d) show the success rates for N LSBs encoding.

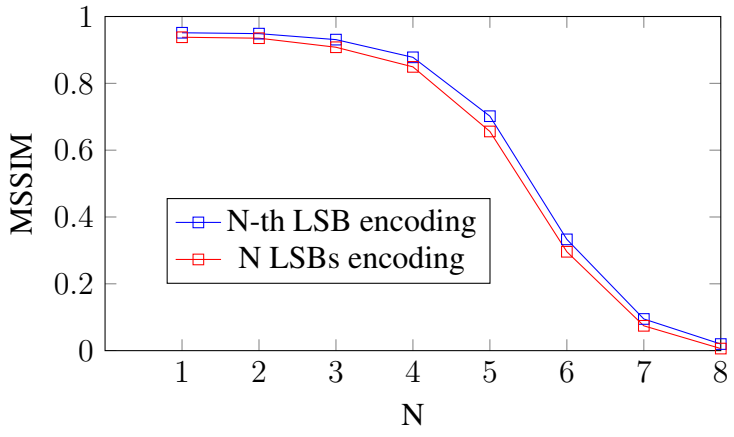
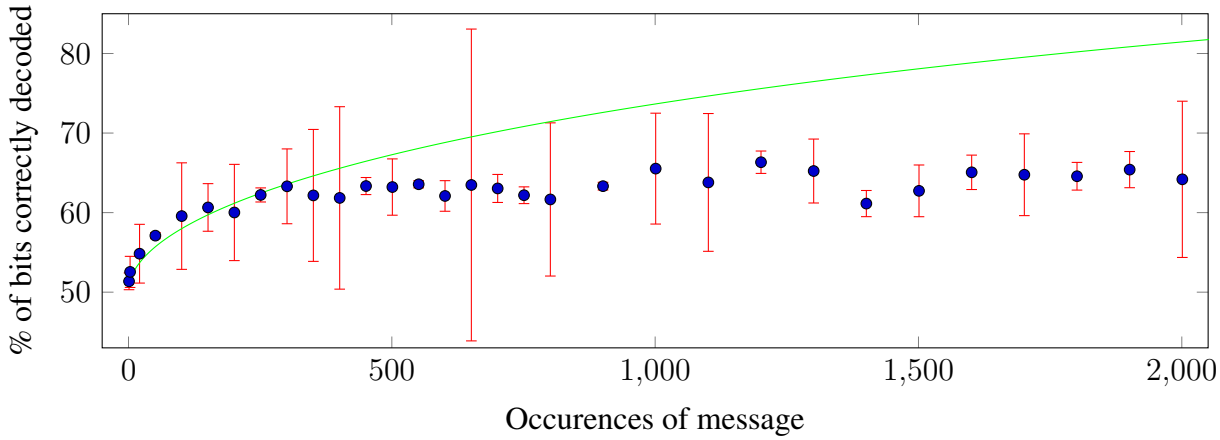
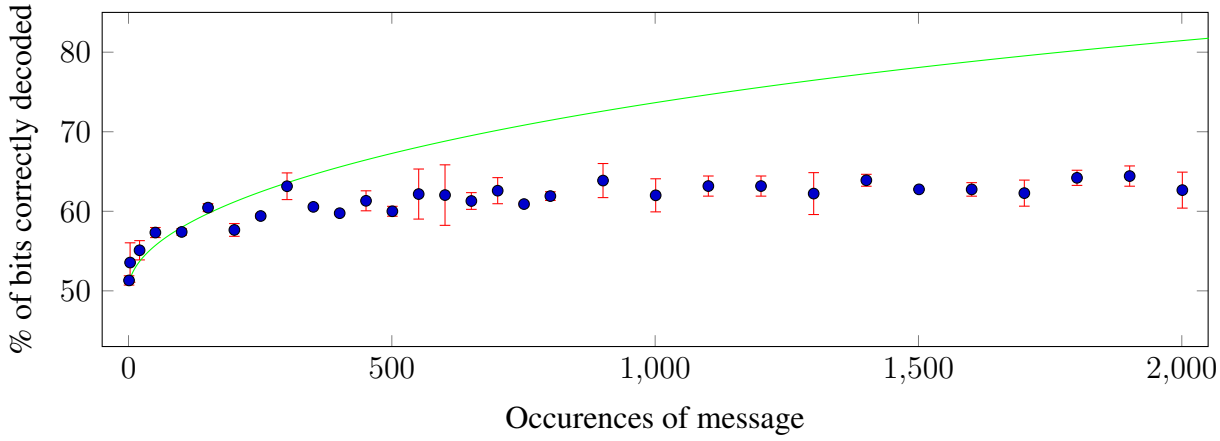


Figure 7: MSSIM values for N-th LSB and N LSBs encoding.

As we would expect, the perceived quality of the stream (approximated here by the MSSIM value) decreases as the significance of the bit(s) used to embed data increases. This is shown in Figure 7. More notably, however, the difference in MSSIM between embedding in just the N-th LSB and embedding using multiple bits is only very slight. The N LSBs embedding method for N = 3 provided an overall success rate of approximately 51%, whilst having minimal perceptible detriment to the image quality. As Figure 7 shows, the MSSIM values rapidly decrease for N > 3, therefore the N LSBs embedding method with N = 3 was chosen to take forward and improve further using the repetition codes.



(a) Simple repetition code (Average pixel value: 96.31).



(b) Alternating repetition code (Average pixel value: 111.62).

Figure 8: Robustness of 3 LSBs encoding with simple and alternating repetition codes. Error bars show the differences between the percentages of 0s and 1s that are correctly decoded, and the green curves show the theoretical values of the binary symmetric channel model, assuming a transmission error rate of 0.49 (based on Figure 6).

Figure 8 shows that both repetition codes initially perform better than the binary symmetric channel model, but performance then levels off after approximately 200 repetitions rather than continuing to improve. The differences between the percentages of 0s and 1s correctly decoded are much greater with the use of a simple repetition code. While the overall performance of the alternating repetition code is not an improvement on the simple repetition code, the differences between the percentages of 0s and 1s that are correctly decoded are drastically reduced.

V EVALUATION

A *Lossless Transmission*

The successful communication of multiple text-based messages, embedded using the simple LSB technique and transmitted losslessly, shows promise for the feasibility of using LSB steganography in live video. The very high MSSIM values of the modified frames demonstrate the imperceptibility of the data embedding.

While the highest resolution achieved was only 128×72, main features in the video are still recognizable, and this may be enough for some applications. At this resolution, we can calculate that the uncompressed data rate would be $128 \times 72 \times 3$ colours$\times 30$ frames per second$\times 8$ bits per pixel value ≈ 6.6Mbps (30 fps was taken as an estimate of an upper bound). Considering that this is the *uncompressed* data rate, and the serialization process includes applying some lossless compression, it is surprising that this is the maximum resolution achievable, since a much higher maximum data rate would be expected, particularly on a loopback network interface. This suggests that the limiting factor on the maximum resolution is not the maximum data rate achievable, but rather that for higher resolutions, the serialization of each frame takes too long to be achieved in real time. If a faster method of serialization and deserialization were implemented, higher resolutions would likely be achievable. A possible route might be to use an existing lossless image format, such as PNG (Portable Network Graphics), to encode each frame individually, rather than using Python's generic serializer/pickler that is able to serialize many different types of data, as methods designed specifically for image encoding may provide superior performance.

B *MJPEG streaming*

The results shown in Figure 6 clearly demonstrate that the likelihood of a message bit embedded using N-th LSB encoding being correctly extracted from a MJPEG stream is heavily dependent on the significance of the bit chosen to embed the message bit. The more significant the bit used for embedding, the more likely the message bit is to be extracted correctly. This is so far unsurprising, as we know that the less significant bits represent fine details of the image, and that MJPEG compression is designed to reduce data usage by sacrificing some of the less perceptible detail.

Figure 6 demonstrates a more interesting phenomenon, however, which is that there is often a *skew* between the percentage of 0s and the percentage of 1s of the original message that are correctly decoded. Comparing subplots (a) and (b), we can see that this skew is dependent both on the significance of the bit used to embed data, and the original pixel values of the scene.

This phenomenon can be attributed to differences in the amount of change required to flip a 'zero' message bit to a 'one', and vice versa. The closer a pixel value is to a value with a flipped message bit, the more likely the message bit is to flip, since small variations in pixel values caused by lossy compression are naturally more likely to occur than larger variations. For most pixel values in an unmodified image, it is equally likely that a '0' message bit will be flipped to a '1', and vice versa. However, since pixel values are bounded between 0 and 255, this is not always the case. Let us use N-th LSB encoding with N = 3 as an example. A pixel value of 0 from the camera is embedded with a '0' message bit, so it keeps the pixel value of 0. In order for the 3rd LSB to be flipped to a '1', a change in pixel value of +4 is required to take the pixel value to 4 (00000100). It is not possible for the message bit to flip as a result of a negative change in the

pixel value, since the pixel value cannot be less than 0. If the message bit were a '1' instead, the modified pixel value would be 4 (00000100). In order for the 3rd LSB to be flipped a minimum change of -1 (to 00000011) would be required. Since a change of -1 is much more likely than a change of +4, an original pixel value of 0 is more likely to cause a message '1' to flip to a '0' than vice versa. In general, the N-th bit of a pixel value cannot flip from a '0' to a '1' by means of a negative change in pixel value if the pixel value is less than 2^{N-1}. By symmetry, the N-th bit of a pixel value cannot flip from a '1' to a '0' by means of a positive change in pixel value if the pixel value is greater than or equal to $255 - 2^{N-1}$ (i.e. each bit more significant than the N-th bit is already set to 1). As the value of N increases, these conditions are true for a wider range of pixel values, which allows for a larger skew. In very bright scenes with many high pixel values, the probability of a message '1' being flipped will be less than of a '0' being flipped. In dark scenes, the opposite is true. Therefore, we can conclude that the probabilities of message 0s and 1s being flipped is dependent on the original pixel values in the scene.

Figures 6(c) and (d) demonstrate the same trend that the greater the significance of the bits used to embed message data, the higher the likelihood of the message bits being correctly extracted. It also demonstrates that using N least-significant bits provides better performance than just using the N-th least-significant bit. Additionally, the use of N LSBs is effective at reducing the skew between the percentages of 0s and 1s of the original message being correctly decoded. These improvements can be explained by the nature of the embedding process, which is designed to maximise the minimum change in each pixel value required for its message bit to be flipped.

As we would expect, as the significance of the bit used to embed message data is increased, the perceived distortion of the carrier image is increased, or in other words, the perceived similarity of the carrier image to the original image is decreased. This is shown clearly by Figure 7. The shapes of the curves in Figure 7 are also noteworthy. For N between 1 and 3, there is minimal difference in the MSSIM values, but values then rapidly decrease as N is increased between 3 and 6, before reducing at a much lesser rate for N between 6 and 8. The MSSIM values for the N LSBs embedding method are *slightly* lower than in N-th LSB embedding. This is to be expected, since more information from the original image is lost during N LSBs embedding.

Figure 8(a) shows the effect that a simple repetition code has on the likelihood of message being correctly decoded. As expected, the use of a repetition code increases the likelihood of original message bits being correctly decoded. However, the results show that this increase does not follow the curve predicted by Equation 2. The percentage of bits correctly decoded initially grows more quickly with the number of occurrences of the message than predicted by Equation 2, but then flattens off slightly below 70% and does not continue to grow, indicating that the binary symmetric channel is an unsuitable model. This may be because while the assumptions behind the model are reasonable, they are not completely correct. For instance, it may be the case that the transmission errors are not independent, as pixels in the same 8x8 block may affect each other during the DCT process.

The differences between the percentages of 0s and 1s that are correctly decoded are increased with the use of a repetition code, as the slight skews are amplified due to the repetition.

The use of an alternating repetition code, where every other repetition of the message is inverted, was not found to increase the success rate of message bits being correctly decoded, as the results, shown in Figure 8(b), formed a similar curve to those of the simple repetition code. However, it was found to be effective at reducing the differences between the percentages of message 0s and 1s that were correctly extracted from the video stream.

VI CONCLUSIONS

We can conclude that the use of LSB steganography in live video streaming is achievable using lossless video transmission. This was demonstrated through the implementation of a one-way "video chat"-style application, where multiple text based messages could be encoded in the video stream before transmission to the client, where the messages were extracted successfully. However, the approach was found to be not altogether practical. The key weakness is the time required to serialize each frame for transmission. This severely limits the achievable video resolution. A quicker form of serialization could lead to improvement, however the bandwidth cost of lossless video streaming is very high, so the video resolution would still be limited. Additionally, the use of lossless compression may also arouse suspicion, since lossy compression is more typical for video transmission.

The use of LSB steganography, and the variants of it explored in this project, in live video streaming subjected to lossy MJPEG compression was not found to be feasible due to the high transmission error rate. The binary symmetric channel was found to be an unsuitable model for the effects of MJPEG compression on the message bits embedded with LSB and its variants. While the use of repetition codes helped to decrease the error rate initially, the error rate did not continue to shrink as the number of repetitions was increased, therefore the error rate was not able to be reduced sufficiently to allow for text-based communication, whilst maintaining sufficiently low visual distortion to the video to avoid arousing suspicion.

There are a number of directions that future work in this area could take. To improve on the LSB steganography method explored here for lossless video streams, the use of lossless video compression standards, such as H.263 lossless (Hanzo et al. 2007, p.310), could be investigated. Successful implementation of such a system could reduce the bandwidth requirements (and therefore increase the maximum achievable video resolution) through the use of more efficient video encoding (e.g. exploiting spacial and temporal correlation as described in Section I Part D).

Further work on steganography for lossy video streams could investigate the feasibility of embedding data in the *frequency* domain, rather than in the spacial domain. For MJPEG streams, modern techniques involving the manipulation of DCT coefficients, such as JSteg and F5, could be employed to hide message data that can be successfully extracted by the client. However, it remains to be seen if such techniques are achievable in real time on current consumer grade hardware, as would be necessary for live video streaming.

The obvious extension to the prototype app would be the implementation of two-way video and text-based communication. This would much improve the usefulness of the application, providing a steganographic alternative to popular video chat applications such as Skype. An alternative route could be to instead create software to allow for steganographic communication over Skype by embedding into the camera feed before it is made available to the Skype application. This would decrease suspicion surrounding the application's use, since it would just appear as regular Skype traffic, which is very prevalent on the World Wide Web. Another further extension of the application could be to allow arbitrary data transfer across the steganographic channel, rather than limiting it to text-based communication. This could allow for transfer of images, video files, and executable files for instance, and could even be extended to function as a full proxy server, whereby the internet traffic from one user's web browser is carried over the steganographic channel to the other user's computer, where it would then complete the connection to the Internet, thus making it appear as if the traffic was coming from a different user, in order to bypass firewall restrictions for instance.

References

Baset, S. & Schulzrinne, H. (2004), 'An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol', *CoRR* **abs/cs/0412017**.

Bhaskaran, V. & Konstantinides, K. (1997), *Image and Video Compression Standards*, 2 edn, Springer US.

Cheddad, A., Condell, J., Curran, K. & Mc Kevitt, P. (2008), Skin tone based steganography in video files exploiting the YCbCr colour space, *in* 'Multimedia and Expo, 2008 IEEE International Conference on', IEEE, pp. 905–908.

Cox, I. J., Kilian, J., Leighton, T. & Shamoon, T. (1996), Secure spread spectrum watermarking for images, audio and video, *in* 'Image Processing, 1996. Proceedings., International Conference on', Vol. 3, IEEE, pp. 243–246.

Eltahir, M. E., Kiah, L. M., Zaidan, B. B. & Zaidan, A. A. (2009), High rate video streaming steganography, *in* 'Information Management and Engineering, 2009. ICIME'09. International Conference on', IEEE, pp. 550–553.

Frączek, W., Mazurczyk, W. & Szczypiorski, K. (2012), 'Hiding information in a stream control transmission protocol', *Computer Communications* **35**(2), 159–169.

Fredrich, J. (2009), *Steganography in Digital Media: Principles, Algorithms, and Applications*, Cambridge University Press.

Fridrich, J. & Du, R. (1999), Secure steganographic methods for palette images, *in* 'International Workshop on Information Hiding', Springer, pp. 47–60.

Fridrich, J., Goljan, M. & Hogea, D. (2002*a*), Attacking the OutGuess, *in* 'Proceedings of the ACM Workshop on Multimedia and Security', Vol. 2002, Juan-les-Pins, France.

Fridrich, J., Goljan, M. & Hogea, D. (2002*b*), Steganalysis of JPEG images: Breaking the F5 Algorithm, *in* 'International Workshop on Information Hiding', Springer, pp. 310–323.

Hanzo, L., Cherriman, P. & Streit, J. (2007), *Video Compression and Communications: From Basics to H.261, H.263, H.264, MPEG4 for DVB and HSDPA-Style Adaptive Turbo-Transceivers*, 2nd edn, Wiley-IEEE Press.

Huffman, D. A. (1952), 'A Method for the Construction of Minimum-Redundancy Codes', *Proceedings of the IRE* **40**(9), 1098–1101.

Lieverse, P., Stefanov, T., van der Wolf, P. & Deprettere, E. (2001), System level design with SPADE: an M-JPEG case study, *in* 'IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)', pp. 31–38.

Marvel, L. M., Boncelet, C. G. & Retter, C. T. (1999), 'Spread spectrum image steganography', *IEEE Transactions on Image Processing* **8**(8), 1075–1083.

Mazurczyk, W. & Szczypiorski, K. (2008), 'Steganography of VoIP streams', *On the move to meaningful Internet systems: OTM 2008* pp. 1001–1018.

Podilchuk, C. I. & Zeng, W. (1997), Digital image watermarking using visual models, *in* 'Electronic Imaging'97', International Society for Optics and Photonics, pp. 100–111.

Provos, N. (2001), Defending Against Statistical Steganalysis, *in* 'Usenix Security Symposium', Vol. 10, pp. 323–336.

Rowland, C. H. (1997), 'Covert channels in the TCP/IP protocol suite', *First Monday* **2**(5).

Shirali-Shahreza, M. (2006), A new method for real-time steganography, *in* 'Signal Processing, 2006 8th International Conference on', Vol. 4, IEEE.

Stanescu, D., Stratulat, M., Ciubotaru, B., Chiciudean, D., Cioarga, R. & Micea, M. (2007), Embedding data in video stream using steganography, *in* 'Applied Computational Intelligence and Informatics, 2007. SACI'07. 4th International Symposium on', IEEE, pp. 241–244.

Sullivan, K., Madhow, U., Chandrasekaran, S. & Manjunath, B. S. (2005), Steganalysis of spread spectrum data hiding exploiting cover memory, *in* 'Electronic Imaging 2005', International Society for Optics and Photonics, pp. 38–46.

Swanson, M. D., Zhu, B. & Tewfik, A. H. (1996), Transparent robust image watermarking, *in* 'Image Processing, 1996. Proceedings., International Conference on', Vol. 3, IEEE, pp. 211–214.

Tkinter - Python Wiki (n.d.), `https://wiki.python.org/moin/TkInter`. Accessed: 2017-01-21.

Upman, D. (1997), 'JSteg', `https://zooid.org/~paul/crypto/jsteg/`. Accessed: 2017-01-21.

Wallace, G. K. (1991), 'The JPEG Still Picture Compression Standard', *Communications of the ACM* **34**(4), 30–44.

Wang, Z., Bovik, A. C., Sheikh, H. R. & Simoncelli, E. P. (2004), 'Image quality assessment: from error visibility to structural similarity', *IEEE Transactions on Image Processing* **13**(4), 600–612.

Westfeld, A. (2001), F5 – A Steganographic Algorithm, *in* 'International Workshop on Information Hiding', Springer, pp. 289–302.

Westfeld, A. & Wolf, G. (1998), Steganography in a Video Conferencing System, *in* 'International Workshop on Information Hiding', Springer, pp. 32–47.

Xu, B., Wang, J. & Peng, D. (2007), Practical Protocol Steganography: Hiding Data in IP Header, *in* 'First Asia International Conference on Modelling Simulation (AMS'07)', pp. 584–588.

Xu, C., Ping, X. & Zhang, T. (2006), Steganography in compressed video stream, *in* 'Innovative Computing, Information and Control, 2006. ICICIC'06. First International Conference on', Vol. 1, IEEE, pp. 269–272.