Re-encoding Resistance: Towards Robust Covert Channels over WebRTC Video Streaming

by

Adrian Daniel Cruzat La Rosa

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2023

Author's Declaration

Abstract

Internet censorship is an ongoing phenomenon, where state level agents attempt to control the free access to information on the internet for purposes like dissent suppression and control. In response, research has been dedicated to propose and implement censorship circumvention solutions. One approach to circumvention involves the use of steganography, the process of embedding a hidden message into a cover medium (e.g., image, video, or audio file), such that sensitive or restricted information can be exchanged without a censoring agent being able to detect this exchange. Stegozoa, one such steganography tool, proposes using WebRTC video conferencing as the channel for embedding, to allow a party within a restricted area to freely receive information from a party located outside of this area, circumventing censorship. This project on itself, is an extension of an earlier implementation, and it assumes a stronger threat model, where WebRTC connections are not peer-to-peer but instead mediated by a gateway server, which may be controlled, or influenced, by the censoring agent. In this threat model, it is argued that an attacker (or censor) may inspect the data being transmitted directly, but has no incentive to change the video data.

With our work, we seek to challenge this last assumption, since many applications using this WebRTC architecture can and will in fact modify the video, likely for non malicious purposes. By implementing our own test WebRTC application, we have shown that performing video re-encoding (that is decoding a VP8 format video into raw format and then back) on the transmitted data, is enough to render an implementation like Stegozoa inoperable. We argue that re-encoding is commonly a non-malicious operation, which may be justified by the application setup (for example to perform video filtering, or integrity checks, or other types of computer vision operations), and that does not affect a regular non-Stegozoa user. It is for this reason, that we proposed that re-encoding robustness is a necessary feature for steganographic systems.

To this end, first we performed characterization experiments on a popular WebRTC video codec (VP8), to understand the effects of re-encoding. Similarly, we tested the effects of this operation when a hidden message is embed in a similar fashion to Stegozoa. We were able to show that, DCT coefficients, which are used commonly as the target for message embedding, change enough to cause loss of message integrity due to re-encoding, without the use of any error correction. Our experiments showed that higher frequency Discrete Cosine Transform (DCT) coefficients are more likely to remain stable for message embedding after re-encoding. We also showed that a dynamically calculated embedding space (that is the set of coefficients that may actually be used for embedding), akin to

Stegozoa's implementation, is very likely to be different after re-encoding, which creates a mismatch between sender and receiver.

With these observations, we then sought to test a more robust implementation for embedding. To do so, we combined the usage of error correction (in the form of Reed-Solomon codes), and a static embedding space. We showed that message re-transmission (that is, embedding in multiple frames) and error correction are enough to send a message that will be received correctly. Our experiments showed that this can be used as a low-bandwidth non time-sensitive channel for covert communications. Finally, we combined our results to provide a set of guidelines that we believe are needed to implement a WebRTC based, VP8 encoded, censorship circumvention.

Acknowledgements

I would like to thank my supervisor, Professor Diogo Barradas, for his input, contributions, and guidance that shaped and made this Thesis possible. I am thankful for the many hours that were spent in discussion through the life of this project, as well as the trust that was put on me to carry it out to completion. I am also thankful to Professors Florian Kerschbaum and Mohammad Ali Salahuddin for agreeing to be the readers of my thesis and providing valuable feedback to this work.

A mis padres Ricardo y Maria, y a mis hermanas Claudia y Nataly, les agradezco por haberme acompañado siempre. Su ayuda y empuje me han hecho llegar hasta donde estoy hoy. Gracias por nunca dudar mis decisiones y siempre haberlas apoyado.

To my parents Ricardo and Maria, and my sisters Claudia and Nataly, I thank them for being with me through this journey. They helped and pushed so I can be where I am today. Thanks for always trusting and supporting my choices.

The journey of writing this Thesis would have not been possible without the many people that supported me, academically or otherwise. Thanks to James, Stefen, and Chris, some of my oldest friends, who I can always rely on, and who I've had the luck to share many a Christmas with. To Devon, Maria, Colin, and Sasha, who I know since undergrad at RIT, and can chat with everyday. To Mattie, Gen, and Albert, for the many lunches we've shared together, and all the support we've had for each other. To all the amazing people I've met through Mambo club, in particular Ciara, Art, Grace, Andrée, Piyush, and Anna-Maria. To my labmates in the CrySP lab, specially to Vecna, Thomas, John, Lucas, Ru, Jason, and Bailey.

Dedication

This is dedicated to Alistar and Yaku, whose memories I keep alive.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

DCT Discrete Cosine Transform xix, xx, xxii, xxiii, xxix, xxx, xxxvi–xxxviii, xl, xliv, xlvii, xlviii, lvii, lxviii

LSB Least Significant Bit xvii, xix

MCU Multipoint Control Unit xxvii, xxviii

ML Machine Learning xx

P2P Peer-to-Peer xvii, xxvi, xxvii, lxv

PSNR Peak signal to noise ratio xix

SFU Selective Forwarding Unit xxvii, xxviii

SSIM Structural index similarity xix

STC Syndrome-Trellis Coding xvii, xx, xxxvi, xxxviii, xxxix, l, li, lvii, lix, lxvi

Chapter 1

Introduction

1.1 Overview

Internet censorship is an increasingly pressing issue, with many authoritative governments around the world actively implementing stringent Internet monitoring and blocking controls [14]. The motivations that lead these governments to deploy tight surveillance and censorship policies are generally tied to a desire to control the free flow of information and exchange of ideas in the Internet [70] towards stifling political dissidence and making it hard for other nations to build awareness over potential crises happening within the country [3, 60].

Throughout the past few years, we have witnessed multiple techniques being employed in censorship apparatuses. These have ranged from the simple usage of keyword-based filters [11] or the outright blocking of Internet destinations [54], to the more intricate monitoring of social media platforms [71], web and games' chat-rooms [12], the throttling of connections to some Internet destinations [2], or the blocking of specific Internet protocols (e.g., those aimed at increasing users' privacy when accessing the web, like Tor [16], or those specifically crafted to avoid censorship, like Shadowsocks [1]).

As monitoring and blocking techniques become more sophisticated, activists and researchers strive to develop new tools whose primary goal is to enable users to circumvent censorship, ensuring oppressed citizens' unrestricted access to online content and the freedom to communicate with one another. Thus, the interaction between censorship techniques and censorship circumvention tools is typically perceived as a cat-and-mouse game, where censors attempt to block novel circumvention tools, and the developers of such tools continually improve those tools, hoping to be one step ahead of censors' capabilities.

Given the above, it is clear that circumvention tools must operate covertly to successfully evade the scrutiny of state-level entities, safeguarding users against potential repercussions. Nevertheless, many of such tools face significant challenges, as they maybe challenging to deploy, or prone to detection through a variety of attacks based on deep packet inspection [16] and statistical traffic analysis [68].

A relatively recent approach for building censorship-circumvention tools that address all of the above challenges is that of generating covert channels over video-conferencing applications. In a nutshell, these tools are perceived as a) being easy to use, since they piggyback data on applications that typical Internet users are familiar with; b) easy to deploy, since they do not require users to set up or interact with any additional infrastructure (all they need is a third-party peer located outside the censored region), and; c) resistant to detection, since they embed covert data in such a way that the traffic generated by video-conferencing applications retains its statistical properties. One common assumption performed by this kind of tools (e.g., DeltaShaper [6] or Protozoa [7]) is that censors only possess the ability to monitor the encrypted traffic being generated by the video-conferencing applications, which is typically encrypted end-to-end, but not the contents of the actual media being exchanged.

In the current Internet landscape, however, these assumptions might not always hold. For instance, Protozoa leverages WebRTC [44] (a framework and set of protocols that enables the exchange of real-time media directly between browsers) to hide covert data in the end-to-end encrypted media exchanged directly between peers. To ensure quality of service requirements and apply other validation procedures, most WebRTC applications today (including popular ones, like Discord [74]) no longer default to connect users directly in a peer-to-peer fashion, but rather utilize media servers, named WebRTC gateways, that decrypt, process, and re-encrypt video streams as needed. It is fathomable that state-level entities might exploit the behaviour of these media servers, either by colluding with service providers or setting up their own in state-controlled media applications (e.g., WeChat [62] in China), to snoop into the media exchanged between peers and block those streams that are suspected of carrying covert channels.

Aware of this potential detection vector, Figueira et al. [22] leverage sophisticated video steganography techniques to embed covert messages in media exchanged over WebRTC. They develop a new tool, Stegozoa, which assumes that a censor might have control over a WebRTC gateway and inspect the contents of media streams. Their evaluation shows that, while Stegozoa's covert channel achieves a smaller throughput than previous tools, it can exchange covert data while resisting both traffic analysis and steganalysis attacks aimed at uncovering covert channels in the media stream.

Despite its advances on censorship-circumvention, one important assumption of Stegozoa is that the censor is considered to be a fully-passive adversary that can only inspect, but not modify, the contents of the media stream being exchanged between two communicating peers. One might then wonder whether Stegozoa can operate within a stronger threat model where the adversary is given the capabilities to manipulate the media being exchanged, e.g., by re-encoding or applying some filter over the exchanged media. Indeed, this is not a far-fetched assumption, given that even legitimate WebRTC applications already provide users with the option to apply image filters over the exchanged media.

In this thesis, we investigate the robustness of Stegozoa's covert channel to perturbations introduced on the media streams exchanged by the communicating peers. Beyond the analysis of tailored perturbations aimed at breaking the operation of Stegozoa, we present a study over the perturbations caused by seemingly innocuous re-encoding operations, performed by WebRTC gateways, which showcase the inability of Stegozoa's implementation to deal with trivial media stream manipulations. In addition, we analyze the steganographic primitives used in Stegozoa's implementation towards understanding why these are unable to survive the above perturbations. Finally, we suggest an alternative steganographic embedding approach, leveraging error-correction codes, that allows for the development of a covert channel that is is robust against media stream manipulations.

1.2 Contributions

Towards carrying out the investigation laid-out in the previous section, this thesis provides the following list of technical contributions:

- The implementation of a laboratory testbed, consisting of a custom WebRTC gateway and a set of WebRTC media peers, which can be used for assessing the effects of media manipulation operations in the contents of WebRTC encoded media.
- A set of experiments that showcase Stegozoa's lack of robustness to media manipulation operations performed by a WebRTC gateway.
- An experimental characterization study of the effects of re-encoding operations on the content of WebRTC video streams' encoded representations, when leveraging the popular VP8 video codec.
- An exploration over the use of different error-correction codes towards the development of a robust, low-bandwidth, steganographic scheme over VP8 that can resist media re-encoding.

1.3 Thesis Outline

This thesis is organized as follows. In Chapter 2, we provide background knowledge on related censorship-circumvention tools, as well as on steganography and steganalysis primitives. In Chapter 3, we showcase Stegozoa's lack of robustness to media manipulation operations. Chapter 4 presents our characterization study on the effects of media re-encoding operations. Chapter 5 describes our experiments towards the design of a robust steganography scheme over WebRTC that can withstand manipulations of the video stream. Lastly, in Chapter 6, we summarize our main takeaways, detail the limitations of our approach, and point towards compelling directions for future work.

Chapter 2

Related Work

In this chapter we present relevant work, and other background information that we deem necessary for the rest of our work. We start in Section 2.1 discussing censorship circumvention and tools that have been implemented for this goal. Next, in Section 2.2, we delve into steganography, a process that is used to embed hidden messages in a cover medium and which can be used to implement censorship resistant technologies. We also discuss work on attacking steganography to detect it, hinder it, or otherwise disable it in this section. Finally, on Section 2.3 we provide an overview of some existing technologies that we used in our research and experiments.

2.1 Censorship Resistant Technologies

As we had originally discussed in Chapter 1, there are ever increasing efforts from state-level agents to control free flow of information on the Internet. In response to this adversarial behaviour, a number of censorship resistant and avoidant technologies have been proposed and studied over recent years. In this section we explore seminal, and state-of-the-art solutions that seek to address this issue.

Tor, a project aiming to provide private, and anonymous access to the Internet [1], is a widespread and highly supported tool for censorship avoidance [36]. The Tor network uses a series of volunteer relays (or nodes), in conjunction with onion encryption, to anonymously connect a client with server. Entry nodes are publicly listed in the Tor directory, making them easy targets for blocking for state level agents [76]. To address this issue, the Tor

[1]The Tor Project: `https://torproject.org`

project also supports "bridges", unlisted entry nodes, that can allow censored clients from accessing the network, under the assumption that a censor would not be able to block all bridges [2].

It has been shown, however, that through methods such as traffic analysis a censor may be able to correlate a client's Tor traffic and block their access to the network [13] (e.g., by ISP level blocking of a bridge). In response, multiple solutions have been proposed to protect clients from this kind of censorship and analysis. Pluggable transports, are a category of circumvention tools supported by the Tor project, four of which are readily accessible [3]:

- **obsf4.** Makes Tor traffic random such that an observer may not be able to determine its type.
- **meek.** Traffic is made to look as if the client was accessing a popular website instead of the Tor network.
- **Snowflake.** Uses volunteer proxy peers to route traffic to make it look as if it was a video call.
- **WebTunnel.** Makes connection to the Tor network appear as if the client is using a website through HTTPS

Other protocols have also been implemented to attempt to address this problem. Projects such as SkypeMorph and StegoTorus aim to mimic other protocols (e.g., Skype and HTTPS) to prevent traffic from being detected by a censor [48, 75]. ScrambleSuit attempts to obfuscate communication using traffic morphing techniques, making the communication harder to fingerprint [77]. Fifield et al. [21], proposed "domain fronting", a mechanism to make HTTPS requests appear to an outside observer to be directed to a "free" domain, while internally, in the HTTP Host header, the true "blocked" domain is placed. This method can be used to reach otherwise blocked nodes.

These systems are not infallible however, and developments in more sophisticated attacks show the ongoing battle between circumvention and detection. Work from Houmansadr et al. has shown for example that "unobservable" solutions such as those proposed above that rely on protocol mimicking can be detected even by a weak adversary [30]. He et al. showed an attack that attempts to determine the type of content being accessed [28]. Habibi Lashkari et al. proposed using timing features to characterize Tor traffic [26].

[2]What is a bridge?: https://support.torproject.org/censorship/censorship-7

[3]CIRCUMVENTION: https://tb-manual.torproject.org/circumvention

Outside the realm of Tor, we find solutions that also seek to enable free, uncensored communication on the Internet. DeltaShaper from Barradas et al. encodes a covert message into a barcode like pattern and then superimposes it on a cover video [6]. This video is then transmitted through a conferencing application such as Skype. While it successfully enables a covert channel, DeltaShaper is weak to image analysis since a clear perturbation is added to the video.

Barradas et al. later propose Protozoa, a system that creates a covert communication channel by replacing the encoded media data within WebRTC video calls with covert resources (concretely, enabling the covert exchange of raw IP traffic) [7]. This solution is vulnerable to man-in-the-middle attacks, where adversaries controlling media relaying servers would be trivially able to determine that the data being exchanged does not follow the expected WebRTC media format. As we discuss in Section 2.3.2, WebRTC is likely to follow a non Peer-to-Peer (P2P) architecture, so the party in control of the WebRTC application (adversarial or otherwise) would be able to identify Protozoa.

Figueira et al. proposed Stegozoa as an improvement on Protozoa [22]. For their implementation, they consider a stronger threat model where WebRTC connections are not peer-to-peer but instead over gateways that may be controlled by an adversary, and they may have transparent access to the data being sent across. In this scenario, the adversary is able to perform traffic analysis and steganalysis on the video stream shared between the two parties. In the case of Protozoa, the adversary would be able to easily determine that the video stream packets have been replaced with the covert data. Stegozoa instead, uses steganography to embed the stego-data by utilizing a combination of Syndrome-Trellis Coding (STC) and Least Significant Bit (LSB) shifting on the client side into the client video stream before it is sent to the gateway server. The data is then recovered on the other end by performing the reverse process. With this approach, Stegozoa was able to resist traffic analysis and steganalysis (achieving close to random guessing in both cases against state of the art network traffic and steganalysis classifiers). Their prototype achieved an average throughput of 8 to 10 Kbps, usable for low bandwidth applications (e.g., downloading tweets and news articles), with a relatively low latency. This implementation achieved usability, covert data can be exchanged between parties with an adversary in the middle with complete access to the data stream, and portability, independent of the target WebRTC platform. Stegozoa, however, remains vulnerable to various intentional and unintentional attacks from an adversary in control of, or at least a high level of influence over, a WebRTC gateway which we discuss in further detail in Chapter 3.

Jia et al. also take advantage of multimedia channels for sending secure covert messages [37]. Their goal is to produce an audio-based steganographic protocol that is not vulnerable to content mismatching attacks, such as those identified by Geddes et al. [24].

In this type of attack, significant differences between covert content and the expected “normal” content are apparent/distinguishable by a classifier. For their case in particular, this attack is possible due to variable bitrate encoding used by most audio/video conferencing applications (such as Skype). They solve this problem by using a generative model that shapes a stream of audio traffic that preserves the timing features of a real conversation between two peers. The tool, Voiceover, was shown to be more resistant to classification compared to cases where the traffic is not shaped to the expect “real” content, with goodput of close to 60 bytes per second. The relatively low goodput, which is better suited for low bandwidth transmissions, can be attributed to the trade-off needed to achieve higher robustness, and the audio medium being used.

Many of the works referenced in this section rely on steganography (video, audio, image, or others) when building these censorship circumvention tools. We further analyze steganography techniques and implementations in the next section.

2.2 Steganography

In this section, we review the current state-of-the-art on steganographic techniques and implementations. These methods allow for concealed embedding of messages within otherwise innocuous mediums (such as image, video, audio). We will then detail how these techniques can be used as the building blocks for censorship resistant communication. In Section 2.2.1 we explore the development and the state of the art in image steganography. Due to the relative closeness of both mediums (video and image), the concepts from this section can be used as the foundation for video steganography. In Section 2.2.2, we build upon those ideas by presenting work that implements them in the video domain, as well as showing other techniques unique to the domain. This set of methods are the most directly applicable to the scenario envisioned in this thesis. Finally in Section 2.2.3 we discuss the concept of robust steganography, that is the ability to resist various passive and active attacks or transformations to the medium used for concealment, and existing implementations. This robustness is what allows for the creation of censorship resistant communication channels, the principal goal of this work.

2.2.1 Image steganography

As implied by the name, this type of steganography uses images as the medium to conceal a message. In this work, we use the term cover image (or later cover video) to refer to unmodified images that are used as the basis for message embedding. These images can be ordinary repurposed images (e.g., photographs or digital illustrations) [33], or entirely synthesized for steganography purposes [42] [59]. Once one or more steganographic methods are applied on these cover images, we obtain a resulting image that has an inserted message that can be extracted by a different party. We use the term stego-image (or stego-video) to refer to this modified image.

The goal of image steganography is to create a stego-image carrying the hidden message that is indistinguishable (visually, statistically, or otherwise) from a cover image. Or stated differently, for the purposes of our work, create images such that an adversary could not accurately determine with high enough confidence that a message is embedded on any given image (stego-image or otherwise). For censorship resistant communications, this quality means that a censor may not be able easily detect and stop communication between covert parties, or if attempts to do so it may incur a high collateral damage cost. Peak signal to noise ratio (PSNR) and Structural index similarity (SSIM) are two common metrics to measure imperceptibility in image steganography [65].

One of the approaches towards undetectable steganography focuses on minimizing the distortion caused in the stego-image during the embedding of stegobits, such as those by Holub et al., and Pevný et al. [29, 57]. Practical implementations of steganography are not likely to create stego-images so distorted (or otherwise "off-looking") that they could be easily identified by a human observer. These implementations may however, add enough noise (in this case noise can be understood as a kind of distortion) to the stego-image may be discovered by statistical, or machine learning models. Pevný et al. proposed a method that used Markov chains at the spatial domain to identify stego noise caused by LSB steganography [56]. More sophisticated methods such as neural networks have also been used to detect stego noise. For example, Qian et al., used deep learning to extract features from stego-images with the goal of steganalysis [58]. We discuss more such methods of detection in Section 2.2.4.

Image steganography methods can be classified according to the domain being targeted, either spatial or transform. In the spatial domain, embedding of the hidden message is done directly on the pixel values representing the image. The domain transform on the other hand, is reached after performing a Fourier Transform (or more commonly Discrete Cosine Transform (DCT)) to a spatial image, and is commonly used for compression, prediction, transmission tasks. Domain transform steganography covers methods that embed the message after the image has gone through this transform.

Spatial domain. In the spatial domain, an earlier implementation by Pevný et al. aimed to minimize the noise within this domain [57]. This method works by creating distortion cost tables from modifying a given pixel, using previous steganalysis work from Pevný et al. [56] to compute them. Hussain et al. [32] used a hybrid adaptive embedding scheme, with choice of least significant bit or rightmost digit embedding, to minimize distortions during the embedding process. Yang et al. [80] proposed a method using generative adversarial networks to create a stego-image while minimizing embedding costs.

Transform domain. There is also a wide diversity of image steganography methods in the transform domain. Fakhredanesh et al. [19] implemented a steganography method for embedding after a discrete wavelet transform to a spatial image. In the realm of discrete cosine transform, we have for example work from El Rahman [17], in which the stego message was embedded in the least significant bits of medium and high frequency DCT coefficients, and Zhu [83], utilizing DCT coefficients to embed a message that is robust to JPEG compression. Similarly, Evsutin et al. [18] provided a scheme that worked after a discrete Fourier transform, and could potentially be used for other tranforms.

A parallel problem to calculating correct embedding costs, that is, determining the impact to image quality of using a given bit for embedding, and minimizing it, is using this information to efficiently embed the stegobits. Filler et al. [23] proposed a method to efficiently embed bits in the cover image while minimizing the costs (as determined by a cost function or matrix). This methodology relies on syndrome-trellis codes (STC) and the Viterbi algorithm. Holub et al. [29] proposed UNIWARD, a universal distortion cost function for image embedding in arbitrary domains. At a high level, this method uses directional filter bank decomposition to identify image areas that are difficult to statistically model in one or more directions (for example, non smooth areas with texture or noisy regions) and assigns them a low distortion cost, while doing the opposite for smoother regions. They then use Syndrome-Trellis coding to efficiently embed stegobits using the calculated costs. Formally, this method uses a directional filter bank composed of three kernels that finds smoothness in horizontal, vertical, and diagonal directions, to produce differential residuals. The cost function is computed as the sum of all these residuals between a cover image and a stegoimage, represented in the spatial domain (pixels). Notably, this method can also be applied at the transform domain by first converting JPEG representation to pixel representation. As previously mentioned, the output of this cost function is used by the STC algorithm to efficiently embed the bits of the secret message. UNIWARD, in its three varieties (spatial, transform, and side informed transform domains) showed steganalysis robustness against Machine Learning (ML) classifiers when compared against state-of-the-art steganographic methods at the time.

2.2.2 Video steganography

Video steganography is the next logical step from image steganography. In a simplified model, a video could be described as a series of still images (or frames) stitched together over time. A naive implementation could then use one of the previously shown methods to embed a message, or part of it, into each "frame" of a cover-video to produce a stego-video. An outstanding feature that is revealed from this model is the increased space afforded for embedding. A large message, such as a file, could be transmitted this way, split over as many frames as needed. Furthermore, in a scenario where the video has an indefinite length (e.g., a video call), data could be transmitted arbitrarily and over time. We now have access to even more use cases, such as covered web navigation and hidden back and forth communication.

Realistically, most forms of video do not quite work in this way, and video steganography has to account for these nuances. For example, videos transmitted over the wire may go through compression (different to image compression), encoding and re-encoding. Similarly, videos are seldom composed of solely key-frames (standalone reference frames), but instead may use one or more forms of prediction to generate "in-between frames" from those key frames. These techniques aim to reduce the space used by the video, but for the purposes of steganography also reduce the available embedding space.

Even with this distinction between both mediums, the goals stay relatively similar. Produce a stego-image/video with minimal visual or statistical distortion (to minimize detectability), with optimal embedding capacity, and robust to intentional and unintentional attacks. For this reason, and due to the closeness of mediums, video steganography can leverage some of the methods used in image steganography, such as STC encoding, as for example in Figueira et al [22].

Video steganography techniques can be divided into three categories: pre-embedding, intra-embedding, and post-embedding, depending on the embedding domain [43].

Pre-embedding. In this category, embedding is done to the uncompressed video (either on the spatial or transform domain). Embedding at this stage means the scheme is independent from any compression or other algorithms that may be used to process the video before transmission. On the other hand, since the embedding is happening before (likely lossy) compression, some of data may be lost. These methods then require some level of redundancy or other recovery methods to guarantee the integrity of the message. In the spatial domain, an early contribution by Cetin and Ozcerit [10] combined least significant bit embedding with spatial histograms to select the most appropriate pixels for embedding. Hu et al. [31] implemented a method to embed a similarly sized video into a raw video stream, using the four least significant bits of pixel values. As for the transform domain, work from Patel et al. [55] first performed a wavelet transform on the cover video, embedded the message in the least significant bits of the transformed video, and then used the audio channel to provide metadata on the embedding. Mstafa et al. [51] used DCT coefficients as the target with embedding, in combination with Hamming and BCH codes for added security and robustness.

Intra-embedding. This type of embedding happens while the video is being encoded (such as by a codec like VP8 or H.264), such as during motion estimation, or frame prediction. A clear advantage of this type of embedding is that it happens after compression, or other lossy operations are performed on the video, meaning there is a lower chance for the stego-message to be corrupted. However, since most codecs are highly efficient and tend to remove a large amount of redundancy from the video, this leaves a much more reduced embedding capacity in the medium compared to pre-embedding. Figueira et. al [22] embedded messages in the DCT coefficients (except for the lowest frequency one) of a VP8 encoded video which was then retrieved during decoding by the recipient. Cao et al. [9] developed a scheme for embedding using the H.264 codec based on intra-prediction modes.

Post-embedding In this embedding type, the message is inserted on the bitstream product of a cover video that has been processed by a codec, or otherwise has been compressed. The benefit of this type of embedding is that this is the last stage before a cover video is transmitted providing the highest degree of integrity, and it may not be codec or implementation specific. However, it is important to note that since heavy modifications to the bitstream may cause it to no longer be properly formatted, an adversary could identify the steganography attempts. Barradas et al. [7] proposed a method that replaced the video stream of a conference call (through an app such as Skype) with the hidden data wanted to be transmitted. While highly efficient, this method is vulnerable to a man-in-the-middle that may access and try to decode the "video" since it no longer follows the format. Xu et al. [78] proposed a method for taking advantage of the binary encoder of H.264/AVC videos to encode a message in the encrypted bitstream without disrupting the video format.

In the previous sections we have discussed some examples of video and image steganography. Various times we've mentioned robust steganography methods, which imply a level of resistance from various types of attacks and analysis of scheme. In the next section we discuss the goals of robust steganography and present how it is implemented.

2.2.3 Robust steganography

Robustness in steganography refers to the ability of the encoding mechanism to resist various changes (malicious or otherwise) that may occur to the communication channel. Ideally, a cover image or video should not suffer any variations during transmission and recovery. This scenario requires no further modifications or alterations to be applied to the media during exchange between involved parties. For example, by sharing a stego-image or video by USB stick from a friend to another. However, in many realistic scenarios, the medium is likely to be compressed, rotated, have noise added to it, cropped, etc. Scenarios like this include posting media to social media or using web conferencing software. In any case, they involve one or more parties in the middle of the communication (be it a server, the user's browser, or the network structure) that may alter the media. For the purposes of our work, we are interested in the model where an opposing third-party agent may attempt to halt or disrupt these communication channels and thus breaking the bridge between the two parties. Robust steganographic methods that may resist such attacks are relevant to our work.

JPEG compression robustness has been widely studied and improved upon. Most of these methods embed their payload at the transform domain (e.g., DCT), as opposed to the spatial domain (e.g., least significant bits of a pixel).

Singh et al. [67] proposed a model to hide a secret image (akin to a watermark) within a cover image by first using Arnold Shuffling on the payload, and using two pseudo random sequences to determine which DCT coefficients to change. This approach showed some robustness, noise and filtering, and while the original message is not completely recovered, it still recovers enough of the embedded watermark.

Qian et al. [59] approach generates synthetic texture images embedded with the desired message from cover texture images. The stego textures can then be used in larger image compositions (for example as background grass). These textures showed resilience to various levels of JPEG compression.

In a similar approach, Tao et al. [72] aim to generate a pre-image, from a base cover image, that when compressed will have the same DCT coefficients as if the payload had been embedded in the base cover image. This method requires to know details about the compression used by the transmission channel and may not be resilient to other kinds of attacks.

2.2.4 Attacking steganography

In this section, we discuss topics relating to steganography detection, destruction, sanitization, and other related attacks. We frame these attacks from the point of view of censoring agents whose main goal is to disrupt the covert communication between two parties. Our aim is to provide a high level understanding of the capabilities of said agents. We divide this section into steganalysis, research in the area of detecting hidden communications, and steganography destruction, attacks that aim to "clean" steganographic channels while preserving the original medium (image or video).

An active warden looking to prevent or disrupt hidden communications may seek to first detect when said covert channels are being used, and then retaliate as they see fit (this can include for example, blocking communications between parties, or disallowing upload of images/videos with embedded steganography to social networks). Here, we provide a look at the state-of-the-art in steganography detection, a process commonly known as steganalysis.

Steganalysis. Current developments in detecting image steganography rely on neural network models. Boroumand et al. [8] proposed SR-NET, which used deep residual networks for image steganalysis, both in the spatial and transform domain. You et al. [81] implemented a method based on how stego noise affects different areas of the image, creating different residuals in the sub-regions of the image. The proposed method uses a Siamese CNN on two different sections of the image to determine their similarity and identify the use of steganography. Their performance is comparable to that of SR-NET, while minimizing the number of parameters.

Li et al. [41] adapted XU-CNN [79] for steganalysis. They used "diverse activation modules" in their network, which activate their outputs differently and then concatenates them for subsequent layers. They showed better performance using multiple parallel modules or subnets as opposed to one net with multiple filters. it reliably detects common adaptive steganography methods including S-UNIWARD [29], HILL[40], CMD-HILL [39].

Sanitization. An active warden may choose to not perform steganalysis, but instead sanitize every input (in this case images or video) they receive from a user, effectively destroying hidden messages if those exist, while preserving the perceived quality of the cover medium experienced by users that do not engage in covert communications. The goal of this approach is not to detect the presence of steganographic marks but instead damage them enough that they cannot be easily recovered by the intended party. Zhu et al. [84] developed a neural network to attempt to remove steganographic content from images uploaded to social networks. Similarly, Zhu et al. [85] furthered this work by implementing a scheme that was able to also remove information that was robustly embedded in the cover.

In this section we have discussed various methods of steganography for both images and videos. We also showed possible attacks on this methods, that aim to either identify their presence to out right block the communication channel, or instead attempt to remove it, disrupt, or sanitize medium such that the message is lost and unrecoverable once it reaches the receipient. In the next section we move away from steganography and instead focus on other complimentary technologies that will also play a role in our research.

2.3 Overview of Related Technologies

In this section we provide relevant details for building a basic understanding over the technologies we will heavily reference throughout the following chapters. Specifically, Section 2.3.1 discusses details on WebRTC and how this framework can be used for supporting video calls and conferencing. Section 2.3.2 discusses different WebRTC deployments from an architectural perspective. Lastly, Section 2.3.3 describes the VP8 video codec, which is the default choice for video encoding and decoding operations on WebRTC media streams.

2.3.1 WebRTC

WebRTC is an open-source project that provides real time communications (RTC) for web browsers and mobile applications through a set of APIs [44]. Most modern browsers natively support WebRTC, and allow users to exchange real-time media. The WebRTC protocol can be divided in four phases that facilitate setting up this interaction.

The first phase is concerned with signaling between peers. The goal of this stage is for a peer to find and start negotiating a session with another peer. To do so, an initiating peer must start a discovery process to find available peers, usually with infrastructure hosted by the application being used. A signaling protocol is then used to mediate session details (such as supported codecs, bandwidth limits, IP addresses). While there is no standardized protocol, Session Initiation Protocol (SIP) is a common choice for this task [63]. Regardless of the chosen method, the details required for establishing a connection are set at the end of this phase.

Next is the connection phase. In this phase, the peers seek to establish a communication channel between them, using the details gathered in the previous step. WebRTC uses the

Interactive Connectivity Establishment (ICE) protocol to enable this process. For a client sitting behind a NAT network, the ICE protocol may rely on the Session Traversal Utilities for NAT (STUN) protocol to facilitate NAT traversal. The STUN server (outside the NAT network), allows the client to self discover the port and IP address assigned to them by the NAT, and open other ports needed for communicating with their peer [47]. In cases where the clients cannot possibly connect directly (for example due to restrictive NAT configurations), the Traversal Using Relays around NAT (TURN) protocol is employed. A TURN relay serves as an intermediary between the clients, where the communications for each client appear to be incoming from the relay [46]. In this configuration, the TURN relay does not have transparent access to packets being exchanged, it only serves as forward point between peers. The WebRTC application commonly provides the needed STUN servers and TURN relays.

The next phase is concerned with securing the data exchanged via WebRTC. Specifically, to secure the media transmitted through the negotiated channel, WebRTC uses the Secure Real-Time Protocol (SRTP) and Datagram Transport Layer Security (DTLS). The SRTP protocol is an extension of the Real-Time Protocol (RTP), and is used to encrypt packets between peers. This protocol is required to use DTLS to negotiate key exchanges for the session. Concretely, DTLS is a UDP protocol based on streaming TLS which allows for private low latency communication channel [61].

The last phase is concerned with the actual communication between peers. At its core, it leverages the RTP protocol for the exchange of messages (which are encrypted through SRTP). In addition, the RTP Control Protocol (RTCP) is used to exchange call metadata with the goal of providing feedback on the quality of the ongoing call to all peers [64]. Stream Control Transmission Protocol (SCTP) may also be used to securely transmit arbitrary data between peers, using DTLS again for encryption negotiation.

This is a high level description of WebRTC for enabling a call between two or more peers. In the following section, we go over common architectures for WebRTC services.

2.3.2 Architectures for WebRTC services

In its original specification, WebRTC was designed to operate in a P2P architecture between clients. Coined the RTC trapezoid, the architecture would have a web application mediate path signaling between two (or more) peers. Once connection details are known to each party, a media path is created between each peer, and data is transmitted using the provided APIs. Each peer is in charge of encoding and decoding incoming and outgoing

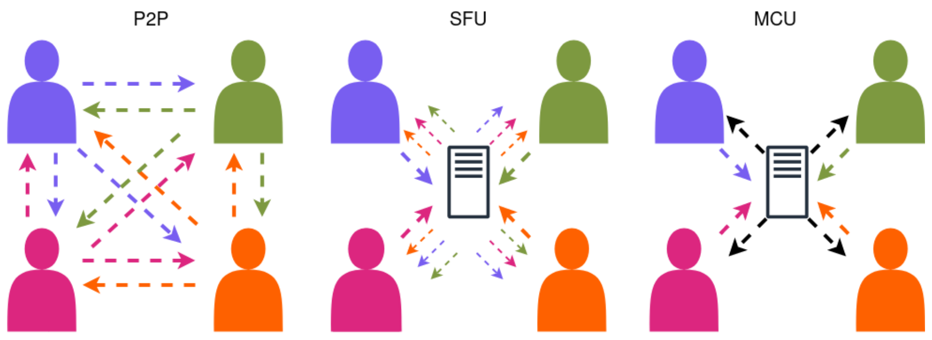


Figure 2.1: WebRTC architecture for P2P, SFU, and MCU for a web call with 4 peers.

media respectively as needed. Transmission of media (voice, video, or generic) carries on between the peers without the involvement of the the third-party web application.

In practice however, it is unlikely that a WebRTC application is implemented in a true P2P architecture due to various constraints. Scalability is one such limitation. In the trapezoid architecture, each peer is required to maintain a media channel with each other peer (for n participants, $n - 1$ channels are setup by each peer) as well as many media encoder/decoders instances to sustain the mesh-like topology. For instance, Jansen et al. [35] studied the performance of WebRTC video calls as more peers were added and observed that data rates for 4 peers were up to 3 times higher than 2 peer connections. To address this issue, WebRTC applications instead use centralized solutions such as SFU and MCU.

In these architectures, the peers do not connect directly with one another, but instead establish a peer connection with a central server. In the case of an SFU-based system, a central server receives one or more data streams (with varying qualities) from each peer, and selects which per-peer stream to forward. MCU-based systems, on the other hand, receive the streams from all participants, mix them into a single data stream (e.g., combining all individual peer video streams into a single video stream), and forwards it to every peer. Also, while these architectures were designed with scalability in mind, they can still be used when a call is held between only two peers. Figure 2.1 shows a diagram of the three architectures.

Notably, what this implies in both non-P2P cases, is that the central server will typically have transparent access to the data being transmitted, since the encryption channel is now established between each peer and the server, as opposed to peer to peer. Due to the inherent lack of end to end encryption, a new WebRTC API, Insertable Streams, has been implemented. This feature would allow for pre-processing of raw video/audio before it is sent to the server. Using this API, the clients are able to add an additional layer of encryption on the media, guaranteeing end to end encryption between peers [4]. While a plausible solution, it is still up to the application provider to enable it in their setups, and it has not yet become standarized.

In Chapter 3, we further elaborate on how the SFU and MCU architectures become a vector for disrupting covert channels that rely on the video stream.

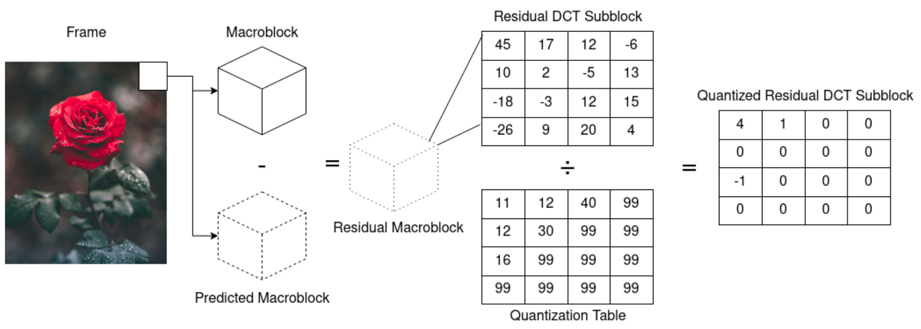
2.3.3 The VP8 codec

VP8 is an open source video compression format codec that aims for high compression efficiency and low decoding complexity [5]. Notably, it is widely used and supported by most popular browsers (e.g., Chrome, Edge, Firefox, Safari) [49]. Next, we address how VP8 represents images, describe the main image encoding abstractions used by VP8, and summarize its encoding and decoding procedures. For the purposes of this document, the following is a reasonably detailed explanation of the process which is intentionally simplified. We refer the curious reader to the official VP8 codec specification [5], which provides full details about the VP8 encoding procedure and mechanisms.

Colorspace format. The VP8 codec works with an 8-bit YUV 4:2:0 format. Put in simple terms, a given video frame is composed of three planes: a “luma” (Y) plane, and two “chroma” (U and V) planes. Each 8-bit pixel in a chroma plane, corresponds to a $2{\times}2$ block of luma 8-bit pixels, giving the luma component a higher resolution than the other two planes. The VP8 encoding algorithm splits a frame into *macroblocks*, where each macroblock is composed of 24 sub-blocks (and in some cases adds a 25th “virtual” block), divided as follows: 16 sub-blocks represent the Y plane, and 8 represent the U and V planes. Each of the sub-blocks corresponds to an array of 16 8-bit pixels from the uncompressed frame.

Image encoding abstractions. At the highest level, the VP8 encoder first generates an *intraframe* (also known as a *key frame*) for the given video input. This frame is then followed by any number *interframes* (also known as *predicted frames*). The difference between these two types of frames is that interframes depend on all previous frames (up to the last intraframe) to be reconstructed, whereas an intraframe is reconstructed using itself as an input only. This loop is repeated as needed until the input is completely encoded. The decoding process takes in these series of compressed frames, starting with an intraframe, to reconstruct the YUV format signal.

[4]https://jitsi.org/blog/e2ee/



Residual DCT Subblock

45	17	12	-6
10	2	-5	13
-18	-3	12	15
-26	9	20	4

Quantization Table

11	12	40	99
12	30	99	99
16	99	99	99
99	99	99	99

Quantized Residual DCT Subblock

4	1	0	0
0	0	0	0
-1	0	0	0
0	0	0	0

Figure 2.2: Frame to quantized sub-block decomposition.

Encoding process. The encoding process can be broken down into the following steps:

First, the type of frame is determined from the state of the encoder. As established previously, the first frame to be output by the encoder is always an intraframe. A frame can also be set as an intraframe if the algorithm determines it is required to convey large changes of information in the overall media (e.g., for a scene cut), or due to quality of service configurations (e.g., mandating the insertion of a key frame every n encoded frames).

Second, and regardless of the frame type, the macroblocks for each frame are processed in a similar fashion, by generating predicted macroblock. The prediction modes for a given macroblock are selected differently depending on whether these are included in an intraframe or an interframe. Macroblock prediction in intraframes rely on information about other macroblocks in the same frame, while macroblocks pertaining to interframes can rely on other macroblocks in the same frame, or across other interframes. After prediction is issued, the encoder then calculates a residue signal between each of the composing sub-blocks of the original and predicted macroblock.

Third, a DCT transform is then applied to each of the residue sub-blocks, resulting in an array of 16 coefficients, ordered from low to high frequency in a zig-zag order. After the transform is applied, the coefficient sub-blocks then undergo a quantization step. In this step, each of the coefficients is divided by the corresponding value on a *quantization table*, and then rounded down. It is important to note that the quantization factors used for quantizing higher frequency coefficients tend to be larger values, causing the high frequency coefficient values to often be rounded down to zero. Figure 2.2 presents an example of how a quantized DCT sub-block is calculated from the original frame during encoding following these steps.

Lastly, the resulting sparse quantized sub-blocks (whose coefficient values will mostly be zero), are then losslessly compressed using a boolean encoder. Each encoded frame output by the encoder is composed of an uncompressed header containing metadata for the whole frame (such as an indication of the frame type), plus the compressed representation of macroblocks that comprise a given frame.

Decoding process. The decoder is mainly responsible for reversing the steps of the encoding process so that video frames can be reconstructed. The main steps are as follows:

First, after the frame metadata is processed, the macroblocks are decompressed and handled in order. A predicted macroblock is generated using either the current frame data, or a previously decoded reference frame (depending on frame/prediction types).

Second, the residue signal of the macroblock is then generated and added to the predicted block. For each residue sub-block in a macroblock, the values are de-quantized, by multiplying them by the values in the quantization table.

Third, an inverse DCT transform is then performed on the de-quantized values and the residue signal is finally added to the predicted frame. (The de-quantization and transform steps may be skipped if a block is marked to have no non-zero values.) The product of processing all the macroblocks is a reconstructed YUV frame.

Lastly, the reconstructed YUV frame is passed through a loop filter to reduce blocking artifacts at macro and sub-block boundaries. Once a frame undergoes all the above steps, the decoder starts the decoding of the next frame in the pipeline.

2.3.4 Summary

In this chapter we first presented ongoing work related to censorship circumvention. We discussed various tools that in one way or another let a user within the area of influence of a censoring agent (such as a state level one) freely access information on the Internet. Then, we focused on one particular method used in such tools, steganography, that allows covert messages to be embedded within otherwise “normal” media, so that an adversary cannot detect the presence of such communication. We gave an overview of work in both image and video steganography, two common and closely related mediums. We also discussed robust steganography, that which can resist active and passive attacks, as well as known attacks to the proposed methods. Finally, we concluded this chapter by providing an overview of technologies we will be using in this work, namely WebRTC, a solution for web communications, and the VP8 codec, a common codec for WebRTC.

Chapter 3

Robustness Concerns for Video Steganography over WebRTC

In this chapter, we study the robustness of existing steganography schemes used to build covert channels over WebRTC-based media streams against adversaries that are able to intercept and modify encoded media streams in transit.

Section 3.1 provides details of our experimental testbed, encompassing the implementation of a custom WebRTC video conferencing service that will be used during our study. This service will leverage a deployment of a WebRTC gateway, allowing us to mediate the transmission of video streams and apply different perturbations in the encoded media comprising those streams.

In Section 3.2, we analyze the robustness of Stegozoa, a state-of-the-art WebRTC based steganography system, against the manipulation of encoded media. We demonstrate its drawbacks against adversaries with the ability to apply perturbations in media streams, and identify a set of characteristics that make the steganography mechanisms fueling Stegozoa inadequate to resist against such adversaries.

3.1 WebRTC Experimental Testbed

To study the implementation and applicability of different steganography schemes over WebRTC media channels, we implemented our own instrumented WebRTC video conferencing application. To this end, we leveraged the Kurento Media Server (KMS) project [20], a C/C++ library of tools and components for implementing video applications for WebRTC.

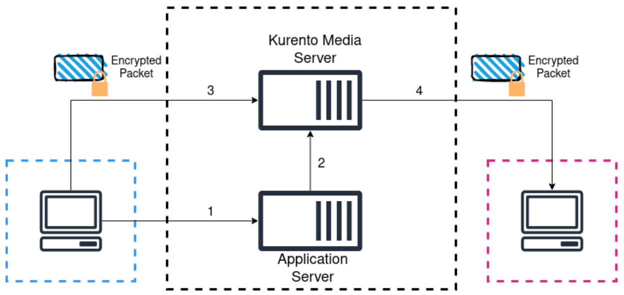


Figure 3.1: Architecture for our WebRTC experimental testbed in our two server configuration. 1) Client reaches the application requesting to start a call with a new user. 2) Application server calls the Kurento API on the media server to handle signaling and creates a communication channel with each of the clients. 3) Once the WebRTC call has been started, the client sends the encrypted video payload to the media server. 4) Server relays the media to the receiver, optionally performing operations on the underlying video.

KMS provides us with the necessary tools for implementing both an *application server* that handles connecting two (or more peers) into a video call, and a *WebRTC gateway server* that mediates the actual call, and through which the encoded media contents flow.

Figure 3.1 shows the overall architecture of our testbed, showcasing how each different server interacts with clients and one another. In the following sections, we provide implementation and configuration details for each of these servers.

3.1.1 WebRTC gateway server

At a high level, the gateway server is composed of interconnecting media elements (native and custom implemented) which create a media pipeline. Examples of two different pipelines are shown on the WebRTC server section in Figure 3.2. For our purposes, the most basic pipeline connects a “sender” WebRTC element with a “receiver” WebRTC element (for one way communication, or duplicated for a full-duplex channel). The “sender” element handles receiving the web packets from the video sending peer, and then passes encoded frames through the pipeline. The “receiver” on the other hand, accepts encoded video frames from the pipeline and handles their transmission to the receiving peer.

We note that while WebRTC is compatible with other video codecs (e.g., H.264, VP9), our server is configured to work only with the VP8 codec, which is the usual default choice. Besides, our WebRTC elements expect to receive and send VP8 encoded frames, but this does not imply that said frames will remain VP8-encoded at every stage of the pipeline, potentially undergoing other transformations and conversion between image/colorspace formats. This is an important feature which we will discuss later in this section.

We now describe important additions to the media pipeline, which can be made in an ad-hoc fashion to include customizable video filtering elements and video conversion layers.

Customizable filter element. The base pipeline configuration can be enhanced by adding more elements at any point in the pipeline. Natively, KMS offers media elements for various tasks such as video recording and playback, signal processing and computer vision operations, and video mixing and dispatching. Of particular interest for our research, it also offers customizable filter elements. These filters can be used for multiple purposes, including adding noise to a frame, perform live changes and retouches to frames, tracking objects (e.g., find a face in a video stream), super-imposing images (e.g., adding a hat to a head when detected). Kurento provides support for the implementation of these filters using the GStreamer and OpenCV libraries.

When connected to the pipeline, each filter element expects a video frame, applies a transformation, and then passes the frame along to the next element in the pipeline. By default, these filter elements expect to receive and transmit raw RGB frames (i.e., frames which are not compressed/encoded). These "expectations" are called element capabilities (or *caps*, for short). The caps of an element (which may include attributes such as resolution, image bit-depth, and audio and video codecs supported) can be understood as the super set of all possible configurations of media an element can handle (either incoming or outgoing). Ideally, each element in the pipeline has at least one configuration that intersects with the caps of another element connected to it, allowing for a seamless interaction between attached elements.

However, as we have mentioned before, our WebRTC elements expect to receive and forward VP8 encoded frames, whereas the filter elements expect raw frames. Thus, a specific conversion layer is required to handle mismatches that could be caused during the conversion of VP8 encoded frames to RGB and back.

Agnostic bin. To handle the above kind of mismatches between the capabilities of a sending (source) element and a receiving (sink) element, the KMS pipeline implements an intermediary mediation layer known an *agnostic bin*. The role of the agnostic bin is to convert the source media into a format the sink can handle (as determined by their capabilities), by performing operations such as transcoding. In the case of our application, when we connect a filter element between the two WebRTC elements, this layer first decodes the VP8 frames into raw (RGB pixel value) frames, and then moves it along the pipeline. Similarly, once the filter finishes processing a frame, the frame gets re-encoded back to VP8 format and passed along the pipeline. Notably, the transcoding mediation happens

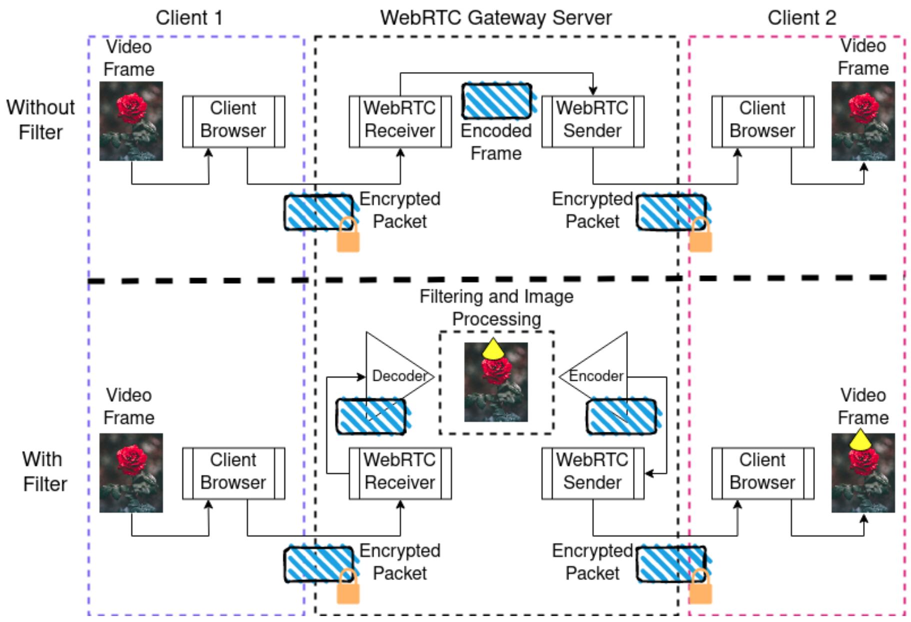


Figure 3.2: WebRTC Server pipeline with and without the use of pluggable filters. In this example, the filter receives a frame and adds a an icon on top of the image.

regardless of the actual implementation of the filter (or any other element) – it only depends on a capability mismatch between two connected elements. That is to say, an "empty" filter element can be added to the pipeline to trigger one or more iterations of VP8 transcoding.

The final server implementation consists of a two pipelines (fully duplexed, one for each peer in the call), that each have a WebRTC sender and a WebRTC receiver element. The caller pipeline also has zero or more filter elements in between the WebRTC elements. Figure 3.2 shows the process of sending a video frame from one peer through our WebRTC gateway server for both with and without filter scenarios.

3.1.2 Application server architecture

Complementing the gateway server, we also have an application server. The responsibility of this server is to allow users to initiate WebRTC calls with one another. The implementation is heavily based on the samples provided by Kurento. It is written in Java using the Spring Boot framework. At a high level, it allows a user to register, and then start a call request with another registered peer. The receiving peer may accept or reject the call. If accepted, the application server performs the necessary API calls to instantiate the pipelines in our gateway server, and performs the required signaling to start a WebRTC call (more details on this process on Section 2.3.1). Once the call is established, any of the peers may end the call and return to the original state. Implementation details are provided below.

Interacting with the gateway server. Interactions with a Kurento server (in this case our gateway server) are done through the Kurento Protocol. This protocol is built on top of JSON-RPC and uses WebSocket for transport. Kurento provides a set of Java APIs that encapsulates this protocol, and allows our server to perform a series of requests on the server to establish a call. When starting a call, the application server first requests a new set of media pipelines to be created (for outgoing and incoming video), and then requests for the WebRTC elements to be instantiated. Depending on the operation mode, the server may also request for a filter element to be created. Once created, another request is made to connect the elements for the connection. Finally, a request to start an SDP negotiation is sent, at which point the gateway server creates the video call for the two peers.

Modes of operation. For simplicity, when starting a new call, the user is provided with a choice of two modes of operation for the WebRTC call. A *clean-channel*, in which no filters are applied in-between the WebRTC media elements on the gateway server, and a *filter-channel* in which at least one filter element is placed in the pipeline. The clean-channel mode, as the name implies, emulates a regular clean call in which no processing is performed over the video stream. Conversely, the filter-channel mode will force the media stream to undergo (at least) a round of VP8 decoding and re-encoding. We note that these modes are exposed to the user simply for making it convenient for us to conduct our experiments. In a realistic scenario, the gateway server implementation may remove these controls away from the user, and instead automatically determine its mode of operation.

3.2 Attacking Stegozoa

As previsouly mentioned in Chapter 2, Stegozoa is a recent example of a censorship-circumvention tool that leverages image steganography techniques to embed covert data in the encoded video media exchanged in WebRTC calls. However, Stegozoa's design does not consider the potential manipulation of encoded media by WebRTC gateways (whether these are explicitly adversarial or not). Next, we present the implementation details of this tool, highlighting possible design aspects that might prevent the correct operation of covert channel if subject to manipulations of the encoded media. Then, we leverage our experimental testbed to showcase an attack on this system. We conclude this section by discussing our key findings.

3.2.1 Stegozoa implementation details

At a high level, Stegozoa instruments the VP8 codec library (libvpx) in the communicating endpoints' machine, towards embedding covert data in the encoded video frames exchanged between peers. Specifically, Stegozoa embeds steganographic messages in the least significant bits of *some* of the quantized DCT coefficients composing an encoded frame, which we call the *embedding space*.

In particular, Stegozoa filters out all 0 and 1 coefficients, as well as DC coefficients, from its embedding space. The reasons for this filtering are twofold: changing coefficients valued at 1 and 0 could greatly affect the quality of the compression algorithms, leading to severe changes on traffic patterns, and changing DC coefficients could bring severe changes to the reconstructed frames, potentially becoming a telltale sign that some steganography system had been put in place by the communicating peers.

Upon filtering out these specific coefficients, Stegozoa takes a parsimonious approach at covert data embedding. Namely, it leverages the STC algorithm to determine which of the coefficients in the embedding space should be used for message embedding (aiming to reduce perceptible changes to frames once these are reconstructed), and sets the least significant bits of each of those coefficients accordingly.

When the steganographically-marked encoded video is received by the communicating endpoint, the receiver performs the reverse process. In a nutshell, once having access to the quantized DCT coefficients during video decoding, STC is performed in a reverse fashion to determine the locations of the embedded message. The covert message is then extracted from the least significant bits of the coefficients identified as carrying covert data.

From the above explanation, it becomes clear that Stegozoa was not designed with the goal to resist an adversary that can directly manipulate the encoded video stream, and thus, thwart the process of identifying which STC-chosen coefficients carry covert data or change the potential contents of the covert message itself. In the next section, we construct a simple proof-of-concept attack on Stegozoa by leveraging our filter-channel construction introduced in Section 3.1, and which, amongst other image perturbations, will also force the WebRTC gateway to perform one round of VP8 decoding and re-encoding before sending the resulting media to a Stegozoa peer.

3.2.2 Breaking Stegozoa's covert channel through re-encoding

The goal of our attack is to showcase that simple media manipulation procedures can disrupt Stegozoa's covert channel, preventing Stegozoa peers from exchanging covert messages. We aim to keep these manipulations simple and imperceptible to regular users, since we believe real-world censors would be interested in preventing the communication of Stegozoa peers while not causing collateral damage to peers that are legitimately using the WebRTC video-conferencing service to communicate amongst themselves.

A basic Stegozoa setup. We used the instructions provided by the Stegozoa developers [22] to deploy the tool in our WebRTC testbed. To verify that our setup works as expected, we connected two Stegozoa peers through a clean-channel connection, and were able to verify that the Stegozoa peers were able to correctly exchange covert messages through the video stream. This confirmed that our deployment is functionally equivalent to that used in the original implementation of Stegozoa, and which was showcased using the Jitsi WebRTC application. Upon setting up an operational testbed, we then proceeded to test our first attack.

Attacking Stegozoa through video filters. Instead of a clean-channel, we connected a video filter element in our pipeline in order to apply various types of noise into the video stream (i.e., VP8 $\rightarrow$ Raw $\rightarrow$ Filter Raw $\rightarrow$ VP8). Our first attack was based on a blurring filter with a configurable window size, and which averaged the value of a given pixel with the pixels around it within the set window. A window of size 3 (3 pixels wide and high), for instance, would average the value of a given pixel with those immediately contiguous to it. Similarly, a window of size 1, would average the value of only the given pixel, essentially resulting in a non-operation. We initially designed this experiment to apply video blurring with an increasing size of pixel averaging windows, until we could verify Stegozoa was no longer able to establish a connection with a peer. However, we verified that using a window of size 1 was enough to prevent the Stegozoa client from establishing a connection.

Attacking Stegozoa through video re-encoding. In order to verify the simplicity of the above attack, we completely removed the filter functionality. Essentially, we set up a "do nothing" filter, which would simply cause the WebRTC gateway to trigger a round of decoding and re-encoding of VP8 video frames (i.e., VP8 $\rightarrow$ Raw $\rightarrow$ VP8). Interestingly, we verified that this operation would be sufficient to prevent Stegozoa clients from correctly reconstructing steganographically-embedded messages from the encoded video stream.

3.2.3 Why does Stegozoa break?

Since the original design of Stegozoa does not consider the construction of a covert channel that resists the manipulation of encoded media (e.g., against censors that attempt to normalize video frames' DCT coefficients LSBs), the results presented in the previous section are not exactly surprising. What is noteworthy, however, is that Stegozoa's embedding mechanism is trivially broken even under non-adversarial conditions. By studying the implementation of Stegozoa, and the details of the VP8 codec, we were able to hypothesize a set of reasons that may cause Stegozoa to become inoperational after simple manipulations of the encoded media.

- **Lossiness of VP8 codec:** The VP8 codec algorithm is highly optimized to minimize the amount of data required to represent a frame. It does so in two main ways, by using inter- and intra-frame prediction methods and by discarding (i.e., turning to zero) high frequency coefficients. While both of these operations are lossy, Stegozoa mitigates this effect by embedding messages after the loss of data happens (i.e., after the quantization of coefficients). Adding a round of video re-encoding, however, re-introduces this lossiness to the steganographic embedding space, and may cause enough changes to the embedding coefficients such that the receiving client is unable to reconstruct the original message.

- **Different VP8 settings:** The threat model for Stegozoa states that the peers have full control of the software running on their machines. That is, they can configure and instrument the VP8 codec used on their machines as they see fit. In our testbed, there exists an additional pair of encoder/decoders that are controlled by the owner of the gateway server. Slight tweaks to the configuration (such as quality of encoding, how often key frames are created, or bitrate) may cause unpredictable effects on the values of the DCT coefficients Stegozoa depends on to carry covert data.

- **Fragility of the embedding algorithm:** Stegozoa uses STC to efficiently embed its message in a given embedding space. However, STC was not designed to be robust against changes in the embedding space, and a single error in the stego-vector (e.g., flipping the least significant bit of a coefficient), triggers cascading errors on the recovery of the steganographically-embedded message (if it can be recovered at all).

- **Fragility of the embedding space:** The Stegozoa embedding space does not use the entirety of the coefficients composing a frame, but instead, dynamically filters out coefficients valued at zero and one at embed time, using the remaining coefficients as an input for the STC algorithm. During message recovery, the same filtering is performed so that the exact same embedding space is re-generated by the other peer. From our observations, zero valued coefficients are not guaranteed to remain static after re-encoding, and the reverse is also true for non-zero coefficients. By changing the embedding space, the input for STC itself also changes, preventing the correct recovery of the covert message.

- **Lack of error correction:** As evidenced above, Stegozoa's steganographic embedding procedure assumes the STC stego-vector remains unmodified during transmission, and thus ignores the need for correcting errors at the stego-vector level. Our observations suggest that slight modifications to the stego-vector deem a Stegozoa message to become unrecoverable.

In the next chapter, we conduct a series of experimental tests using the VP8 codec to validate the above hypotheses. Specifically, we aim to quantify the observable changes on encoded media upon re-encoding operations, and assess their potential impact on steganographic schemes like those used in Stegozoa.

Chapter 4

Characterizing VP8 Re-encoding

In this chapter, we perform a set of experiments on the VP8 codec to better understand the effects of re-encoding operations performed over a video stream. Our experiments will be focused on analysing the changes caused to quantized DCT coefficient values, which are the vehicle of choice for multiple video steganography tools, including Stegozoa, after each round of encoding. In Section 4.1, we provide details on the testbed we designed to carry out these experiments. Then, in Section 4.2, we describe our characterization experiments and present our results.

4.1 Local Testbed Configuration

To conduct our experiments, we implemented a different testbed from that focused on the previous chapter. Specifically, we implemented an offline testbed in which we isolate VP8 codec operations without additional networking-induced considerations, and therefore focus only on observable changes in encoded media caused solely by re-encoding procedures.

4.1.1 Testbed design and data collection

Our offline testbed consists of a modified VP8 codec deployment in a local machine with a 2.8GHz 11th Gen. Intel i7 CPU and 16GB RAM. To collect the necessary data for our analysis, our testbed requires the implementation of a set of sensors and actuators within the codec codebase, which we implement through instrumentation code (see Section 4.1.2).

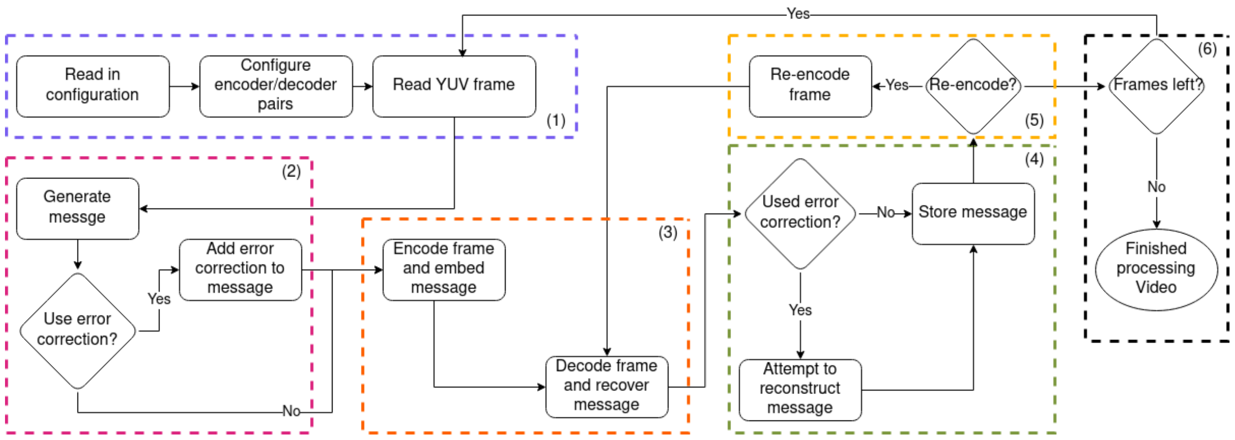

Read in configuration
Configure encoder/decoder pairs
Read YUV frame
(1)
Yes
Generate messge
(2)
Use error correction?
Yes
Add error correction to message
No
Encode frame and embed message
(3)
Decode frame and recover message
Re-encode frame
Yes
Re-encode?
(5)
Used error correction?
No
Store message
(4)
Yes
Attempt to reconstruct message
Frames left?
(6)
No
Finished processing Video

Figure 4.1: Workflow of our offline VP8 re-encoding testbed.

The entirety of our testbed is implemented in ∼1 300 lines of C code for the main test setup, with an additional 100 lines for instrumenting the VP8 codec as previously mentioned.

Testbed workflow. We refer to the steps in Figure 4.1 to describe the workflow we follow for performing our embedding experiments using the testbed. This worflow includes a total of six phases, which we describe below. Observational experiments (those that simply analyze the effects of re-encoding) follow a similar simplified process. Throughout our description, we highlight which steps may be skipped for these experiments.

- **(1) Configuration and setup phase.** We read in the parameters for an experiment, including the source YUV video, codec settings, number of re-encoding layers to execute, and error correction configuration. We then configure a pair of VP8 encoders and decoders, and additionally a new pair for each iteration of re-encoding wanted. The first frame of the video is read into memory. Table 4.1 shows a summary of the configuration parameters available at this stage.

- **(2) Message generation phase.** We generate a random bitstream array whose length is determined during the configuration phase. The probability of 1s and 0s can be biased if desired. Then, if required, additional redundancy can be added to the message by using an error correction algorithm whose parameterization is defined during the configuration phase. This step may be skipped for tests where we are only observing the re-encoding behaviour and not embedding a message.

- **(3) Encoding and decoding phase.** We start the encoding process with the current frame in memory using the original encoder. Our hook code runs during this

Testbed-specific parameters	
Parameter	**Description**
Embedding Coefficient	Index of the sub-block coefficient used for embedding. Range is 0 to 15.
Embedding Bit	Index of the bit in the embedding coefficient to shift to embed message. Range from 0 to 15, where 0 is the least significant bit.
Message Size	Size in bits of the message to embed. Range from 0 to the number of sub-blocks in a frame.
Error Correction	Enable or disable error correction on the embedded message.
Error Correction Level	Used to determine the amount of redundancy for error correction.
Iterations	Number of re-encoding iterations to perform on a video. A value of 0 means no re-encoding happens.
Max Frames	Maximum number of frames that are processed per video.
VP8-specific parameters	
Parameter	**Description**
Width	Width in pixels of the source video.
Height	Height in pixels of the source video.
Frame rate	Number of frames encoded per second of video.
Deadline	Maximum time in ms to process a frame. Higher values can improve visual quality at the cost of performance.
Bitrate	Bitrate per second that the encoder should target.
Key Frame Interval	Maximum number of predicted frames encoded in a row without a key frame. Value of 0 allows any number of predicted frames in a row.

Table 4.1: Testbed configuration parameters.

process, calling our embed function with the message we generated on the previous step. The resulting encoded stego-frame is passed to the original decoder. Again, our hook is called, this time invoking the recovery function, to extract the message from the same bits it was embedded on. We also store the new decoded raw frame in memory. Same as in the previous phase, the message recovery process may be skipped for experiments where we do not embed any data.

- **(4) Message reconstruction phase.** In case that error correction has been enabled during the encoding of a message, we attempt to reconstruct the original message once it is retrieved from a frame, storing the resulting message in local storage. If no error correction has been applied, we directly store the retrieved message.

- **(5) Re-encoding phase.** If our testbed is configured to perform re-encoding operations and we have not yet reached the last re-encoding layer, the raw frame decoded in step (3) is re-encoded using the encoder defined at configuration time. We note

Parameter	Value	Notes
Frames per second	30	Match source video.
Width	1 280	Match source video.
Height	720	Match source video.
Deadline	300 000	Match Kurento Media Server default.
Bitrate	20 000	Match Kurento Media Server defualt.
Key Frame Interval	0	Match Kurento Media Server default.

Table 4.2: VP8 codec configuration.

that, in our experiments, we only use the VP8 encoder, and perform no other message embedding procedures at this stage. The new encoded frame is passed to the corresponding decoder to generate a new decoded frame, and we then attempt the extraction of the stego-message in phase 4. If no more re-encoding operations are to be performed in phase 5, the process continues to phase 6.

- **(6) Process next frame phase.** If there are frames left on the original source video, or if we have not reached the desired number of frames to be processed in the experiment, we jump to phase 1 and read in the next available raw frame from the source video. Otherwise, we are done processing the current video and we read the next configuration file, if available.

Data collection. For our characterization experiments We used 20 YUV 4:2:0 video samples, with at a 1280 × 720 resolution, at 30 frames per second with a duration of 10 seconds each (i.e., 300 frames). The video samples used in our characterization are uniformly sampled from four different categories, including: a) *chat*, where a person talks directly to a camera, simulating a video call; b) *collaboration*, showing a screen capture of someone using collaborative coding software; c) *gaming*, showing a video game broadcast, or; d) *sports*, depicting video broadcasts of multiple sport activities. While videos in the first category can more closely resemble a typical video conferencing call, we drew videos from other categories as it would be trivial for a steganographic system's users to arbitrarily set the cover media (e.g., injecting any pre-recorded video into their webcam feed).

We configured the VP8 codec used in our testbed to match the default codec options used in our Kurento-based WebRTC testbed presented in Section 3.1 (Table 4.2 provides a summary of the chosen values for each configuration parameter). Kurento is configured for a production environment, with a set of parameter choices that enable for reasonable trade-offs between the codec performance and the resulting media quality perceived by users. Importantly, this also means that our local testbed configuration matches the one we used for testing Stegozoa, which we found to be inoperable after re-encoding operations performed by Kurento.

In our experiments, and with the help of our testbed, we saved the DCT coefficient values obtained after the original encoding of each frame, as well as the coefficient values of each frame after one round of re-encoding. Specifically, given our parameterization of VP8, each frame will be divided internally by the codec into 3 600 macroblocks, each containing 16 Y sub-blocks with 16 residual DCT coefficients. This amounts to a total of 92 1600 DCT coefficients per frame, or otherwise a one-to-one correspondence with the number of pixels in the original video.

4.1.2 VP8 codec instrumentation

For our implementation, we instrumented a version of the libvpx library, which holds the source for the VP8 codec. We modified the libvpx code such that we have access to the quantized coefficients from the residual signal as soon as they become available during encoding and decoding. Our objective was to create a set of "hooks" on the libvpx code, such that when the quantized coefficients become available, a user defined function is called to do some processing on them. This processing could be passive (such as inspecting values and storing to be used later), or active (modifying coefficient values to embed a stego-message). Next, we detail how we performed the codec instrumentation and describe our processing functions. In Figure 4.2, we present a diagram of the encoding and decoding algorithms showing the location of our hooks.

Encoder instrumentation. Focusing on the encoding side of the process, first we had to modify the encoder, so that we can pass user defined processing functions during instantiation. We modified the C structure for the encoder to include a flag which will be set to true if we want our hook code to be exectued. Similarly, we added a function pointer member to this structure, which will hold the user defined processing function that takes in a pointer to the macroblock being processed and an optional context pointer that may be used during processing. We then modified the encoding algorithm at two points. Depending on whether a intra or interframe is being encoded, the functions `vp8cx_encode_intra_macroblock` and `vp8cx_encode_inter_macroblock` are called, respectively. Regardless, each of these functions, perform the prediction, DCT transform, and quantization steps mentioned earlier. After these operations are performed, we add our hook that checks that the processing flag is set to true, and if that is the case, calls the user provided processing function. Next, the boolean encoding process continues as normal.

Decoder instrumentation. For the decoder, on the other hand, we do similar modifica-

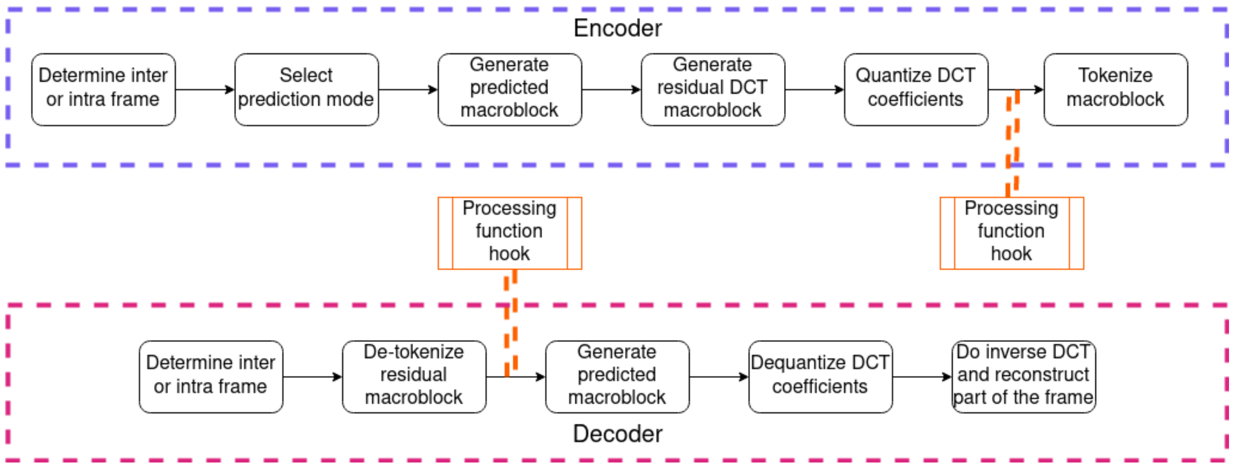

Encoder
Determine inter or intra frame
Select prediction mode
Generate predicted macroblock
Generate residual DCT macroblock
Quantize DCT coefficients
Tokenize macroblock
Processing function hook
Processing function hook
Determine inter or intra frame
De-tokenize residual macroblock
Generate predicted macroblock
Dequantize DCT coefficients
Do inverse DCT and reconstruct part of the frame
Decoder

Figure 4.2: Macroblock encoding and decoding process with processing hooks.

tions. We modify the structure for the decoder to add a similar check flag and processing function pointer. For the decoder algorithm, we also modified the process at two points, however, due to different reasons. As it was mentioned in the decoding overview, a macroblock may be skipped if there are no non-zero coefficients. As we are interested in processing all coefficients, regardless of state, we add our hooks at either of these branching points. In the skip path, we inject our hook code right before the decoder context is reset. Conversely, in the other scenario, our code is called after the boolean decoding, and before the coefficients are dequantized. Importantly, while it is possible to modify the quantized coefficients at this stage, since this would be the message extraction phase of the communication channel there is no need to do so. For this reason, the processing function called by the hook are expected to be passive (that is, purely observational).

Processing Functions As the name implies, the processing functions perform user defined processing of quantized coefficients during encoding and decoding, which get called by our hook code. They do so at a macroblock level, similar to how VP8 processes a frame macroblock by macroblock, and following the same order. The parameters for these functions are very similar for the encoding and decoding phases. They both receive a pointer to the current macroblock (these are the same structures VP8 uses), the type of frame (intra or inter frame), and a context void pointer. The context pointer is user defined, and holds persistent data that may be used across various calls to the processing function. Since it is readily available, the decoder processing function also takes in a macroblock index, indicating the ordinal for the macroblock in the processing order. The functions return a status code corresponding to the success of performing the operations.

We have implemented three types of processing functions for our experiments. One for observing and recording a set of coefficients from each macroblock (which may be all of them), one for embedding data into a macroblock, and one for recovering embedded data from a macroblock. All three of them perform operations only on the Y component sub-blocks of a macroblock (the first 16 in order). We decided to focus only on this component, as changes to the Y component typically cause fewer distortions to an image, causing this component to be a usual target for video steganography techniques. However, we note that our instrumentation hooks could be easily extended to work on the other components too.

Observing function. This function has the most straightforward implementation and can be hooked to either the encoder or the decoder. As each macroblock is received, the coefficients of each block are written to a comma-separated file in order. Subsequent re-encoding iteration values are written in a new line. The saved files can then be compared.

On the other hand, the embedding and recovering functions have complementary implementations, and heavily make use of the context pointer. For the embedding function, the context has a list of indexes of the coefficients used for embedding (we can think of all the Y coefficients of a frame as a long list of numbers, and we use this as their index), embedding bit (for example the 0th, or least significant bit), the message to embed across the entire frame (a bit array), the index for the next message bit to embed, the index of the current macroblock, and the size of the message in bits. The recover function uses a similar context, except the message array starts empty and gets populated as the message is recovered, and keeps track of a recover index instead of an embed bit.

Embedding function. This function is hooked to the encoder, and its goal is to allow us to embed (part of) a stego-message into a given macroblock undergoing processing. To do so, the context passed as a parameter includes an array of ordered embedding coefficient indexes, as well as a pointer to the current embedding index (which starts at the first element in the array). Every coefficient in a frame can be given a unique index (e.g., the first coefficient of the first sub-block of the first macroblock would have index 0), the array is a subset which will be used for embedding the stego-message. If the current embedding index corresponds to one of the coefficients in the macroblock, we replace the ith bit in the binary representation of the corresponding coefficient with the next message bit, where i is the value of the embedding bit parameter. After, the end of block function is called and the current embedding index pointer is then moved to the next element in the array. This embedding and updating process is repeated until either the message is exhausted or the next embedding index falls outside of the range of the macroblock. Once we are done with a macroblock, we increase the current macroblock index in the context and return the control to the VP8 codec.

Recover function. This function is hooked to the decoder, and we use it to recover the message we had previously embedded. The function reverses the embedding process, and uses a similar context, except the message array starts empty and gets populated as the message is recovered, and keeps track of a recover index instead of an embed bit. Once again, we calculate the list of available coefficient indexes in the macroblock, using the same formula as during the embedding. Similarly, if the next coefficient index to recover is found in this macroblock, we save into the message array the value of the bit at the ith position in the binary representation of the coefficient. The process is repeated until the next recover index is outside the bounds of this macroblock, or we have recovered the entire message (as specified by the message size parameter). No end of block update is needed at this stage since we are not modifying any of the coefficient values. The control is then returned to the codec.

End of block update function This function is not called directly but used by the embedding function. After embedding the message bit, we perform an "end of block update". The reason for this operation is because the macroblock object keeps track of an end of block value for all sub-blocks. This value corresponds to the 1-based index of the last non-zero coefficient of a block in zig-zag order, or 0 if all values in the sub-block are 0. The VP8 algorithm uses this value for efficiency purposes, including during the binary compression, to reduce the number of values that need to be processed. Since our embedding could change a non-zero value to zero, or vice versa, it is imperative that we update the end of block to correspond to the new true end of block. Failing to do so may cause a value that is now non-zero to be compressed as a 0 (because it was ignored), or VP8 decoder to flag a macroblock as corrupt and stopping the entire process.

4.2 VP8 Characterization

This section presents the results of the experiments we performed to characterize the changes on the values of the DCT coefficients of encoded frames, after multiple rounds of encoding with the VP8 codec. First, we aim to get a grasp of how the DCT coefficients composing a typical encoded VP8 frame look like (Section 4.2.1). Then, we aim to characterize the changes observed once we subject encoded video to a decoding and re-encoding procedure, like the one performed in Kurento (Section 3.1.1). Following, based on these observations, we analyze why Stegozoa is rendered inoperable after re-encoding operations (Section 4.2.2), and what requirements must be met for the design of a steganographic system that can survive re-encoding (Section 4.2.3).

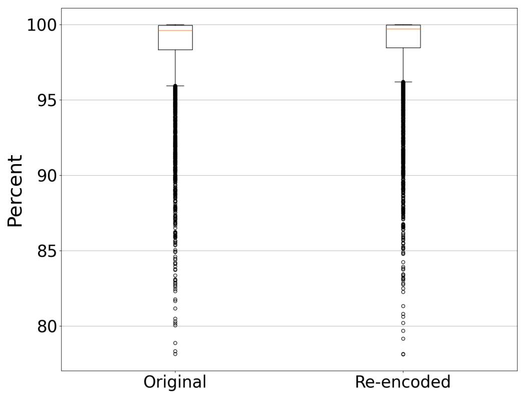


Figure 4.3: Box plot of percent of coefficients with value 0 per frame.

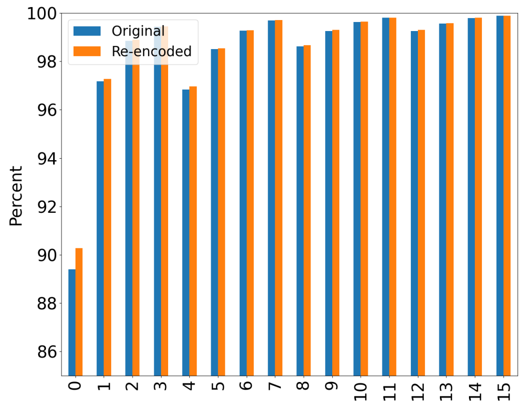


Figure 4.4: Percent of zero coefficients per position.

4.2.1 General VP8 characterization

The goal of our first characterization experiment is to establish a baseline of the contents of encoded media frames. When doing so, we were particularly interested in understanding the percent of DCT coefficients that assume a value of 0 after the quantization step during encoding. Given that the codec was designed with high compression efficiency in mind, our expectations are that most values will be 0. Thus, our first experiment aims to contextualize and ground this expectation. We also hypothesize that an iteration of re-encoding would preserve the values coefficients in similar ranges.

In Figure 4.3 we present a summary of the percentage of zero valued coefficients per frame for both the original video samples and the re-encoded versions. The average value of 0 coefficients is 98.43% and 98.52% for the original and the re-encoded video, respectively. Similarly, we have a mean of 99.61% and 99.70% and a standard deviation of 2.78 and 2.71, respectively. Notably, both plots show a long fading tail of outlier examples that extend down to 75% range. It is likely that these outliers are key-frames, due to those generally encoding more data. Even with these outliers, it is clear that the quantization step is turning most of the coefficients into zero. A similar comparison can be done at a per-macroblock level. The median for both the original and re-encoded videos is 100% and the standard deviations are 4.87 and 4.79, respectively. Or in other words, we can expect at least 50% of the macroblocks to be "empty". This is another proof of the efficiency of the codec, since the empty macroblocks will be skipped during the process.

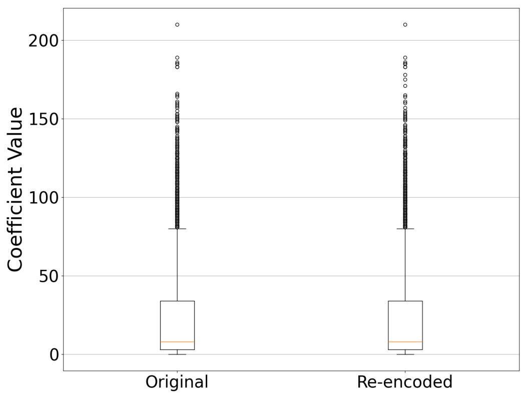


Figure 4.5: Box plot of the maximum positive coefficient value per frame.

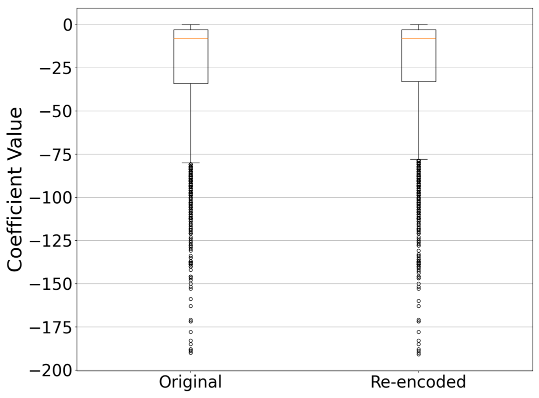


Figure 4.6: Box plot of the minimum negative coefficient value per frame.

Spliced differently, we can also look at the average percent of zero coefficients for each of the 16 coefficients. By design, VP8 codec preserves lower frequency (in zig-zag order) coefficients during the quantization step. Figure 4.4 shows the percent of zero values per coefficient order. Lower frequency (those that have the most impact on the image quality) are less likely to be zero than those of higher frequency. This holds true in our observations.

With the same goal to establish a baseline for the codec, we also look at the value ranges for the quantized coefficients. Intuitively, we would expect that larger coefficient values may be more likely to survive re-encoding. While coefficients are represented as 16 bit signed integer, our initial observations show the maximum values for the coefficients do not get close to this ceiling. By measuring a realistic range, we can have a bound on how large a change we can make to a coefficient during embedding.

We now summarize the largest positive and negative values among quantized coefficients in a frame. Focusing first on the positive values (Figure 4.5), we can observe that non outlier values are all less than 100 units, with a maximum observed value of 210, for both the original encoding and the re-encoding. Conversely, for negative values (Figure 4.6), the non outliers are within 80 units, with the largest observed values at -190 and -191, respectively. From these observations, we can conclude that for non outlier values (both negative and positive), any changes to the least 6 significant bits of the value will not create a new value larger than any previously observed. At worst, a change to the 6th bit would result in a value of 127 (0111111 in 2's compliment binary), which is within the observed values. It is important to note that this ceiling is not taking into account the actual magnitude of the change, and we expect that while this is a best case scenario, embedding may be limited to lesser bits than the 6th bit.

In general, the results from these experiments match our expectations of the codec. The characterization experiments have shown that VP8 is highly efficient both in terms of data being transmitted (high density of zero values can be highly compressed), and processing (empty macroblocks and frames can be ignored). With these observations in mind, the rest of our experiments in this section focus on how re-encoding directly affects covert channels that rely on these coefficients.

4.2.2 Embedding space changes

In this subsection, we describe our experiments towards measuring possible covert data embedding space changes that re-encoding may cause to Stegozoa. Previously, we mentioned that one of the factors that may prevent Stegozoa's correct operation are changes to this space. As a reminder, Stegozoa filters out the first (DC) coefficient of any sub-block, and similarly filters out coefficients that are either 0 or 1. Every coefficient not being filtered is considered to be part of the embedding space. Changes to this space would cause the underlying Stegozoa embedding algorithm (STC) to produce the wrong output (corrupted stego-message). We seek to analyze if there is any consistency on this space after re-encoding by measuring increases and decreases, as well as what percent of sub-blocks remain constant.

To measure changes to the embedding space, we proceed as follows: for each pair of coefficients (extracted from the original and the re-encoded frames, respectively), we compare if they are inside or outside the embedding space. We qualify it as a *space increase* when the original coefficient would be filtered out, but the re-encoding coefficient would have not, and vice-versa for a *space decrease*. We also count the number of frames where all sub-blocks are "constant", i.e., sub-blocks without any changes in the embedding space in either direction.

The total number of space increases per frame observed in our experiments lead to a mean of 540 and a median of 39, with a standard deviation of 1191.95 and a maximum increase of 19068 (2.0% of all coefficients in a frame). Similarly for space decreases, we observed a mean of 719.13, a median of 65, with a standard deviation of 1499.00 and a maximum decrease of 20233 (2.1% of all coefficients in a frame). From these summary statistics, it is clear that these changes in space are fairly spread out. Further, we measured that 75% of all the frames had at least 1 space increase and 2 space decreases. These figures go up to 2 space increases and 5 space decreases for the 70th percentile, or up to 11

Video Category	**Sample 1**	**Sample 2**	**Sample 3**	**Sample 4**	**Sample 5**
Chat	31.67%	30.00%	30.33%	27.33%	34.33%
Collaboration	41.33%	39.00%	24.33%	40.00%	28.67%
Gaming	5.67%	3.33%	0.33%	0.33%	2.67%
Sports	10.33%	0.33%	0.67%	47.00%	0.00%

Table 4.3: Percent of embedding space constant frames per video sample.

space increases and 23 space decreases for the 60th percentiles. Considering a system like Stegozoa relies on this space to be stable (where one change would be enough to disrupt the reverse STC algorithm), we can infer why a message cannot be sent through this channel without being corrupted.

We now focus instead on the number of frames that exhibit a constant embedding space. A summary of our findings is presented in Table 4.3, where we analyze the number of constant frames in each of 5 different videos pertaining to a different video baseline. For samples in the chat and collaboration categories we see a consistent level constant frames, clustered around 30%. On the other hand, for gaming and sports, with the exception of the 4th sample of the latter, the percent of constant frames plummets to 10% or less. One possible explanation for this behaviour is that collaboration and chat samples may have a smaller number of major changes in the video, and may have more "empty" or mostly zero-valued coefficient frames, compared to gaming and sports. Regardless, in terms of affecting the correct functionality of Stegozoa, this shows us that even in the best case scenario (the more stable samples, chat and collaboration), it would not be able to calculate the same embedding space for two thirds of all frames. This is assuming that the actual values have not changed (which is not guaranteed), and that Stegozoa determined there was enough embedding space on the given frame.

4.2.3 Embedding survivability

Next we seek to measure the survivability of an embedded message after re-encoding. For this experiment, we generate and embed a stego-message into a VP8 encoded frame following the flow detailed in Figure 4.1. We then extract the stego-message from the same coefficients we used for embedding, after a round of re-encoding. The survibility rate is the total number of matching bits between the original and extracted message over the total bit length of the message.

Our goal is to find the best performing configuration, between embedding coefficient (1 through 15), and embedding bit (least significant bit, second least significant, and so on).

Coefficient	**All frames**	**Only key frames**
1	34.04%	20.75%
2	30.18%	14.78%
3	30.06%	14.17%
4	33.38%	21.35%
5	32.25%	19.60%
6	31.01%	12.48%
7	**22.88%**	11.37%
8	29.98%	18.03%
9	33.01%	14.06%
10	28.31%	3.65%
11	25.43%	**3.14%**
12	29.18%	16.04%
13	**25.07%**	13.77%
14	26.11%	**3.38%**
15	**21.84%**	**1.24%**

Table 4.4: Least significant bit embedding, per coefficient, message error rate after re-encoding. DC (0th) coefficient excluded from analysis.

Embedding bit	**All frames**	**Only key frames**
0	**25.16%**	**2.94%**
1	41.81%	9.99%
2	52.68%	12.63%
3	56.79%	17.66%
4	56.74%	30.18%
5	59.86%	39.04%

Table 4.5: Variable bit embedding, on 11th coefficient, message error rate after re-encoding.

Commonly, to make it harder to perceive that an image carries a steganographic message, the least significant bit is chosen for message embedding. For this reason, we conduct our first set of experiments holding the embedding bit constant (least significant), and changing the embedding coefficient. We then repeat this experiment holding the embedding coefficient constant (choosing the best performing from the previous experiment), and varying the embedding bit.

Table 4.4 shows the results of the first set of experiments. We present the embed message error rate for all non DC (0th index) coefficients, highlighting the best performing ones. For each frame of each sample, we embedded a random message 3000 bits long in the given coefficient in a sub-block using least significant bit shifting. We then re-encoded each frame, and recovered the message by reversing the process. The error rates presented are measured by comparing each original message with its corresponding recovered message, across all samples, and calculating the percent of bits that were not preserved (or flipped). Lower values represent higher survivality rates for the message, and we present our results both for all frames, as well as only key frames.

Notably, in all cases, error rates are lower on key frame only embedding. A likely explanation for this behaviour is that during encoding (or re-encoding) these type of frames carry more data than the predicted counterparts, since they are reconstructed without any other frames as reference. In other words, the codec will perform less lossy operations during the quantization process in order to preserve that information. We hypothesize that this is what allows messages to better survive a round of re-encoding. It is important to note however, that while a client can configure the codec being used by WebRTC to force more frequent key frames, the codec in the server decoding and re-encoding the media cannot be configured in such a way. For this reason, we believe that a steganography method may rely on the "naturally occurring" key frames (as determined by the codec itself), but not force more frequent key frames.

Regardless of the type of frame, we also observed that error rates tend to decrease as we use higher frequency coefficients (in zigzag order). We reason that a one unit increase (for example embedding a 1 in a coefficient that was otherwise 0) in high frequency quantized coefficients, corresponds to a much higher increase in the corresponding non-quantized coefficient, since this value is multiplied by a higher quantization factor. This implies a higher distortion or impact in the cover image. Similarly, higher frequency coefficients tend to be valued at zero. By embedding on these coefficients, the statistical features are disturbed. We understand both these effects as trade-offs between encoding robustness and observability (both in terms of steganalysis and image analysis).

Given these results, we have chosen the 11th coefficient as the best candidate for the embedding bit experiments. On key frame only embedding, it has comparable performance to the other two best coefficients, while having a lower index in the zigzag order. Similarly, it ranks 4th for all frame embedding, and within a percent of the third best performing.

We present the results of the embedding bit experiments in Table 4.5. As with the previous experiments, we see a clear improvement in performance when embedding happens only on key frames. Similarly, the performance of the 0th bit embedding is comparable to what we had previously measured.

Most surprisingly perhaps, changing embedding from least significant bit towards higher bits causes higher rates of message error rates. Intuitively, we expect a bigger change to the quantized coefficient value to be more likely to survive re-encoding. However, since most coefficient values are zero, our embedding on higher bits, taking the 4th bit for example, would change the binary value from 00000 to 10000. We hypothesize that, during the re-encoding, the value may be quantized to a slightly lower but non-zero value (in the example it would change to 01111), causing the recovery process to wrongly extract a zero instead of a 1. We do not further investigate the reasons for this effect, but experimentally conclude that 0th (least significant) bit embedding is the best choice for our use case.

4.2.4 Characterization conclusions

In this section we have experimentally observed the behaviour of the VP8 codec both during regular encoding and after re-encoding. We have shown the high efficiency of the codec, which translates in a high density of 0 coefficients in a given frame. We also made observations on the range of values (positive and negative) that coefficients normally take. This information is useful guiding our choices in subsequent experiments. We measured changes in embedding space, which helped us explain one reason why Stegozoa does not properly operate after re-encoding, as well as gave a justification to look into a fixed or static embedding space. Finally, we performed experiments on message survivability after re-encoding. We were able to experimentally choose parameters, for both embedding bit and coefficient, that can guide our development of a more robust scheme. The observations from this section are used throughout Chapter 5.

Chapter 5

Towards Robust Video Steganography over WebRTC

This chapter, based upon our previous observations from Chapter 3 and Chapter 4, explores athe design of a video steganography system providing re-encoding robustness. In Section 5.1, we present our threat model and discuss our design goals for a system that can work under this model. Section 5.2 discusses some of the implementation decisions we made to align with our goals. Finally, in Section 5.3 we collect all of our observations and experiments to present recommendations for implementing a VP8 re-encoding robust system. In this section we also discuss the limitations of our work, as well as areas of possible future work based on our findings.

5.1 Threat Model and Design Goals

In the context of our research, the goal of the adversary is to disrupt covert communication channels that leverage WebRTC video conferencing systems, all without having to rely on detecting such methods and avoiding to disturb the communications of legitimate users. We assume that our adversary has control of the WebRTC gateway servers that mediate video calls (either by owning or otherwise influencing/colluding with the owners of such servers) and is then able to freely manipulate the VP8 encoded video being exchanged by peers. For instance, the adversary is able to decode encoded media into raw video frames, optionally adding noise, and then re-encode each frame back to VP8 before transmitting it to the intended receiver. We point out that this kind of "attack" is also within the power of a non-malicious or non-colluding agent. As an example, projects like Kurento (leveraged in our experiments), allow non-malicious WebRTC services' operators to apply video filters to WebRTC media streams, possibly rendering existing covert channels like those produced by Stegozoa inoperational.

We leave several possible attacks outside the scope of our threat model. Firstly, we assume that the software and hardware being used by the peers engaging in communication are trusted, and not under the influence of the adversary (e.g., free from malware compromises that could allow the adversary to interfere with local operations like covert message embedding). Secondly, due to potential collateral damage, the adversary will not completely halt WebRTC communications within their sphere of influence, and will avoid to severely disrupt video streams so as to ensure a reasonable QoS for legitimate users of videoconferencing services.

Design goals. Given our threat model, which defines a stronger active adversary model than the passive one considered by Stegozoa, we set the goal of improving the robustness of WebRTC video calls as a channel for censorship resistance communication. Throughout this chapter, we seek a more robust approach to video steganography that can survive interference with the media elements responsible for carrying covert data. We set the following goals for the design of such a system:

- **VP8 re-encoding resistance:** Given the insights obtained from our VP8 encoding characterization, we seek to devise steganographic message embedding methods that offer robustness against simple re-encoding operations. The sender of a message should be able to add enough redundancy to the message such that the receiver is able to recover and correct the message after a bounded set of perturbations are applied to encoded media within the WebRTC gateway server.
- **WebRTC application independence:** The embedding and error correction methods shall not rely on the carrier application. That is, the system should be usable regardless of the WebRTC video conferencing application being used.
- **Reasonable robustness/throughput trade-off:** Our robust message encoding scheme shall only use as much redundancy as required for ensuring message survivability, towards maximizing the available space for useful message contents.

In the next section, we describe multiple choices that we made when implementing a prototype system towards achieving these goals.

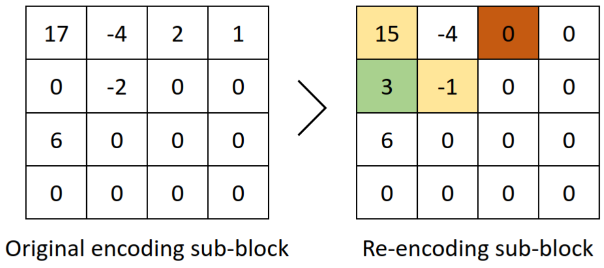
5.2 Towards Robust Steganography in VP8

In this section, we start by elaborating on the robustness issues caused by re-encoding operations over the VP8 codec, and then propose strategies to increase the robustness of covert data transfers over VP8. We note that the solutions we introduce exhibit a set of trade-offs between robustness and other desirable characteristics of steganography. For example, adding error correction will reduce the goodput (non-redundant throughput) of our system, in exchange for the ability to recover the message successfully. Similarly, statically choosing an embedding space solves the dynamic space problem, but may impose observable changes to the cover video (since we are not actively choosing least impactful locations).

5.2.1 Robust embedding spaces

One major challenge we identified for existing steganographic systems that make use of VP8 (throughout Chapter 4) was that such systems could face difficulties in their operation should they rely on dynamically-calculated embedding spaces. Previously, we defined the embedding space as the set of values that can be selected for embedding a stego-message or, particularly for our case, the set of DCT coefficients in video frames. A coefficient found to be within the embedding space does not necessarily imply that this coefficient will be selected or changed by the stenography algorithm, but that it might be so. Regardless of how it is selected, the assumption is that both the sender and receiver must be able to calculate (or have agreed) on the same set of coefficients as the embedding space, since this will be used as the input for the message extraction algorithm (such as reverse-STC for Stegozoa). Through this subsection, we argue that in order to achieve better robustness, static embedding spaces are the more practical choice.

Issues with dynamic embedding spaces. Our characterization experiments showed that one cannot rely on the assumption that a dynamic embedding space will remain constant after one ore more layers of re-encoding. Using the filtering mechanism from Stegozoa as an example (where quantized coefficients with values 1 or 0 are filtered out of the embedding space), we were able to observe that the embedding space may not only experience shrinkage (non-zero and non-one values changing to either zero or one), but also have the opposite effect (by having coefficients valued at zero and one changing to other values) after the media is re-encoded. Our experiments showed that even relatively large values may still be re-encoded as a zero or one, which means that we cannot reliably assume that more aggressive value filtering would ensure that the embedding space remains



Original encoding sub-block

17	-4	2	1
0	-2	0	0
6	0	0	0
0	0	0	0

Re-encoding sub-block

15	-4	0	0
3	-1	0	0
6	0	0	0
0	0	0	0

Figure 5.1: Sample sub-block before and after re-encoding. Coefficient values that did not change after re-encoding in white. Cells in yellow are coefficient values that changed after re-encoding, but did not change the embedding space. Cells in green and red are coefficients that changed in value and also changed the embedding space, increase and decrease respectively.

constant during re-encoding operations. Similarly, we showed that after re-encoding, a value that was originally zero or one could turn into a relatively large value, showing another reason why this kind of more aggressive filtering would not be sufficient. In Figure 5.1 we illustrate this phenomenon by using a sample sub-block before and after a layer of re-encoding.

We hypothesize that predicting how re-encoding would affect dynamic embedding spaces would not be entirely practical, especially if there is no control over the codec parameters used by the adversary's WebRTC gateway during re-encoding.

Advantages of static embedding spaces. Instead of computing a dynamic embedding space, we propose a more straightforward embedding scheme, using a static embedding space. Strawman examples would be, for instance, to include every coefficient or every second lowest frequency coefficient of each block in the embedding space. In this case, we are guaranteed that the space will not change regardless of re-encoding operations. It is important to highlight that when we describe the space as static, we mean it to be agreed beforehand by the sender and receiver of a covert message, and not dynamically determined during embedding. This does not imply that the actual embedding space may not be refreshed; for example, using one of the previous scenarios, the embedding space could switch to every 3rd coefficient instead of every 2nd coefficient after a certain number of messages are sent. The only requirement is that the space is consistently agreed on between both communicating parties before having to process a frame or macroblock, and therefore not susceptible to changes due to re-encoding.

It is worth noticing that the selection of the elements composing embedding space can have direct effects on the ability of an adversary to perform steganalysis of the embedded content. As we have previously shown, each frame is mostly composed of 0 coefficients. By choosing to include 0 in the embedding space, this ratio of 0 to non 0 values may be decreased enough to hint at adversary that something is happening to the cover medium. Similarly, by reducing the number of 0 coefficients in a frame, the compressed frame will be larger in size than if it had not been changed. This feature may also be used by the adversary performing steganalysis.

For our implementation, we aim to maximize re-encoding robustness; from our observations, a static embedding space provides better opportunities for message recovery since this avoids possible inconsistencies caused by the dynamic calculation of the embedding space. We recognized this as a trade-off, that may be fine tuned in later implementations. We discuss a similar necessary trade-off in the next subsection.

5.2.2 Error correction

As we have shown in Chapter 4, the process of re-encoding a video may cause considerable changes on the coefficients used for stego-message embedding. To address this issue, we seek to add redundancy to the stego-message we are embedding such that it may be recoverable post re-encoding. Following from the previous section, we also highlight how choosing a static embedding space also helps with error correction efforts. As we saw with Stegozoa, changes on the coefficients included or excluded in its embedding space caused a mismatch in input for the STC algorithm, between embedding and recovering the message. Paired with variations in the actual values of the coefficients, recovering the stego-message becomes impractical. Instead, by selecting a predetermined set of coefficients known both by the sender and receiver, we simplify the error correction problem by only focusing on the effects of the channel noise (re-encoding) without the additional overhead of having to determine which coefficients are actually part of the message.

For our experiments we chose Reed-Solomon error correcting codes (RS codes for short). RS codes are a popular solution that have been used in CD, DVD and Blu-Ray discs [53], QR Codes [69], and satellite based message transmissions [66]. Even more relevant, they have been successfully used in image and video steganography error correction [4, 38, 82]. RS codes also allow us to fine tune the level of redundancy used in our message given the noise we have measured in our channel. We now provide details on RS, our implementation and evaluation.

Reed-Solomon (RS) error correcting codes. RS codes operate on blocks of data called symbols. A code block with n total symbols is composed of k message symbols and

Symbol Size	**n**	**c**	**t**	**k**	**Redundancy Ratio**
2	3	1	2	1	66.67%
3	7	1	2	5	28.58%
4	15	2	4	11	26.67%
5	31	5	10	21	32.26%
6	63	12	24	39	38.10%
7	127	26	52	75	40.95%
8	255	58	116	139	45.50%

Table 5.1: Estimated parameters (n, c, t, and k) for a given symbol size using calculated error rate of 3.14% for key frame only embedding. Additionally, the redundancy ratio (t/n), as a reference of the efficiency of the configuration, where lower values are better.

Symbol Size	**n**	**c**	**t**	**k**	**Redundancy Ratio**
2	3	2	4	-	-
3	7	5	10	-	-
4	15	11	22	-	-
5	31	24	48	-	-
6	63	53	106	-	-
7	127	111	222	-	-
8	255	231	462	-	-

Table 5.2: Estimated parameters (n, c, t) for a given symbol size using calculated error rate of 25.43% for embedding on every frame. K and message ratios are not provided since the number of correction symbols exceeds the number of symbols in the message.

t (or $n - k$) check symbols. This code is able to detect up to t corrupted symbols, and correct up to *t/2* corrupted symbols. Additionally, the implementation we chose to use (standalone copy of Linux Kernel implementation [1]) is configured in such a way that the maximum number of symbols in a code word is $2^b - 1$, where b is the length in bits of each symbol. As an example, a symbol size of 8 bits implies a maximum size of 255 symbols, and at most being able to detect 254 corrupted symbols and correct 127 corrupted symbols (assuming that all but one symbol is redundancy, or $k = 1, t = 244, t/2 = 127$).

It is relevant to mention at this point that corrupted symbols may have any number of corrupted bits (from 1 to b). Since each individual bit of a symbol is embedded on a different coefficient, changes to the coefficient due to re-encoding may cause no new corrupted symbols (if the symbol already had a wrong bit), or up to one corrupted symbol.

[1] https://github.com/quiet/libfec/tree/master

Redundancy Symbols	**Error rate**	**Error rate (key)**	**Full message**	**Full message (key)**
116	24.75%	2.28%	4.53%	71.66%
166	24.35%	1.91%	8.48%	75.00%
216	24.12%	1.11%	13.43%	83.33%
254	25.29%	2.50%	34.68%	86.66%

Table 5.3: Experimentally observed error and complete message survivability rates at different RS redundancy levels for both key and all frame embedding

Assuming that the embedding bit of a coefficient has a chance to be flipped (from 0 to 1 or vice versa) of P, and each symbol is composed of b bits, the probability that a symbol will not be corrupted is $(1 - P)^b$. Briefly, longer symbols are less likely to not be corrupted.

Given the measurements from our characterization experiments in Section 4.2.4, we can tailor the RS parameters to VP8 re-encoding. We concluded that the 11th (raster order) coefficient and the 0th (least significant) bit provided one of the best trade-offs between low message error rate and frequency. For all frames we observed an error rate of 25.43%, and an error rate of 3.14% on only key frames. If we are to take these as the probability P of any bit to flip, then we can estimate that the number of corrupted symbols per message, c, is equals to $(1 - 0.2543)^b$ and $(1 - 0.0314)^b$, respectively. With c, we can also estimate the number of redundancy symbols t needed, as $t = 2 * C$, and k as $k = n - t = 2^b - 1$.

Tables 5.1, and 5.2 provide a summary of the values for n, c, t, and k for values of b from 2 to 8, using our observed error rate for both key and all frame embedding. From our estimations, no matter the size of the symbol, on average, more symbols will get corrupted per code word than what are possible to fix using RS codes if we are to embed on all available frames. On the other hand, for only key frame embedding, we observe smaller error rates. The key frame only table also presents the redundancy ratio (t/k or the total number of error correction symbols over the total number of symbols). A lower redundancy ratio implies that more bits of the code word can be used for the actual message being transmitted. At 8 bit symbols, on average we will need 45.5% of the message for redundancy, and the ratios tend downwards as we reduce the symbol size (up until we reach 2 and 3 bit symbols). Since encoding a message of the same size would require more instances of RS encoding and decoding, we decided to test on 8 bit symbol sizes as they show good performance.

Message survivability with RS error correction. To measure in practice the survivability of messages encoded using RS codes, we perform a similar experiment to those in Section 4.2.1. This time, after we generate a random bit stream message, we encoded it using a RS code, setting the total message to 255 8-bit symbols. We also set the number of redundancy symbols to the estimated 116 (and in increments of 50 until we reach the maximum of 254). During recovery, we attempt to reconstruct the original message, and store the recovered bits. We compare both the original and recovered message to measure both the per bit error rate, as well as the total number of frames in which the whole message was received correctly. We conduct these data embedding experiments for both key frames only and all types of frame, although we expect the latter not to perform well.

The results of our experiments are presented on Table 5.3. In general, it appears that error correction marginally improved overall error rates, particularly for key frame embedding only. Since RS codes work in an all or nothing manner (it either corrects the entire message or not), we can infer this marginal decreases in error rates are due to the relatively low number of complete messages that were fixed (shown in the following columns). However, this is still an improvement over no error correction being used.

Analyzing all frame embedding, it is clear that even extreme levels of error correction are not enough to reliably transmit a recoverable message. We can calculate the expected number of message re-transmissions, R, before it is correctly received using the formula $R = 1/P$, where P is the probability of a message being received (in this case our complete message rate). Similarly, we can calculate the bit rate, br (or expected number of useful bits per message sent) using the formula $br = k_{bits}/R$, where k_{bits} is the number of non redundancy symbols times the number of bits per symbol. For the 4 values, we observed 50.37, 62.94, 41.90, and 2.74 bits per message transmission (each message being 2040 bits).

We can perform the same calculations on only key frame embedding. For the used levels of redundancy, we observed the bit rates of 796.85, 534, 259.98, and 6.93 bits per message transmission. Interestingly, we can observe clear diminishing returns as we increase the number of parity symbols, as they stop providing message correctness for the number of symbols needed.

Message transmission rate. At first glance, it would appear that key frame embedding may be orders of magnitude better than all frame embedding. However, we have to bear in mind the frequency of key frames in the encoded videos. For each of the 300 frame samples we studied, only 3 of those frames were key frames. That is, only 1 in 100 frames would be a candidate for embedding in the second case. In other words, if we adjust our per message bit rate to a per frame bit rate by multiplying by the percent of useful frames (0.001 and 1, respectively), we obtain new values of 7.96, 5.34, 2.59, and 0.0693 bits per frame for key frame only embedding, while all frame embedding bit rates remain the same.

With this new calculated values, we can conclude that all frame embedding with 166 redundancy symbols will provide the best message transmission throughput. Due to the diminishing returns, we can infer that there is a "sweet spot" of how much redundancy to use, trading off between survivability and useful symbols. We also point out that this kind of message transmission is reliant on having both a re-transmission mechanism, and a general expectation that "eventually" the message will survive. This calculated bit rate does not include the overhead required for this mechanism. We argue that in cases that a message needs to be reliably transmitted with seldom need for re-transmission, it is better to use key frame embedding in exchange for a much lower long run throughput. This is can be specially true if the transmitted video is more likely to have more key frames encoded by the codec than our samples (perhaps one with more scene cuts). Regardless, we propose both methods as valid options for different use cases.

5.3 Re-encoding Robust System Recommendations

In this section we draw conclusions from all of our experiments to provide recommendations towards a re-encoding resistant censorship circumvention system. We offer suggestions necessary for a resilient system under our threat model. We provide design recommendations given our observations from our experiments in Section 5.3.1. Then we conclude this section by going over the limitations from our experiments, and discuss new areas that may be worth being explored to improve this design in Section 5.3.2.

5.3.1 Design considerations

Codec modification. As we have shown from our testbed design, the VP8 codec can be modified to insert "hook" functions which can handle embedding and recovering of a stego message. These modifications should be done on the VP8 codec library used by the browser of choice of the client (e.g., Chrome, Firefox, or Edge). Our implementation provides both the points in the process where these functions should be hooked, as well as the foundation to apply bit embedding and recovery. In our tests, we limited our embedding to a single set of coefficients (for example, only 11th coefficients), however by providing different embedding locations, a more complex scheme can be used. As we previously showed, any scheme must also correctly update end of block pointers after embedding, which we also provide in our implementation.

Messaging control. In our implementation, we generate messages statically; for each frame, we create a random bit stream and embed it into the frame. A proper censorship circumvention tool will need to implement some kind of data exchange layer that accepts user-generated messages to be sent through the channel to meet the needs of the user (e.g., to relay small instant-messaging alike messages). Similar to Stegozoa's implementation, there must exist a middle control layer that communicates with the "hook" code and provides not only the message to be embedded as well as other needed parameters such as the indexes of the embedding coefficients. As we showed in our error correction experiments, this layer should also implement reliable communication and be able to handle message re-transmission since there is an expectation that some messages (or the majority, for all frame embedding) will be corrupted beyond recovery. We suggest a kind of dynamic channel adjustment, perhaps increasing or decreasing the number of times a message is being sent as the average survivability is observed. This control unit should also be able to take advantage of the lesser error rates of key frame embedding to send messages that require a higher level of integrity without re-transmission (such as synchronization messages).

Choice of bit and coefficient embedding. Following the results of our experiments, we showed that embedding on the highest frequency coefficients (i.e., 11th, 14th, 15th) decreases the likelihood of a bit being flipped due to re-encoding. Similarly, we showed that, perhaps initially intuitively, embedding on the least significant bit of a coefficient improves the survivability of the message. This result plays in favor of undetectability, since the distortion introduced due to embedding is minimized using least significant bit embedding.

Embedding space. We showed that relying on dynamically calculated embedding space may prove hard (or in practice impossible) due to the unpredictability of the VP8 codec during re-encoding. We believe this is the main reason our attack rendered Stegozoa inoperable. Thus, we suggest instead using a statically determined embedding space (e.g., "every other coefficient" or "only embed on the 5th sub-block of each macroblock"). We recognize the trade off between observability and encoding robustness, however we believe this is a necessary choice. We also point out that, paired with a mechanism such as message control, the peers using the system may update the rules they use for determining this static space.

Error correction. In our experiments we showed that high levels of redundancy in error correction are needed due to the high bit error rates, regardless if the embedding happens only on key frames or on all frames. In general increasing the amount of redundancy increases the survivability rate of the message, however we observed diminishing returns in terms of rate of message transmission. As we observed, relying on both message re-transmission and just enough error correction allows for a more efficient system.

Zero-biased messages. In our characterization experiment we showed that, as expected, that most coefficients have a value of zero. We argue then that a message with a higher frequency of zeros will be less likely to be corrupted, compared to the even probability random messages we generated. While we have not provided any methods to generate such messages, if a scheme is able to bias the message composition towards zero values, we expect better survival rates.

Choice between key and all frame embedding. As we showed, messages embedded in key frames are more likely to be recoverable after a layer of re-encoding, compared to a message embedded on a predicted frame. However, we also showed that due to their relative infrequency, relying on key frames only is overall less efficient in terms of useful bits per message. In scenarios were the message is encoded and decoded only by the sender and receiver (such as in P2P mode, or in our filter-less architecture), the clients can force their instance of the codec to inject more key frames in the stream. In our attack scenario, on the other hand, there exist at least one pair of encoder/decoder that are configured by the owner of the gateway server, and are not accessible to the clients. The implication here is that the server encoder configuration may reduce the number of produced key frames to a minimum, therefore making this technique unreliable.

Usability. Given the relatively low effective bit rates we observed with our experiments, paired with the need to rely on message re-transmission, we argue that a tool like this is better suited for low bandwidth and non-time sensitive data exchanges. Scenarios such as key exchanges, transmission of Tor bridge addresses, tweet and short message transmission, are a good example of use cases that can be accomplished with this system. It is necessary to point out, that in the case of cryptographic keys for example (or other sensitive information), the data could be extracted by an adversary that exactly knows the embedding bit and coefficients, as well as error correction scheme, and other configurations being used. Similarly, such data needs complete integrity to be usable for both parties. For the former problem, we argue that the configuration parameters described can be understood as a shared secret between the clients, and additional levels of security can be used on the exchange to mitigate this issue. On the latter problem, while it is likely that some messages will be irrecoverable, with re-transmission we can assure that eventually the crypto bits will be received correctly. In the case of RS codes, we can assume that the correct message has been received once it can be recovered, since it works in all or nothing fashion. For the case of continued data transmission, since the channel can remain open for as long as needed, a user may dynamically request new messages to be sent through this channel.

The above recommendations serve as the first step for developing a re-encoding resistant covert channel over VP8 video streams. In the next subsection, we discuss areas of possible improvement that can be built on top of our findings.

5.3.2 Limitations and future work

In this section, we discuss some of the limitations of our work and provide pointers for future developments towards building a full-fledged and robust censorship resistant communication tool over VP8 video streams.

Choice of error correction codes. In our tests, we chose to use RS codes for error correction. RS codes are good for error detection as well as detecting erasures in the data. In our case, we know data will not be erased (we always receive the expected number of coefficients), only possibly corrupted. We argue that there may be other error correction implementations that may be more efficient and effective at recovering the embedded message. Based on previous work, Bose–Chaudhuri–Hocquenghem codes [50] and Low Density Parity-Check codes [15] may be options that can be further explored.

Embedding schemes. In our experiments we decided to embed one bit in each coefficient in our embedding space. Schemes such as STC look at the entirety of the possible embedding space instead and minimize embedding distortions. In Section 3.2.3, we discussed that while STC is intended for efficient embedding, it is fragile to errors, which was one of the reasons Stegozoa was rendered inoperable. However, combining it with error correction could potentially allow for the message to be correctly extracted, while minimizing distortions on the cover. Work from Kin-Cleaves et al. [38], and Guan et al. [25] have previously attempted combining error correction with STC embedding. For this work, to provide a baseline for robust embedding, we decided not to add such schemes. Our implementation could be extend to use them.

Media cover selection. In this work, we did not extensively discuss the selection of the cover video medium used to embed the stego message and then transmit to the other party. We deemed these problem to be outside of the scope of our research, however recognizing them is also an important consideration to implement a robust solution. For one, a cover video that naturally creates more key frames than our samples could help take advantage of their high survivability rates. The video cover could also be chosen so that it is more likely to go through re-encoding without much change in the coefficients for a given frame. The inverse could also be done, by preprocessing the cover media to understand how it behaves under re-encoding, one could choose the most stable coefficients as the targets for embedding.

Unobservability. One of the limitations of our experiments is that our focus is on re-encoding robustness while forgoing an evaluation of how detectable potential distortions in the cover media may be. Instead, we focused our presentation on the best case scenario for messages to survive through the encoding/decoding process while acknowledging these trade-offs. However, more work is required to properly assess the unobservability and undetectability of the steganographic embedding while maintaining a reasonable level of robustness. We leave for future work finding the correct choices between these two important properties.

Network conditions. We performed our message survivability experiments in an offline testbed. We did not perform a closer analysis on how data transmission through an actual network may also cause the VP8 codec to dynamically adjust its operation. Due to network conditions, encoding quality, video bitrate, resolution and other configuration parameters may be dynamically adjusted by both the VP8 codec and by the WebRTC server. These changes may cause differences in the DCT coefficients being used for embedding, potentially causing higher error rates than we observed on our offline testbed.

Chapter 6

Conclusions

In this work, we started by presenting a picture of the ongoing struggle to provide free and uncensored access to the internet. Powerful state-level agents pour considerable resources in controlling the exchange of information on the web to quell political dissent. We have also presented the continued efforts to mitigate, avoid, or otherwise circumvent this kind of control. In particular, we focused on steganography-based systems as a potential solution. These schemes rely on embedding a hidden message in a cover medium such as audio, video or packet traffic, in such a way that it cannot be detected by any other party besides the intended recipient. One such project we studied, Stegozoa, used WebRTC video conferencing applications to effectively create a covert channel between a party with free access to the internet and another within the control of an oppressing regime.

With our work, we showed that systems like Stegozoa are not infallible. Due to the common architecture used by WebRTC servers, even a non-malicious agent could disrupt covert channel by causing the video to be re-encoded. We dedicated a portion of our work analyzing why VP8 re-encoding may cause this kind of disruption. In particular, we analyzed the changes caused to DCT quantized coefficients, a series of values used by the VP8 codec in conjunction with frame prediction methods to compress and then later recreate a video frame, as they are a common medium for stego-message embedding (such as in the case of Stegozoa). By implementing our own WebRTC video application, we confirmed that video re-encoding caused enough modifications on the cover medium to make it so that a stego message is not recoverable by the recipient. We isolated the effect of re-encoding by creating an offline testbed, and were able to show that it makes impractical to dynamically determine an embedding space (as it was being done with Stegozoa). We also showed experimentally the error rates of a transmitted message depending on the chosen index of the embedding coefficient and the embedding bit. Our results confirmed that bits embedded on higher frequency coefficients are less likely to be corrupted, and that least significant bit embedding is the best choice for embedding.

These experiments were carried out both for key frame only embedding and embedding on every frame. As a reminder, VP8 encoded videos consist of two types of frames, key or intra frames, which can be described as self-contained unit that only relies on itself to reconstruct the raw frame it represents, and predicted or inter frames, which unlike key frames, rely on previous key or predicted frames for reconstruction. Embedding data on key frames resulted in much lower error rates when compared to embedding on both key and predicted frames.

With this information, we were then able to test the usage of error correction methods in order to transmit a message that is recoverable, or eventually recoverable. Given the error rates we observed during our previous experiments, combined with the choice of RS codes for our method of error correction, we were able to estimate the level of redundancy needed to transmit a message preserving its integrity. In our experiments we tested on this estimated level, as well as more conservative (higher redundancy) levels, both when embedding on key frames only and on every frame. We showed that using a combination of re-transmission and relatively high levels of error correction may provide the best data transmission rates when using RS codes for error correction when embedding messages on a cover video. We also showed that embedding on key frames only greatly increases the integrity of the messages being transmitted, however their relative low occurrence made them worse than simply transmitting the message multiple times on multiple frames.

Finally, we combined all the observations from our experiments to provide a set of guidelines that we believe are needed to robustly transmit a stegonographic message through a VP8 encoded video medium. We provided our suggestions as building blocks for implementing a censorship circumvention solution that relies on WebRTC conferencing applications. From our observations and results, we believe these to be necessary for designing such a system under our proposed threat model. Finalizing our recommendations, we also discussed limitations of our experiments, and proposed areas of future work that can be built upon our work.

References

[1] Alice, Bob, Carol, Jan Beznazwy, and Amir Houmansadr. How China Detects and Blocks Shadowsocks. In *Proceedings of the ACM Internet Measurement Conference*, pages 111–124, New York, NY, USA, October 2020. Association for Computing Machinery.

[2] Collin Anderson. Dimming the Internet: Detecting Throttling as a Mechanism of Censorship in Iran, June 2013. arXiv:1306.4361 [cs].

[3] Simurgh Aryan, Homa Aryan, and J. Alex Halderman. Internet Censorship in Iran: A First Look. In *Proceedings of 3rd USENIX Workshop on Free and Open Communications on the Internet*, Washignton, D.C., August 2013.

[4] Ananya Banerjee and Biswapati Jana. A robust reversible data hiding scheme for color image using reed-solomon code. *Multimedia Tools and Applications*, 78(17):24903–24922, September 2019.

[5] Jim Bankoski, Paul Wilkins, and Yaowu Xu. Technical overview of VP8, an open source video codec for the web. In *Proceedings of the 2011 IEEE International Conference on Multimedia and Expo*, pages 1–6, July 2011.

[6] Diogo Barradas, Nuno Santos, and Luís Rodrigues. DeltaShaper: Enabling Unobservable Censorship-resistant TCP Tunneling over Videoconferencing Streams. *Proceedings on Privacy Enhancing Technologies*, 2017(4):5–22, October 2017.

[7] Diogo Barradas, Nuno Santos, Luís Rodrigues, and Vítor Nunes. Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communications by Parasitizing on WebRTC. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 35–48, Virtual Event USA, October 2020.

[8] Mehdi Boroumand, Mo Chen, and Jessica Fridrich. Deep Residual Network for Steganalysis of Digital Images. *IEEE Transactions on Information Forensics and Security*, 14(5):1181–1193, May 2019.

[9] Mingyuan Cao, Lihua Tian, and Chen Li. A Secure Video Steganography Based on the Intra-Prediction Mode (IPM) for H264. *Sensors*, 20(18):5242, January 2020. Number: 18 Publisher: Multidisciplinary Digital Publishing Institute.

[10] Ozdemir Cetin and A. Turan Ozcerit. A new steganography algorithm based on color histograms for data embedding into raw video streams. *Computers & Security*, 28(7):670–682, October 2009.

[11] Abdelberi Chaabane, Terence Chen, Mathieu Cunche, Emiliano De Cristofaro, Arik Friedman, and Mohamed Ali Kaafar. Censorship in the Wild: Analyzing Internet Filtering in Syria. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 285–298, Vancouver, BC, Canada, November 2014.

[12] Jedidiah R. Crandall, Masashi Crete-Nishihata, Jeffrey Knockel, Sarah McKune, Adam Senft, Diana Tseng, and Greg Wiseman. Chat program censorship and surveillance in China: Tracking TOM-Skype and Sina UC. *First Monday*, June 2013.

[13] Alfredo Cuzzocrea, Fabio Martinelli, Francesco Mercaldo, and Gianni Vercelli. Tor traffic analysis and detection via machine learning techniques. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 4474–4480, Boston, MA, December 2017.

[14] Ronald Deibert, John Palfrey, Rafal Rohozinski, and Jonathan Zittrain, editors. *Access Controlled: The Shaping of Power, Rights, and Rule in Cyberspace*. The MIT Press, 2010.

[15] I. Diop, S. M Farss, K Tall, P. A. Fall, M L Diouf, and A K Diop. Adaptive steganography scheme based on LDPC codes. In *Proceedings of the 16th International Conference on Advanced Communication Technology*, pages 162–166, Pyeongchang, Korea (South), February 2014. Global IT Research Institute (GIRI).

[16] Arun Dunna, Ciarán O'Brien, and Phillipa Gill. Analyzing China's Blocking of Unpublished Tor Bridges. In *Proceedings of the 8th USENIX Workshop on Free and Open Communications on the Internet*, Baltimore, MD, August 2018.

[17] Sahar A. El Rahman. A comparative analysis of image steganography based on DCT algorithm and steganography tool to hide nuclear reactors confidential information. *Computers & Electrical Engineering*, 70:380–399, August 2018.

[18] Oleg Evsutin, Anna Kokurina, Roman Meshcheryakov, and Olga Shumskaya. The adaptive algorithm of information unmistakable embedding into digital images based on the discrete Fourier transformation. *Multimedia Tools and Applications*, 77(21):28567–28599, November 2018.

[19] Mohammad Fakhredanesh, Mohammad Rahmati, and Reza Safabakhsh. Steganography in discrete wavelet transform based on human visual system and cover model. *Multimedia Tools and Applications*, 78(13):18475–18502, July 2019.

[20] Luis López Fernández, Miguel París Díaz, Raúl Benítez Mejías, Francisco Javier López, and José Antonio Santos. Kurento: a media server technology for convergent WWW/mobile real-time multimedia communications supporting WebRTC. In *Proceedings of the 2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks"*, pages 1–6, June 2013.

[21] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies*, 2015(2):46–64, 2015.

[22] Gabriel Figueira, Diogo Barradas, and Nuno Santos. Stegozoa: Enhancing WebRTC Covert Channels with Video Steganography for Internet Censorship Circumvention. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 1154–1167, Nagasaki Japan, May 2022.

[23] Tomáš Filler, Jan Judas, and Jessica Fridrich. Minimizing Additive Distortion in Steganography Using Syndrome-Trellis Codes. *IEEE Transactions on Information Forensics and Security*, 6(3):920–935, September 2011.

[24] John Geddes, Max Schuchard, and Nicholas Hopper. Cover your ACKs: pitfalls of covert channel censorship circumvention. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 361–372, Berlin, Germany, 2013.

[25] Qingxiao Guan, Peng Liu, Weiming Zhang, Wei Lu, and Xinpeng Zhang. Double-Layered Dual-Syndrome Trellis Codes Utilizing Channel Knowledge for Robust Steganography. *IEEE Transactions on Information Forensics and Security*, 18:501–516, 2023.

[26] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. Characterization of Tor Traffic using Time based Features:. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, pages 253–262, Porto, Portugal, 2017.

[27] A.I. Hashad, A.S. Madani, and A.El Moneim A. Wahdan. A Robust Steganography Technique Using Discrete Cosine Transform Insertion. In *Proceedings of the 2005 International Conference on Information and Communication Technology*, pages 255–264, Cairo, Egypt, 2005.

[28] Gaofeng He, Ming Yang, Junzhou Luo, and Xiaodan Gu. A novel application classification attack against Tor. *Concurrency and Computation: Practice and Experience*, 27(18):5640–5661, 2015. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.3593.

[29] Vojtěch Holub, Jessica Fridrich, and Tomáš Denemark. Universal distortion function for steganography in an arbitrary domain. *EURASIP Journal on Information Security*, 2014(1):1, January 2014.

[30] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. The Parrot Is Dead: Observing Unobservable Network Communications. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, pages 65–79, May 2013.

[31] Sheng Dun Hu and Kin Tak U. A Novel Video Steganography Based on Non-uniform Rectangular Partition. In *In Proceedings of the 2011 14th IEEE International Conference on Computational Science and Engineering*, pages 57–61, Dalian, August 2011.

[32] Mehdi Hussain, Ainuddin Wahid Abdul Wahab, Noman Javed, and Ki-Hyun Jung. Hybrid Data Hiding Scheme Using Right-Most Digit Replacement and Adaptive Least Significant Bit for Digital Images. *Symmetry*, 8(6):41, June 2016. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[33] Mehdi Hussain, Ainuddin Wahid Abdul Wahab, Yamani Idna Bin Idris, Anthony T.S. Ho, and Ki-Hyun Jung. Image steganography in spatial domain: A survey. *Signal Processing: Image Communication*, 65:46–66, July 2018.

[34] Lakhmi C. Jain, Sheng-Lung Peng, and Shiuh-Jeng Wang. *Security with Intelligent Computing and Big-Data Services 2019: Proceedings of the 3rd International Conference on Security with Intelligent Computing and Big-data Services (SICBS), 4–6 December 2019, New Taipei City, Taiwan.* April 2020.

[35] Bart Jansen, Timothy Goodwin, Varun Gupta, Fernando Kuipers, and Gil Zussman. Performance Evaluation of WebRTC-based Video Conferencing. *ACM SIGMETRICS Performance Evaluation Review*, 45(3):56–68, March 2018.

[36] Eric Jardine. Tor, what is it good for? Political repression and the use of online anonymity-granting technologies. *New Media & Society*, 20(2):435–452, February 2018. Publisher: SAGE Publications.

[37] Watson Jia, Joseph Eichenhofer, Liang Wang, and Prateek Mittal. Voiceover: Censorship-Circumventing Protocol Tunnels with Generative Modeling. *Free and Open Communications on the Internet*, 1:67–80, 2023.

[38] Christy Kin-Cleaves and Andrew D. Ker. Adaptive Steganography in the Noisy Channel with Dual-Syndrome Trellis Codes. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–7, Hong Kong, Hong Kong, December 2018. IEEE.

[39] Bin LI, Ming Wang, Xiaolong Li, Shunquan Tan, and Jiwu Huang. A Strategy of Clustering Modification Directions in Spatial Image Steganography. *IEEE Transactions on Information Forensics and Security*, 10(9):1905–1917, September 2015.

[40] Bin Li, Ming Wang, Jiwu Huang, and Xiaolong Li. A new cost function for spatial image steganography. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 4206–4210, Paris, France, October 2014. IEEE.

[41] Bin Li, Weihang Wei, Anselmo Ferreira, and Shunquan Tan. ReST-Net: Diverse Activation Modules and Parallel Subnets-Based CNN for Spatial Image Steganalysis. *IEEE Signal Processing Letters*, 25(5):650–654, May 2018.

[42] Jia Liu, Yan Ke, Zhuo Zhang, Yu Lei, Jun Li, Minqing Zhang, and Xiaoyuan Yang. Recent Advances of Image Steganography With Generative Adversarial Networks. *IEEE Access*, 8:60575–60597, 2020.

[43] Yunxia Liu, Shuyang Liu, Yonghao Wang, Hongguo Zhao, and Si Liu. Video steganography: A review. *Neurocomputing*, 335:238–250, March 2019.

[44] Salvatore Loreto and Simon Pietro Romano. Real-Time Communications in the Web: Issues, Achievements, and Ongoing Standardization Efforts. *IEEE Internet Computing*, 16(5):68–73, September 2012.

[45] Xiaojing Ma, Zhitang Li, Hao Tu, and Bochao Zhang. A Data Hiding Algorithm for H.264/AVC Video Streams Without Intra-Frame Distortion Drift. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(10):1320–1330, October 2010.

[46] Philip Matthews, Jonathan Rosenberg, and Rohan Mahy. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). Request for Comments RFC 5766, Internet Engineering Task Force, April 2010. Num Pages: 67.

[47] Philip Matthews, Jonathan Rosenberg, Dan Wing, and Rohan Mahy. Session Traversal Utilities for NAT (STUN). Request for Comments RFC 5389, Internet Engineering Task Force, October 2008. Num Pages: 51.

[48] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. SkypeMorph: protocol obfuscation for Tor bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 97–108, Raleigh, NC, USA, October 2012.

[49] Mozilla. Codecs used by WebRTC - Web media technologies | MDN, July 2023.

[50] Ramadhan J. Mstafa and Khaled M. Elleithy. A DCT-based robust video steganographic method using BCH error correcting codes. In *Proceedings from the 2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pages 1–6, April 2016.

[51] Ramadhan J. Mstafa and Khaled M. Elleithy. A novel video steganography algorithm in DCT domain based on hamming and BCH codes. In *Proceedings of 2016 IEEE 37th Sarnoff Symposium*, pages 208–213, September 2016.

[52] Ramadhan J. Mstafa, Khaled M. Elleithy, and Eman Abdelfattah. A Robust and Secure Video Steganography Method in DWT-DCT Domains Based on Multiple Object Tracking and ECC. *IEEE Access*, pages 1–1, 2017.

[53] Johnny Phuong Nguyen. *Applications of Reed-Solomon codes on optical media storage.* PhD thesis, San Diego State University, 2011.

[54] Arian Akhavan Niaki, Shinyoung Cho, Zachary Weinberg, Nguyen Phong Hoang, Abbas Razaghpanah, Nicolas Christin, and Phillipa Gill. ICLab: A Global, Longitudinal Internet Censorship Measurement Platform. In *In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP)*, pages 135–151, May 2020. ISSN: 2375-1207.

[55] Khushman Patel, Kul Kauwid Rora, Kamini Singh, and Shekhar Verma. Lazy Wavelet Transform Based Steganography in Video. In *2013 International Conference on Communication Systems and Network Technologies*, pages 497–500, April 2013.

[56] Tomáš Pevný, Patrick Bas, and Jessica Fridrich. Steganalysis by subtractive pixel adjacency matrix. In *Proceedings of the 11th ACM workshop on Multimedia and security*, MM&amp;Sec '09, pages 75–84, Princeton, NJ, USA, September 2009.

[57] Tomáš Pevný, Tomáš Filler, and Patrick Bas. Using High-Dimensional Image Models to Perform Highly Undetectable Steganography. In Rainer Böhme, Philip W. L. Fong, and Reihaneh Safavi-Naini, editors, *Information Hiding*, Lecture Notes in Computer Science, pages 161–177, Berlin, Heidelberg, June 2010. Springer.

[58] Yinlong Qian, Jing Dong, Wei Wang, and Tieniu Tan. Deep learning for steganalysis via convolutional neural networks. In *Media Watermarking, Security, and Forensics 2015*, volume 9409, pages 171–180. SPIE, March 2015.

[59] Zhenxing Qian, Hang Zhou, Weiming Zhang, and Xinpeng Zhang. Robust Steganography Using Texture Synthesis. In Jeng-Shyang Pan, Pei-Wei Tsai, and Hsiang-Cheh Huang, editors, *Advances in Intelligent Information Hiding and Multimedia Signal Processing*, Smart Innovation, Systems and Technologies, pages 25–33, Cham, 2017. Springer International Publishing.

[60] Reethika Ramesh, Ram Sundara Raman, Matthew Bernhard, Victor Ongkowijaya, Leonid Evdokimov, Anne Edmundson, Steven Sprecher, Muhammad Ikram, and Roya Ensafi. Decentralized Control: A Case Study of Russia. In *Proceedings 2020 Network and Distributed System Security Symposium*, San Diego, CA, 2020.

[61] Eric Rescorla and Nagena Modadugu. Datagram Transport Layer Security. Request for Comments RFC 4347, Internet Engineering Task Force, April 2006. Num Pages: 25.

[62] Lotus Ruan, Jeffrey Knockel, Jason Q. Ng, and Masashi Crete-Nishihata. One App, Two Systems: How WeChat uses one censorship policy in China and another internationally. Technical Report Citizen Lab Research Report No. 84, University of Toronto, November 2016. Section: Free Expression Online.

[63] Eve Schooler, Jonathan Rosenberg, Henning Schulzrinne, Alan Johnston, Gonzalo Camarillo, Jon Peterson, Robert Sparks, and Mark J. Handley. SIP: Session Initiation Protocol. Request for Comments RFC 3261, Internet Engineering Task Force, July 2002. Num Pages: 269.

[64] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RTP: A Transport Protocol for Real-Time Applications. Request for Comments RFC 3550, Internet Engineering Task Force, July 2003. Num Pages: 104.

[65] De Rosal Igantius Moses Setiadi. PSNR vs SSIM: imperceptibility quality assessment for image steganography. *Multimedia Tools and Applications*, 80(6):8423–8444, March 2021.

[66] Priyanka Shrivastava and Uday Pratap Singh. Error Detection and Correction Using Reed Solomon Codes. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2013.

[67] Siddharth Singh and Tanveer J Siddiqui. A Security Enhanced Robust Steganography Algorithm for Data Hiding. *International Journal of Computer Science Issues*, 9(3), May 2012.

[68] Mohammad Hassan Mojtahed Soleimani, Muharram Mansoorizadeh, and Mohammad Nassiri. Real-time identification of three Tor pluggable transports using machine learning techniques. *The Journal of Supercomputing*, 74(10):4910–4927, October 2018.

[69] Tan Jin Soon. QR Code. *Synthesis Journal*, 2008(2008):59–78, 2008.

[70] Taiyi Sun and Quansheng Zhao. Delegated Censorship: The Dynamic, Layered, and Multistage Information Control Regime in China. *Politics & Society*, 50(2):191–221, June 2022. Publisher: SAGE Publications Inc.

[71] Yun Tai and King-wa Fu. Specificity, Conflict, and Focal Point: A Systematic Investigation into Social Media Censorship in China. *Journal of Communication*, 70(6):842–867, December 2020.

[72] Jinyuan Tao, Sheng Li, Xinpeng Zhang, and Zichi Wang. Towards Robust Image Steganography. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(2):594–600, February 2019.

[73] Matias Tassano, Julie Delon, and Thomas Veit. FastDVDnet: Towards Real-Time Deep Video Denoising Without Flow Estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1354–1363, Seattle, WA, USA, 2020.

[74] Jozsef Vass. How Discord Handles Two and Half Million Concurrent Voice Users using WebRTC, September 2018.

[75] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. StegoTorus: a camouflage proxy for the Tor anonymity system. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 109–120, Raleigh, NC, USA, October 2012.

[76] Philipp Winter and Stefan Lindskog. How China Is Blocking Tor, April 2012. arXiv:1204.0447 [cs].

[77] Philipp Winter, Tobias Pulls, and Juergen Fuss. ScrambleSuit: a polymorphic network protocol to circumvent censorship. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 213–224, Raleigh, NC, USA, November 2013.

[78] Dawen Xu and Rangding Wang. Context adaptive binary arithmetic coding-based data hiding in partially encrypted H.264/AVC videos. *Journal of Electronic Imaging*, 24(3):033028, June 2015. Publisher: SPIE.

[79] Zhongwen Xu, Yi Yang, and Alexander G. Hauptmann. A discriminative CNN video representation for event detection. In *In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1798–1807, Boston, MA, USA, June 2015.

[80] Jianhua Yang, Kai Liu, Xiangui Kang, Edward K. Wong, and Yun-Qing Shi. Spatial Image Steganography Based on Generative Adversarial Network, April 2018.

[81] Weike You, Hong Zhang, and Xianfeng Zhao. A Siamese CNN for Image Steganalysis. *IEEE Transactions on Information Forensics and Security*, 16:291–306, 2021.

[82] Yi Zhang, Xiangyang Luo, Chunfang Yang, and Fenlin Liu. Joint JPEG compression and detection resistant performance enhancement for adaptive steganography using feature regions selection. *Multimedia Tools and Applications*, 76(3):3649–3668, February 2017.

[83] Zhiqiang Zhu, Ning Zheng, Tong Qiao, and Ming Xu. Robust Steganography by Modifying Sign of DCT Coefficients. *IEEE Access*, 7:168613–168628, 2019.

[84] Zhiying Zhu, Sheng Li, Zhenxing Qian, and Xinpeng Zhang. Destroying robust steganography in online social networks. *Information Sciences*, 581:605–619, December 2021.

[85] Zhiying Zhu, Ping Wei, Zhenxing Qian, Sheng Li, and Xinpeng Zhang. Image Sanitization in Online Social Networks: A General Framework for Breaking Robust Information Hiding. *IEEE Transactions on Circuits and Systems for Video Technology*, 33(6):3017–3029, June 2023.