Original Paper
Received: May 27, 2018    Accepted: June 04, 2019

Covert Channel Implementation Using Motion Vectors Over H.264 Compression

YORAM SEGAL*
OFER HADAR**

Abstract

Embedding information inside video streaming is a hot topic in the world of video broadcasting. Information assimilation can be used for positive purposes, such as copyright protection. On the other hand, it can be used for malicious purposes such as a hostile takeover, remotely, on end-user devices. The basic idea of information assimilation technology within a video is to take advantage of the sequence of frames that flows between the video server and the viewer. Casting foreigner data into each frame such a hidden communication channel is created namely - covert channel. Attackers find the multimedia world in general and video streaming, an attractive backdoor for cyber-attacks. Multimedia covert channels provide reasonable bandwidth and long-lasting transmission streams, suitable for planting malicious information and therefore used as an exploit alternative. In this article, we propose a method to protect against attacks that use video payload for transferring confidential data using a covert channel. This work is part of a large-scale study of video attack methods. The goal of the study is to build a generic platform that will investigate the reliability of video sequences. The platform allows to encoding and decoding video. A plugin can be added to each encoder or decoder. Each plugin is an algorithm that is studied and developed in the framework of this study. One of the algorithms in this platform is information transmission over video using motion vectors. This method is the topic off this article.

Keywords: Covert Channel; H.264; Video Streaming; Cyber-attacks; Malicious information

* Communication Systems Engineering Department, Ben Gurion University of the Negev (BGU), Beer-Sheva, Israel; (yoramse@post.bgu.ac.il)
** Communication Systems Engineering Department at Ben-Gurion University of the Negev (BGU), Beer-Sheva, Israel; (ofer.hadar2@gmail.com)
Introduction

Nowadays, we are witnessing the increasing consumption of multimedia content in general, and specifically, of video services worldwide. With the increment of internet service suppliers’ bandwidth size to the public, there is increasing demand for fast, accessible and high-quality online music and video services. To comply with this need, compression standards that significantly reduce storage size and bandwidth usage were introduced (e.g. H.264, JPEG, H.265).

New multimedia platforms are introduced to our lives frequently (e.g. Spotify, Netflix) and the relative part of digital media in internet traffic is increasing. Current studies show that globally, IP video traffic will be 82 percent of all IP traffic (both business and consumer) by 2022, up from 75 percent in 2017 (Cisco, 2018).

The growth of video traffic and the incredible amount of data it contains give many opportunities to manipulate the data for benign uses like video watermarking and encryption for increased security and copyright protection, or malicious uses such as taking advantage of video encoding process to embed malicious data within videos.

Due to the quite permissive multimedia standards and their popularity, manipulation of the video data is possible without exceeding protocol standards and with minimal effect and disruption of video quality. Multiple techniques have been reported for steganography and watermarking (Katzenbeisser, 2000). An overview of digital image steganography is presented in Neufeld and Proc (2013). In Morkel et al (2005) basic building blocks for steganography in compressed video were examined: the embedding operation and the choice of embedding alternatives.

It is shown in Amsden et al (2014) that Facebook Cover Photos can effectively hide information using Discrete Cosine Transform (DCT) coefficient embedding algorithms. Watermarking solves the challenge of illegal video distribution and manipulation. Watermark’s robustness is critical for avoiding attackers’ watermark disruption. Some methodologies were developed in Harsh and Abhishek (2009) for compressing the robustness of different watermarking techniques. The watermarking algorithm presented in Lu Jianfeng and Yang Fan (2011) embeds the watermark into the video by adjusting intermediate frequency coefficients.

An innovative approach for cyber-attack/defense applying a Smart threshold and Anomaly Correction to compressed domain DCT coefficients is described in Amsalem et al (2015). In this paper we focus on manipulations of compressed domain Error estimation of Motion Vectors.

Video compression protocols, such as H.264, for example, divide video frames to Macro Blocks (MB) and estimate their movement between frames (represented by Estimated Motion Vectors, or EVMs). To minimize the number of bits required to represent transmitted EMVs, the algorithm assumes that neighboring MBs statistically move in similar vector values (size and direction). Therefore, it is possible to transmit only the EMVs delta values compared to neighboring MBs. To find the EMV delta value, a Median of all neighboring MBs EMV is calculated and only the delta is sent (error estimation of the Motion Vector).
The Cyber-attack algorithm takes advantage of lack of sensitivity of movie viewers to small deviations of Macro-Block (object) movements from their original path. Viewers are not likely to notice minor noise around moving MBs. Moreover, since the viewer does not know the accurate real position of MBs in the original video movie, they are not likely to notice minor displacement changes that affect MBs position accuracy.

**Glossary**

**Challenges**

The nature of video encoding standards makes them a viable target for manipulation and additional data insertion, in a concealed manner. Such manipulations could be used for steganography and copyright protection of videos via watermarking. Amirjan and Mansouri (2017) in addition to software algorithms, hardware implementation is also possible (Dalbouchi et al., 2017). Additional uses include addition of hidden data and video encryption for copyright protection and secure video transfer (Long et al, 2018). Another use could be a cyber-attack by embedding malicious data in the video stream (Yahav, 2016).

In a similar fashion, it is possible to use the video stream to establish a covert channel for data transfer, the data which is embedded and hidden in the video stream. No matter the objective, there are several key points that should be considered when manipulating video media:

- **Real time processing:** Cause minimal delay and take into consideration video rate and quality.

- **Perceptual quality:** Maximizing the amount of embedded data while minimizing the video quality degradation and keep a high user Quality of Experience (QoE).

- **Keep video size and bitrate closest to the original video:** The embedding of additional data can increase the stream bit rate or size, so a big change to the stream size could be used to identify the video as manipulated.

The additional data could be embedded either in a video header or payload. Embedding data in the payload is harder to detect because the manipulated multimedia content is well formed, and the quality of the infected content is not degraded. As opposed to embedding data in the video header which is easier to detect by common means such as anti-virus software. Using the video payload as medium and concealing the data in the compressed domain enables the manipulation to occur in real time, under certain restrictions.

**The New Research Platform**

The Matlab based research platform which we create as part of our work is designed to allow easy access and manipulation of various parameters as well as produce results in the form of graphs and statistical data, as well as output compressed bit-streams and decoded movie files. The platform is designed to be a starting point for future projects and research.
In its current implementation, the platform is designed to take folders containing video files (sorted to categories or otherwise by the user) and encode the input files using parameters defined by the user. The parameters include technical parameters such as input size (height and width), frames per second, etc. and, the usage of different modes of operation and selectable layers (such as encryption, compression and error checking). The output of the encoding process is bitstreams, sorted to output folders and statistical information on the process itself, which could be used to create graphs and manipulated as needed.

The platform can also perform the reverse decoding process for given bitstreams and parameters (such as encryption keys, if applicable) and create and output de-coded video from the bitstream, along with statistical information.

![Platform Block Diagram](image)

**Fig. 1. Platform Block Diagram**

### The Method

#### Implementation

The focus of this project is to supply a proof of concept to the establishment of a covert channel for hidden data transfer, in the compressed domain, by means of steganography (in real time) on one hand and extracting this hidden data by using a compatible video decoder on the end user side, on the other.

To achieve this, our second objective in this project is to create a MATLAB based research platform which will allow us to efficiently plan and control all the different parameters and
aspects that affect the channel (See Fig. 1). Moreover, our intention is to make the platform as versatile and generic as possible, to allow it to be used in different future projects and researching different aspects of the subject such as possible attacks and defenses in the compressed domain.

Additionally, after the implementation of the channel and data transfer it is necessary to review and conclude what are the optimal parameters to be used in different contexts. The suggested channel establishment have been implemented in the H.264 standard (Richardson, 2010) since it is widely used and offer flexibility in the compression process. H.264 standard defines syntax for compressed video and a method for decoding this syntax to produce a displayable video sequence (Richardson, 2014; Sullivan, 2003).

The hidden data to be transferred in the covert channel was embedded in the video during the encoding process in the motion vector calculation stage, and likewise extracted by the compatible decoder on the receiving side.

Since we strive to provide a proof of concept, there is no need to try and implement the covert channel on advanced standards such as H.265 since they are based on the same principles as H.264 and we believe our current implementation is enough to prove viability.

To measure the effect on video quality after the decoding process in comparison to the original video, we used the MSE (Mean Square Error), PSNR (Peak signal-to-noise ratio) and SSIM (Structural Similarity Index) metrics to measure the video quality after the covert channel implementation.

**Method Implementation**

In general, we established a covert channel for transferring embedded data in the video payload, between the either source - which already provides the manipulated video with the embedded data, or a intercepted video stream and preform a process similar to an "man in the middle" attack (Fouant, 2010) in which we receive the video stream before it's intended destination, partially decode it, embed hidden data in the pay-load, re-encode and deliver it to its intended destination.

On the destination side, a compatible video decoder must be used to extract the embedded hidden data that was added to the video stream. While it is possible to implement the process on either the DCT coefficients (Barni, 1998; Segal and Hadar, 2018) or using motion vectors parameters, in this work we focus on implementing the covert channel using motion vectors (See Fig. 2).
The H.264 standard employs predictive Motion Vectors (MV) coding technique. This technique uses the Median Motion Vector (MMV) for estimating the movement of neighboring Macro Blocks (MB). The motion compensated inter-frame prediction technique achieves high compression by reducing data, effectively exploiting temporal correlation, using accurate displacement (dx, dy) relative to MMV - Error estimation of MVs (See Fig. 3).

Fig. 2. Encoding and Decoding process

Fig. 3. Median Motion Vector (MMV) and search criteria
Encoder / Decoder Implementation

In this work, in order to easily manipulate the encoding and decoding process as well as have the ability to gather and analyze data, as well as generate plots and graphs, we will use a Matlab implementation of an H.264 codec (Muhit, 2016) which is similar in its operation to the real H.264 codecs and provides us with all the needed functionality to implement our proposed covert channel.

While the source implementation covers the main functionality of the H.264 codecs it is not an exact replica and some features are either simplified or missing entirely; such as the input file has to be in a fixed size and only the Luma component is used in the compression process and audio is being ignored as well. In the internal process, the implementation does not support B-Frames. To prevent confusion with the 'true' H.264 codec standard, in this work we shall refer to our implementation as 'H.264*' where applicable.

Encoding Process

The manipulation of the video in the encoding stage is done as per the H.264 protocol encoding process with the additional data embedded in the P-frame macro blocks. P-frames are encoded with reference to previous frames and contain intra-macroblocks - blocks that originate in the picture domain and contain information directly from previous frames and predicted macroblocks - blocks that could be predicted with good estimation using already available information from neighboring blocks and previous frames.

Every frame is constructed from a header and a collection of macroblocks. We intend to the predicted macroblocks inside the P-frames as candidates for hidden data embedding and use them to transfer the data over the covert channel.

Every macroblock contains a header describing the parameters needed to reconstruct the block in the video decoding stage, amongst them are the motion estimation vectors (dx, dy) which describe the location of the block in the frame and a point of reference which is also part of the header information. For each such block, we would like to check if it is a suitable candidate for data embedding if it meets our criteria and if possible, use it to transfer data.

In the predicted- macroblock construction stage during the encoding process, a median vector is calculated using the neighboring blocks. In total 3 neighbors are used: left, upper left and upper neighbor. A median is calculated for every axis separately and the resulting reference vector (X, Y). The estimated motion vector describes the difference between the resulting vector and the motion vector used to estimate the predicted macro block’s location.

The embedding process consists of scanning all the macroblocks in the frame and determining if they could be used to transfer data. For each block we will calculate the reference vector and compare them to the estimated motion vector. if the vector is within a given threshold, the conditions are met, and we could replace the motion vectors with encoded data and at a later stage, even provide a correcting matrix to prevent the vector manipulation from damaging the decoded frame.
The embedding process

We will use a plain text file as the source of the additional information we wish to embed and transfer covertly over the video stream. As mentioned before, not every macroblock will be suitable for data embedding but only those within a given threshold, which is selected at the beginning of the process and is agreed upon by both the encoder and decoder, with a ¼-pixel resolution. A suitable macroblock, for which the motion estimation vectors (Dx, Dy) are within:

$$\sqrt{Dy^2 + Dx^2} \leq \text{threshold} \quad (1)$$

In other words, the Euclidian distance of the vector is lesser than the threshold value. If the criteria are met for the block, we could replace the motion estimation vector values to values representing an encoded form of the data we wish to embed in the block (See Table 1).

Table 1. Example of a coding table

| x/y | -0.25 | 0   | 0.25 |
|-----|-------|-----|------|
| -0.25 | 000 | 001 | 010  |
| 0    | 011 | ∅   | 100  |
| 0.25 | 101 | 110 | 111  |

Moreover, since all options are checked, and each division could possibly contain a different number of embedded bits, a sophisticated indexing mechanism is used to keep track of the embedded data, so no duplication or missing bits could occur in the embedding process (See Fig. 4).

![Figure 4. Variable distance selection (word size)](image)

- 3 bit
- 4 bit
- 5 bit
Decoding Process

If the video is decoded with a compatible decoder and the threshold parameters are known and shared between the encoder and decoder. The extraction of the embedded data is done during the decoding process itself. Every frame is scanned for suitable macroblocks who stand within the criteria, and every such frame is considered to contain not an actual video information but embedded data and is treated accordingly.

For every such block the motion estimation vectors data is read and translated back using a reverse coding table (See Table 1). Example of a coding table to a number of bits, which in turn and concatenate and form a stream of bits representing the hidden data that was embedded in the video. After that, it is a simple matter to translate the bit stream to readable data along-side the decoded video resulting from the decoding process.

Additional layers in the channel

To make the channel more secure, robust and resistant to either defensive mechanisms or noise, in the creation of the bit stream during the encoding process additional algorithms are used to encrypt, compress and error-correction (See Table 2). All these layers are optional and selectable in the encoding process and require the decoder to 'know' which layers are used to be able to decode the stream correctly.

- Encryption: For encrypting the stream, we use a Matlab implementation of a DES encryption algorithm (Wu, 2012). While better and more secure encryptions are available, this implementation is enough for our proof of concept and could be changed in the future to a different encryption if needed.

- Compression: To reduce the stream size as much as possible, we use a Matlab implementation of a lossless compression algorithm (Kleder, 2005).

- Error Correction: To make the channel more resistant to noise and other interference, A Matlab implementation of a Reed-Solomon code is used to encode the stream, and on the decoder side to decode, find and correct errors.

Table 2. Imbedding data in the video covert channel

| Encoding steps | Example                                      |
|---------------|----------------------------------------------|
| 1             | Embedder data                                | Alice went to the park                        |
| 2             | Encryption: DES                              | Vc+=! N*/^w (2 %^& X35F                      |
| 3             | Text to ascii                                | 001010101…010101010                        |
| 4             | Compression: ZIP                            | 10101…01000                                   |
The Simulator/Demo

The simulator of the defense algorithm uses MATLAB script for compressing and decompressing video stream in H264 format. In the simulator "Attack" mode, the simulator allows the user to type a hidden text that should embed into the video as a covert channel. In this mode, during the calculation of the motion vectors, the hidden information is fed into the compressed video, according to the following steps:

- Lossless data compression - LZW format.
- Data Encryption Standard - DES
- Error correction - Reed Solomon.
- Encoding the binary information according to the dictionary of the algorithms.
- Embed the data in the compressed video.

Two options are available in the attack mode:

- Preserve video quality - In this case the motion compensation error, recalculated, according the new MV, to preserve the video quality.
- Preserve approximate original video rate - in this case the old motion compensation used with the new MV, to preserve the video size.

To recover the hidden text, the decompressor side will perform the same actions in a reverse order. The simulator enables some statistical analysis and measurement of the survivability of the hidden channel. By adding a noise to the payload and measuring the hidden channel immunity.

Results

Movie Database

The video database was formed by movies with different characteristic such as: magnitude of motion between frames, amount of details or changes in a frame in the spatial domain (detailed frames tend to hold in the medium and high frequencies range more data than frames with greater smooth areas), natural and artificial objects in the video, etc.
Experiments attack description

Consider the three types of FRAMES available in the H.264 compression method:

- **I-FRAME**: Encoded without reference to other frames. Contains only INTRA-MACROBLOCK, and as a result, the compression depends only on the information that exist in the frame itself and does not use motion block prediction technique from adjacent frames.

- **P-FRAME**: Encoded with reference to previous frames. Contains INTRA-MACROBLOCK and PREDICTED-MACROBLOCK, i.e. blocks originating from the image plane (INTRA-MACROBLOCK) and blocks that can be approximated existing data in previous frames (PREDICTED-MACROBLOCK).

- **B-FRAME**: according to P-FRAME, encoded with reference to previous frames and following the sequence of time, if they may contain information that is more accurate than the current MACROBLOCK. Contains INTRA-MACROBLOCK and PREDICTED-MACROBLOCK.

Video in the compressed domain: Each frame contains a HEADER and a collection of MACROBLOCKS. We will partially decode the compressed frame (See Fig. 5):

- If the frame is an I-FRAME, its content does not contain PREDICTED-MACROBLOCK and therefore it is not suitable for the compressed domain attack.

- If the frame is a P-FRAME or B-FRAME, its contents contain PREDICTED-MACROBLOCK, therefore we will use the blocks headers as hidden channels container where we will add the malicious information.

![Video frame structure](http://www.itye.org/archives/2869)

**Fig. 5. Video frame structure**

Source: [http://www.itye.org/archives/2869]
To embed the malicious information inside the video, we are interested in PREDICTED-MACROBLOCKS. These blocks can be manipulated to hide information without disrupting or damaging the compression format. Manipulation is based on the limitations of human vision and will be realized as explained below.

Each MACROBLOCK contains a HEADER that describes the parameters needed to display the block in the image being constructed (See Fig. 6). Among other things, there are vector values (DX, DY), which describe the location of the block in the following frame relative to a reference point. The reference point is also in the HEADER.

We would like to determine whether the same PREDICTED-MACROBLOCK is suitable for malicious information and has a minimal influence on the image quality. This setting will be based on the predefined THRESHOLD VALUE parameter and will remain constant throughout the process (The threshold will serve as a secret key that will be known to the transmitter – encoder side and to the receiver – decoder side).

Setting the reference point: At the encoding and decoding stage of the PREDICTED-MACROBLOCK, Median Motion Vector (MEDIAN) is calculated via the PREDICTED-MACROBLOCK neighbors. We have calculated the MEDIAN on each axis separately and receive the reference vector (X, Y). The ESTIMATED MOTION VECTOR is describing the difference between the MEDIAN and the true MOTION VECTOR used to estimate the current PREDICTED-MACROBLOCK position.

The search for the appropriate PREDICTED-MACROBLOCK is done by scanning all the macro blocks in the FRAME, from the top left of the image to the lower right corner. In the first step, we are seeking in the MACROBLOCK headers, for a PREDICTED-MACROBLOCK.
We have reviewed the following terms: Consider the reference vector \((X, Y)\) – the median vector and its ESTIMATED MOTION VECTOR values. If the reference vector is above pre-defined threshold, and the ESTIMATED MOTION VECTOR is smaller than another threshold, the conditions are met, therefore we will use the HEADER of that MACROBLOCK as a one place holder for our hidden channel.

The algorithm will change that header with a new MACROBLOCK location according to the data we want to embed. At the decoding stage, the attacker performs the same process, search for the vectors that met the pre-defined hidden channels thresholds and extracts the information from them. Therefore, an attacker will be able to implement malicious data in all the locations where the conditions are met. If an attacker has no new data to transmit, he will send pre-defined null value (e.g. zeros).

**Embedding Data protocol**

We have created a text file with the malicious code we want to embed in the video compression domain. At some point, we may have to pad the malicious information with zeros, in order to match the size of the data to the dictionary coding method below, thus allow us to read the file in real time, from LSB to MSB. We will set a malicious data encoding within the video according to Table 3:

| DX, DY | VALUE | BIT SEQ |
|--------|-------|---------|
| 0, -1 | 0     | 111     |
| 1, -1 | 1     | 000     |
| 1, 0  | 2     | 001     |
| -1, -1| 7     | 110     |
| -1, 1 | 3     | 010     |
| -1, -1| 6     | 100     |
| 0, 1  | 4     | 011     |

| DX, DY | VALUE | BIT SEQ |
|--------|-------|---------|
| -1, -1| 0     | 111     |
| -1, 1 | 5     | 100     |
| 0, 1  | 6     | 011     |
| 0, 0  | 7     | 101     |

In Fig. 7 you can see a picture which is divided into blocks. On the left you can see the original image without the malicious code. In the right-hand picture you can see the red mark on the blocks that met the threshold conditions and in which the information was planted.
In this experiment we imbedded malicious data into five different video files with different motion behaviors - Each for 10 seconds long. We set a variable threshold value for the MEDIAN Motion Vector component (Median threshold) while setting fix delta threshold to $\sqrt{2}$. We examined the amount of information that we were able to embed in each video (bits per second) vs. video quality (PSNR). On the other hand, we examined the PSNR value obtained from the original video (which was fully deployed without information) for each MEDIAN threshold value. The results of the experiment can be seen in Fig. 8.

Note that for the “Talk show” we received a low transfer rate compared to the other video files. This can be explained by the fact that the “Talk show” has low traffic and is characterized mainly by movements with a small range of motion, which explains the small amount of motion vectors that meet the threshold criteria.

On the other hand, we notice that for the festival movie that was filmed by drone we received (for certain threshold values) a high data transfer rate. As we increased the value of the MEDIAN element threshold (reference vector) we could combine less data in the video but on the same time we have improved the PSNR result (better video quality).
Fig. 8. Experiment 1 Results

Delta threshold vs PSNR experiment (Median is fix)

In this experiment, we combined data into five different 10-second video streams, set the threshold value of the MEDIAN component to 20,000 (MMV=|20,000|), and a free variable threshold value for the DELTA element (\(\sqrt{1.5} - \sqrt{5.5}\)). We examined the amount of data we were able to embed in each video (bits per second) and the PSNR between the infected video and the original video per DELTA threshold value.

Importantly, the compression algorithm allows setting a deviation in fractions of a pixel (1.5-5.5). The motivation is to improve compression efficiency. Instead of trying to predict the location of a macroblock with pixel accuracy, we interpolate neighboring pixels and thus forecast an expected position at the resolution of pixel fractions. The results of the experiment can be seen in Fig. 9.

As with the first experiment, we note that for the Talk Show video we received a low transfer rate compared to our other video files. It can be explained by the fact that in the Talk Show, the number of movements is low and therefore it is characterized mainly by small distance movements which explain the small number of motion vectors that met the threshold.

In general, as we increased the value of the threshold conditions of the DELTA component, we were able to transmit a larger bandwidth while reducing the quality of the image.
- lower PSNR value. On the other hand, we notice that for the Festival movie that was filmed by a drone we received (for certain threshold values) a high bit rate.

![Graph of Experiment 2 Results](image)

**Fig. 9. Experiment 2 Results**

### Experiments Result Analysis

After investigating the algorithm on one specific video file (frames: 100, size: 360 x 640, frame rate: 25 frame/sec, overall bitrate: 2128 kbps, format: AVC/H.264, color space: YUV, Chrome subsampling: 4:2:0, Bit depth: 8 bits), we found out that it is possible to create a covert channel of more than 80Kbps bandwidth inside that video. Changing the threshold value (and thus changing the number of selected MVs’ Error estimation based on their size criteria) changes the covert channel bitrate. The value that determines the attack strength is the threshold level $Tr$. The lower the $Tr$ value is, the less damage is caused to video quality. However, the covert channel bandwidth is also reduced as $Tr$ decreases. Tests have shown that it is possible to increase $Tr$ beyond $\sqrt{2}$ (which corresponds to 3 bits), such that more MVs’ Error estimation are impacted, while introducing only a slight impact to video quality. Therefore, higher covert channels bandwidth can be achieved, while retaining reasonable video quality.

The $Tr$ value can be modified for different purposes. Whenever security is more important than image quality, we would recommend using a high $Tr$ value, while whenever security is less important than video quality, we would recommend using a low $Tr$ value.

The picture of Fig. 5 below illustrates varying number of MBs that meet the threshold criteria for different $Tr$ values. The yellow squares indicate Macro Blocks with MVs’ Error
estimation that meet the criteria of being smaller than $\sqrt{2}$ (e.g., $|\text{MV Error estimation}| \leq \sqrt{2}$), while the red squares indicate MBs corresponding to a larger $Tr$ value.

Fig. 10. Picture illustrating different number of MBs that meet different threshold ($Tr$) criteria

After inspecting the impact on video, we were able to create a cost-effectiveness graph of our entire defense algorithm, shown in Fig. 11. In the graph we can see the number of MBs that potentially belong to the covert channel with respect to various values of $Tr$. The numbers also correspond to varying changes of MVs’ Error estimation, minor in the case of small $Tr$ values and higher as $Tr$ increases. This approach allows users to perform risk analysis of their systems and set the acceptable quality degradation based on covert percentage.
We performed several experiments to examine the information assimilation capabilities under different conditions and for various video files. Our research and experiments demonstrate that an Error estimation of Motion Vector threshold criteria of $\sqrt{2}$ can support a covert channel transmission bitrate (CCB) of ~80kbs. This bandwidth was calculated according to formula (2).

\[
CCB = \frac{P \times F_r}{F_T} \sum_{i=1}^{N} F_i \quad (2)
\]

Where:  
$F_i=1$ if $MMV \geq D_{MMV}$ and $r \leq T_r$, else $F_i=0$  

$F_r$ - Frame Rate  
$F_T$ - Total Number of frames in the movie  
$N$ - No. of MB in the all movie  
$P$ - No. of bits per element in the dictionary

**Result data format**

Running the video database through the system in the different modes allowed us to gather statistics about various parameters of the videos in the database. These results are

![Graph indicating the smallest number of vectors per frame for different threshold](image-url)
generated after each system run in the format of an excel sheet. Example for a table generated for a single video, at a single chosen distance (see Table 4):

| Mode | Encode | Decode | SSIM | PSNR | MSE | stream length | stream size | BPS |
|------|--------|--------|------|------|-----|---------------|-------------|-----|
| 1    | 17.6613| 3.82204| 0    | 0    | 0   | 260524.6     | 41.80595703| 0   |
| 2    | 16.3703| 3.26996| 0.98954| 43.6636| 13.1316| 260524.6     | 41.80595703| 8369.36|
| 3    | 17.1072| 4.95223| 0.75144| 20.8829| 707.904| 501716.2     | 79.64306641| 12.8571|
| 4    | 17.0892| 4.99269| 0.73114| 20.2551| 823.983| 501716.2     | 79.64306641| 7855.71|
| 5    | 15.3739| 3.31049| 0.98562| 33.4831| 36.0412| 278509.6     | 44.69345703| 12.8571|
| 6    | 15.2612| 3.29157| 0.99724| 52.7244| 0.01001| 278509.6     | 44.69345703| 7545.86|

Each row in the table represents a different mode from 1 to 6:

- **Mode 1**: Baseline - This mode uses the standard video encoder and decoder to encode and decode the videos, and serves as a baseline for comparison to the other modes.
- **Mode 2**: Modified decoder - This mode uses the standard video encoder for the encoding process, but decodes the video using our modified decoder. This mode represents how a client side with modified decoder views regular, unmodified videos that contain no additional information to them.
- **Mode 3**: Modified encoder – This mode uses our modified encoder to process videos and embed them with additional data, and the standard decoder to decode the video stream. This mode represents how a video containing additional embedded information is viewed by regular standard video decoders, i.e. not the intended destination of the video.
- **Mode 4**: Modified encoder and decoder – This mode uses both our modified encoder and decoder to process the video and decode both the video and the additional information. This mode represents how the video is viewed on the intended destination of the embedded video.
- **Mode 5**: Modified encoder with corrected matrix - In this mode we use our modified encoder with the additional feature of a corrected matrix to encode the videos. This mode represents how a video containing additional embedded information is viewed by regular standard video decoders, i.e. not the intended destination of the video.
- **Mode 6**: Modified encoder with corrected matrix and decoder - In this mode we use our modified encoder with the additional feature of a corrected matrix to encode the videos, and our modified decoder to decode them. This mode represents how the video is viewed on the intended destination of the embedded video.

Color coded for the ease of visual distinction. The columns are as follows:

- **Encode** – The time it took the encoder to encode this video, in seconds.
- **Decode** – The time it took the decoder to decode this video stream, in seconds.
- **SSIM** – Calculated SSIM factor for this video in this mode.
- **PSNR** - Calculated PSNR for this video in this mode.
- **MSE** - Calculated MSE for this video in this mode.
• Stream length – The length (number of symbols) of the encoded stream, as shown in MATLAB.
• Stream size – The physical space it takes on the hard disk to store the encoded stream, in KB.
• BPS – Calculated rate of Bits per Second of embedded data transferred in the duration of the video.

Averaged results by video type category

In Table 5 we show the averaged results for each video category, at 0.25 distance:

| Table 5: Averaged results by video type category |

It is important to note the mode 1 serves as a baseline for comparison to the other modes, and represent the standard encoder and decoder; there for all image quality parameters count as 0. Also, as no additional data is embedded in the video the BPS is 0. From the results shown at Table 5 we can notice the following conclusions:

• The modified decoder produces very good results in terms of image quality (high SSIM factor) when decoding an un-manipulated video. In real application terms, that means the video produced by the modified decoder shows almost no visual quality degradation and is unlikely to be noticed by the end user.
• Encoding times are significantly longer compared to decoding times, as conforms to the H.264 standard, since all the "hard work" is done by the encoder.
• The addition of the corrected matrix produces significantly better results in all aspects: the SSIM factor is better and stream length and size are smaller.
• Embedding the additional data increases both stream size and length, although this is significantly reduced in modes including the corrected matrix.
• Videos with less movement in the background (Cartoon, nature, studio) are more efficiently compressed and produce smaller and shorter encoded streams.
• Comparing the most likely to be used mode – mode 6, shows that videos in the cartoon category provide the highest BPS and is on average, a good candidate for data embedding. The nature category makes a close second choice, while the gaming category provides the worst results.
• On average the gaming category takes the longest to encode and decode and produces the biggest encoded streams. Possibly due to the high amount of overall movement in the videos.

**Immune process**

Using our knowledge of how the covert channel implementation uses various distance vectors to find suitable locations to embed additional data in, we have modified the video decoder to operate differently and will refer to this mode of operation as 'immunity mode'. In this mode, upon detecting a motion estimation vector of up to a selected size, the decoder will randomly select a different value for the vector's representation.

This change of operation is intended to disrupt the translation of the embedded data, to such a degree that it would be unusable and will not be a potential threat to the end user. The cost of such operations is a degradation of visual quality in the decoded video, as parts of it are scrambled. The size of the vectors to be replaced could be modified, and while choosing to replace bigger vectors will provide better chance of scrambling any embedded information that could be embedded, visual quality suffers accordingly up to a point that the video is no longer watchable.

**Final Remarks**

In this paper, a unique technic was suggested for preventing cyber-attack using video streams. The new idea is based on manipulating motion vectors of compressed H.264 standard video streams. The paper is focused on prevention (not detection) of the malicious data from attacking the victim.

We demonstrated cyber-attacks that take advantage of compressed domain video payload and propose an immunity technique for improving defense efficiency. The new idea is based on manipulating motion vectors of compressed H.264 standard video streams. We have demonstrated that the attacker can achieve a covert channel of 80 kbps bandwidth.
We have demonstrated that the attacker can achieve a covert channel of 80 kbps bandwidth. This technique can be accomplished by using proxies and in near real time, without the need to decode the video. Therefore, it will consume very low CPU processing time and can become very attractive to attackers. The protection technique is very simple. It uses the same technique as the attack to insert random noise at the client side. Client can choose their protection level. If they trust the video source they can decrease the Tr value and improve video quality. On the other hand, if they suspect the video source, yet insist to view the video content, they can increase the Tr value (while reducing the video quality) and watch it safely.

In future research, we plan to demonstrate an algorithm to prevent such attacks. The algorithm will work in a similar way to that of the attack, but unlike the attack, the prevention will pick random motion vectors and will change them randomly without significantly reducing the video quality.

Acknowledgments

This work was supported by the Israel National Cyber Bureau. The authors gratefully thank Yossi Gidron and Stas Karpenko, for implementing the Matlab platform and to Michael Edry and Liran Odiz for implementing the attack algorithm.
References

Amirjan, P. and mansouri, A. (2017). "Video Watermarking Using Quantized Coefficients," in ICEE, 2017.

Amsalem, Y., Puzanov, A., Bedinerman, A., Kutcher, M., Hadar, O. (2015). "DCT-based cyber defense techniques," Proc. SPIE 9599, Applications of Digital Image Processing XXXVIII, 95991F (September 22, 2015).

Amsden, N. D., Lei Chen, Xiaohui Yuan. (2014). "Transmitting hidden information using steganography via Facebook," International Conference on Computing, Communication and Networking Technologies (ICCCNT).

Barni, M., "A DCT-domain system for robust image watermarking," Signal Processing, vol. 66, pp. 357-372, (1998).

CISCO (2018). "Cisco Visual Networking Index: Forecast and Methodology, 2017–2022," 7 November (2018). [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.

Dalbouchi, R., Dahri, S., Elhaji, M. and Zitouni, A. (2017). "Software/Hardware implementations of a video watermarking scheme based on motion vectors," ICEMIS, pp. 3-5.

Fouant, Stefan (2010). "Man in the Middle (MITM) Attacks Explained: ARP Poisoning," ShortestPathFirst (November 18, 2010)

Harsh, K Verma, Abhishek, Raman Kumar. (2009)., "Robustness of the Digital Image Watermarking Techniques against Brightness and Rotation Attack," International Journal of Computer Science and Information Security, IJCSIS, Vol. 5, No. 1.

Katzenbeisser, S. (2000). Information Hiding Techniques for Steganography and Digital, Artech House.

Kleder, Michael (2005). "Matlab lossless compression algorithm" [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/8899-rapid-lossless-data-compression-of-numerical-or-string-variables.

Long, M., Peng, F. and Li, H (2008). "Separable reversible data hiding and encryption for HEVC video," Journal of Real-Time Image Processing, vol. 14, no. 1, pp. 171-182.

Lu Jianfeng, Yang Shenhua, YANG FAN, Li (2011). "A MPEG2 Video Watermarking Algorithm Based on DCT Domain," Digital Media and Digital Content Management (DMDCM).

Morkel, T., Eloff, J.H. P. and Olivier, M. S. (2005). "An Overview of Image Steganography," Proc. the Fifth Annual Information Security South Africa Conference (ISSA2005).
Muhit, A. "H.264 Baseline Codec," (2016). [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/55901-h-264-baseline-codec-v2?s_tid=prof_contriblnk.

Neufeld, Andreas, Proc, Andrew D. Ker. (2013). "A study of embedding operations and locations for steganography in H.264 video,". SPIE. 8665, Multimedia Watermarking, Security, and Forensics.

Richardson, I. E. G (2010), The H.264 advanced video compression standard, John Wiley & Sons, (2010).

___, "H.264 encoder decoder scheme," [Online]. Available: http://slideplayer.com/slide/6832629/23/images/11/H.264+ENCODER+AND+DECODER+BLOCK+DIAGRAM+ [23,24].jpg.

Segal, Y., Hadar, O. (2018). “Covert Channel Cyber-attack based on Watermarking in the DCT domain,” ICSEE 2018, (December 2018)

Sullivan, G. (2003), " The H.264 advanced video compression standard," IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, vol. 13, pp. 560-576.

Wu, Yue. (2012). "DES encryption in Matlab" [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/37847-data-encryption-standard-des

Yahav, L. (2016). "DCT-based Cyber Attack in the H.264 Coding Standard"