Watermark-based Sensor Data Authentication

by

Zhe Feng

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Engineering
(Computer Engineering)
in the University of Michigan-Dearborn
2019

Master's Thesis Committee:

Associate Professor Hafiz Malik, Chair
Assistant Professor Samir Rawashdeh
Professor Weidong Xiang

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Professor Hafiz Malik of the Electrical and Computer Engineering Department at University of Michigan - Dearborn. He is always warm and welcoming and patient. He gave me the initial idea, guided me to the right direction when I was lost and encouraged me to overcome the challenges I met. His office becomes my favourite place in the whole university.

I would also like to thank other professors, colleagues and friends who provided help to me during this project. Without their help, this thesis would not be possible.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

ABSTRACT

Sensors have been widely used in robots, Internet of Things and automobiles. The data sent from sensor is used to detect objects, measure environment and make decision or conclusion. Since the sensor data is so critical for the system, the data from sensors must be authenticated before it is processed. The obvious approach is to use encryption. But this approach is not suitable for real-time streaming and may fail because of the noise or lossy compression. Besides, the receiver side must decrypt the data before displaying it and the encryption and decryption takes time when the data is huge, e.g. video streaming. In this paper, we propose an approach which combines encryption and watermarking to authenticate the sensor data. It has two phases and is spatial, invisible and blind-detected. We designed the approach carefully and try to achieve real-time performance. The experiment shows it is robust and fast.

CHAPTER I: INTRODUCTION

Sensors have been widely used in automobile, Internet of Things(IoT) and robot. They are used for different purposes such as detecting objects and measuring environment or carrier's information. The sensor data could be further sent to a system for making decisions or conclusion. Take automobile as an example, because of the high number of deaths caused by motor vehicle accidents (*Wikipedia contributors*, 2019a), a vehicle-related safety approach called Advanced Driver Assistance Systems (ADAS) is proposed. ADAS has a lot of functionality including but not limited to speed assistance, navigation, collision avoidance, lane keeping and intersection support. They all use sensors or external data to achieve their functionality. For example, the navigation system needs to position the vehicle by using inertial sensors and GPS. To position the other objects, the lidar and infrared sensors are used (*Lu et al.*, 2005).

Some well-known attacks to the wireless sensor networks(WSN) have been developed. For example, in Denial of Service(DoS) attack, the attacker sends enormous packets to the victim sensor so that it can exhaust the resource of that sensor. In sybil attack, the malicious node pretends to be other nodes in the network and degrade the distributed system performance. For the attacks during the transmission, attackers can use a more powerful device to monitor the network and then modify/intercept/fabricate/interrupt the packets and fools the nodes in network (*Pathan et al.*, 2006).

Since the sensor data is so critical for the system, the data from sensors must be authenticated before it is processed. The cryptographical approach may not be suitable for some cases because the encryption and decryption take time. This delay could be obvious when the data is comparably huge, such as video stream. But ADAS system may need real-time data to make decision in time. For example, if the position data is delayed, the navigation system may provide outdated information to the driver and the driver may miss the right exit. And the decryption may fail because of

error bits caused by transmission.

On the other hand, the watermark has been used widely to protect copy right. And there are also some reaches on watermark application on data security. These are discussed in next chapter. Compared to traditional security schemes, watermark is light weight and requires less computing resource.

In our approach, we propose an approach using watermarking techniques. We try to achieve data authentication in real-time. Our approach is spatial and invisible and blind-detected. It is also semi-fragile against jpeg compression. We focus on image authentication and try to prevent some known attacks. In our attack model, we assume the attacker attacks from the inside and cannot compromise any important entities nor discover the credentials and parameters used in transmission. The external attacks such as presenting a picture in front of the camera are not considered.

Although we use camera sensor in our work, the algorithm can be applied to other sensors as long as the data can tolerate some level of distortion. The experiment shows that we can authenticate the data correctly without bring too much impairment to the data itself.

The details of algorithm will be discussed in later chapters. The rest of the document is arranged as follows: chapter 2 discusses the related works, the detail of the approach is presented in chapter 3, the chapter 4 shows how the experiment is set up and its result is shown in chapter 5. The chapter 6 is the further thoughts of the project and the final chapter presents the concluding remark.

CHAPTER II: RELATED WORK

Because of the high-speed internet, online video piracy is getting easier and more popular. Now it is a big concern of the film industry. Then watermark is used by the industry to claim their copyright. A watermark system consists of an encoder and a decoder. Before the movie is distributed, the watermark is added into the movie by the encoder and then the watermarked movie is released. The Internet Service Provider (ISP) has the decoder and will reject the request if the movie to be downloaded contains the watermark. (*Asikuzzaman and Pickering*, 2018)

There are many watermark embedding techniques for different domains such as compressed domain (e.g. MPEG-2), spatial domain (e.g. LSB-based), transform domain (e.g. discrete Fourier transform). Some issues exist for video watermark. For example, the imperceptibility of the watermark means the watermark is invisible to human visual system. The payload of the watermark means the number of watermarking bits is embedded. The blind detection of the watermark system means the we can extract the watermark from the watermarked video without referring to the original video. (*Asikuzzaman and Pickering*, 2018)

Also, watermark systems are facing some attacks. In signal processing attacks, the watermark's energy is reduced because the pixel values are changed. This may happen if compression, such as MPEG-2, is adopted. In geometric attacks, the frames could be upscaled, rotated, cropped and downscaled to an arbitrary resolution and thus may remove the watermark. The temporal synchronization attacks may insert frames into original frames, drop frames, swap frames and change the rate of the frame (e.g. double the frame rate). (*Asikuzzaman and Pickering*, 2018)

Juma et. al. proposed their approach which uses watermark to protect the integrity of sensor data in their paper (*Juma et al.*, 2008). They proposed two schemes "Simplified Sliding Group Watermark" (S-SGW) and "Forward Watermark Chain" (FWC). In S-SGW scheme, they use hash function on each data reading along with secret key K. Then the synchronization point can be deter-

mined by the value of hash modulo secret parameter m and current group is longer than minimum group length L. Watermark can be calculated when two such groups are found. In FWC scheme, the difference is that the watermark is calculated based on single group itself. Thus, FWC is even faster. (*Juma et al.*, 2008)

For the image authentication, *Hu et al.* proposed their watermark algorithm that uses two semi-fragile watermarks. A watermark is semi-fragile if it is robust to acceptable content-preserving manipulations but not to malicious distortions. In their algorithm, the two watermark is generated by extracting the image feature from the low frequency domain. One is to classify intentional content modification and the other is to indicate the modified location. The authentication does not need the original image or watermark. The experiment shows their algorithm is practicable. (*Hu and Han*, 2005)

Watermarking can also be applied to audio. *Malik et al.* proposed a novel watermarking scheme applied on audio. It is based on frequency-selective spread spectrum. Compared to similar works, it doesn't use the whole audible frequency range to embed watermark. Instead, it randomly selects subbands(s) signal(s) to do that. There are two blind watermark detection methods. One is based on estimation correlation and the other is based on normalized correlation. The proposed approach has better fidelity, secure embedding and other nice attributes. (*Malik et al.*, 2008)

Recently, *Azeem et al.* proposed a novel approach which uses physical-fingerprinting of controller area network. They found that each Electronic Control Unit(ECU) has unique frequency response. So, in their approach, they use this unique physical property of ECUS to identify which ECU is sending the data. (*Avatefipour et al.*, 2017) (*Hafeez et al.*, 2017) (*Tayyab et al.*, 2018) (*Hafeez et al.*, 2018)

CHAPTER III: DESIGN

3.1 Introduction

In this chapter and the experiment in later chapters, we use data from camera sensor as the example. But generally, this solution can be applied to other kinds of sensors as long as it can accept an amount of intentional modification.

There are two phases in the proposed approach. The first phase is the negotiation phase and the other phase is the watermarking phase. In the later phase, if it is a transmitter, it will embed the watermark into the video in this phase. If it is a receiver, it will extract the watermark from the video and verify the watermark in this phase.

In the negotiation phase, the two entities will negotiate the parameters used during the transmission and watermarking. After the negotiation, the transmitter takes live video, watermarks each frame and sends them to the receiver. And the receiver receivers the data and authenticate each frame. If authentication fails, there will be a warning message displayed on the screen.

Because the video steam consists of continuous image capture from the camera. For clarity, we name each such image capture a "frame" and denote it by F_i where the i is the index of the frame in the sequence. Then we define the original video stream V as an infinite sequence of frames:

$$V = \{F_1, F_2, ...\}$$

The watermark embedding phase generates a new frame F_i' for each original frame F_i. And we define the watermarked video steam V' as an infinite sequence of frames:

$$V' = \{F_1', F_2', ...\}$$

The following sections of this chapter describe those phases in detail.

3.2 Negotiation Phase

In this phase, the transmitter initiates the negotiation with the receiver. They negotiate the parameters that are used in the transmission. These parameters include but not limited to the percentage of the data to be watermarked, the transmission model and credentials or keys used during watermarking.

Although the transmitted sensor data doesn't need to be confidential, these parameters themselves have to confidential. Otherwise the attacker can simply eavesdrop the parameters and forge the data to fool the receiver.

To achieve the confidentiality, we investigated some symmetric (*wini J*, 2015) and asymmetric cryptosystems (*Kuhn et al.*, 2001). Then we choose the RSA public key cryptosystem. It is only used in negotiation phase and thus the delay caused by encryption and decryption is acceptable because this phase only occurs once in the whole transmission. In our design, we use the following functions:

1. *Sign(data, Priv)* which uses a private key and creates a signature for input data. The *Priv* stands for the private key.

2. *Verify(signature, Hash Value, Pub)* which uses a public key and verify the identity of the signature.

3. *SHA384(data)* which generates the hash for input data. There are some other hash functions available for RSA. We choose SHA384.

4. *Encrypt(data, Pub)* which encrypt data with a public key. The *Pub* stands for the public key.

5. *Decrypt(data, Priv)* which decrypt the data with a private key.

The public keys and private keys for the sensors and ADAS can be pre-installed in the devices

or retrieved from certification authority (CA). If the keys are pre-installed, the hardware security module (HSM) is need.

We have considered two options to do this. One is encrypt-then-sign and the other is sign-then-encrypt.

3.2.1 Option I: encrypt-then-sign

If we use encrypt-then-sign, the transmitter first encrypts the parameters data with receiver's public key. Then it uses its own private key to sign the encrypted message and generate the signature. The encrypted message along with the signature is sent to the receiver. For the receiver, it does the reversed process. It first verifies the signature with transmitter's public key and then decrypt the message and get the parameters.

For the transmitter, the process can be expressed by algorithm 1 and the data flow diagram is shown in figure 3.1.

Algorithm 1: Transmitter process when using encrypt-then-sign

Cipher Text $\leftarrow Encrypt($*Parameters*$, Pub_R)$;

Hash Value $= SHA384($*Cipher Text*$)$;

Signature $= Sign($*Hash Value*$, Priv_T)$;

$Transmit($*Cipher Text*$||$*Signature*$)$;

In the pseudocode, the *Cipher Text* is the data after encryption. Pub_R means the public key of receiver. $Priv_T$ stands for the private key of transmitter. And "||" means concatenation and this concatenation should also provide some metadate so that the receiver can split the data correctly.

For the receiver, once it received the data, the process can be expressed in algorithm 2 and the data flow diagram is shown in figure 3.2.

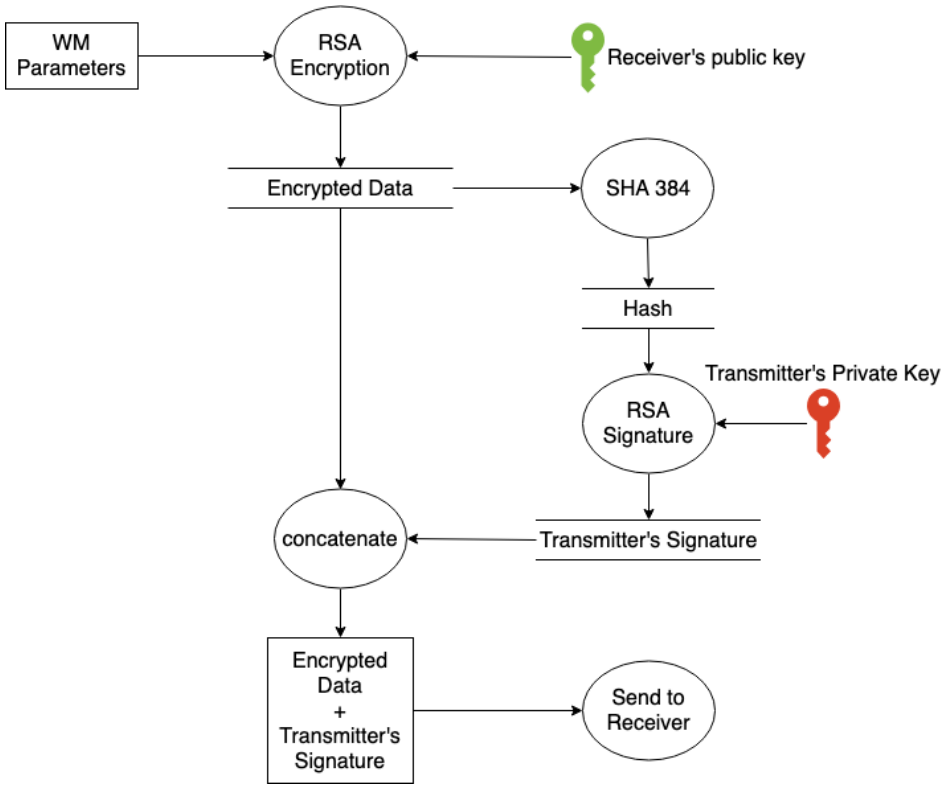


Figure 3.1: Option I's Data Flow Diagram for Transmitter

Algorithm 2: Receiver process when using encrypt-then-sign

Hash Value $\leftarrow SHA384($*Cipher Text*$)$;

if $Verify($*Signature*, *Hash Value*, $Pub_T)$ **then**

    Parameters $\leftarrow Decrypt($*Cipher Text*$)$;

else

    Reject the request;

end

3.2.2 Option II: sign-then-encrypt

If we use sign-then-encrypt, the transmitter first calculates the hash and signs the parameters data with its own private key and generate the signature. Then it uses receiver's public key to encrypt the whole message that consists of parameters data and the signature. The encrypted whole message is sent to the receiver. After transmission, the receiver decrypts the whole message with

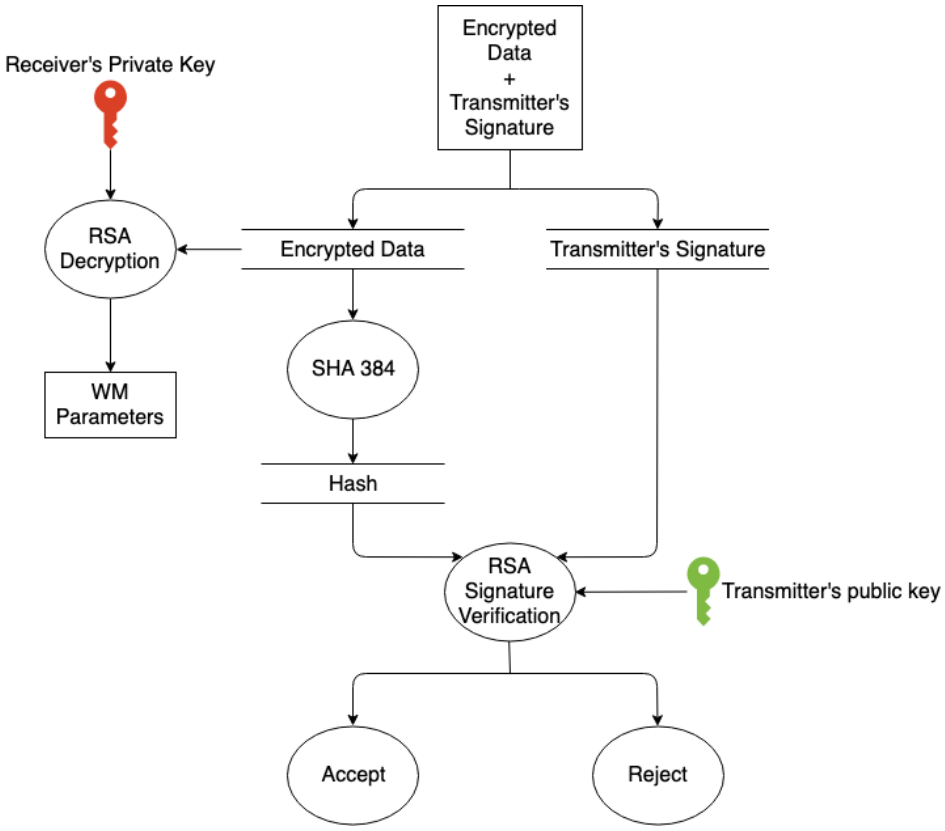


Figure 3.2: Option I's Data Flow Diagram for Receiver

its own private key and verifies the signature with transmitter's public key.

Similarly, for the transmitter, the process can be expressed by algorithm 3 and the data flow diagram is shown in figure 3.3.

Algorithm 3: Transmitter process when using sign-then-encrypt

Hash Value $\leftarrow SHA384($*Parameters*$)$;

Signature $\leftarrow Sign($*Hash Value*$, Priv_T)$;

Cipher Text $\leftarrow Encrypt($*Parameters*$||$*Signature*$, Pub_R)$;

$Transmit($*Cipher Text*$)$;

For the receiver, once it received the data, the process is expressed by algorithm 4 and the data flow diagram is shown in figure 3.4.

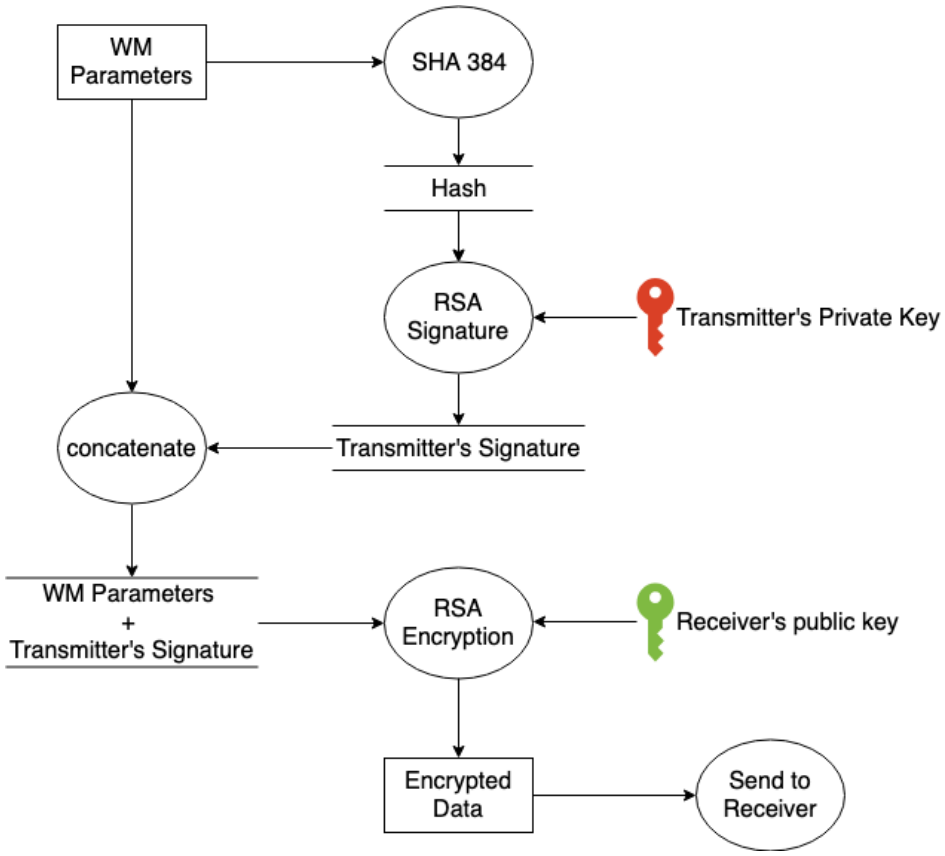


Figure 3.3: Option II's Data Flow Diagram for Transmitter

Algorithm 4: Transmitter process when using sign-then-encrypt

Parameters||*Signature* $\leftarrow Decrypt($*Cipher Text*$)$;

Hash Value $\leftarrow SHA384($*Parameters*$)$;

if $Verify($*Signature*, *Hash Value*, $Pub_T)$ **then**

    Accept parameters;

else

    Reject the request;

end

In our case, these two options are both suitable. So both of them are implemented and can be chosen by run-time arguments provided to the program.

After this phase, the transmitter and the receiver should have enough information for the next phase.

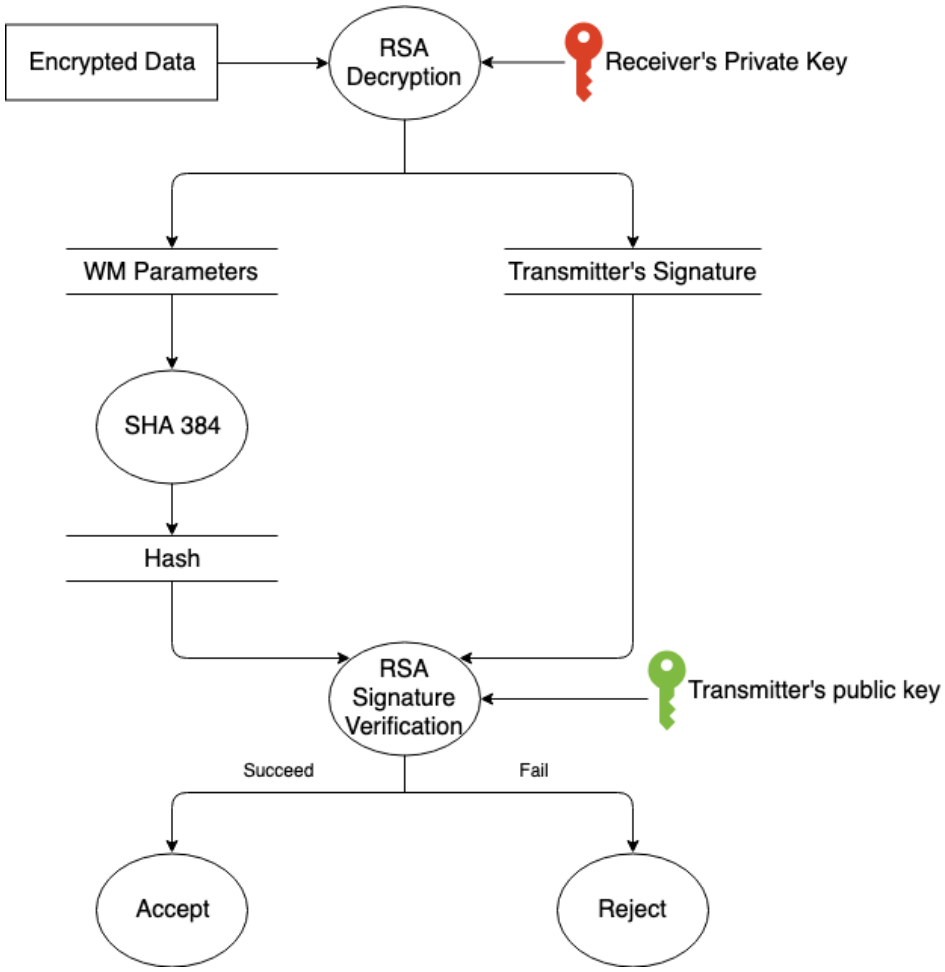


Figure 3.4: Option II's Data Flow Diagram for Receiver

It is worth noting that there is a potential replay attack in this phase. The attacker eavesdrops the network and record the negotiation request from transmitter and following frames. Although the attacker cannot decrypt data and get the parameter, he can replay it to the receiver directly and then replay the frames he has eavesdropped. Because the all the data is legitimate, so the receiver will be fooled and accept the frames just like it was from the transmitter.

To prevent this, we have to also ensure the data freshness. We added a time-stamp field among the parameters, so when the receiver receives a negotiation request, it will first validate the time-stamp. If the time-stamp is too old, it will reject the request. To make this work, the time skew between them shall not be too large.

3.3 Watermarking Embedding Phase

The transmitter takes the video streaming and embeds watermark into each frame. We defined a special algorithm to generate the different watermark information for each frame. So even if our attacker reveals the watermark information of a frame, he cannot use that information to discover watermark information of other frames.

In general, the transmitter captures the image from the camera, transforms it into a more suitable representation, embeds the watermark and transforms it back to image. Then it compresses the image by jpeg and transmits the data to the receiver.

In a more formal way, we define a function $Generate\_Watermark$ that generates unique watermark information for each frame:

$$W_i = Generate\_Watermark(F_i)$$

where F_i is the *i-th* frame and W_i is the generated watermark.

Note although it only takes the current frame data as the input, it also uses the some internal information including the parameters negotiated in previous phase.

The transformation function $Transform$ does the transformation to the frame. The embedding function $Embed$ embeds the watermark into the frame. The reverse transformation function $Transform^{-1}$ reverses the transformation. The pseudocode can be found in algorithm 5:

Algorithm 5: Watermark Embedding

$W_i \leftarrow Generate\_Watermark(F_i)$;

$T_i \leftarrow Transform(F_i)$;

$T_i' \leftarrow Embed(T_i, W_i)$;

$F_i' \leftarrow Transform^{-1}(T_i')$;

$Transmit(Compress(F_i'))$;

Where T_i stands for the data in transformed representation. Then we use a compress function $Compress$ (in our case, it is JPEG with default quality parameter) and transmission function

$Transmit$ to send data to receiver.

If we remove the intermediate variables in the formula, the watermark embedding process can be expressed as:

$$F_i' = Transform^{-1}(Embed(Transform(F_i), Generate\_Watermark(F_i)))$$

The flow diagram of process is also shown in figure 3.5.

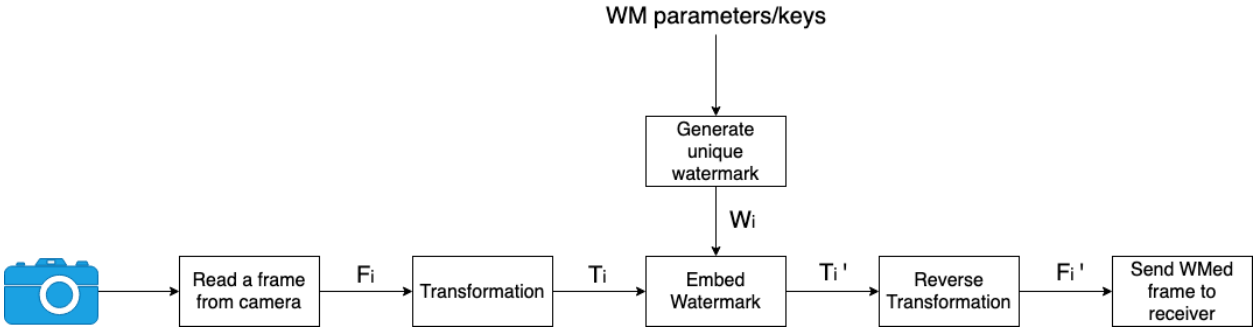


Figure 3.5: Flow Diagram of Watermark Embedding Phase

To improve the performance, the implementation is little different than what is described in this chapter. And there are other features that can be enabled for this phase. They are described in later sections.

3.4 Watermarking Verification Phase

The receiver receives the data from transmitter and recovers the video frames and verifies the watermark. It must use the same special algorithm to generate the watermark information for each frame. In general, it follows the similar process but in a reversed way. It restores the frame from the received data, do the transformation and extracted the watermark. Then it calculates the correlation between the extracted watermark and the generated watermark. If the correlation is above the threshold, the authentication is successful; Otherwise, the received data is suspicious.

Similarly, using the same notation, the process can be expressed in algorithm 6.

Algorithm 6: Watermark Embedding

$W_i \leftarrow Generate\_Watermark(F_i)$;

$T_i = Transform(F_i)$;

$W_i' = Extract(T_i, W_i)$;

if $Verify(W_i', W_i)$ **then**

    Inform the user the frame is safe;

else

    Warn the user the frame is suspicious;

end

Where the $Extract$ is the function which uses generate watermark information to extract the watermark from the received frame. And $Verify$ also uses the generated watermark information to calculate the correlation and compare it with the threshold.

The correlation definition is simple and we choose it because of real-time requirement:

$$\frac{W_i' \cdot W_i}{\mid W_i' \mid\mid W_i \mid}$$

Where the W_i is the locally generated watermark and the W_i' is the watermark extracted from the received video. The correlation could be with or without the absolute symbol. Because there are thousands of watermarked pixels, the correlation should be relatively very small if W_i and W_i' are generated with different watermark parameters. Thus, the sign of them doesn't matter. For example, correlation -0.05 and 0.05 both indicate that the received video is forged. So, the correlation could also be:

$$\left| \frac{W_i' \cdot W_i}{\mid W_i' \mid\mid W_i \mid} \right|$$

The threshold value is chosen based on the experiment data.

The control flow diagram is shown 3.6.

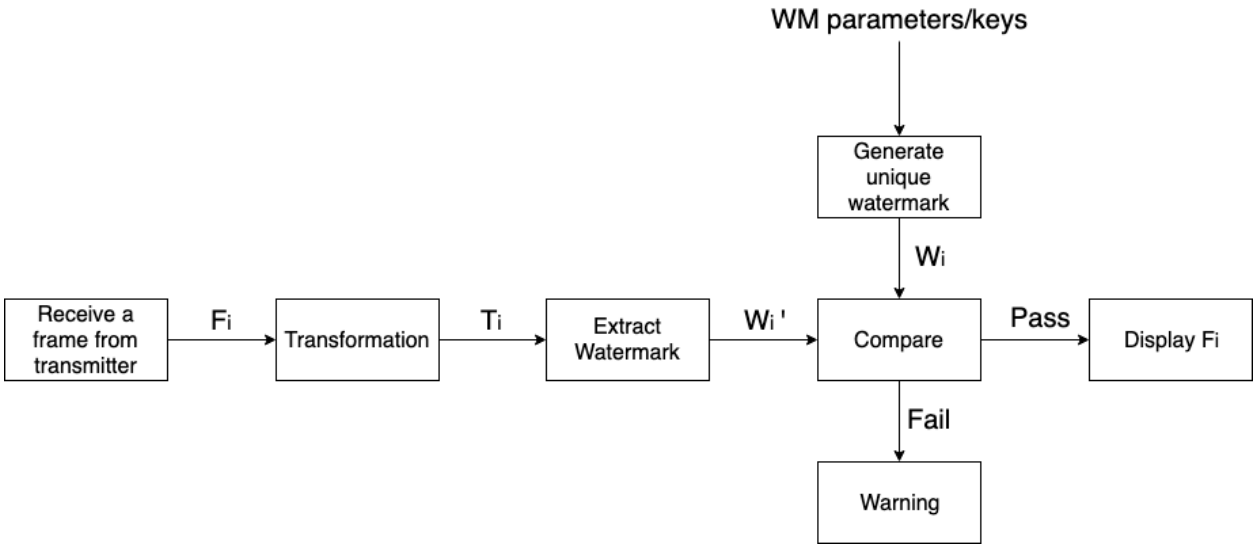


Figure 3.6: Watermark Verification Phase

3.5 Block-Based Feature

In the watermark phase, the watermark information can be generated for each block of frames. For example, the first fifty frames use the same watermark information. Then the new watermark information is generated for the next fifty frames.

In this way, the overall performance should be better. Because for the frames in the same block, we don't have to generate new watermark info for each frame.

But this feature may make the algorithm vulnerable to the frame insertion attack because a replied frame is verified as valid.

The idea can be illustrated in figure 3.7.

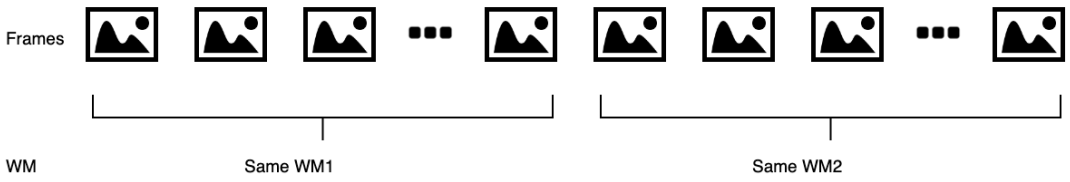


Figure 3.7: Block-Based Feature Illustration

3.6 Synchronization Frame Feature

It is possible to have some frames lost during the transmission. In this way, the transmitter and the receiver may generate different watermark information for the same frame. And this leads to verification failure for all the frames after the lost ones.

To solve this issue, we proposed a feature called “synchronization frame feature”. This feature uses fixed watermark information called “synchronization watermark” that must be known to the both sides before the transmission. Because the attacker should not know the block size and the location of synchronization frames, we assume that the attacker cannot precisely insert frames between the synchronization frames and cannot delete all the synchronization frames of a block.

The transmitter uses the synchronization watermark for the last few frames in each block. The receiver calculates the correlation between the watermark in the received frame and the synchronization watermark. If the correlation reaches the threshold, it will regard it as a synchronization frame. Then it prepares the watermark information for next block and wait for the next non-synchronization frame. Once it finds a non-synchronization frame, it starts the verification process for next block.

In this way, even if there are few frames lost or deleted by deletion attack during the transmission, the impact is constrained in that block of frames.

The idea is also illustrated in the figure 3.8.

And also, to fight against the insertion attack, the receiver would not generate new keys for next block unless it receives at least one synchronization frame.

It is also worth noting that we should calculate the synchronization watermark even though we can calculate the synchronization watermark in advance. This is to prevent the potential side-channel attack. If we don’t do it, the attacker can monitor the network and may find that few frames are prepared relatively faster. And further the attacker may discover the size of the block and number of the synchronization frames.

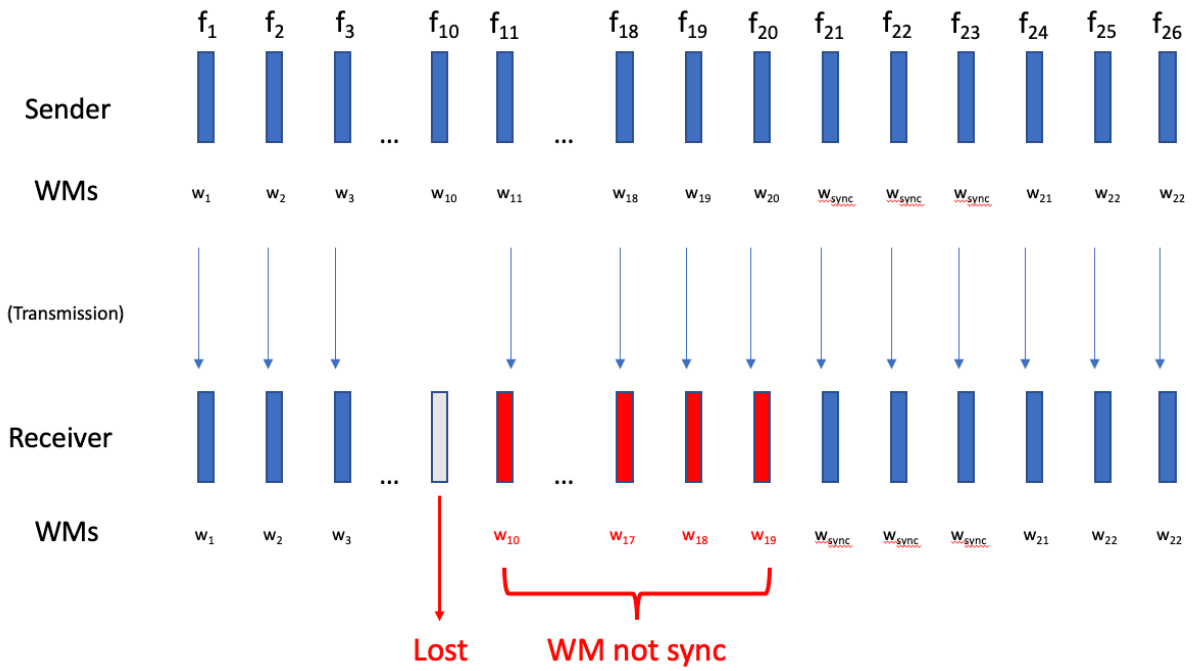


Figure 3.8: The 10-th frame is lost during the transmission and the verification of all later frames in the same block fail. But in next block, the keys are synchronized again.

3.7 Separate Thread to Generate Watermark Information

To improve the performance even further, the whole process to generate watermark information for frames or blocks can be delegated to a separate thread. To implement this, a new thread, called "generating thread", is created for this task. It communicates with the main thread through a queue.

Then the only difference is that all the watermark information generation in the process is moved to the thread and the main thread retrieves them instead of generating itself.

The idea is also illustrated in figure 3.9.

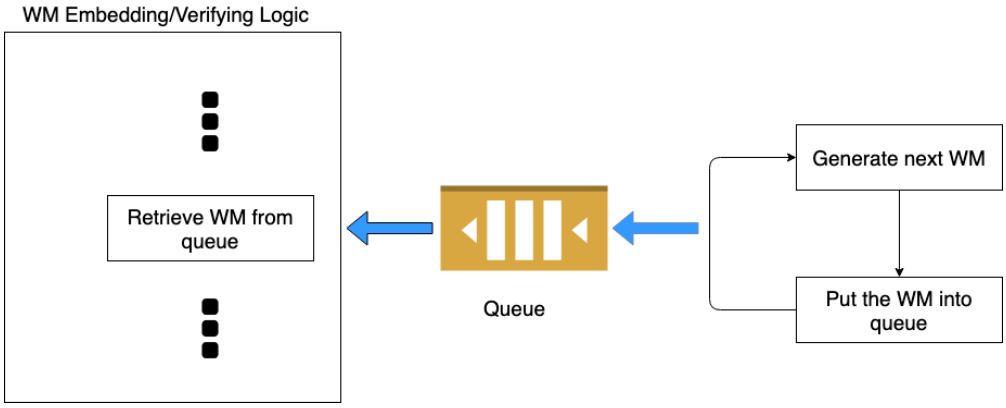


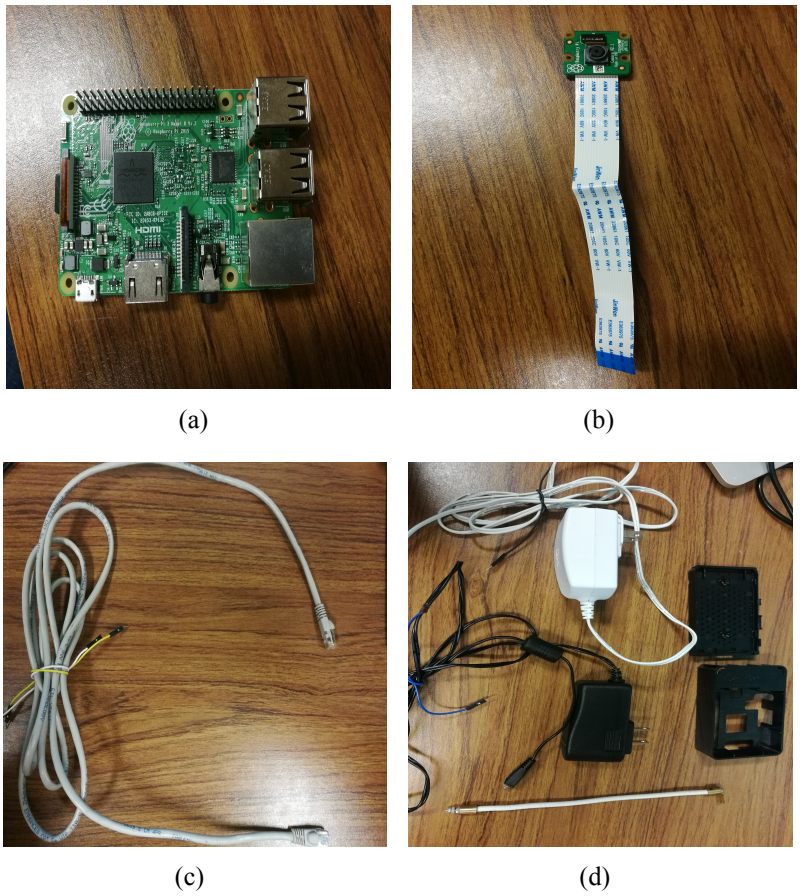
Figure 3.9: Separate Thread to Generate Watermark Information

CHAPTER IV: IMPLEMENTATION

4.1 Hardware

The following devices are used in the experiment:

1. Two Raspberry Pis (Raspberry Pi 3 Model B Rev 1.2) and their power supply. One is the transmitter and the other is the receiver.
2. One Pi camera. Version 2.1. It is mounted on the transmitter.
3. An Ethernet wire.
4. A computer which is used to run the object recognition algorithm.
   (a) Operation system: Ubuntu 18.04
   (b) CPU: Intel Core 2 Quad CPU Q9950
   (c) Memory: 4GB



(a) (b)

(c) (d)

Figure 4.1: The used devices: (a) Raspberry Pi board; (b) Ethernet; (c) Computer; and, (d) Other peripherals

4.2 Technologies

The following technology is used:

1. Python. An elegant script language. We developed our demo quickly with it. We use some bitwise operators to optimize the calculation.

2. OpenCV. Open Source Computer Vision Library (OpenCV) (*Wikipedia contributors*, 2019c). It is an open source computer vision and machine learning software library. It is designed for computational efficiency and real-time applications. It also provides Python APIs. We delegate the time-consuming tasks to Numpy so that the overall speed is really fast.

3. Numpy. A fundamental library provides for scientific computing. We need a lot of computation for the watermark. Numpy (*Wikipedia contributors*, 2019b) is the most efficient one we found for this task.

4. YOLOv3. There are many object recognition algorithms using deep learning, such as AlexNet (*Krizhevsky et al.*, 2012) and ZFnet (*Zeiler and Fergus*, 2014), and other approaches such as support vector machine (*Pontil and Verri*, 1998) and through kinect using Harris Transform (*Hafeez et al.*, 2014). We used third version of "You only look once" (YOLO) algorithm (*Redmon and Farhadi*, 2018) which is relatively new. In the experiment, we use the pre-trained network directly and OpenCV 3.4.2 implementation of YOLOv3.

5. Linux bash script. We need to test our watermark algorithm with different parameters and also test the YOLOv3 performance on them. We prepare the data and write bash script to automate the whole process.

And in the experiment, we pre-install the RSA keys in the two Raspberry Pis.

4.3 Watermark Embedding Implementation Detail

Video data is relatively large, and it takes time to transmit each frame. During our implementation, we found that if we do the video capture and transmission synchronously, we may lose some frames because the program has to wait for the transmission and then take the next frame.

We solve this problem by creating another thread called "sending thread" besides the main thread. The main thread takes video frames, apply the watermark and then puts them into a thread-safe first-in-first-out double-ended queue. The other thread retrieves data from the queue and sends it to the receiver.

The detailed pseudocode is described in algorithm 7.

Algorithm 7: Detailed Watermark Embedding

$Init()$;

$SendingThread(queue, receiv)$;

while $F_i \leftarrow Capture()$ **do**

$W_i \leftarrow Generate\_Watermark(F_i)$;

$T_i \leftarrow Transform(F_i)$;

$T_i' \leftarrow Embed(T_i, W_i)$;

$F_i' \leftarrow Transform^{-1}(T_i')$;

$SendToQueue(queue, F_i')$;

end

In the pseudocode, the $Init()$ initializes and prepare objects such as camera, queue and socket. The $SendingThread()$ starts a thread which reads data from queue and sends data to the receiver. The $SendToQueue()$ sends frame to the shared queue.

For the sending thread, it uses simple algorithm 8.

Algorithm 8: Sending Thread

while $F_i' \leftarrow Read(queue)$ **do**

$Transmit(Compress(F_i'))$;

end

The idea and control flow are also shown in figure 4.2.

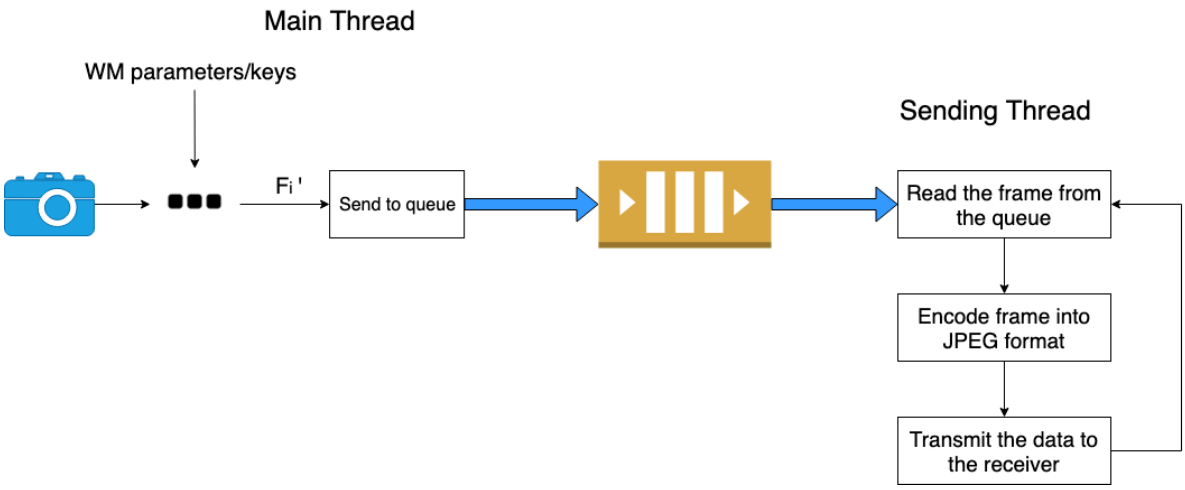


Figure 4.2: Control Flow

CHAPTER V: EVALUATION

In this chapter, we show the result of our experiment. And during our experiment, we found that there is always a trade-off between the parameters that negotiated at first. Unless otherwise specified, we use a group of "proper" parameters values which have a balance between WM speed, compressed frame size, video impairment and verification rate.

The picture of experiment looks like in figure 5.1:

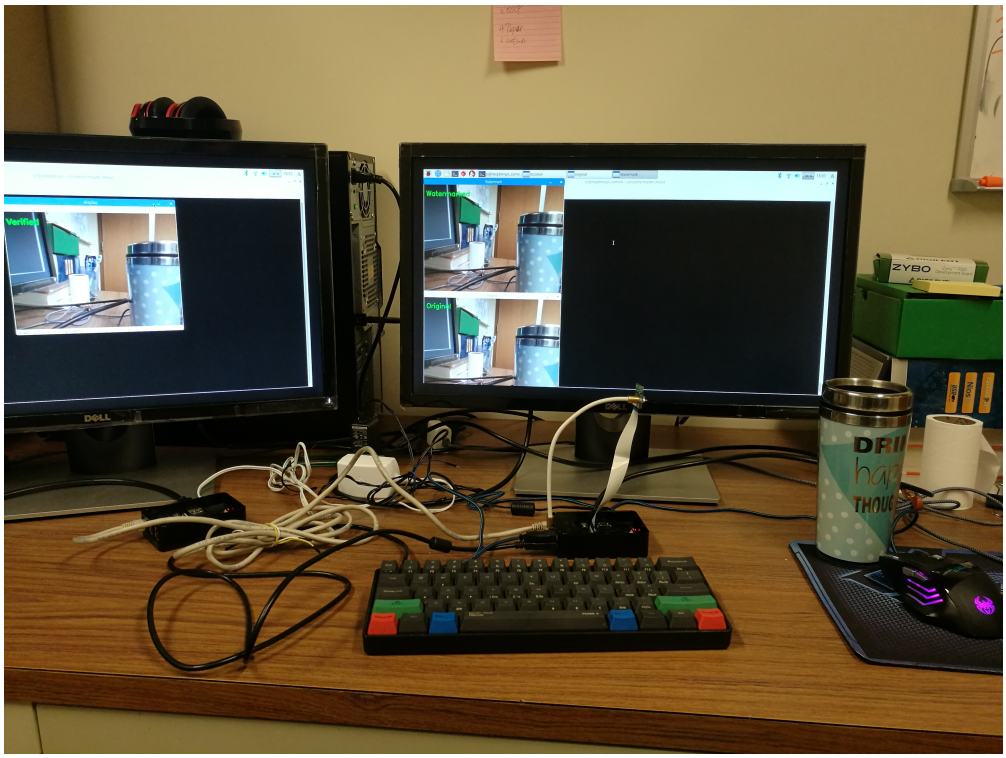


Figure 5.1: Experiment Setup

5.1 Evaluation Targets

We want to acquire the following information of our algorithm via experiment:

1. How much is the correlation between:
   (a) frames with same watermark
   (b) frames with different watermark
   (c) frames with watermark and those without watermark
2. What value should the threshold be?
3. How long would it take to generate watermark, apply watermark, transmit the frame, extract the watermark, and verify the watermark?
4. Is the performance of object recognition affected by the watermark?
5. How is the performance of when other features are enabled?
6. What is the performance of the algorithm under intended noise in the transmission?
7. What is its performance under some temporal attacks such as frame insertion, deletion, swapping?

5.2 Correlation and Threshold

Theoretically, if there is no loss during the compression and transmission, the correlation should reach 100%. But since we use lossy compression for each frame before the transmission, there is degradation and loss of quality and thus the correlation is not 100%.

The following experiments are performed with compression. For each experiment in this section, we tested ten batches and each batch has 30 frames and we calculated the average correlation of each batch. Based on the experiment, we believe we can choose 50% as the threshold.

5.2.1 Watermark Detection: Same Watermark

This is the idea case. With the same watermark parameters, the transmitter transmits the frames with same watermark generated by the receiver. Theoretically, it should be very high. But we also

need to know its value in our implementation.

Part of the experiment data is listed in table 5.1.

Batch	Average(%)	High(%)	Low(%)
1	64.38	73.90	55.79
2	65.62	71.91	50.86
3	66.28	72.30	59.04
4	64.94	70.78	55.99
5	64.52	72.39	56.48
6	63.71	73.37	54.84
7	64.91	72.45	55.49
8	64.69	73.38	57.81
9	65.34	72.63	58.19
10	66.65	72.15	60.50

Table 5.1: Correlation between frames with same watermark

5.2.2 Watermark Detection: Different Watermark

If the transmitter uses different parameters to generate the watermark, or an attacker who doesn't have the parameters sends forged watermark to the receiver, the receiver will receive frames with different watermark. The theoretical correlation should be low.

In this experiment, we let the transmitter use different watermark parameters. The difference between parameters is very tiny.

Part of the experiment data is listed in table 5.2.

As we can see, although the difference is really tiny, the correlation rate drops dramatically. So our algorithm is very sensitive to the parameters.

5.2.3 Watermark Detection: No Watermark

We also tested the correlation between frames with legitimate watermark and frames without any watermark. This could happen if the attacker just sends the original video frames. Theoretically, the correlation should also be low.

Part of the experiment data is listed in table 5.3.

Batch	Average(%)	High(%)	Low(%)
1	-0.17	11.50	-13.08
2	-0.38	6.85	-9.98
3	1.00	13.60	-16.39
4	-0.28	11.97	-16.27
5	-0.58	15.87	-12.72
6	0.39	14.90	-11.16
7	0.29	10.09	-8.55
8	0.16	11.44	-11.46
9	0.04	11.80	-8.43
10	-0.10	13.09	-14.81

Table 5.2: Correlation between frames with different watermark

Batch	Average(%)	High(%)	Low(%)
1	-2.02	6.60	16.41
2	0.38	7.32	-17.71
3	-0.41	12.86	-13.46
4	-0.33	8.02	-11.32
5	0.08	12.75	-13.92
6	-0.97	8.09	-8.17
7	0.34	12.24	-12.45
8	0.38	15.56	-20.69
9	0.99	10.30	-13.61
10	-0.28	8.37	-13.65

Table 5.3: Correlation between frames with watermark and without watermark

5.3 Detection Performance

Based on the threshold we choose, we experiment the *precision* and *recall* under different circumstances. The definition of *precision* and *recall* is same as commonly used in machine learning classification:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Where TP means true positive, FP means false positive and FN means false negative. If the

receiver believes the frame is legitimate, we will call this verification is *positive*. Otherwise, we call it *negative*.

5.3.1 Legitimate Data

This is the idea scenario between the transmitter and the receiver. So all of the frames are legitimate and the ideal precision and recall should be 100%.

No.	N	TP	FP	TN	FN	Precision(%)	Recall(%)
1	2067	2067	0	0	0	100	100
2	2212	2211	0	0	1	100	99.95
3	2584	2583	0	0	1	100	99.96
4	2864	2864	0	0	0	100	100
5	2117	2117	0	0	0	100	100
6	2577	2577	0	0	0	100	100
7	2288	2287	0	0	1	100	99.96
8	2787	2786	0	0	1	100	99.96
9	2258	2258	0	0	0	100	100
10	2204	2204	0	0	0	100	100

Table 5.4: Detection performance of legitimate sensor data. We included some very rare case.

From the experiment, we can see that the algorithm has almost 100% precision and recall. And the false positives we found in the experiment, their correlations range from 49.25% to 49.99% correlation, which are very close to the threshold.

5.3.2 Replace with Forged Data

In this experiment, we replace a portion of the legitimate frames with the original frames which contains no watermark. Each frame has 10% probability to become the original one.

Part of the experiment data is listed in table 5.5.

In previous test, the precision and recall are both 100%.

No.	N	TP	FP	TN	FN	Precision	Recall
1	2719	2442	0	277	0	100	100
2	2403	2172	0	231	0	100	100
3	2142	1934	0	208	0	100	100
4	2191	1945	0	246	0	100	100
5	2481	2243	0	238	0	100	100
6	2378	2132	0	246	0	100	100
7	2792	2488	0	304	0	100	100
8	2501	2267	0	234	0	100	100
9	3031	2729	0	302	0	100	100
10	2643	2379	0	264	0	100	100

Table 5.5: Detection rate of forged sensor data

5.3.3 Replace with Replayed Data

In this experiment, we replace a portion of the legitimate frames to the replayed frames and test whether the receiver can detect them. For each watermarked frame, the transmitter has 10% chance to replace it with previous watermarked frame.

Part of the experiment data is listed in table 5.6.

No.	N	TP	FP	TN	FN	Precision	Recall
1	2031	1825	0	205	1	100	99.95
2	2193	1972	0	221	0	100	100
3	2344	2094	0	250	0	100	100
4	2274	2046	0	228	0	100	100
5	2473	2221	0	252	0	100	100
6	2134	1922	0	212	0	100	100
7	2281	2040	0	241	0	100	100
8	2493	2229	0	264	0	100	100
9	2033	1835	0	198	0	100	100
10	2269	2056	0	212	1	100	99.95

Table 5.6: Detection rate of replayed sensor data

5.3.4 Swap Attack

In this experiment, we evaluate the performance of the algorithm under swap attack. To simulate the attack, for each pair of frames, they have 10% chance to be swapped. And we regard the two

swapped frames are both invalid.

The experiment result is shown in table 5.7.

No.	N	TP	FP	TN	FN	Precision	Recall
1	2030	1660	0	370	0	100	100
2	2128	1714	0	414	0	100	100
3	2294	1866	0	428	0	100	100
4	2258	1840	0	418	0	100	100
5	2108	1689	0	418	1	100	99.94
6	2048	1682	0	366	0	100	100
7	2149	1731	0	418	0	100	100
8	2165	1795	0	370	0	100	100
9	2115	1737	0	378	0	100	100
10	2138	1732	0	406	0	100	100

Table 5.7: Detection rate under swap attack

Without the synchronization feature, the frame insertion and dropping attack will cause the receiver to detect all the frames after the insertion or dropping.

5.4 Speed

Like many cryptography algorithms, in our algorithm, we also need to generate a lot of random numbers so that it is almost impossible for the attacker to forge the same watermark information.

We calculated the time used to do the watermarking for each frame. And we also record the time used to generate the random numbers and the time used to in other parts of the algorithm.

The table 5.8 contains some experiment data from transmitter. The first column shows watermark percentage, i.e. how much percentage of the image is watermarked. Other columns show the percentage of the time used in the corresponding process. The transmission speed is not included because its speed is irrelevant to our algorithm. In our implementation, it normally takes 0.013 seconds to transmit a 480x640 frame.

From the experiment data, we found that the time used to generate the watermark increases as the watermark percentage increases while the time used to embed the watermark almost keeps

WM Percentage(%)	WM Generation(%)	WM Embedding(%)
1	13.59	86.41
5	24.19	75.81
10	34.72	65.28
15	41.39	58.61
20	47.12	52.88
25	51.69	48.31
50	67.64	32.36
75	74.81	25.19
100	78.92	21.08

Table 5.8: Speed

same. This is because the watermark embedding uses binary operation on the whole image while the watermark generation is based on how much of the image is to be watermarked.

5.5 Object Recognition Performance

Since we add some dither into the video, we want to know whether the watermark degrades the performance of object recognition and how severe it is. We use the pre-trained parameters from Darknet github repository (*Redmon*, 2013–2016).

In this experiment, we didn't do it in real-time like previous experiments. This is because the Yolov3 takes time to recognize the objects in each frame and that's why we need a PC to do the calculation. We actually tried to run the Yolov3 on Raspberry Pi directly so we don't need to copy videos to the PC. But It turned out one Raspberry Pi might take a whole week to analyze the videos.

We define the performance *P* of the Yolov3 on watermarked videos in following formula:

$$P = \frac{\sum_{i=0}^{N} \frac{m_i}{n_i}}{N}$$

Where the N is the number of frames and n_i is the number of the objects recognized in the original i-th frame and m_i is the number of the same objects recognized in the corresponding watermarked i-th frame.

And to simplify the experiment, we put only single object in the video or different multiple

objects in the video. In this way, to determine whether the object is recognized, we only need to check whether the object's ID is in the generated object ID list.

We tested the performance of Yolov3 under different percentage of watermark. We used four videos: static single-object video, moving single-object video, static multiple-object video, moving multiple-object video. And their performance is shown in graph 5.2. And the screen-shot of the generated video is shown in 5.3.

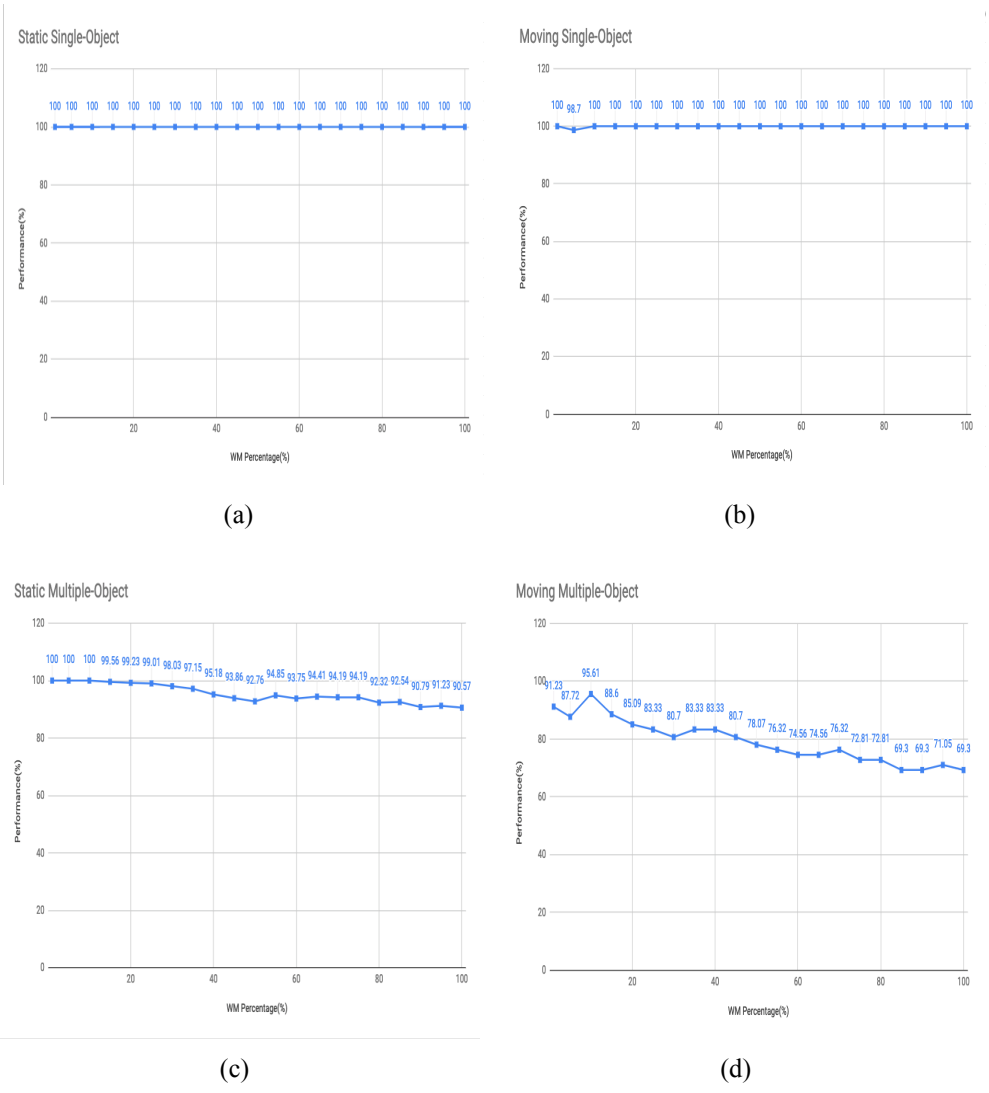
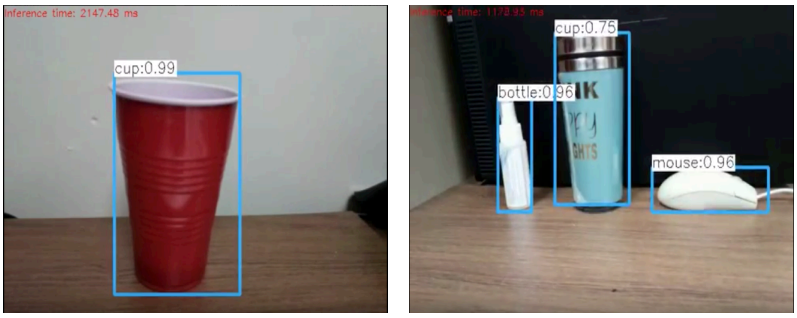


Figure 5.2: The used devices: (a) Yolov3 applied on a static single-object video; (b) Yolov3 applied on a moving single-object video; (c) Yolov3 applied on a static multiple-object video; and, (d) Yolov3 applied on a moving multiple-object video



(a) (b)

Figure 5.3: Screen-shot of the Yolov3 result: (a) for single-object video; (b) multi-object video

5.6 Performance under Intended Noise

Although we have proved that our watermark algorithm is semi-fragile via JPEG encoding, we also tested its performance under certain degree of intended noise. Without the knowledge of the algorithm, the noise is added in a blind way so that every pixel may have some bits inverted because of the noise. The noise is generated from random numbers. The degree of the noise is defined as how much percent of the watermark information could possibly be corrupted by the noise. For example, 25% degree of noise means that the generated random noise could corrupt 25% of the generated watermark at most. Note that the noise is added to the video before the JPEG encoding and transmission.

We did the same experiment in section 5.3.2 using noise degree of 25%, 50%, 75% and 100% respectively. The experiment shows that the precision and recall stays almost 100% for 25% and 50% noise degree. With 75% noise degree, the precision is still 100% but recall drops to 72.29%. With 100% noise degree, the precision and recall drops to 0 because all the frames are suspicious to the receiver.

5.7 Sync Frames Feature under Attacks

5.7.1 Insertion Attack

In this experiment, we inject one frame randomly into the video stream from transmitter. Then we compare the logs generated by the transmitter and the receiver to know whether the receiver get synchronized again in next block. As mentioned in section 3.6, we assume the attacker cannot insert the frames precisely between the synchronization frames.

The log is shown in 5.4. For the log, we can see the synchronization works as expected. The 27th frame is duplicated on purpose and all the following frames of that block become suspicious. And after receiving the 40th frame, the receiver refuses to generate next watermark information because it hasn't received the synchronization frames. But after the synchronization frames, the watermark information in next block becomes synchronized.

5.7.2 Deletion Attack

This experiment is similar to the previous one. But instead of insertion, we delete some frames from the video stream from transmitter. As mentioned in section 3.6, we assume the attacker cannot precisely delete all the synchronization frames of a block.

From the logs shown in 5.5, we can see that three frames are deleted after the 27th frame and all following frames become suspicious. But after the synchronization frames, the next block becomes synchronized.

5.8 Speed with Separate WM-generating Thread

Theoretically, if there are enough CPU cores, this feature should increase the speed of both the transmitter and the receiver. But somehow, in our experiment, the result shows that there will be some unexpected delay which may occurs randomly between frames. And this random delay makes the video sometimes fast and sometimes slow. This is unacceptable.

We believe this problem may be caused by threading competition. But more evidence may be needed to confirm it.

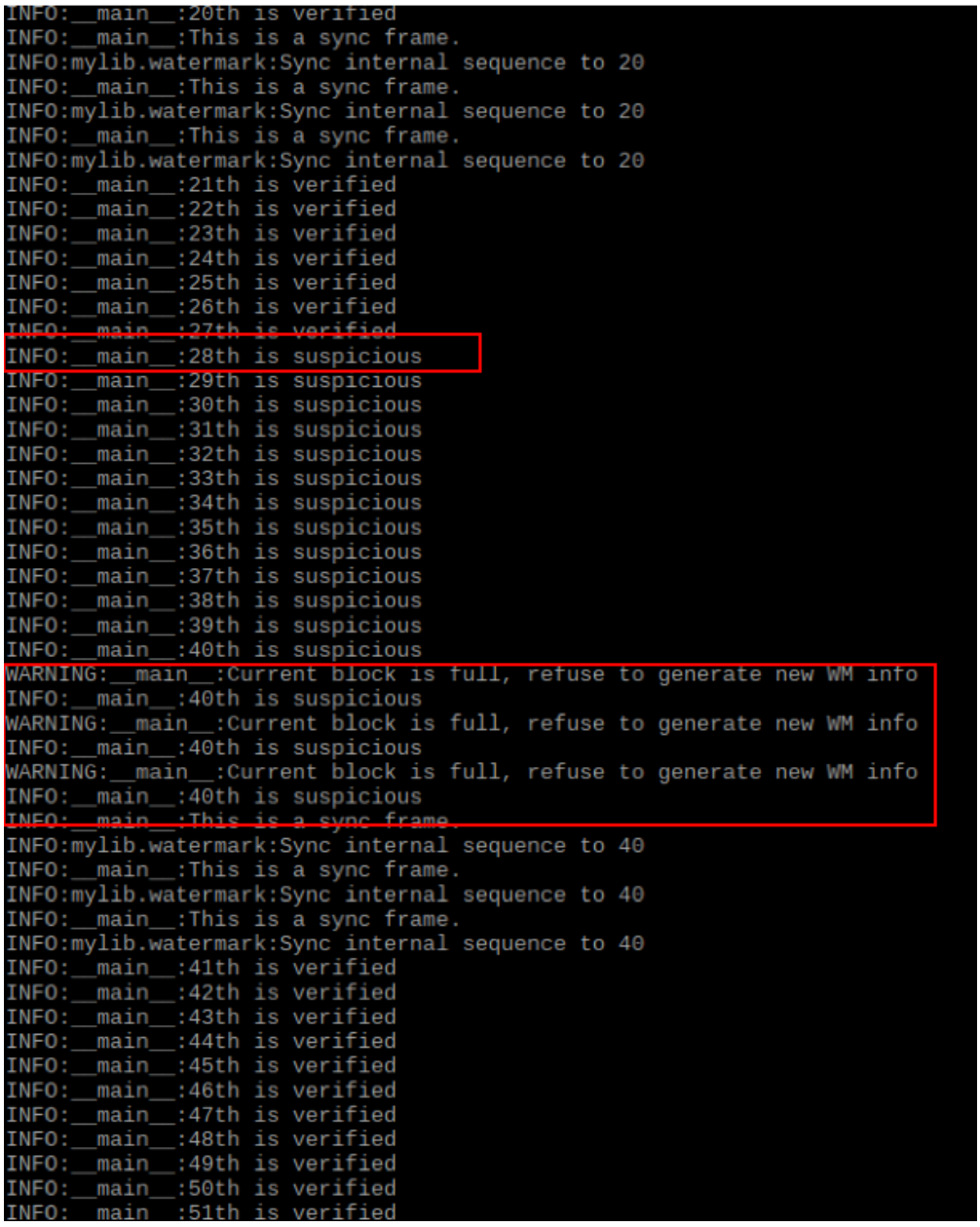


Figure 5.4: Program output from receiver under insertion attack

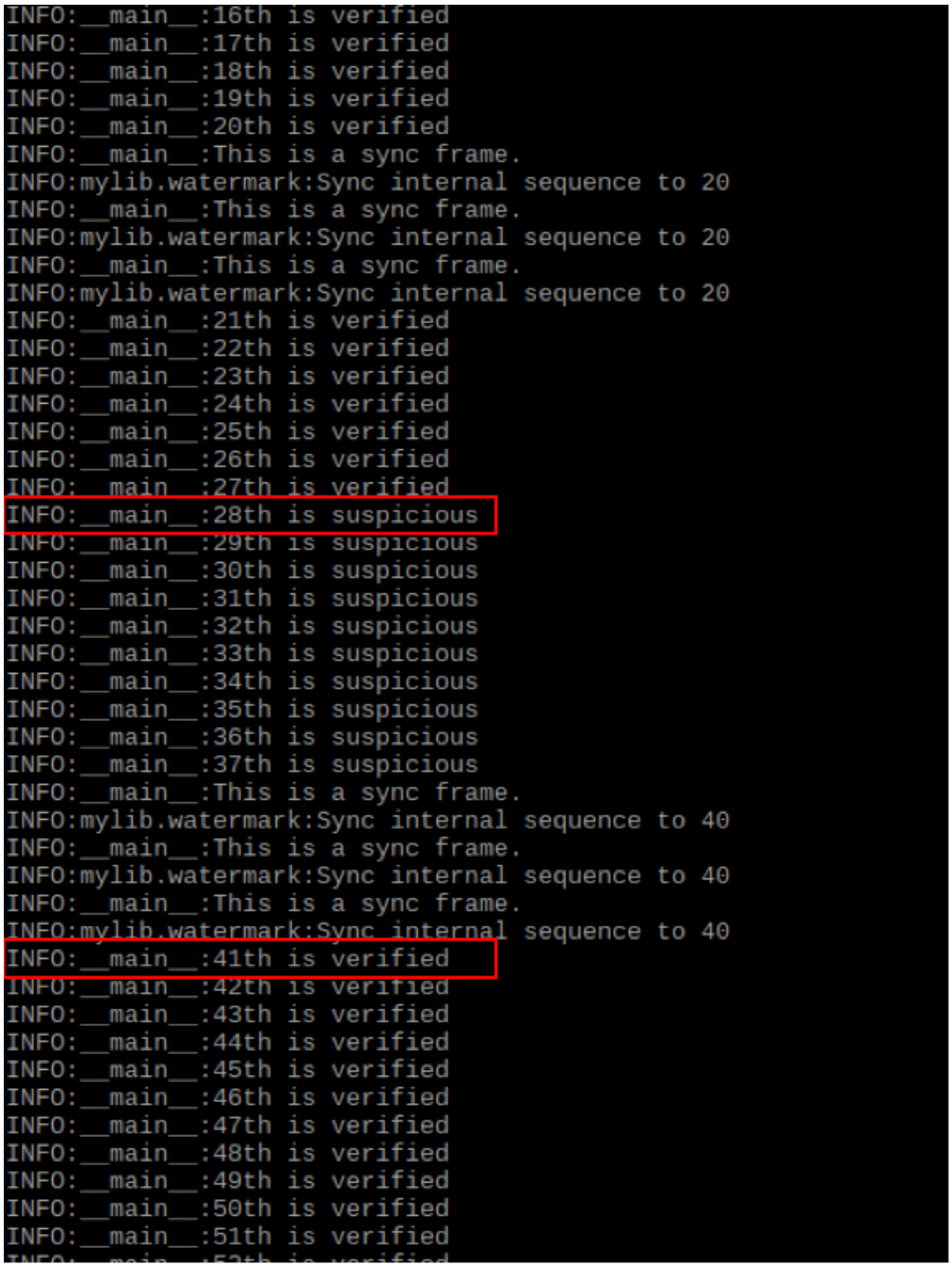


Figure 5.5: Program output from receiver under deletion attack

CHAPTER VI: FURTHER THOUGHTS

6.1 Further Thoughts

Our approach is not perfect. There are still some further works to do.

We tried to implement the project in C/C++ or delegate the time-consuming task from Python to C/C++ module (*Python contributors*, 2019). But we meet some problems such as generating the pseudo-random numbers. As mentioned before, our algorithm needs to generate a lot of random numbers for each frame. This can be done by a single Numpy function call. But we are searching for its C++ counterpart.

We also thought about implementing the project on FPGA. FPGA can be much more powerful than Raspberry Pi. But the implementation seems even more challenging.

The Numpy uses Mersenne Twister pseudo-random number generator (*Matsumoto and Nishimura*, 1998) which is fast but not cryptographically secure. Currently we didn't find its replacement in Numpy. Although this should not be a problem for our algorithm because the adversary should not be able to perceive the generated random numbers at all, further investigation on this is needed.

Besides, we assume that the attacker cannot precisely insert frames between the synchronization frames nor delete all the synchronization frames. But if he randomly chooses the insertion and deletion location, it is possible that he might eventually be able to do that. This case can not be handled by current synchronization feature. We believe we could add some block sequence information into the synchronization watermark information. But how to precisely preserve and detect those information after the lossy compression and transmission error is still under research.

And there is a potential attack in which the attacker only changes a very smaller portion of the frame, e.g. adding a very small stop sign. The correlation should decrease but may not decrease so much that it becomes suspicious. Depending on how small the modification is, the current solution

may or may not detect it. In other words, the integrity may not be guaranteed by current solution.

CHAPTER VII: CONCLUSION

Sensors are used everywhere today. The security of the sensor data is critical for the decision-making systems to achieve their functionality. Therefore, the sensor data must be authenticated before it is used. In this paper, we present our approach to authenticate data by mainly using watermark techniques. We use camera sensors as the example but this approach may also suitable for other sensor data. Our algorithm is spatial, invisible and blind-detected. And it is semi-fragile to some compression. We designed, implemented and evaluated our approach. The experiment result shows that the algorithm almost has 100% precision and recall. We also added some feature to the original algorithm to make it faster and robust under certain temporal attack. Our implementation is relatively generic and modularized. Any further ideas shall be applied to it easily. So we think our work is beneficial for both the current industry and further research.

BIBLIOGRAPHY

Asikuzzaman, M., and M. R. Pickering (2018), An overview of digital video watermarking, *IEEE Transactions on Circuits and Systems for Video Technology*, *28*(9), 2131–2153.

Avatefipour, O., A. Hafeez, M. Tayyab, and H. Malik (2017), Linking received packet to the transmitter through physical-fingerprinting of controller area network, in *2017 IEEE Workshop on Information Forensics and Security (WIFS)*, pp. 1–6, IEEE.

Hafeez, A., H. Arshad, A. Kamran, R. Malhi, M. A. Shah, M. Ali, and S. Malik (2014), Object recognition through kinect using harris transform, *European Scientific Journal, ESJ*, *10*(10).

Hafeez, A., H. Malik, O. Avatefipour, P. R. Rongali, and S. Zehra (2017), Comparative study of can-bus and flexray protocols for in-vehicle communication, *Tech. rep.*, SAE Technical Paper.

Hafeez, A., M. Tayyab, C. Zolo, and S. Awad (2018), Finger printing of engine control units by using frequency response for secure in-vehicle communication, in *2018 14th International Computer Engineering Conference (ICENCO)*, pp. 79–83, IEEE.

Hu, Y.-P., and D.-Z. Han (2005), Using two semi-fragile watermark for image authentication, in *2005 International Conference on Machine Learning and Cybernetics*, vol. 9, pp. 5484–5489, IEEE.

Juma, H., I. Kamel, and L. Kaya (2008), Watermarking sensor data for protecting the integrity, in *2008 International Conference on Innovations in Information Technology*, pp. 598–602, IEEE.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012), Imagenet classification with deep convolutional neural networks, in *Advances in neural information processing systems*, pp. 1097–1105.

Kuhn, D. R., V. C. Hu, W. T. Polk, and S.-J. Chang (2001), Introduction to public key technology and the federal pki infrastructure, *Tech. rep.*, National Inst of Standards and Technology Gaithersburg MD.

Lu, M., K. Wevers, and R. Van Der Heijden (2005), Technical feasibility of advanced driver assistance systems (adas) for road traffic safety, *Transportation Planning and Technology*, *28*(3), 167–187.

Malik, H., R. Ansari, and A. Khokhar (2008), Robust audio watermarking using frequency-selective spread spectrum, *IET Information Security*, *2*(4), 129–150.

Matsumoto, M., and T. Nishimura (1998), Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, *8*(1), 3–30.

Pathan, A.-S. K., H.-W. Lee, and C. S. Hong (2006), Security in wireless sensor networks: issues and challenges, in *2006 8th International Conference Advanced Communication Technology*, vol. 2, pp. 6–pp, IEEE.

Pontil, M., and A. Verri (1998), Support vector machines for 3d object recognition, *IEEE transactions on pattern analysis and machine intelligence*, *20*(6), 637–646.

Python contributors (2019), Extending python with c or c++, [Online; accessed 24-February-2019].

Redmon, J. (2013–2016), Darknet: Open source neural networks in c, `http://pjreddie.com/darknet/`.

Redmon, J., and A. Farhadi (2018), Yolov3: An incremental improvement, *arXiv preprint arXiv:1804.02767*.

Tayyab, M., A. Hafeez, and H. Malik (2018), Spoofing attack on clock based intrusion detection system in controller area networks.

Wikipedia contributors (2019a), Motor vehicle fatality rate in u.s. by year — Wikipedia, the free encyclopedia, [Online; accessed 15-February-2019].

Wikipedia contributors (2019b), Numpy — Wikipedia, the free encyclopedia, [Online; accessed 26-February-2019].

Wikipedia contributors (2019c), Opencv — Wikipedia, the free encyclopedia, `https://en.wikipedia.org/w/index.php?title=OpenCV&oldid=879892622`, [Online; accessed 26-February-2019].

wini J, Y. (2015), Key distribution for symmetric key cryptography: A review, *International Journal of Innovative Research in Computer and Communication Engineering*, *03*, 4327–4331, doi:10.15680/ijircce.2015.0305047.

Zeiler, M. D., and R. Fergus (2014), Visualizing and understanding convolutional networks, in *European conference on computer vision*, pp. 818–833, Springer.