# Securing the video feed of IP surveillance cameras

***A B.Tech Project Report Submitted***
***in Partial Fulfillment of the necessities***
***for the Degree of***

**Bachelor of Technology**

*by*

**Rishi Pathak**
(170101054)

*beneath the steering of*

**Prof. J.K. Deka**

**to the**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI**
**GUWAHATI - 781039, ASSAM**

# <u>CERTIFICATE</u>

*This is to certify that the work contained during this thesis entitled "**Securing the video feed of IP surveillance cameras**" is a bonafide work of **Rishi Pathak (Roll No. 170101054)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati beneath my direction and that it has not been submitted elsewhere for a degree.*

Supervisor: **Prof. J.K. Deka**

Professor and Head,

April, 2021 Department of Computer Science & Engineering,

Guwahati. Indian Institute of Technology Guwahati, Assam.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The age of the Internet of Things(IoT) is upon us. All the items around us are slowly getting smart attached to their name. This is because they now can communicate some information over some internet/network. The network they are connected to can be wired or wireless. This is the next step of machines infiltrating the lives of humans with connected devices and related services.

The Internet of Things domain has garnered much attention over the last few years, mainly due to its applicability across many domains. Its applications are only limited by technology and human imagination. The technology is still in its growing years, but it promises us an era of convenience and increased productivity in our daily lives and different industries.

This architecture allows devices to sense and control objects remotely through wired and wireless network infrastructures. This ends up creating opportunities that enable interactions between the physical world and intelligent machines. One of the applications of this new type of architecture is surveillance. The surveillance systems provide multiple sensors which send information back to a centralized processing station and displays. It is mainly used for monitoring assets and humans for inappropriate behavior.

## 1.1 The main problem

IoT devices' growing market brings new conveniences for customers, presenting new challenges for protecting privacy within the house. Several smart home devices like IP surveillance cameras have always-on sensors which capture users' offline activities in their homes and transmit information regarding these on the net.

There was a massive DDoS attack against Dyn, which is a DNS provider, in October 2016. This attack broke a reasonably large part of the Internet, causing hundreds of websites and services to go down.[CCT17] Dyn did not make the actual size of the attack public, but the researchers assume that the DDoS attack may be larger than the one that hit OVH, a French web service and hosting provider, which peaked at 1.1 TBps, the largest DDoS attack known till this date.[Smi13]

Another study studied Motorola's Focus 73 [Min15] outdoor security camera, and it found that the pictures and videos that are taken by the camera can be redirected to a mobile application. Another attack showed that it is possible to scan for cameras connected to the Internet and reverse the root shell to determine the device's control credentials. In addition to this, by interfering with the device's DNS settings, the attacker can cut off the alerts that the camera sends to the customer and see the video clips sent to a cloud storage service. This is an extensive security concern and a breach of users' privacy without their knowledge of the system compromise.

## 1.2 Challenges with digital video surveillance systems

Usually, IP surveillance systems are made up of three modules: video capture units, network transmission, and central control modules.[ZDYL05] Video capture units are the camera part of the system. They can be analog or digital. They are paired with a video encoder device that converts the analog input to digital output if they are analog. This module captures raw videos, compresses the data, encodes them into popular formats like MPEG and H261. The network transition module sends this encoded video over LAN or the Internet. The central control module is the end module of the system where we can view or record the video stream. It also can be used to control the camera by sending out commands.

The data we are mainly concerned with here is the video stream. The source of digital video data are cameras and video servers. The receiver of digital video can be a personal computer(PC) or a digital video decoder(DVD). The main features of digital video data are -

- Large amount of data - The size of video streams can be huge and may require up to 4Mbps network speed. Video surveillance systems require real-time monitoring of the watched area. Therefore there is a big requirement for network output and process capability.
- Time-sensitive - Video data is real-time and hence really time-sensitive. Capturing the content after the event has happened is not of much significance for the attacks.

Hence the main challenge we face here is the significant security threats to the internet protocol-based video surveillance systems. Attackers can capture the video frames merely by listening on the network transmission channel. Security cannot be overlooked in Internet-based video surveillance systems. This paper analyzes IP-based video surveillance systems' security concerns over public networks and tries to design a more secure system than previous works. The new system we propose here is more secure, cost-efficient, and flexible. We inspect our design's performance and security and try to enhance the security of IP-based video surveillance systems directly.

## 1.3 Motivation and Objective of the project

After the DDoS attack on Dyn in October 2016[CCT17], many researchers actively started evaluating IP surveillance cameras' vulnerabilities. One such paper was written in 2017 by Cusack and Tian, in which they used different tools such as Angry IP Scanner, WireShark, ophcarack, Burpsuite, Cain & Abel, etc. to find the vulnerabilities of the IP surveillance cameras. The paper found that although code injection and techniques for exploiting using directory traversal were not allowed., there were many other points of vulnerability they found.[CT17] This paper motivated me to take this topic and work towards making a cost-effective algorithm for fixing the security concerns about privacy in IP surveillance cameras.

The objectives of the thesis are:

- We try to learn about the IP-based surveillance camera systems' security and privacy concerns.
- We explore the work done in this research area and try to find the drawbacks of their proposed method.
- We also propose a new secure, flexible and cost-effective algorithm to solve the drawbacks of existing works
- We implement the proposed method and detail all the findings from its thorough

testing

## 1.4 Organization of Report

The remainder of this paper is organized as follows. Chapter 2 discusses the previous work that has been done in the area and its drawbacks. Chapter 3 proposes a cost-effective system approach and its implementation in Java. Chapter 4 details the configuration of the test bench on which implementation was developed and tested. It also presents the analysis of all the findings from thorough testing of the algorithm. Chapter 5 concludes the thesis with future outlooks of the proposed algorithm.

# Chapter 2

# Literature Review

Security has been a hot topic in the research community for the past couple of decades. Significantly since the arrival of IoT, the security concerns has increased manifolds. As we know that, IoT devices are usually tiny and have much less space in them. So most of the companies creating such devices are not much concerned with the devices' security aspect. Typical protocols do not tend to work in IoT due to a lack of needed space for fitting large computing devices.

People always had their concerns regarding privacy since IP Surveillance cameras came into existence. They are always concerned about their privacy, thinking someone might hack into these cameras and look into their private lives. This concern dates back to the 1990s, when such camera systems were first made. After the October 2016 DDoS attack on Dyn, a DNS provider[CCT17], in which most of the internet services and websites broke down using a botnet of such cameras, the disquiet about their security has been an all-time high.

With this said, there has been much research going on in video streaming to prevent such attacks and privacy breaches. Current research provides many ways to secure the data stream. Some implementations limit themselves to specific video formats like MPEG, while other implementations are for generic video streams or provide some variations in the algorithms. Some of them include -

## 2.1 The basic method

The basic method usually uses the basic security operations to secure the data. It treats the stream data as the regular IP data and does not understand its encoding and compression. As we already saw that the video streaming data can be huge and may require internet connections up to 4Mbps for streaming real-time. So the performance of the basic method is not very good. Special hardware units like the AES encryption hardware unit are usually used for speeding up the performance. However, these hardware units are usually not that flexible and do not allow reconfiguration to provide specific security needs. The basic method's main disadvantage is that they are not very cost-efficient and are usually very slow.[ZDYL05]

## 2.2 Selective encryption

Unlike the basic method, selective encryption does not treat the stream data as the regular IP data. It is based on understanding the video data's encoding and compression and uses this knowledge to take advantage of the video stream's particular characteristics. For example - The I frames carry more information in the MPEG format than the B and P frames. So selective encryption of the I frames can be more efficient and provide us with various security degrees.

Usually, the implementation of security operations goes hand to hand with the implementation of the data's encoding and compression. So in selective encryption, a relatively high-performance processor is required, which is more costly than having two low-performance processors. Another issue that selective encryption faces are that of insecurity. Most algorithms have not gone through formal and strict cryptanalysis and can be broken by simple attacks. Also, it is not very flexible as this encryption is based on video characteristics and cannot be changed by the consumer.

There are various implementations of selective encryption. Let us discuss some of the implementations and issues with them.

### 2.2.1 Header Encryption

Header encryption is one of the most basic selective encryption and was the first of the four encryption levels proposed by Meyer and Gadegast in 1995[mey95]. In header encryption, it simply encrypts all the headers of data streams. It was rejected because the reconstruction of headers in MPEG is relatively simple. The video encoders produce CBR streams, and they use the same header in all.

### 2.2.2 Prediction based encryption

In the MPEG videos, B and P frames are predicted from I frames. Therefore Spanos and Maples[SM95] proposed that encrypting the I pictures would be sufficient. However, this meant that they need to encrypt 25-50% of the stream or. 1-3 frames per second. This method failed because the encryption was not strong, and researchers were able to discern some parts of the video with the help of decoded P and B frames.

### 2.2.3 DCT Coefficient encryption

DCT techniques allow the frames in MPEG to be represented in frequency than the time domain. Frames can be represented in less information in frequency compared to time. Different researchers proposed encryption of different DCT coefficients like Kunlemann and Reinema[KR97] proposed encryption of low order coefficients. There was another proposal by Cheng and Li[CL97] in which they proposed encryption of higher order DCT coefficients. This method had issues like it required deep parsing into compressed bits stream at both encoder and decoder side.

### 2.2.4 Zig Zag scan permutation

One paper[Tan97] proposes that the order in which DCT coefficients are scanned from 8×8 matrix to a 1×64 linear vector can be permuted before encoding it. Many researchers rejected this proposal by making cryptanalytic attacks on this method. Some also proved that non-optimal scanning of coefficients causes an efficiency overhead of 40%.

### 2.2.5 Half encryption/ One time pad

In this type of encryption, half of the bytes are encrypted using both a standard encryption algorithm like DES, and then these bytes are also XORed with the other half of bytes. This was a pretty good algorithm due to low correlation between bytes of MPEG streams and was pretty secure. The only drawback one could say about this algorithm is that it does not reduces the complexity dramatically.[QN97][QN98]

### 2.2.6 Random corruption algorithm

Griwodz[Gri98] proposed random corruption of video stream approximately 1% can complete our goal leading the video stream unwatchable or undecidable. The original bits are sent to the receiver only when content decoding is enabled. This method was critiqued by researchers for leaving the content format incompatible. The algorithm's main problem was its possibility of imitating distinctive bit patterns that can be used to intermediate the video bitstream.[LSK+03]

## 2.3 Conclusion

This chapter provided details of some of the existing algorithms for encrypting the video stream of IP surveillance cameras. In the next chapter, we discuss a new algorithm for encrypting the video feed. We also discuss the implementation of the proposed algorithm.

# Chapter 3

# Proposed Work

In the previous two chapters, we have looked over the world of IoT, and specifically, how does IP Surveillance cameras, one application of IoT, works. We have also seen the security concerns in these cameras. Moreover, we have seen how the researchers have been working to solve these concerns for the past two decades.

## 3.1 Proposed Algorithm

As we discussed in chapter 2, there were different approaches to solve the security concern. These can be categorized into two categories: The basic method and the selective encryption.[ZDYL05] We saw the limitations of both the methods there. They suffer from security and performance problems. We propose a more flexible and randomly adjustable algorithm to solve the privacy breach problem to avoid this problem.

We will treat the data as regular IP data like in the basic method. We will then selectively add encryption to the video stream data. i.e., Without understanding the data's specific characteristics, we will encrypt some part of the data and leave the other part. For flexibility, users can control the degree of security by setting the percentage of encrypted video.

So the algorithm goes here. There will be two ends of the whole algorithm -

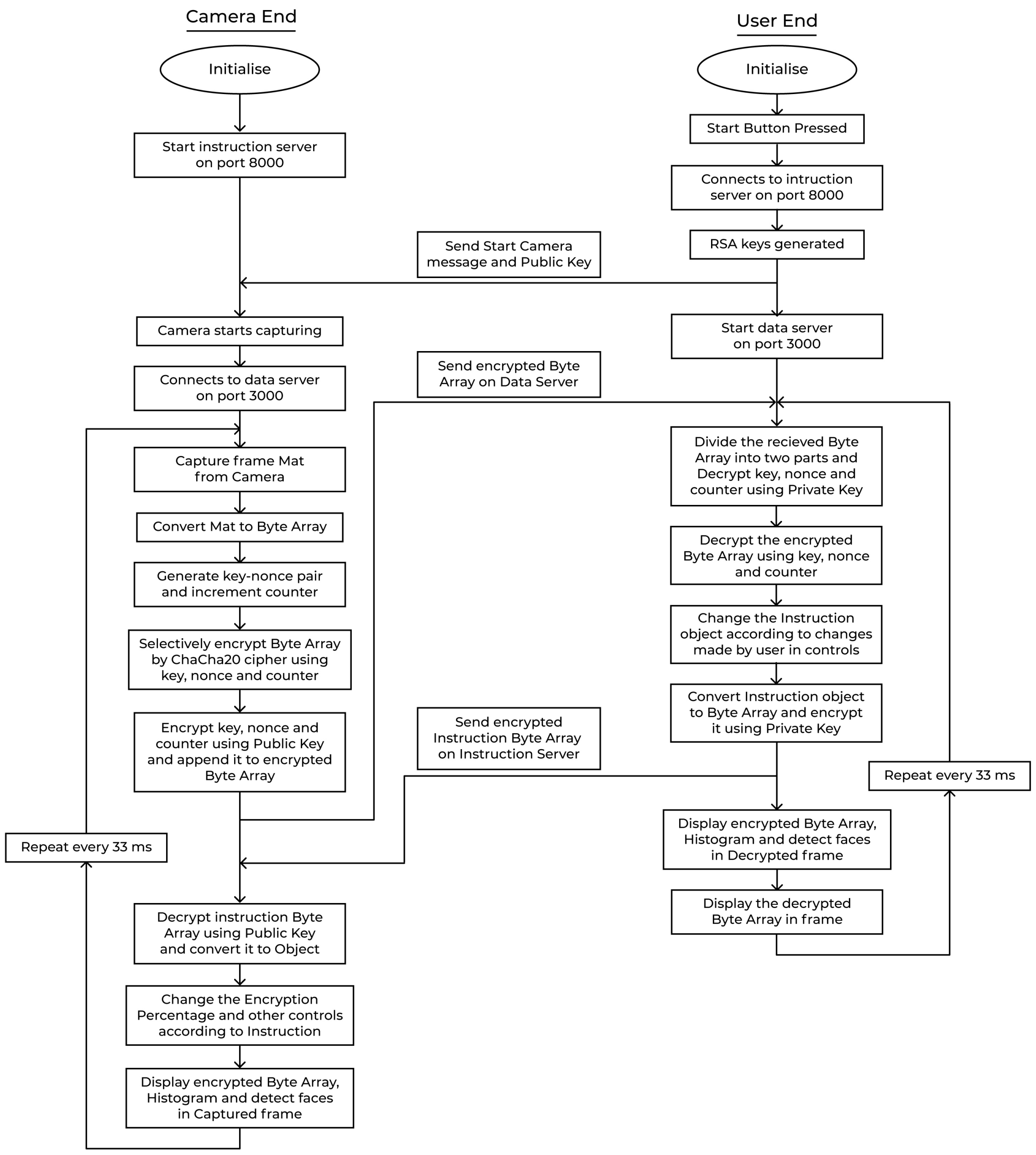
- Camera end - This end gets instruction from the User end, captures video 30 frames per second from the camera, encrypts data, and transmits it to the User end
- User end - This sends all the instructions and receives and decrypts data from the Camera end and displays it.

The two ends will be connected through two TCP connections. One will be used for sending data and the other for sending instructions. We will use a unique design cryptographically secure pseudorandom number(CSPRNG) known as ChaCha20 at the camera end for encrypting the data[Cha]. Google has selected ChaCha20-Poly1305 message authentication code as a replacement for RC4 in Transport Layer Security, [Wik21] which provides enough credibility for the CSPRNG. We will use a stream cipher for combining the sequence generated by CSPRNG with our stream data at some controllable length of bits based on the user's degree of security. They will be combined using the XOR (exclusive OR) operation bit by bit. ChaCha20 uses a 256-bit key, 32-bit initial counter, and 96-bit nonce to encrypt the stream data. We will use the RSA cryptosystem for encrypting instructions and ChaCha20 requirements. RSA cryptosystem makes use of two keys, a public key used for encryption and a private key for decryption or vice versa. In our algorithm, both keys are generated at the user end. The public key is sent to the camera end, where it is used to encrypt the ChaCha20 key, nonce, and counter, which are later appended to the encrypted frame.

Decryption at the user end is done in the same method. First, we decrypt the key, nonce, and counter through the private key. The ChaCha20 block function then uses these key, nonce, and counter to expand the key to keystream, which is then XORed with the encrypted frame data which CSPRNG generated to get back the original data. So the data can be converted from no secrecy to encrypted form with a bit of performance overhead. In the whole process, the private key remains at the user end and is never transmitted. Therefore, this algorithm with high-speed and secure stream cipher and key encryption cryptosystem will provide enough flexibility, sufficient security, and real-time encryption to the users.

The problem we are solving consists of video streaming. Although we are using TCP, certain packets can still be lost on the way to the user. So we should also find a way to synchronize the data sender and receiver with each other. Here, ChaCha20 successively calls the block function, with the same key and nonce, and increasing counter parameters. Every keystream block is XORed with a data block of an equivalent length before going forward to make the successive block, saving some memory. There is no demand for the data to be an integral multiple of 512 bits. If there is a further keystream after the last block, it is removed. [Cha] So when the packet is lost in the transmission, the receiver detects it using the packet sequence number and then makes for this loss by generating a keystream block of 512 bits. So the sender and receiver will remain synchronized.

Another important aspect of surveillance systems is key management. We suggest that we change the RSA keypair every fixed amount of time, say 5 minutes. The public key that we change after every interval can be sent to the camera end, and the private key remains at the user end. The central control unit can set this interval according to the user's degree of security and the overhead generated in encryption.



Camera End
User End
Initialise
Initialise
Start instruction server on port 8000
Start Button Pressed
Connects to intruction server on port 8000
RSA keys generated
Send Start Camera message and Public Key
Camera starts capturing
Start data server on port 3000
Connects to data server on port 3000
Send encrypted Byte Array on Data Server
Capture frame Mat from Camera
Divide the recieved Byte Array into two parts and Decrypt key, nonce and counter using Private Key
Convert Mat to Byte Array
Generate key-nonce pair and increment counter
Decrypt the encrypted Byte Array using key, nonce and counter
Selectively encrypt Byte Array by ChaCha20 cipher using key, nonce and counter
Change the Instruction object according to changes made by user in controls
Encrypt key, nonce and counter using Public Key and append it to encrypted Byte Array
Convert Instruction object to Byte Array and encrypt it using Private Key
Send encrypted Instruction Byte Array on Instruction Server
Repeat every 33 ms
Display encrypted Byte Array, Histogram and detect faces in Decrypted frame
Repeat every 33 ms
Display the decrypted Byte Array in frame
Decrypt instruction Byte Array using Public Key and convert it to Object
Change the Encryption Percentage and other controls according to Instruction
Display encrypted Byte Array, Histogram and detect faces in Captured frame

**Fig. 3.1** Flow Chart for Proposed Algorithm

## 3.2 Solution

We implemented the above-proposed algorithm for testing its performance and feasibility. The algorithm was coded in Java in Eclipse IDE for Java developers. The project uses a JavaFX platform, an open-source, next-generation client application platform for developing the software interface. It has support for desktop computers and web browsers on Windows, macOS, and Linux. It also uses OpenCV for capturing frames from the camera and other image processing functions proposed in the algorithm.

Like in the proposed algorithm for IP surveillance cameras' security concern, we developed two parts of the software: the camera end and the user end. Both the interfaces look nearly identical but differ significantly in the internal working. In the real world, both the software ends will be on different computers. The camera end will be on a chip/small-sized computer with no screen to display the interface. However, for development and testing purposes, we have made an interface for the camera end too. It displays the frame captured by the camera, the encrypted frame, the histogram of the captured frame, and the user's controls. There would be no screen for the camera end in the real world, so these controls or buttons do not need to be touched for changing the transmitted bytes. The camera interface can be completely controlled from the user end. Nevertheless, if needed, controls can be changed from the camera end interface too.

The user end interface is also quite similar to the camera end interface with a couple of extra buttons. There are two ImageViews: one displaying the encrypted frame received from the camera end and the other ImageView for displaying decrypted final frame. There is a histogram of the decrypted frame. The controls include showing the grayscale frame, showing a custom logo as a watermark, and decrypting the encrypted video frame. Other controls include a choice of pre-trained face detection classifier and encryption percentage of the video stream. In the bottom part of the interface, there are three buttons. The

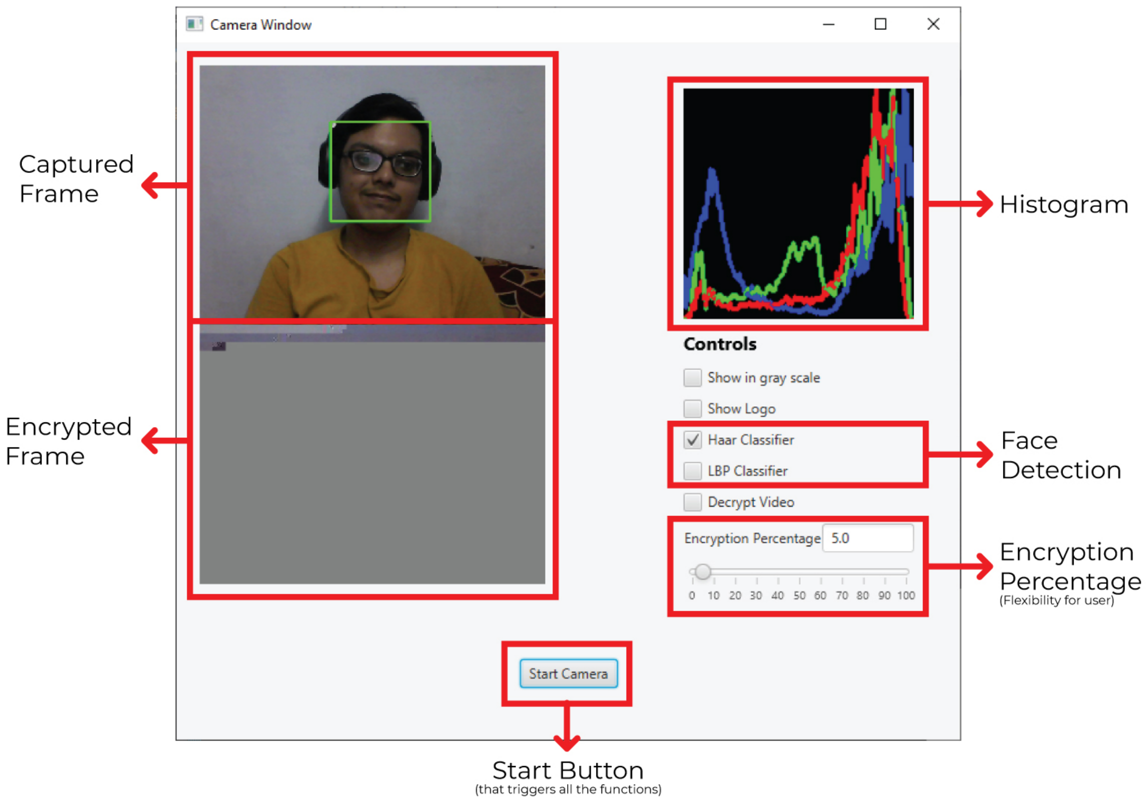


**Fig. 3.2** Camera end Interface

Start Camera button is the main button which triggers all the function and starts the video stream. After the user clicks the Start Camera button, it turns to the Stop Camera button, which stops the video stream when clicked. There is a Snap button on the Start Camera button's left, with a floppy disk as its icon. It saves the current decrypted frame as an image on disk. Then there is a third record button on the Start Camera button's right, with a red disk as its icon. It records the decrypted frame as a video on the disk till it is pressed again.

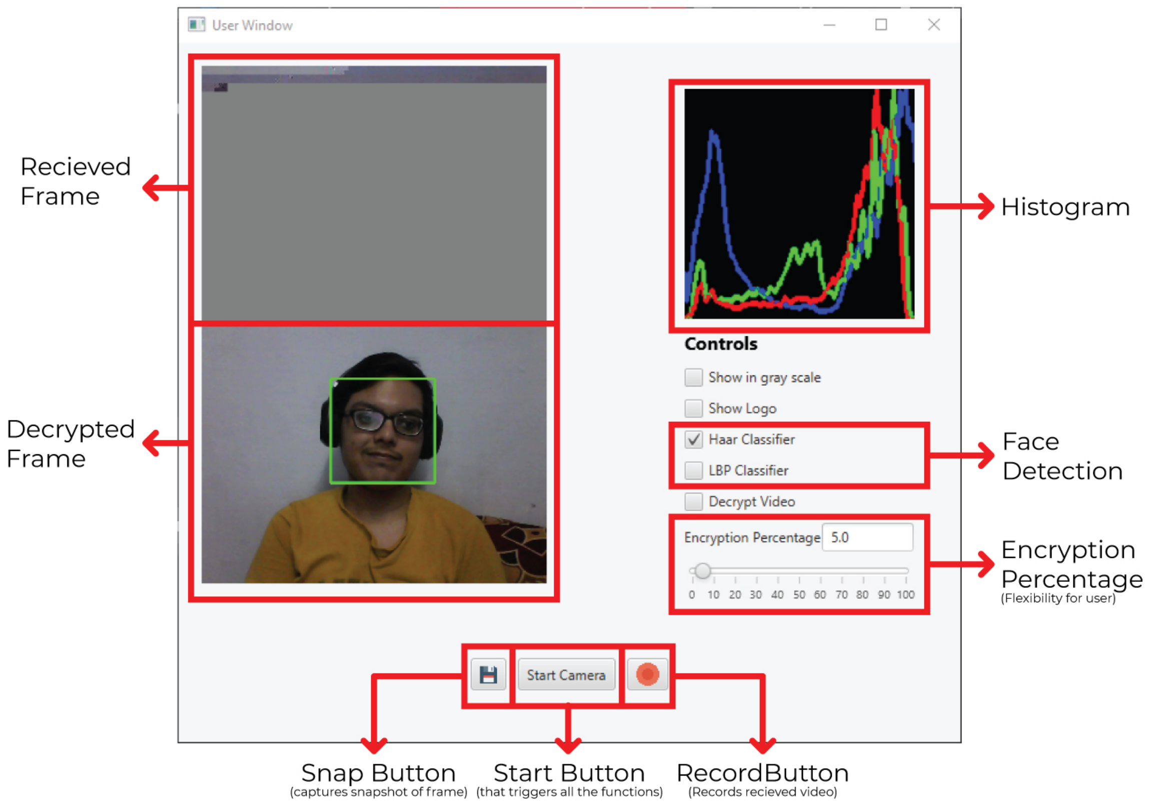


**Fig. 3.3** User end Interface

There were two encryption algorithms used in the solution, namely RSA and ChaCha20. A 2048-bit key was used for the RSA cryptosystem, which was used to encrypt the instructions. For the ChaCha20 cipher, a 256-bit key, 96-bit nonce, and a 32-bit counter were used to encrypt the data stream. The encryption percentage of the data stream could be set anywhere between 0% and 100% by the user. The RSA key pair gets updated periodically, and the frequency is set according to the encryption percentage.

| **RSA key** | **ChaCha20 key** | **ChaCha20 nonce** | **ChaCha20 counter** |
|---|---|---|---|
| 2048 bits | 256 bits | 96 bits | 32 bits |

**Table 3.1** Security configurations

## 3.3 Conclusion

This section discussed the problem's width and how we were motivated to take the problem as the topic for the BTP. We also proposed a cost-effective and secure algorithm for the problem, which gives users more flexibility. We also implemented the algorithm using Java for testing purposes.

# Chapter 4

# Experiment Results

An efficient and more secure algorithm was proposed in the previous chapter. We also showed how we implemented the proposed algorithm in Java. In this chapter, we will talk about the platform we have used to implement the algorithm. We will also try to explain in detail all the findings from the thorough testing of the solution.

## 4.1 Test bench

We have implemented our proposed solution to IP surveillance systems' security concerns and analyzed the algorithm's performance. For this, we have used a laptop with a web-cam/USB camera as a test bench for the experiment. We have used the same laptop as the surveillance system's control center to receive the data and view the video stream. The system's transmission module is mobile hotspot WiFi with TCP for the video streaming and sending the instructions. The test bench's hardware configuration for the experiment is Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.70 GHz with 8.00GB of installed RAM and Windows 10 as the operating system.

For the algorithm, we use a public encryption key of size 2048 bits for the RSA cryptosystem. Our main encryption stream cipher ChaCha20 will use a 256-bit key and a 96-bit nonce. The percentage of encryption is decided by the user and can range from 0 to 100%.

## 4.2 Results

We ran the solution program with the encryption percentage set as 100%. We found virtually no lag between the frame captured by the camera and decrypted frame at the user end. Both frames were running smoothly with no frame drops visible. Even at 100% encryption, the algorithm runs very efficiently, and encryption and decryption complete within milliseconds.

While running the tests, we noticed that after pressing the start camera button at the user end, there is a delay of 2-3 seconds before the frames appear on the screen. The delay maybe because of the long process that takes place before the frames are displayed. This process includes forming a TCP connection for instructions, generating the RSA key pair at the user end, and then sending the public key over the connection. After all this, the camera starts capturing and displays the frame on the screen after the algorithm's first iteration completes.

We did another experiment to find enough value of encryption percentage. The minimum encryption percentage such that video captured from the camera is scrambled enough that its content is not decipherable. We encrypted the video such that the header information of individual frames remains unencrypted. We found that as the encryption percentage values increased from 0% to 4%, the scrambled ness of the video gradually increased, but few video frames were partially decipherable even at 4%. However, as the encryption percentage increases to 5%, the encrypted video frames cannot be deciphered by the average human eye. When the percentage increases to 10%, the encrypted frame becomes too encrypted and cannot be displayed by most decoding software. This percent depends on how selective encryption is implemented and can change drastically based on it.

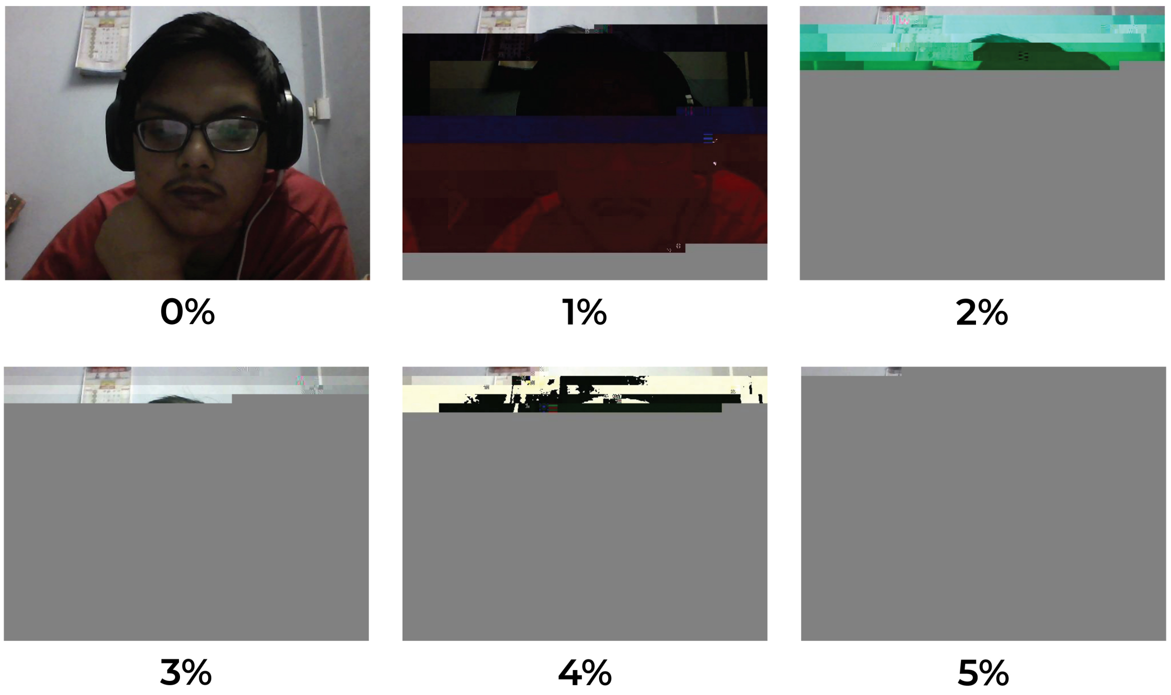


**Fig. 4.1** Encrypted frame wrt increasing encryption percentage

Next, we also measured the time required by the ChaCha20 cipher and RSA cryptosystem in one iteration of the algorithm. For encryption percentage set to 5%, we found that in the camera end, ChaCha20 cipher takes an average of 0.173 ms and a minimum of 0.037 ms for encrypting captured frame bits. RSA takes an average of 0.415 ms and a minimum of 0.188 ms for decrypting instruction at the same end. At the user end, things turn out a little different. ChaCha20 takes only 0.125 ms on average to decrypt the received byte array with a minimum of 0.035 ms. However, the whole algorithm's bottleneck is the RSA algorithm's time to encrypt 130-byte instruction. RSA takes 8.213 ms on average, while the minimum time was 3.583 ms which is enormous compared to the time taken by ChaCha20.

We also tried using different cameras for capturing. The above results were from the test bench machine's webcam. We used Logitech C270 HD USB Camera to test and analyze if any difference comes in the time taken by ChaCha20 and RSA. With encryption percentage set to 5%, the average time of ChaCha20 for encrypting frames was 0.109 ms while the min time was 0.036 ms. In comparison, RSA took a minimum time of 0.186 ms and an average time of 0.399 ms for decrypting the instructions. At the user end, ChaCha20, on average, took 0.118 ms with a min of 0.040 ms to decrypt the received frame. The slowest one was RSA encryption here, too, with an average of 6.97 ms and a minimum of 3.581 ms. We can see from the graph that the time taken by ChaCha20 and RSA is slightly lower for the USB camera. We can also see that decryption takes slightly less time than encryption in ChaCha20, while this difference is enormous in RSA.

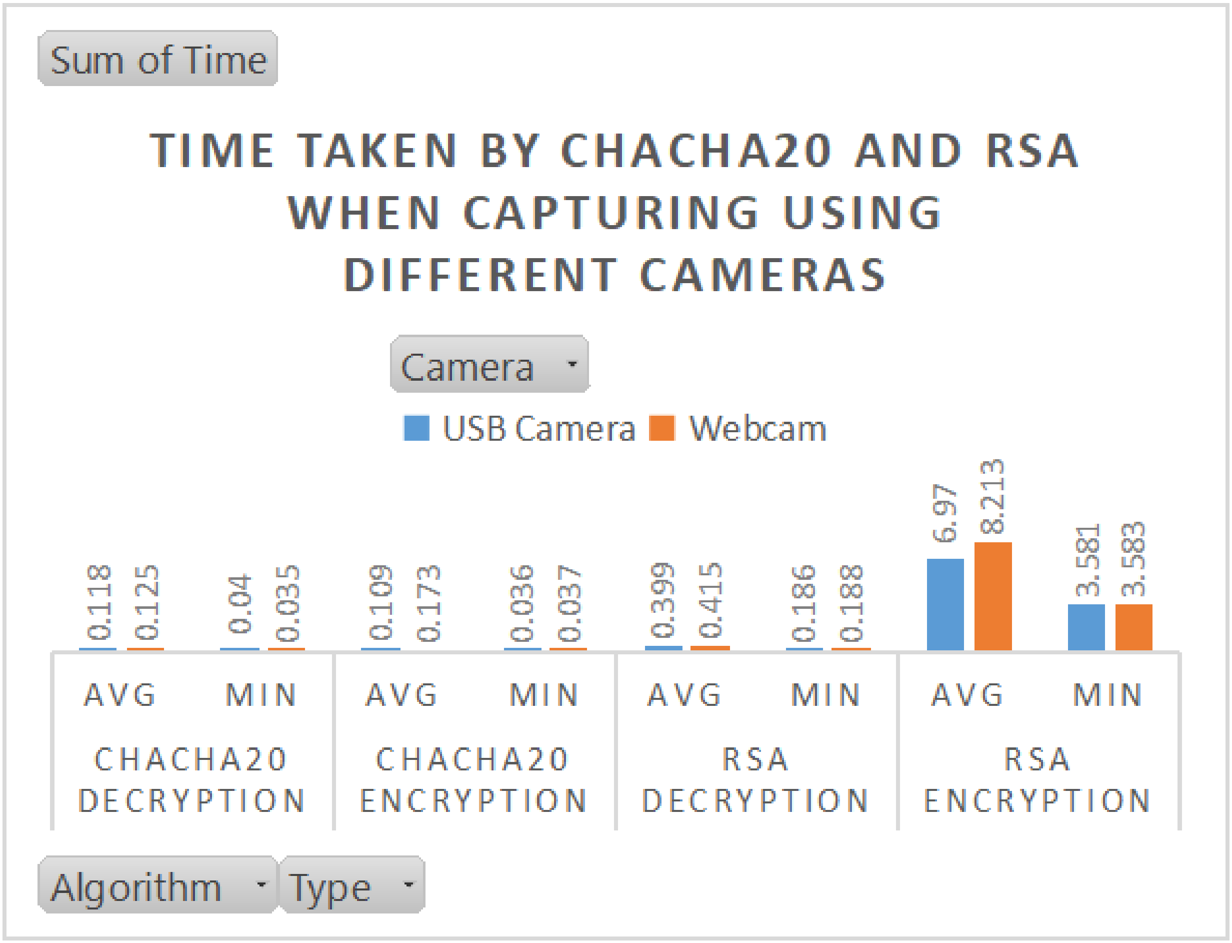


**Fig. 4.2** Time taken by ChaCha20 and RSA when capturing using different cameras

We also ran the tests with different encryption percentage values and found out how it

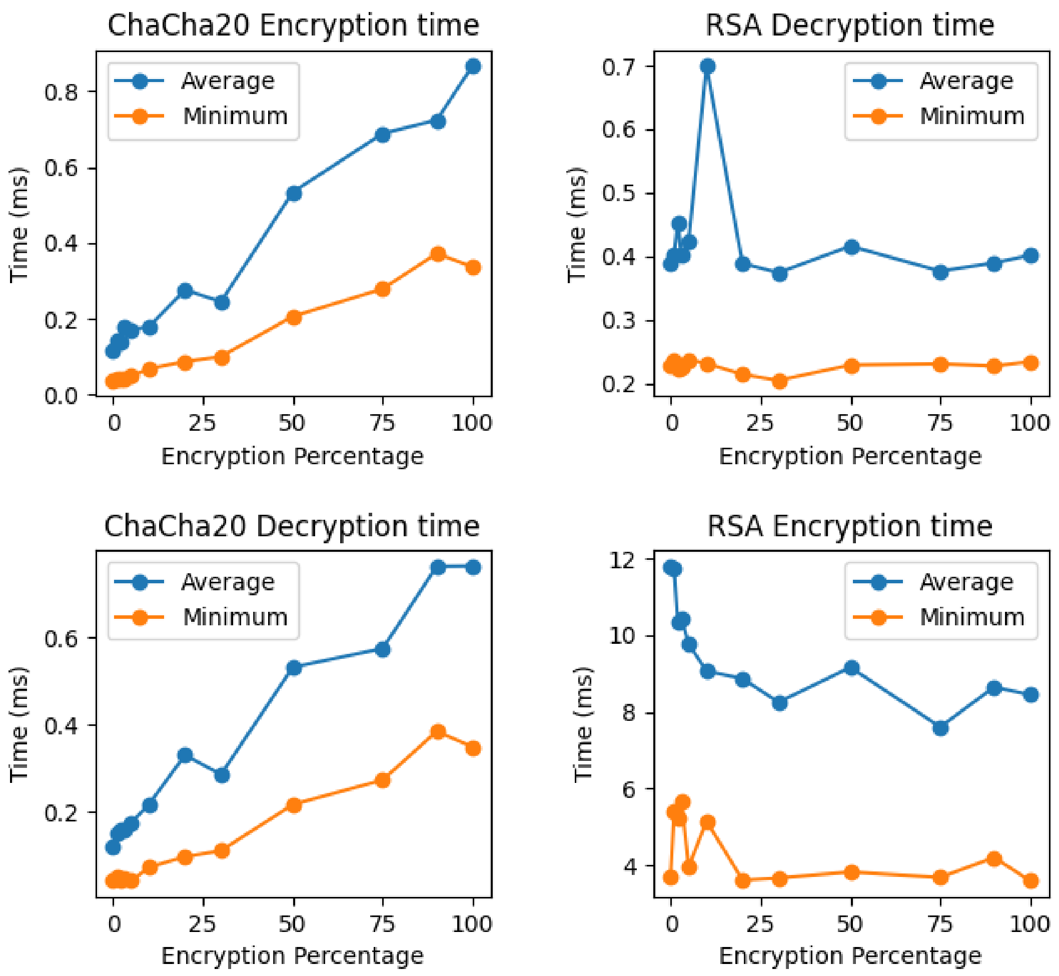


**Fig. 4.3** Trend of time taken by ChaCha20 and RSA when capturing wrt Encryption Percentage

affected the time taken by ChaCha20 and RSA to encrypt and decrypt data and instruction, respectively. For this test, we ran the algorithm for approximately 1500 frames each time after setting the initial encryption percentage value. The values of encryption percentage used for this test were 0, 1, 2, 3, 5, 10, 20, 30, 50, 75, 90 and 100. From figure 4.3, we can study the trends. We can see that with an increase in encryption percentage, the time taken by ChaCha20 increases gradually for both encryption and decryption. It is expected because, with an increase in encryption percentage, the number of bits ChaCha20 has to encrypt also increases proportionally, which gets reflected as the increase in time of encryption and decryption. In contrast, for decryption, except for one time, RSA takes about the same amount of time every time. However, in the case of encryption of instruction at the user end, RSA's time gradually decreases.

## 4.3 Conclusion

In this chapter, we talked about the configuration of the test bench we have used to implement and test the project. We also described the security configuration we used for the implementation of ChaCha20 cipher and RSA cryptosystem. We also talked about the experimental results generated by our solution. Further, we analyzed the results in a detailed manner.

# Chapter 5

# Conclusion and Future Work

IP Surveillance systems with better security and flexibility for the user are becoming important day by day. With attacks like the DDoS attack against Dyn of October 2016 happening globally, the need for cost-efficient and secure systems has never been more. The privacy attacks against the IP-based surveillance camera have reached another level.

This thesis report discussed the importance of security in IP-based video surveillance systems and the increasing threat to them. We also talked about the existing solutions to the privacy and security concerns of these systems. We saw the algorithms in detail and the drawbacks and loopholes in their approaches.

Then we proposed a new algorithm to address the privacy concerns of the IP-based surveillance camera systems. Our method uses ChaCha20 cipher to selectively encrypt the video streaming data, which is cost-effective and very secure. We also use the RSA cryptosystem to manage a pair of keys which is used to encrypt and decrypt the key, nonce, and counter used by ChaCha20 to encrypt the data. RSA keys are also used to encrypt the instructions sent to the camera by the user. This double encryption provides the data confidentiality it requires. The encryption percentage can be set by the user, giving flexibility and total control of the security to the user. We also change the RSA keypair after a fixed interval of time, which is decided by the user's encryption percentage.

We then presented our implementation of the algorithm we proposed. We implemented the algorithm in Java using Eclipse IDE. We used an open-source cross-operating system platform called JavaFX to make the interface of the implementation. We used OpenCV to capture the video from the camera and then apply our algorithm to it. The simulation of our implementation showed that our method balances both cost-efficiency and security requirements. ChaCha20 was able to encrypt and decrypt KBs of video data in fractions of a millisecond, and RSA decryption at the camera end also took comparable time. The videos were not decipherable when the encryption percentage was set to 5%, and the encrypted bytes cannot be displayed when it was increased to 10%. So we can say that there was significantly less computational cost at the camera end.

For this project's future outlooks, we can choose a more efficient and secure asymmetric cryptosystem than RSA. As seen in the results, RSA encryption takes the most time than other parts of the algorithm. We can also investigate the security properties of our algorithm and check if it is compatible with IPSec. Stronger cryptanalysis can be done on our approach to check its security strength. We can also try to implement our approach on an embedded device that can be used practically in the industry.

# References

[CCT17] Vulnerability database. Retrieved from CCTV Calculator. https://www.cctvcalculator.net/en/knowledges/vulnerability-database/, 2017.

[Cha] ChaCha20 and Poly1305 for IETF Protocols, at. https://tools.ietf.org/html/rfc7539.

[CL97] Howard Cheng and Xiaobo Li. On the application of image decomposition to image compression and encryption. In *Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security II*, page 116–127, GBR, 1997. Chapman amp; Hall, Ltd.

[CT17] Brian Cusack and Zhuang Tian. Evaluating ip surveillance camera vulnerabilities. 2017.

[Gri98] Carsten Griwodz. Video protection by partial content corruption. In *Multimedia and Security Workshop at ACM Multimedia*, volume 98. Citeseer, 1998.

[KR97] T. Kunkelmann and R. Reinema. A scalable security architecture for multimedia communicationstandards. pages 660–661, 07 1997.

[LSK$^+$03] Tom Lookabaugh, Douglas Sicker, David Keaton, Wang Guo, and Indrani Vedula. Security analysis of selectively encrypted mpeg-2 streams. *Proceedings of SPIE - The International Society for Optical Engineering*, 5241, 11 2003.

[mey95] Meyer, J. and Gadegast, F., Security Mechanisms for Multimedia-Data with the Example MPEG 1 Video, at. http://www.gadegast.de/frank/doc/secmeng.pdf, 1995.

[Min15] GeoVision (GeoHttpServer) webcams - Remote file disclosure. Retrieved from Exploit Database:. https://www.exploit-db.com/exploits/37258/ , 2015.

[QN97] Lintian Qiao and Klara Nahrstedt. A new algorithm for mpeg video encryption. In *In Proceedings of The First International Conference on Imaging Science, Systems, and Technology (CISST'97*, pages 21–29, 1997.

[QN98] Lintian Qiao and Klara Nahrstedt. Comparison of mpeg encryption algorithms. *Computers and Graphics*, 22(4):437–448, January 1998.

[SM95] G. Spanos and T. Maples. Performance study of a selective encryption scheme for the security of networked, real-time video. In *Computer Communications and Networks, International Conference on*, pages 2–10, Los Alamitos, CA, USA, sep 1995. IEEE Computer Society.

[Smi13] Hacks to turn your wireless IP surveillance cameras against you. Retrieved from CSO Online:. https://www.csoonline.com/article/2224469/hacks-to-turn-your-wireless-ip-surveillance-cameras-against-you.html, 2013.

[Tan97] Lei Tang. Methods for encrypting and decrypting mpeg video data efficiently. In *Proceedings of the Fourth ACM International Conference on Multimedia*, MULTIMEDIA '96, page 219–229, New York, NY, USA, 1997. Association for Computing Machinery.

[Wik21] Wikipedia contributors. Salsa20 — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Salsa20oldid=1016224245, 2021. [Online; accessed 9-April-2021].

[ZDYL05] Zhaoyu Liu, Dichao Peng, Yuliang Zheng, and J. Liu. Communication protection in ip-based video surveillance systems. In *Seventh IEEE International Symposium on Multimedia (ISM'05)*, pages 8 pp.–, 2005.

phase-II-panel-3-170101054

ORIGINALITY REPORT

| 10% | 6% | 7% | % |
|---|---|---|---|
| SIMILARITY INDEX | INTERNET SOURCES | PUBLICATIONS | STUDENT PAPERS |

PRIMARY SOURCES

| | | |
|---|---|---|
| 1 | Zhaoyu Liu, Dichao Peng, Yuliang Zheng, J. Liu. "Communication Protection in IP-based Video Surveillance Systems", Seventh IEEE International Symposium on Multimedia (ISM'05), 2005<br>Publication | 3% |
| 2 | ro.ecu.edu.au<br>Internet Source | 2% |
| 3 | docplayer.net<br>Internet Source | 2% |
| 4 | spot.colorado.edu<br>Internet Source | <1% |
| 5 | tools.ietf.org<br>Internet Source | <1% |
| 6 | heim.ifi.uio.no<br>Internet Source | <1% |
| 7 | www.ecs.umass.edu<br>Internet Source | <1% |
| 8 | www.ess.washington.edu<br>Internet Source | |

<1 %

| | | |
|---|---|---|
| 9 | www.it.iitb.ac.in<br>Internet Source | <1 % |
| 10 | norma.ncirl.ie<br>Internet Source | <1 % |
| 11 | www.coursehero.com<br>Internet Source | <1 % |
| 12 | Yu Li, Baojia Zhang, Yue Ji. "Privacy Preserving Distance Learning with Smart Phones", 2016 12th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN), 2016<br>Publication | <1 % |
| 13 | Fuwen Liu, Hartmut Koenig. "Chapter 9 Puzzle – A Novel Video Encryption Algorithm", Springer Science and Business Media LLC, 2005<br>Publication | <1 % |
| 14 | Wang, Qiu Hua, and Xing Jun Wang. "Study of Selective Video Encryption for the H.264 Standard", Applied Mechanics and Materials, 2013.<br>Publication | <1 % |
| 15 | read.pudn.com<br>Internet Source | <1 % |
| 16 | www.mdpi.com | |

Internet Source

<1%

|  |  |  |  |
|---|---|---|---|
| Exclude quotes | Off | Exclude matches | Off |
| Exclude bibliography | On |  |  |