DEPARTMENT OF ELECTRICAL ENGINEERING & ELECTRONICS

Final report for project ‘Build A WiFi-based Key Generation System Using Raspberry Pi’

Abstract

The WIFI-based key generation system is designed and built on Raspberry Pis(Alice and Bob), applying the unpredictable features of the wireless channel to realize protection on the physical layer. The key generation procedure contains four steps, which are channel probing, quantization, information reconciliation and privacy amplification. Alice and Bob send data packets to each other and measure the RSSI as an indication of the randomness of the wireless channel. Then the received analog data is converted to binary sequence by using the mean-value threshold. After that, mismatched bits between Alice and Bob are corrected based on the BCH code. Finally, the hash function is used to remove revealed information from the public channel. The NIST statistical test is applied to analyze the randomness of the generated keys. In addition, the performance of the key generation system in different scenarios is discussed. Furthermore, a GUI system is designed to present the results of this system more intuitively. By testing, it is found that this system can be applied in mobile scenario and objecting moving scenario.

Content

1. Project specifications....5
1.1 Copy of the original project specifications....5
1.1.1 Package one: Literature Review(4 weeks)....5
1.1.2 Hardware Assembly and Setup....5
1.1.3 Key Generation Procedure....6
1.1.4 Performance Testing and Analysis(3 weeks)....7
1.2 Verification table and Gantt chart....7
1.3 Major revisions to the project specification....8
1.3.1 Key Generation Procedure....8
1.3.2 Performance Testing and Analysis....8
1.3.3 GUI system....8
2. Introduction....9
2.1 Project Description....9
2.2 Open Systems Interconnection model....10
2.3 Wireless security....12
2.3.1 Physical layer attack....12
2.3.2 Security techniques....12
3. Literature Survey....14
4. Industrial relevance, real-world applicability and scientific/societal impact....15
5. Theory....17
5.1 Received Signal Strength Indicator....17
5.2 Randomness....17
5.3 IEEE 802.11 protocol....17
5.3.1 Network architecture....17
5.3.2 Distributed coordination function....18
5.3.3 MAC frame....19
5.4 Key generation protocol....20
5.4.1 Principles....20
5.4.2 Procedure....21
5.5 Evaluation metric....25
5.5.1 Cross-correlation....25
5.5.2 Autocorrelation function....25
5.5.3 Key disagreement rate....26
5.5.4 Randomness test....26
6. Design....27
6.1 Hardware configuration....27
6.2 Software programming....28
6.2.1 Channel probing....28
6.2.2 Quantization....33
6.2.3 Information reconciliation....34
6.2.4 Privacy amplification....36

6.2.5 Tests....36
6.2.6 GUI design....37
7. Experimental method....44
8. Results and Calculations....44
8.1 Scenarios description....44
8.2 Results....45
8.2.1 Channel probing....45
8.2.2 Quantization, information reconciliation and privacy amplification....47
8.2.3 Randomness tests....51
9. Discussion....51
9.1 Performance analysis....51
9.1.1 Performance of two quantization methods....51
9.1.2 Performance under different scenarios....52
9.2 Contribution, limitations and improvements....52
10. Conclusions....52
References:....54
Appendices:....57
A. Gantt Chart....57

1. Project specifications

1.1 Copy of the original project specifications

The project will last for 18 weeks. To complete the project on time, the project is broken down into four packages and each package is divided into several tasks. This section will describe these packages clearly and define them according to the SMART standard which involves specific, measurable, achievable, relevant, and time-bound.

1.1.1 Package one: Literature Review(4 weeks)

The first step in this project is to get familiar with the relevant theories and principles of wireless communication. A lot of materials and papers need to be read. IEEE 802.11 should be understood clearly as the standard supported by the Wi-Fi system. Secondly, the physical layer and MAC layer protocols of Wi-Fi are the basic knowledge of this project, hence it is necessary to research their concepts. Thirdly, Python programming in the Linux system is required in this project, so the operation of the Linux system and Python language should be learned in advance. Finally, the most significant step is having a good command of the key generation procedure and related parameters involved in it.

1.1.2 Hardware Assembly and Setup

There are two tasks shown as follows.

a. Raspberry Pi Assembly(0.5 week)

In this project, one Raspberry Pi 4b 4GB and one Raspberry Pi 64b 2GB are chosen as the development platforms. Additionally, two 16GB Micro SD cards are used for storing code and data.

b. Preparation for Experimental Environment and Test(1.5 weeks)

In this project, one Raspberry Pi 4b 4GB and one Raspberry Pi 64b 2GB are chosen as the development platforms. Additionally, two 16GB Micro SD cards are used for storing. Before starting the project, the configuration of development environment and relevant tests should be completed. Firstly, the Linux system should be installed in an SD card using mirror programmer WIN32 Disk Imager. Secondly, the Secure Shell(SSH) tool will be used to log in to Raspberry Pi remotely. Then, the Virtual Network Console(VNC) Server should be installed in Raspberry Pi and the VNC Viewer will be used to launch the remote desktop. The final step is to configure the Nexmon Channel State Information Extractor and test it if the CSI in the wireless channel between two users can be extracted normally.

1.1.3 Key Generation Procedure

The key generation procedure(shown in Figure 2) is the most important part of this project, which is divided into four tasks- channel probing, quantization, information reconciliation, and privacy amplification. This package will take about nine weeks in total. More details about the four tasks will be described.

a. Channel Probing(4 weeks)

During the channel probing step, CSI will be measured from the wireless channel between Alice and Bob to extract the randomness existing in the temporal, spatial, and frequency domain. The channel sampling is to be processed based on time division duplex(TDD) systems. Alice will send a request packet to Bob, then Bob will send acknowledge(ACK) back to Alice after receiving the packet and measure the CSI. Similarly, Alice will measure the same parameter after receiving the ACK. Alice and Bob will get a set of keys by repeating the sampling of channel parameter CSI. It is worth noting that due to the limitation of the TDD system, hardware cannot receive and transmit a packet at the same time, hence there will be a time delay before Bob sending back the response packet. And the time delay will vary according to the packet type. The longer the time delay, the larger the difference of channel measurements between Alice's and Bob's might be. Therefore, a suitable packet should be chosen to shorten the delay time between transmitting and receiving. Furthermore, during wireless communication, there will be interference of noisy and useless packets from other wireless devices which can lead to mismatches between the measured parameters from Alice and Bob. Therefore, some countermeasures should be taken such as applying signal preprocessing algorithms to reduce error[1].

b. Quantization(2 weeks)

At this stage, the analogue measurements CSI will be converted into binary sequences. The quantization level(QL) determines the key bits of each measurement[8]. The mean and standard deviation-based quantizer will be used in the quantization process. The positive threshold t^+ and negative threshold t_- will be calculated based on the mean value m and standard deviation s. The formulas are as follows:

$$t^+ = m + \alpha \times s \quad (1)$$

$$t_- = m - \alpha \times s \quad (2)$$

where α is the turning parameter. The samples above t+ will be converted to 1, while samples below t− will be converted to 0. In addition, the measurements between t+ and t− will be neglected.

c. Information Reconciliation(2 weeks)

As mentioned in section 4.3.1, due to the limitation of TDD systems and the influence of noise, there will be mismatches between Alice's and Bob's channel measurements CSI at the

same frequency. Consequently, bit errors will appear after quantization, hence information reconciliation is required to correct the mismatch. Two methods are usually employed in this process, which are error detection protocol-based approaches(EDPA) and error correction code-based approaches(ECCA)[1]. BCH code, one of ECC, will be used to correct mismatches in this project. This code mainly relies on exclusive OR to implement the operation. Additionally, the bit differences should be within the code's correction capacity, otherwise, the information reconciliation procedure could be invalid.

d. Privacy Amplification(1 week)

Since the information transmission of previous steps is carried out in the public channel, part of the channel information can be revealed to eavesdroppers which will make them find the generated key. Thus, privacy amplification should be applied to eliminate the leaked information. This step will be implemented by using hash functions.

1.1.4 Performance Testing and Analysis(3 weeks)

In this section, the performance of the key generation system should be tested. The real-time channel measurements of CSI will be displayed on the PC using MATLAB. In addition, the AES encryption and decryption will be used to test the feasibility of the generated keys. The test will be carried out in various scenarios, and the performance will be compared and analyzed. The key generation rate(KGR) and key disagreement rate(KDR) will be partial metrics. The KDR can be calculated based on the following equation:

$$\mathrm{KDR} = \frac{\sum_{i=1}^{N} |K^{A}(i) - K^{B}(i)|}{N} \tag{3}$$

where N is the length of Keys[2]. A smaller KDR means a better key generation performance.

1.2 Verification table and Gantt chart

Table 1: Verification table

Parameters	Verification
Channel state information	Display on MATLAB
Positive and negative thresholds for quantization	Review calculation
Binary numbers	Test using calculated thresholds
Key disagreement rate(KDR)	Test using given formula
Key generation Rate(KGR)	Test using given formula

Then original and revised Gantt charts are shown in Appendix A.

1.3 Major revisions to the project specification

1.3.1 Key Generation Procedure

a. Channel probing

- **Channel measurement**
  In the original project specification, the Channel State Information(CSI) is to be used to present the wireless channel information. Nevertheless, after having a deeper understanding of the tool 'the Nexmon Channel State Information Extractor', it is found that the CSI can not be extracted from the devices that are conducting communication, which means the third device is required. In addition, the modification of the internal code of this tool is extremely complicated. Therefore, the tool 'scapy' is used instead and the parameter to be measured is changed to Received Signal Strength Indicator(RSSI). Furthermore, since the tool does not have a mechanism to return ACK, Data packet is used at both sides. Due to the change, this period was extended by two weeks

- **Packet matching**
  Since in actual situations, packet loss will happen during transmission between two devices. The step packet matching is added to equalize the packet number received by Alice and Bob.

b. Quantization

In addition to the mean value-based quantization method, another method is added which is differential-based quantization.

c. Information reconciliation

Cyclic redundancy check is added to test the sameness of the generated keys.

1.3.2 Performance Testing and Analysis

The randomness test is added in this step and the NIST statistics test suite is used to realized this step.

1.3.3 GUI system

In order to demonstrate the key generation procedure and the results more directly, GUI designing is added in this project.

2. Introduction

The proliferation of the Internet of Things(IoT) makes it possible to coordinate decisions and share information between physical objects[3]. As the best medium to connect the IoT devices, wireless communication is getting widespread attention and many wireless technologies are developed such as IEEE 802.11, BLE, IEEE802.15.4, and LoRaWAN. However, security issues can appear because of the broadcast nature of wireless communication, for example, eavesdroppers can attack users who are transmitting information. The WiFi-based key generation system is proposed to provide security protection for WiFi system. In this report, the knowledge related to this project is described. In addition, the literature survey about the previous work that has been done in this field and the applicability of this project are mentioned. The theories involved and the designing process are introduced in detail. Furthermore, the results obtained are shown and the performance of the outcome is discussed.

2.1 Project Description

This project is to build a WiFi-based system that generating cryptographic keys for two legitimate users by applying the unpredictable features of the wireless channel. The key generation procedure can be divided into four stages, which are channel probing, quantization, information reconciliation, and privacy amplification(shown in Figure 1). During channel probing, data packets are transmitted between the two users, Alice and Bob, and the channel parameter Received Signal Strength Indicator(RSSI) is measured at both sides. After that, the decimal RSSIs are converted to a sequence of binary numbers at the stage of quantization. Then, information reconciliation is carried out to correct the mismatched bits between the two generated keys. Finally, revealed keys in the previous stage are eliminated by the stage privacy amplification. After key generation, randomness test, Advanced Encryption Standard(AES) encryption test are conducted on the generated keys. In addition, the performances of this system under diverse scenarios are tested and analyzed. Graphical User Interface(GUI) is designed to improve the operability and practicality of the system. This project is realized mainly based on programming using python and the knowledge of MAC layer and physical layer protocols. Two raspberry pis are used to demonstrate the system(shown in Figure 2).

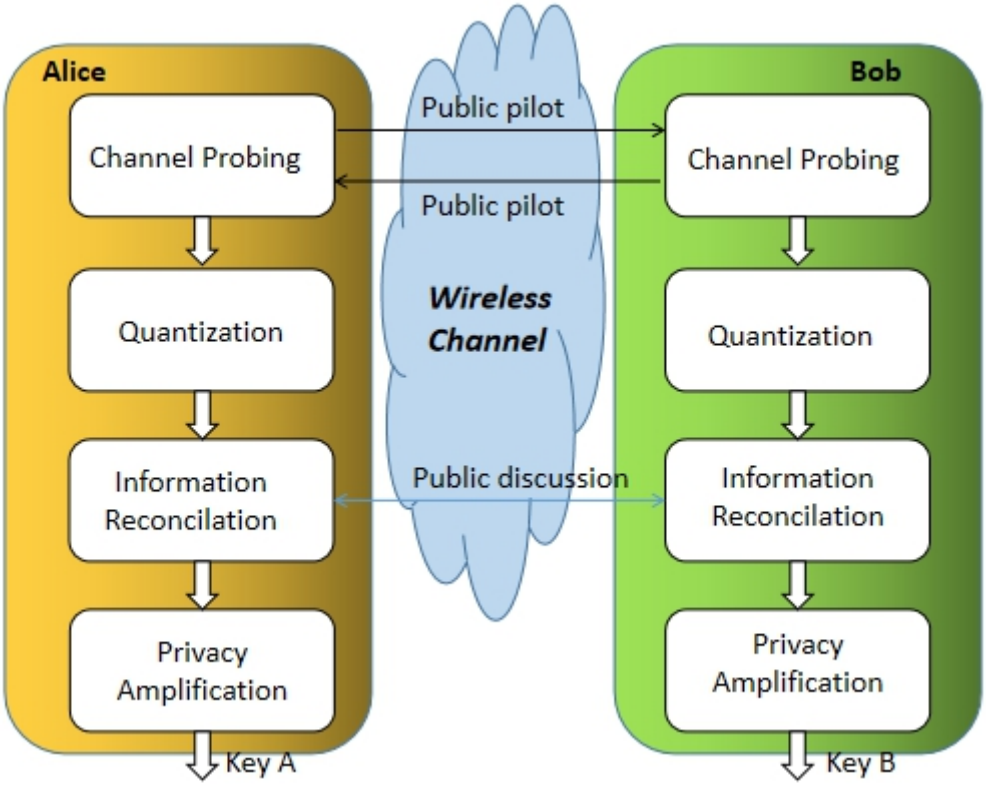


Figure 1: Key generation procedure

Figure 2: Alice and Bob.

2.2 Open Systems Interconnection model

In 1983, the Open Systems Interconnection(OSI) was proposed by the International Standard Organization(ISO), aiming to standardize the open communication between different computing systems, which is robust, interoperable, and flexible[4]. This model divides the work of network communication into seven layers, which are the physical layer, data link layer, network layer, transport layer, session layer, presentation layer, and application layer(shown in Figure 3).

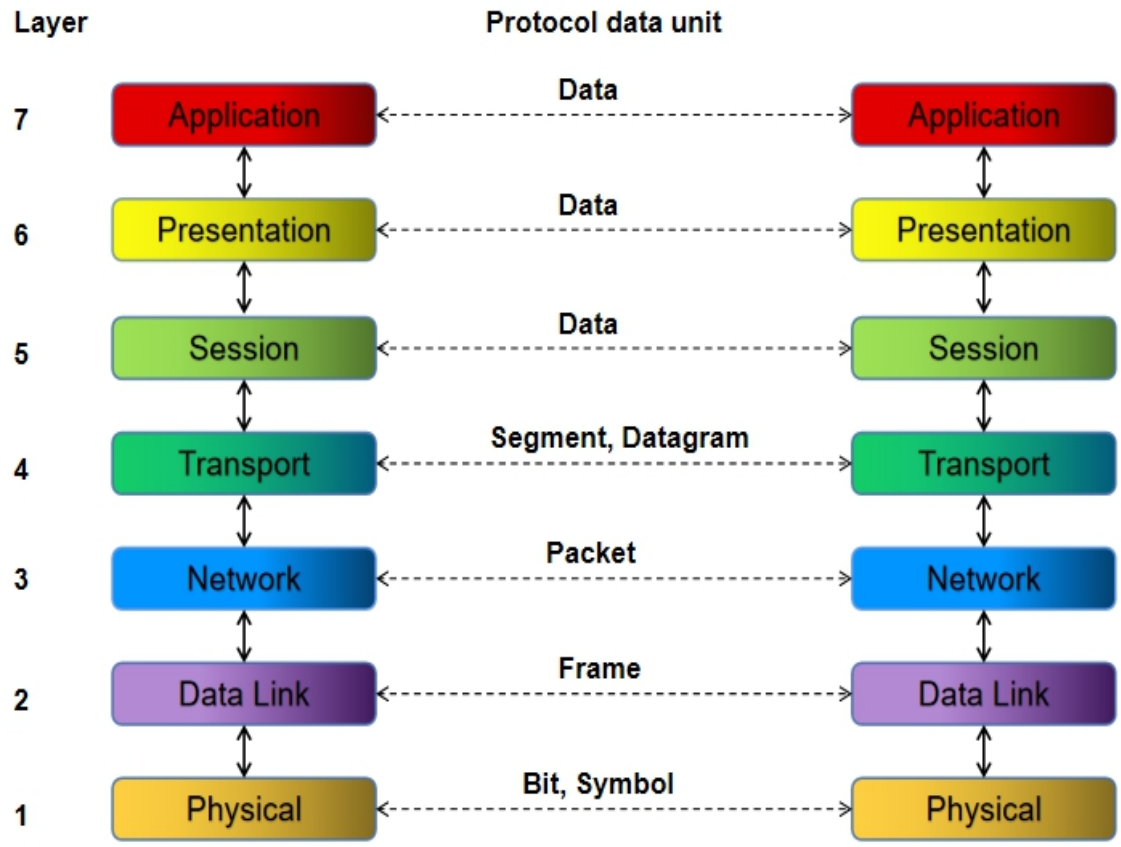


Figure 3: Open Systems Interconnection model.

The functions of the seven layers are described in [4] and[5]:

- Physical layer: This layer is responsible for establishing physical connection between Data Link entities to convey bit stream. For example, cables, hubs, optimal fiber, etc.
- Data Link layer: This layer includes two sublayers, which are the Media Access Control(MAC) layer and the Logical Link Control(LLC) layer. This layer is used for packaging the data into frame and transmitting information between physical devices within the same logical network. In addition, it has the function of correcting errors that occur in the physical layer. For example protocols, Ethernet, IEEE 802.2, ATM, IEEE 802.11, etc.
- Network layer: This layer is used for delivering packets between different networks by the routing technologies and the logical address. For example protocols, IP, ICMP, ARP, etc.
- Transport layer: This layer is responsible for delivering whole message between hosts, flow control, and error control. For example protocols, TCP, UDP.
- Session layer: This layer is responsible for setup, coordination, data exchanges between applications at each end. For example protocols, RPC, PPTP, etc.
- Presentation layer: This layer is responsible for the format translation of the data during network communication such as encryption and data compression. For example protocols, SSL, TLS, etc.

- Application layer: This layer is responsible for providing services for users to access the network. For example protocols, HTTP, FTP, DNS, etc.

2.3 Wireless security

Each OSI layer has a unique security technology based on its own protocol. However, most of the security methodologies are designed for the MAC layer and above. By contrast, the lowest layer, the physical layer, is out of security consideration, which makes it easier to be attacked[6].

2.3.1 Physical layer attack

Due to the broadcast nature of wireless communications, the physical layer are extremely susceptible to two main kinds of physical layer attacks, which are jamming and eavesdropping attacks[7]. The jamming attack(also known as DoS attack) refers to a malicious node deliberately interfere with the wireless network to prevent data transmission between legitimate users, which can affect the network availability of authorized users. The eavesdropping attack means an unauthorized user seeking to monitor the data communication between two legitimate users[7].

2.3.2 Security techniques

(1) Classic cryptography

Cryptography achieves information protection by applying mathematical algorithms and protocols. Based on the keys used at the two parties, this approach can be divided into asymmetric, and symmetric encryption[8]. For asymmetric encryption(as known as public-key cryptography), a pair of different public and private keys are used(shown in Figure 4). The PKC includes three main applications, which are encryption, key distribution and digital signatures[8]. For the symmetric encryption, the identical key is used for both users(shown in Figure 5). One example of the symmetric algorithm is the Advanced Encryption Standard(AES)[9].

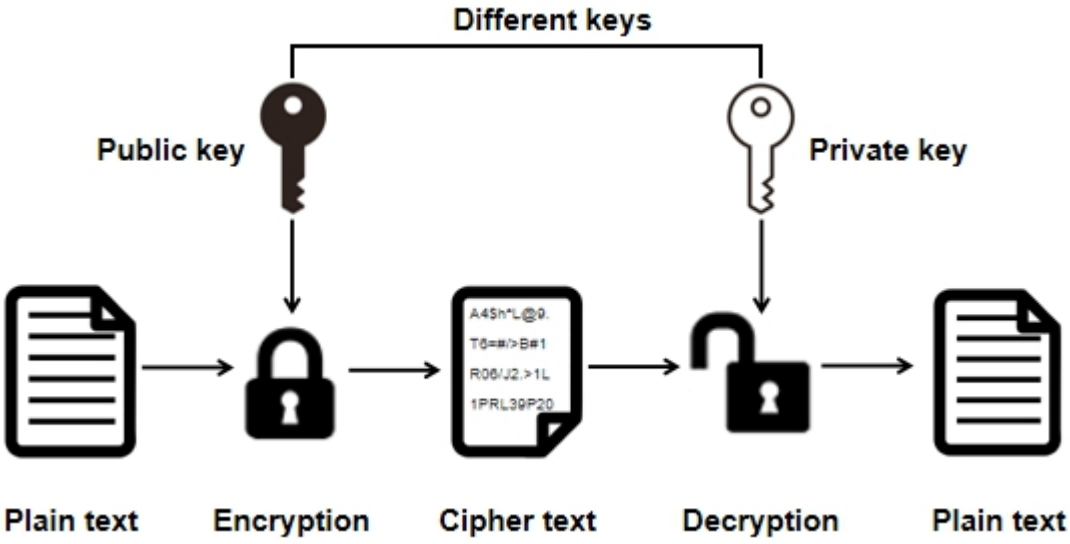


Figure 4: Asymmetric encryption.

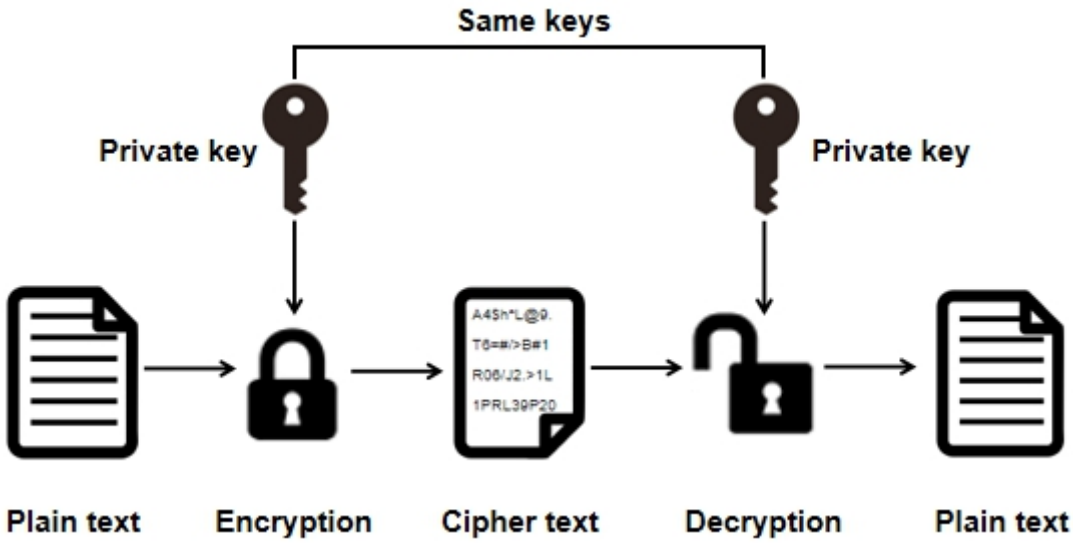


Figure 5: Symmetric encryption.

(2) Physical layer security

The physical layer security approach can be divided into two techniques, keyless security transmission and key generation[10]. For the keyless security transmission, a wiretap channel model is designed(as shown in Figure 6). When the channel capacity of the legitimate exceeds that of the wiretap channel, perfect security without encryption can be realized. However, this technique is not suitable for practical applications due to the requirements of complex precise channel state information(CSI) and complex algorithm. For key generation protocol, the keys are generated from the wireless channel based on the randomness feature of the wireless environment. Four stages are included in the key generation procedure which will be discussed in section VII. Due to the limitation of the key generation rate, a hybrid

cryptosystem combining symmetric encryption and key generation is commonly applied(shown in Figure 7)[1].

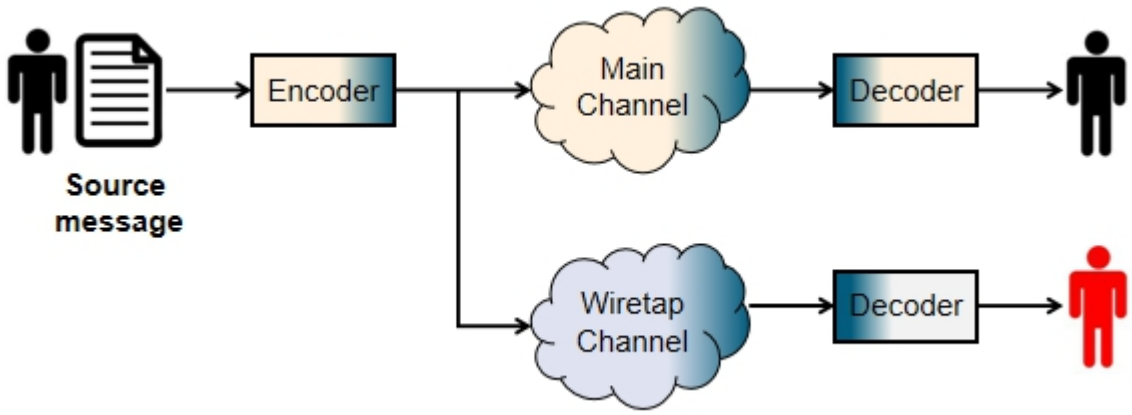


Figure 6: Keyless security transmission model.

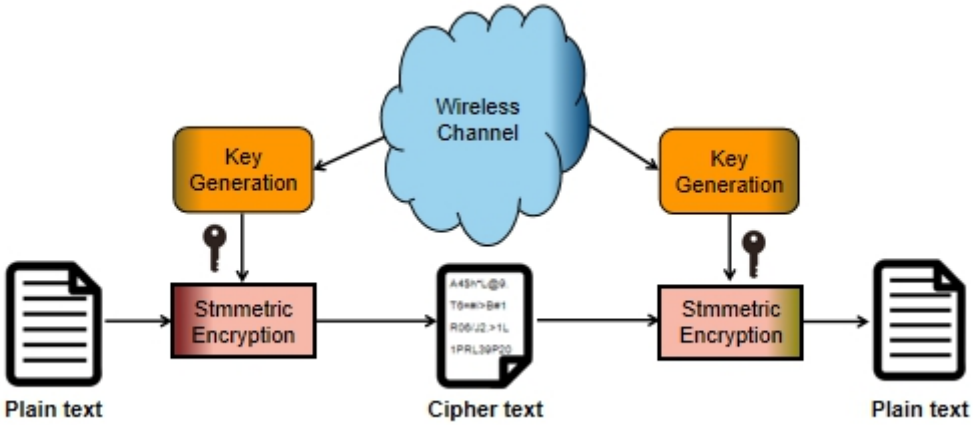


Figure 7: Hybrid cryptosystem.

3. Literature Survey

The information-theoretical basis of the physical layer key generation was laid by Ahlswede, Csiszar[10] and Maurer[11] in 1993. After then, a large number of researches were carried out in this field. As the most significant index to indicate the randomness of the wireless channel, RSSI and CSI are usually used in the research about key generation[1]. RSSI is a coarse-grained parameter, while CSI is a fine-grained parameter. Compared with CSI, RSSI is easier to be measured but more vulnerable to stalking attacks and predictable channel attacks[1].

For the research about RSSI-based key generation, Y. Wei, et al[12] proposed an adaptive channel probing scheme, which can be used to edit the probing rate of RSSI based on environment variation. This technique is realized by applying the Proportional Integral Derivative(PID) controller and therefore the expected Key Generation Rate(KGR) can be

controlled. For the research about CSI-based key generation, H. Liu, et al[13] applied the Orthogonal Frequency Division Multiplexing(OFDM) subcarriers to measure CSI. In addition, the Channel Gain Complement(CGC) algorithm was designed to deal with the issues of channel non-reciprocity.

In recent years, researchers are attempting to apply the physical layer key generation in different applications such as Bluetooth, LoRa, IEEE 802.15.4, etc. For example, S. N. Premnath, el al[14] developed the Bluetooth-based key generation method with robust to heavy WiFi traffic by combing an extremely wide bandwidth with random frequency hopping. It was proved that this technique can generate secret keys with much lower transmit power.

4. Industrial relevance, real-world applicability, and scientific/societal impact

The evolution of the IoT is gradually changing the world and a number of new concepts were proposed such as smart home[15], smart city[16], which brings great convenience to people's life. For example, people can remotely access, control, and monitor the appliances or sensors that are equipped in a smart home[15]. In addition, the number of IoT devices is increasing dramatically. According to[17], the whole economic impact caused by IoT every year is expected to be in the range from f $2.7 trillion to $6.2 trillion by 2025. Moreover, it is estimated by Cisco that there will be 500 billion IoT devices by 2030[18].

The IoT devices are physically connected by the networking infrastructure including wired and wireless communication networks. Among the two types of communication, wireless infrastructures are considered the most potential and outspread part of IoT due to their flexibility and low installation costs[19]. A number of wireless techniques are developed used for different applications, such as WiFi, ZigBee, LoRa, etc. Nevertheless, while benefiting from the convenience of wireless technologies, people's privacy information security is faced with great challenges. Due to the characteristic of the wireless channel and the defect of classical cryptosystems which focuses on upper layer encryption, wireless communication is vulnerable to passive attacks(eavesdropping) and active attacks(jamming)[6]. For instance, the physical packet header and MAC header are sent in plaintext, which will reveal side-channel information(SCI) such as mapping schemes, data rate, etc[20]. Through analyzing the SCI, attackers can conduct various attacks on users. Furthermore, the cryptographic methodologies with complex algorithms usually have higher requirements about the power consumption, code space, and silicon area[21]. Therefore, many low-cost IoT devices overlook the security issues in order to obtain better hardware performance and efficiency. Weak keys are used in these IoT devices and hence increases the risk of information leakage. According to the research carried out by Kaspersky Lab[22], 93% of IoT malware

propagation is realized by the brute-forcing of passwords- repetitive attempts using different passwords.

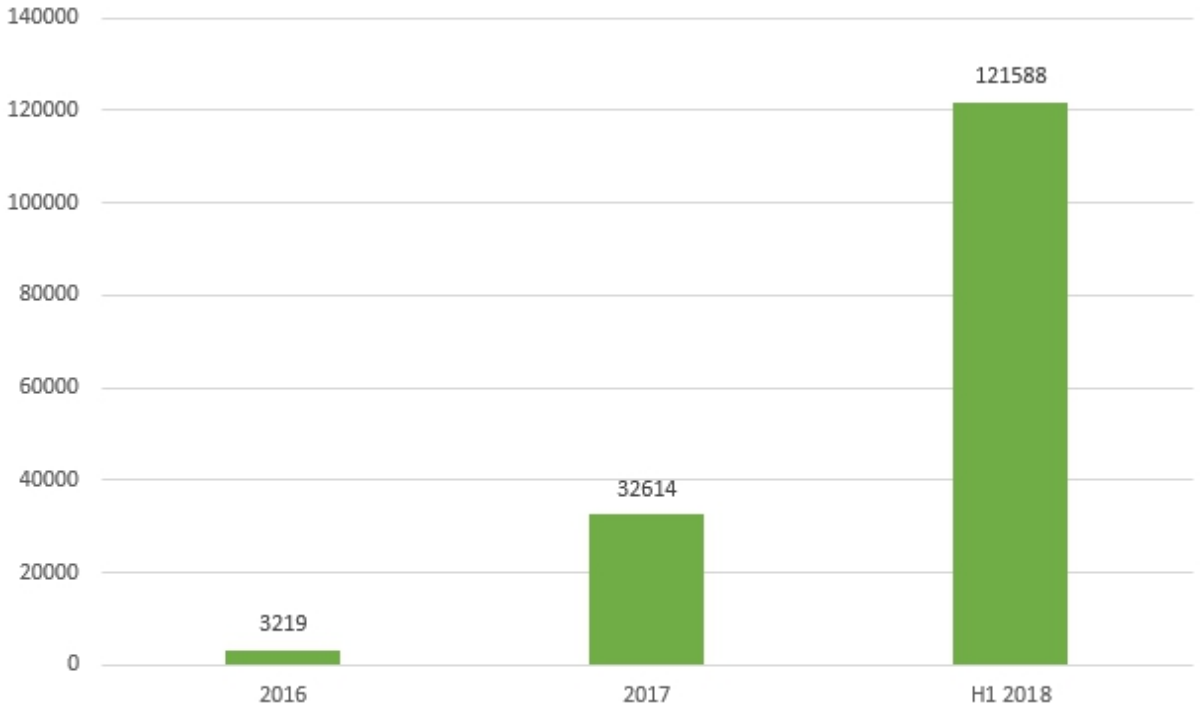


Figure 8: Number of malware samples for IoT devices collected by Kaspersky Lab[22].

Currently, people's protection of information security is far from enough. It can be seen from Figure 8 that, with the increase in the number of IoT devices, the number of information security attacks is also booming up year by year. The impact of some hacking attacks can be catastrophic. The Mirai botnet attack[23] in late 2016 is such an example. A malware family infected a large number of IoT devices with insecure default keys such as IP cameras and routers. Then Mirai carried out massive distributed denial of service(DDoS) attacks on DNS servers by controlling those devices. Many websites in Europe and the USA were brought down and millions of users are affected because of this. Therefore, the development of a secure password for IoT devices is imminent and enhancing physical layer encryption(PLE) has become a significant part of the cryptosystem. By applying physical layer key generation, the entire packet can be protected including the MAC header and physical layer header, and therefore greatly reduces the risk of being attacked by eavesdroppers[24]. In addition, due to the lightweight nature of physical layer key generation, it can be applied in many low-cost IoT devices.

Body area network is an emerging technology in recent years, which consists of several small sensors mounted on the human body[25]. Devices under this network can measure vital signs and share these statistics with each other by wireless communication so that patients can obtain real-time health information. Nevertheless, there is a huge challenge to secure these devices. Private medical data is recorded and handled in these wearable wireless sensor devices, which must be adequately secured. Meanwhile, considering the low power and computational capabilities, conventional cryptosystems are not suitable for these devices[25].

Thus, the physical layer key generation provided a solution to prevent their medical devices from eavesdropping by constructing secret keys from the unsecured wireless channel.

Based on the above description, it can be found that the technique of physical layer key generation can secure the lowest layer of the OSI system from attacks effectively. In addition, this technique provides a feasible low-cost and low-computing encryption scheme for low-cost IoT devices. Although there are still several issues, such as the low key generation rate and the limitation of CSI measurement, physical layer key generation has great market potential.

5. Theory

5.1 Received Signal Strength Indicator

The RSSI is a parameter used to indicate the quality of a wireless channel. It can be measured in almost all wireless techniques such as IEEE 802.11, LoRa, Bluetooth, etc. Nevertheless, the specific calculation method is determined by the vendors[1].

5.2 Randomness

During wireless communication, the signal power will attenuate with propagation which is called fading[26]. Based on the TX-RX separation, fading can be divided into large-scale fading including path loss and shadowing, and rapid small-scale fading which is multipath fading. Since the determinism of path loss and slow changing of shadow fading which are not suitable for key generation, the key generation is mainly contributed by multipath fading[1]. In addition, refraction, reflection and scattering happening in the wireless environment generate the randomness of the wireless channel.

5.3 IEEE 802.11 protocol

The IEEE 802.11 provides the operation standards for the physical layer and MAC layer of the wireless local area network(WLAN) working at 2.4/5 GHz. The IEEE 802.11 standard was approved in 1991, and from then diverse IEEE 802.11 standards were developed to make improvements or extensions based on the original 802.11 standards[27]. Nowadays, the IEEE 802.11families have been widely applied in the house, office and various public areas. In addition, most PCs, smartphones, tablets, and other IoT devices support the IEEE 802.11 standards.

5.3.1 Network architecture

In the IEEE 802.11 network, there are four basic physical components, which are access point(such as routers), station(such as laptops), wireless medium and distribution system[28](shown in Figure 9). In the network system, the basic building block is the basic service set(BSS), which defines the coverage area in which the stations can communicate

with each other. Based on whether the access point(AP) is involved in the BSS, the BSSs can be divided into Independent BSS(IBSS) and Infrastructure BSS[28](shown in Figure 10). In the Independent BSS(which is also known as hoc BSS), stations can communicate with each other directly within the basic service area. In the infrastructure BSS, the access point is used to connect to all the stations within the service area. The communication between two stations should take two hops[28]. The source station should transmit data to the access point, and then the data is transmitted from the access point to the destination station.

Figure 9: WLAN system.

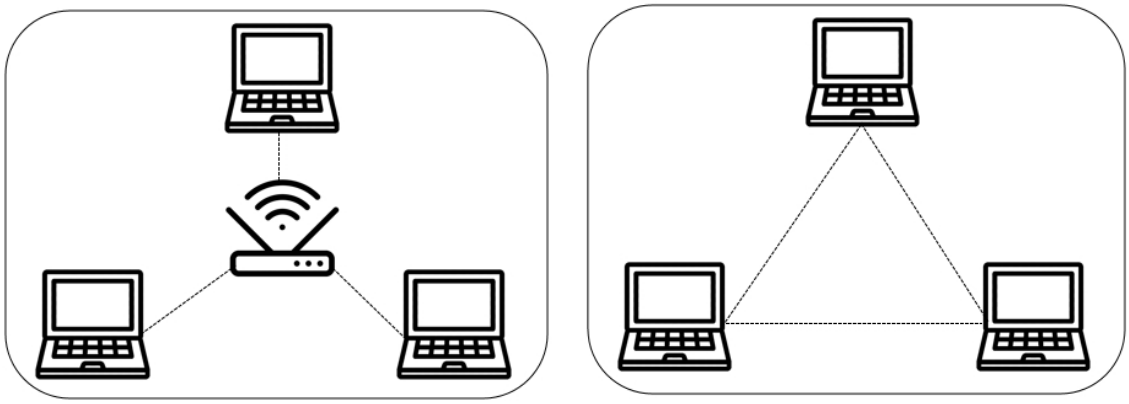


Figure 10: Infrastructure BSS (left) and Independent BSS (right).

5.3.2 Distributed coordination function

The distributed coordination function(DCF) is the fundamental medium access method that all stations should support, based on carrier sense multiple access mechanisms with collision avoidance(CSMA/CA) protocol[29]. The DCF consists of two techniques used for packet transmission which are two-way handshaking and four-way handshaking[30]. For the first technique which is also the basic access method (shown in Figure 11(a)), stations should conduct carrier-sensor mechanism to monitor whether the wireless channel is idle or busy. If the channel is idle, the source station could start transmitting when the medium is sensed idle for more than a distributed interframe space(DIFS). Otherwise, when the channel is busy, after a DIFS, the source station should wait for an extra binary exponential backoff period before starting transmission. When the destination station receives the packet, a positive acknowledgment(ACK) is sent back to the source station after a short interframe space(SIFS) to inform the successful transmission. In the second technique (shown in Figure 11(b)), the

request-to-send(RTS)/clear-to-send(CTS) mechanism is applied. When the wireless medium is idle for a DIFS, an RTS packet is sent and a CTS packet should be received as a response before starting data packet transmission. If this process succeeds, data packet and ACK can be transmitted between two stations. The time intervals between RTS, CTS, data and ACK are all SIFS time.

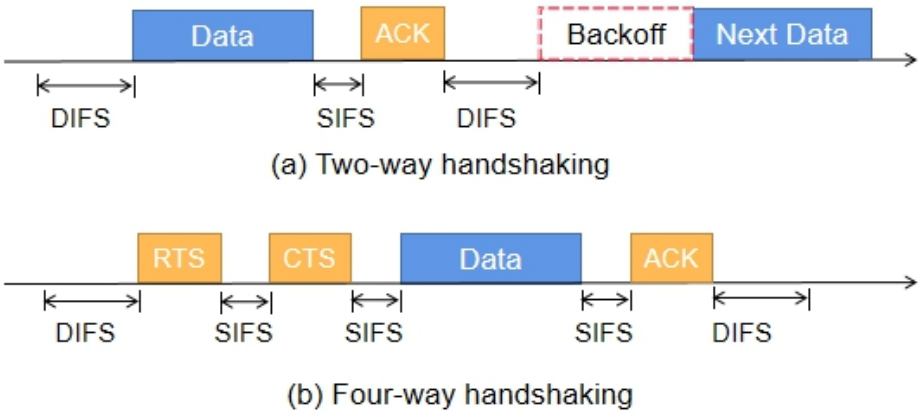


Figure 11: IEEE 802.11 access method.

5.3.3 MAC frame

Figure 12 shows the generic IEEE 802.11 MAC frame format. The function of each component of the frame is described as following[28]:

① Frame control field:
- Protocol version: determines the version of the 802.11 protocol applied in the frame.
- Type and subtype: indicates which type of frame is used.
- To DS and From DS: determines whether the frame is destined for the distribution system.
- More fragments: used for some large management frames and large data frames.
- Retry: set to be 1 when the frame requires to be retransmitted.
- Power management: indicates whether the station is at power-save mode or active mode.
- More data: shows whether there is an available frame at the access point.
- Wired equivalent privacy: shows if WEP or WPA is executed.
- Order: determines whether the frames or fragments are transmitted based on Strictly Ordered Service.

② Duration/connection ID: indicates the time allocated in a channel

③ Address: identifies the destination address, source address, receiver address and transmitter address.

④ Sequence control: defragments and discards duplicate frames.

⑤ Frame body: carries the transmitted data.

⑥ Frame check sequence: also known as cyclic redundancy check(CRC) which is used for checking the integrity of the received frames.

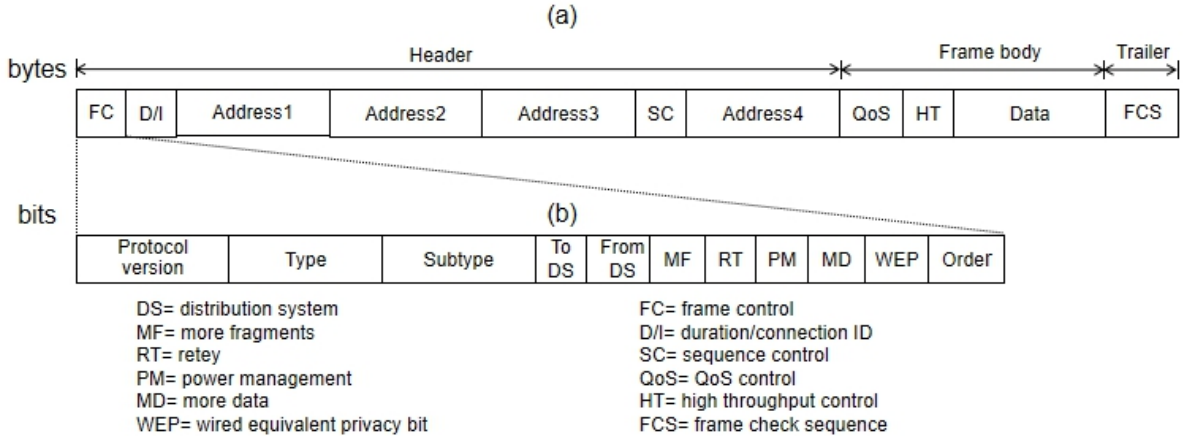


Figure 12: (a) IEEE 802.11 MAC frame format (b) Frame control field.

There are three types of frames, which are management frames, control frames and data frames[28]. The management frames are used to serve the wired network, such as probe request/response, association request/response, beacon, etc. The control frames can manage stations' access to wireless channel. The data frames are used to transmit data with higher-level protocols. In this project, data frame is applied as it is suitable for multiple transmissions in the real wireless communication.

5.4 Key generation protocol

5.4.1 Principles

- **Channel reciprocity**

Channel reciprocity refers to the identical channel variations in both the forward and backward links[31], which means Alice and Bob can obtain the same RSSIs. Therefore, this principle is the basis to realize key generation. However, in practice, asynchronous measurement is inevitable during channel probing due to the operating mechanism of the time division duplex(TDD) mode. In addition, the consistency of channel responses is affected by hardware noises[32]. Hence, the generated keys are not always symmetric because of the limitation of non-simultaneous measurements and noises.

- **Spatial decorrelation**

Based on Jake's model[33], the eavesdroppers will experience an uncorrelated fading when they are located approximately half-wavelength away from the legitimate users, which is called spatial decorrelation. Therefore, the eavesdropper Eve will not be able to extract effective information from the transmission between Alice and Bob. This feature guarantees the security of key generation. Nevertheless, uniformly distributed and infinite scatters are required in the surrounding environments, which may not satisfied in all conditions. For example, in an environment with strong line-of-sight(LOS) components and poor scattering,

there are still correlations between the signals observed by attackers and the signals received by legitimate users[34]. Hence, eavesdropping prevention is an essential part that requires to be considered.

- **Temporal variation**

Temporal variation indicates the variation of the wireless channel over time, introduced by the movement of the transmitter, receiver or the objects in the surrounding environment[1]. This process will effect the reflection, refraction and scattering of fading channel, and therefore ensures the randomness of key generation. In addition, the performance of the key generation system can be affected by the rate of channel variation. For instance, an extremely slow fading channel may lead to inefficiency of key generation, while an extremely fast fading channel may reduce the correlation of two-way measurements. Based on the above problems, one solution is applying an adaptive probing method to adjust channel probing rate based on channel variations as described in [12]. Furthermore, in a extremely static case, the channel is nearly stable over time which cannot generate randomness for key generation. Artificial randomness generation was proposed to keep key generation normally in the static environment[35].

5.4.2 Procedure

The key generation procedure consists of four stages, which are channel probing, quantization, information reconciliation and privacy amplification (as shown in Figure 2). The channel parameter RSSIs are measured through channel probing. Then the analog measurements are converted to binary numbers by quantization. After information reconciliation, the mismatched bits between two sequences are corrected. Finally, privacy amplification is employed to remove the leaked information.

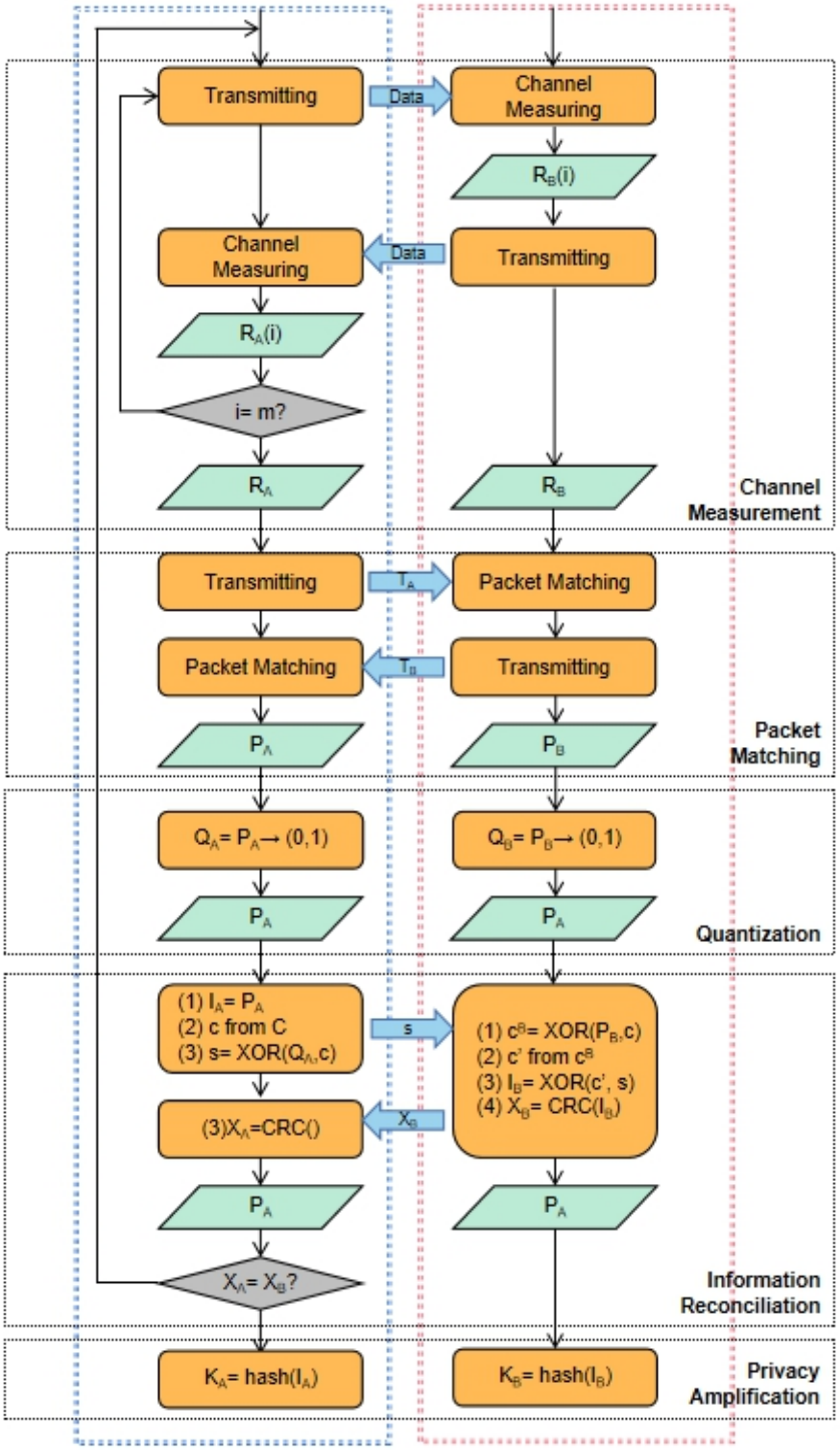


Figure 13: Key generation procedure.

A. Channel probing

The first stage channel probing consists of two parts, which are channel measurement and packet matching.

- **Channel measurement**

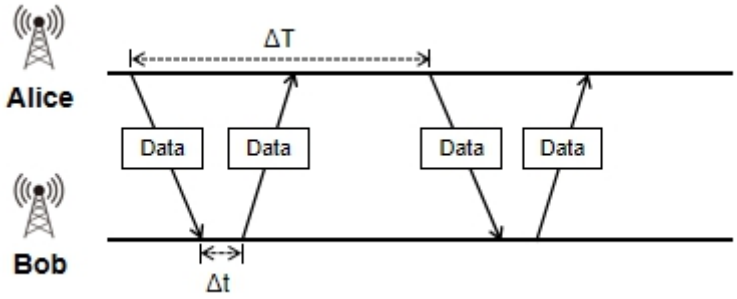


Figure 14: Channel measurement in TDD mode.

At channel measurement, packet exchange occurs and RSSIs are measured. Since the key generation system usually works in TDD mode, the data packet cannot be sent at both Alice and Bob sides simultaneously. In this case, the station Alice firstly sends a data packet to access point Bob. When receiving the packet, the corresponding RSSI, $R_B(i)$, can be measured by Bob. After a short time interval Δt, Bob sends a data packet back to Alice, who will also be able to measure the RSSI, $R_A(i)$. In addition to the RSSI, the timestamps of receiving packets can also be recorded. At this stage, one cycle of transmission is completed. This cycle is carried out every time interval ΔT. In order to eliminate the correlation between samples, the time interval ΔT should be larger than the coherence time T_C, where T_C is defined by the function[36]:

$$T_C = \frac{0.423}{f_D} = \frac{0.423c}{vf_C} \tag{4}$$

where f_D is the Doppler spread, c is the speed of light, v is the relative speed of the receiver and transmitter, f_C is the carrier frequency. In addition, the repetition of the cycle is determined by the required key length. After channel measurement is finished, both Alice and Bob can obtain a set of measurements, R_A and R_B.

- **Packet matching**

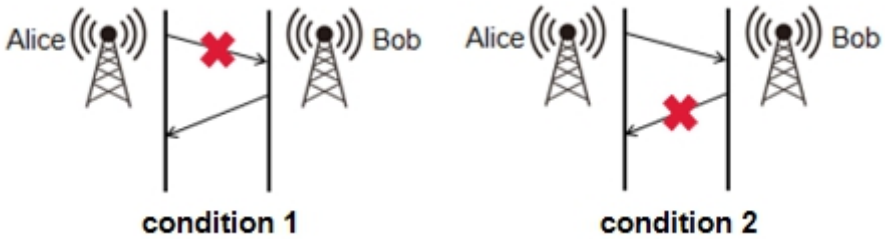


Figure 15: Packet loss.

Nevertheless, in practice, due to the instability of the wireless environment, there will be packet loss during transmission(as shown in Figure 14) which will lead to the different number of packets received by Alice and Bob. In addition, the time difference between the measurements of Alice and Bob should be small enough to satisfy the principle of channel reciprocity. Therefore, packet matching is required to equalize the packet number between the two users and remove the packets that are

received out of the range of coherence time. Firstly, all the key-independent timestamps T_A are transmitted from Alice to Bob. Then, the time difference between the measurements of Alice and Bob can be calculated and the mismatched packets are removed at Bob side. Then the timestamps measured by Bob, T_B, are transmitted to Alice and the mismatched packets are removed at Alice side. After packet matching, symmetric RSSIs, P_A and P_B, are obtained by Alice and Bob.

B. Quantization

In this stage, the measured decimal RSSI values are converted to a binary sequence. There are two methods applied for this stage, which are mean and standard deviation-based quantization and differential value-based quantization.

- **Mean and standard deviation-based quantization**
  In this method, the upper and lower thresholds, t^+ and t_-, are calculated based on the mean value **m** and the standard deviation **s** (as shown in formulas (1) and (2)). Alice and Bob can compare the received RSSI with the thresholds to obtain the binary numbers. The RSSIs that are larger than the upper threshold represents 1s and that are smaller than the lower threshold represents 0s. Otherwise, the RSSIs are dropped.

- **Differential value-based quantization**
  This method is conducted based on the variation of adjacent measurements[37]. The upper and lower thresholds, t_D^+ and t_D^-, are calculated based on the measured RSSI and the parameter resolution. The formulas are shown as below:

$$t_D^+(i) = R(i-1) + \epsilon \quad (1)$$

$$t_D^-(i) = R(i-1) - \epsilon \quad (2)$$

  where R(i-1) indicates the value of the previous RSSI, and ϵ is the parameter resolution that is used to eliminate the effect caused by hardware noise. The RSSIs that are larger than the previous one represent 1s, and the RSSIs that are smaller than the previous one represent 0s. Otherwise, the RSSIs are dropped.

Since in this section, the dropped bits of Alice and Bob are different, the dropped bits should be shared between the two uses through the public channel. Therefore, few measurements are removed after quantization which will decrease the key generation rate. An alternative solution is assigning α or ϵ to be 0 so that all measurements can be converted to binary numbers. Nevertheless, this can lead to lower entropy and higher bit disagreement[38]. Hence, a tradeoff between the key generation rate and key disagreement rate should be considered. After this stage, Alice and Bob acquire the binary sequences, Q_A and Q_B.

C. Information reconciliation

Due to the limitation of transmission mechanism, the bit-differences are generated after quantization even the measured RSSIs of Alice and Bob are extremely similar. Thus, the stage information reconciliation is employed to correct the mismatched bits between Alice and Bob. Two types of approaches are based to realize this stage, which are error detection protocol-based approaches(EDPA), such as the Cascade, BBBSS and Winnow schemes, and error correction code-based approaches(ECCA), such as low-density parity check(LDPC), BCH, and Golay codes[39]. BCH code is used in this project.

Firstly, a codeword **c** is selected randomly from the BCH code set **C** by Alice. Then, the syndrome is calculated by the exclusive-OR operation between the codeword and the binary sequence, where **s**= XOR($Q_{A,}$ **c**). After that, the syndrome **s** is sent to Bob. When Bob completely receives the syndrome, the codeword $\mathbf{c^B}$ is calculated to correct the errors, where $\mathbf{c^B}$= XOR(Q_B, **s**). Next, by processing BCH decoding on $\mathbf{c^B}$, another codeword **c'** is obtained, where **c'** should be equal to **c**. Finally, the new binary sequence I_B is calculated by exclusive-OR operation, where I_B= XOR(**c'**, **s**). The binary sequence for Alice after Information reconciliation does not change, so I_A= Q_A. At this stage, the keys generated by Alice and Bob should be the same, i.e. $I_{A=}$ I_B.

Nevertheless, when the error bits exceed the maximum correction capacity, different keys can be generated. In order to confirm the key agreement, the cyclic redundancy check(CRC) is applied. If the CRC results for both Alice and Bob are identical, all the errors have been corrected. Otherwise, the key generation fails and requires to be restarted.

D. Privacy amplification

Finally, since the signal transmission of previous steps, such as channel probing, quantization and information reconciliation, is carried out in the public wireless channel and the coherence time is hard to be estimated in practice, the measured bits of legitimate users can be correlated and be revealed to the public channel, which may cause Eva to infer the key based on the leaked information[40]. Therefore, privacy amplification should be conducted to eliminate the revealed keys. The universal hash families are commonly applied to deal with this problem, such as the cryptographic hash functions, leftover hash lemma, and Merkle-Damgard hash function[2]. Through randomly selecting from a publicly known set of hash functions, smaller-length keys with fixed size can be generated from the longer input streams. When the stage is finished, a pair of identical keys with higher-level security is generated.

5.5 Evaluation metric

5.5.1 Cross-correlation

The similarity of signals received by the two users Alice and Bob can be indicated by the cross-correlation, which is defined as:

$$\rho^{A,B} = \frac{E\{X^A X^B\} - E\{X^A\}\{X^B\}}{\sigma^A \sigma^B} \quad (5)$$

where E{} represents the expectation operation, X^A and X^B are the measurements by users, and σ denotes the standard deviation of measurements X.

5.5.2 Autocorrelation function

The variation of the signal can be calculated by the autocorrelation function(ACF) of signal, which is defined as:

$$r^A(t, \Delta t) = \frac{E\{(X^A(t)-\mu^A)(X^A(t+\Delta t)-\mu^A)\}}{\sigma_A^2} \quad (6)$$

where μ^A indicates the mean value of measurements X and σ is the standard deviation of X.

5.5.3 Key disagreement rate

The KDR between the key generated by two users can be calculated after the stage quantization based on the formula described in (3). A high KDR can lead to key generation failure.

5.5.4 Randomness test

Randomness is the basis for ensuring the security of the generated keys, hence it can be used as a significant indicator for evaluating a key generation system. The National Institute of Standards and Technology(NIST) statistical test suite is widely applied in testing the randomness of binary sequences created by random number generators(RNGs) and pseudorandom number generators(PRNGs)[41]. This suite consists of 15 tests, which are described in table 2. The p-value can be calculated in each test, and the comparison between the p-value and the significant level α determines whether the randomness requirement is satisfied.

Table 2: Nist statistical test suite[41].

Test	Purpose
Frequency (Monobit) Test	Test the proportion of zeros and ones for the entire binary sequence
Frequency Test within a Block	Test the proportion of zeros and ones within M-bit blocks
Runs Test	Test the total number of runs in the sequence, where a run is an uninterrupted sequence of identical bits
Longest Run of Ones in a Block	Detect the longest run of ones within M-bit blocks
Binary Matrix Rank Test	Test the rank of disjoint sub-matrices of the binary sequence to check the linear dependence among fixed length substrings of the original sequence
Discrete Fourier Transform (Spectral) Test	Detect periodic features based on the peak heights in the Discrete Fourier Transform of the sequence
Non-overlapping Template Matching Test	Detect generators that produce too many occurrences of a given non-periodic pattern
Overlapping Template Matching Test	Test the number of occurrences of pre-specified target strings
Maurer's "Universal Statistical" Test	Detect whether the sequence can be significantly compressed without loss of information or not
Linear Complexity Test	Determine whether the sequences is complex enough to be considered random or not
Serial Test	Test the frequency of all possible overlapping m-bit patterns across the entire sequence

Approximate Entropy Test	Compare the frequency of overlapping blocks of two consecutive/adjacent lengths (m and m+1) against the expected result for a random sequence
Cumulative Sums (Cusum) Test	Test the maximal excursion (from zero) of the random walk defined by the cumulative sum of adjusted (-1, +1) digits in the sequence
Random Excursions Test	Test the number of cycles having exactly K visits in a cumulative sum random walk
Random Excursions Variant Test	Detect deviations from the expected number of visits to various states in the random walk

6. Design

The design of this project is realized by software programming and hardware configuration.

6.1 Hardware configuration

In this project, the main device that are used in the Raspberry Pi. Raspberry Pi is a tiny single-board computer with random-access memory(RAM) processor. Due to its low cost, open design and modularity, Raspberry Pi is widely applied in the experiments of diverse fields, such as education, home automation, industrial automation, etc. Two version 4 Model B Raspberry Pis with 2GB and 4GB RAMs respectively are chosen to implement key generation procedure. In addition, Debian-based Linux distribution is employed in the Raspberry Pis. The built-in USB Ethernet adapter is mainly used in this project, which supports 2.4/5 GHz IEEE 802.11 b/g/n/ac protocols. Nevertheless, the monitor mode is required during channel probing and this mode is not supported by the Raspberry Pi 4B. Therefore, the working environment of the tool Nexmon[42], which is a C-based firmware patching framework for Broadcom/Cypress WiFi chips, should be configured to enable the monitor mode of Raspberry Pis.

Another hardware used is a seven-inch touchscreen display. In order to enhance the operability of the key generation system, the graphic user interface is designed and manipulated by the touchscreen display. Since only one display is introduced in this project, the display is used at the station side and the access point side should be controlled remotely on PC, the software Virtual Network Console(VNC) server is applied to launch the remote desktop. Furthermore, a cable is used at the access point to avoid losing control of the device when it is in monitor mode during channel probing. The relevant devices are shown in Figure 16.

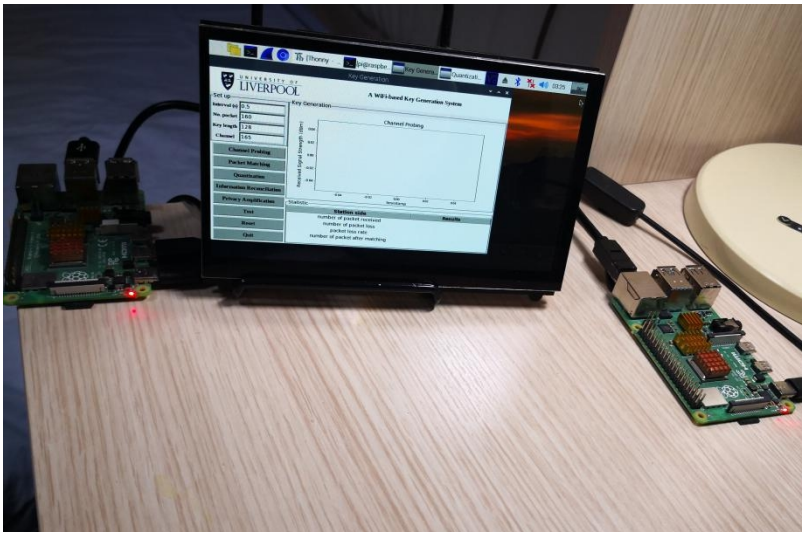


Figure 16: Hardware configuration.

6.2 Software programming

The key generation procedure including channel probing, quantization, information reconciliation, privacy amplification, and the tests including randomness test, AES encryption test, as well as GUI design are all realized by programming using python.

6.2.1 Channel probing

In this stage, the channel parameter RSSI is measured during WiFi wireless communication. Alice works as the station, while Bob works as the access point. In addition, both Alice and Bob should operate in monitor mode to obtain the channel information.

a. Channel measurement

Data packets are transmitted between Alice and Bob, and RSSI are extracted when the packets are received. Firstly, the data frame should be constructed at both Alice and Bob sides via the tool scapy, which is shown as follows:

- Alice (Station):

```
interface='mon0'         # wireless interface on monitor mode
destination=bssid='dc:a6:32:ca:6c:da'    # destination MAC address
ssid='Access piont'
source='dc:a6:32:a2:7a:0c'    # source MAC address
channel=self.channel_value.get()

# construct data frame
def Dataframe(source,channel,ssid,dst,bssid, intfmon):
    essid = Dot11Elt(ID='SSID',info=ssid, len=len(ssid))
    WPS_ID = "\x00\x50\xF2\x04"
    WPS_Elt = Dot11Elt(ID=221,len=9,info="%s\x10\x4a\x00\x01\x10"%
              WPS_ID)
```

```
    dsset = Dot11Elt(ID='DSset',info=chr(channel))
    frame = RadioTap()/Dot11(type=2, subtype=0, addr1=destination,
            addr2=source, addr3=bssid)\
             /Dot11()/essid/WPS_Elt/dsset
     # Update timestamp
     frame.timestamp = current_timestamp()
     # Update sequence number
     frame.SC = next_sc()
```

- Bob (Access point):

```
interface = 'mon0'    # wireless interface on monitor mode
destination = bssid = 'dc:a6:32:a2:7a:0c'# destination MAC address
ssid = 'Access point'
source= 'dc:a6:32:ca:6c:da'    # source MAC address
channel=self.channel_value.get()

# construct data frame
def Dataframe(source,channel,ssid,destination,bssid,interface):
     essid = Dot11Elt(ID='SSID',info=ssid, len=len(ssid))
     dsset = Dot11Elt(ID='DSset',info=chr(channel))
     frame = RadioTap()/Dot11(type=2 ,subtype=0, addr1=destination,
             addr2=source, addr3=bssid)\
             /Dot11()/essid/dsset
     # Update timestamp
     frame.timestamp = current_timestamp()
    # Update sequence number
    frame.SC = next_sc()
```

The frame is constructed by defining different layers based on the OSI model[43]. *RadioTap()* is an additional layer assisting in information transmission. Then the MAC header by *Dot11* which indicates IEEE 802.11 protocols. In the MAC header, the frame type is selected based on Table 3. In addition, the addresses are specified, where addr1 is the MAC address of the receiver, addr2 is the MAC address of the transmitter, and addr3 is the Basic Service Set Identifier which is the same with addr1. After that, the frame function is defined by *Dot11()*. Finally, the SSID, supported speeds, as well as the channel used for transmission, are determined in the *Dot11Elt* layer.

Table 3: Type and subtype identifier[44].

Type	Description	Subtype	Description
00	Management	1000	Beacon
00	Management	0100	Probe request
00	Management	0101	Probe response
00	Management	1011	Authentication
00	Management	1100	Deauthentication
00	Management	0000	Association request
00	Management	0001	Association response
01	Control	1101	ACK
10	Data	0000	Data
11	Reserved	0000-1111	Reserved

After the frame is constructed, the function *sendp()* is applied to send frame from the transmitter to the receiver. The *frame* refers to the constructed frame, *iface* represents the network interface, *verbose* is used to show the detailed information of operation, and *count* means one frame is to be transmitted.

```
1.  sendp(frame, iface=interface, verbose=verbose, count=1)
```

When a frame is sent, devices are switched to receiving mode to capture frames using the function *sniff()*. By assigning *timeout* to be 1 at the Alice side and 3 at the Bob side, packet retransmission is set to happen only at the station side when there are packet loss. In addition, a frame filtering function is created to filter frames by defining the frame type and addresses of the required frame, which is the same at both sides. When the frame is received, the RSSI can be extracted from the *RadioTap* layer, which is stored in the variable *frame[RadioTap].dBm_AntSignal.* Furthermore, the timestamp of receiving the frame is stored together with the RSSI in a text file.

- Alice (Station):

```
1.  sniff(iface=interface, prn=PacketSniff, stop_filter=stopfilter,
        timeout=1)
```

- Bob (Access point):

```
1.  sniff(iface=interface, prn=PacketSniff, stop_filter=stopfilter,
        timeout=3)
```

Then code for packet filtering is shown as follows:

```
1.  def PacketSniff(frame) :
2.    if frame.haslayer(RadioTap) :
3.       if frame.type == 2 and frame.subtype == 0:  # data packet
4.          if frame.addr2.upper() == destination.upper() and
               frame.addr1.upper() == source.upper() :
5.             rcv_time = datetime.now()
```

```
            timestamp.append(rcv_time)
            print("%s Received a data packet from AP %s RSSI %sdBm")
                  % (rcv_time, frame.addr2, frame[RadioTap].
                  dBm_AntSignal)
            rssi.append(int(frame[RadioTap].dBm_AntSignal))
            # extract RSSI
            f = open("STA.txt","a")
            f.write(str(rcv_time)+" "+str(frame[RadioTap].
                    dBm_AntSignal)+"\n")
            f.close()
```

The time interval between two packet exchanges cannot be too short due to security issues of correlated samples. Therefore, 0.5s is chosen as the time interval. In addition, the repetition times of packet exchange should be slightly larger than the required key length, considering the conditions of packet loss during channel probing as well as the dropped bits in the stages of packet matching and quantization.

b. Packet matching

Timestamps are transmitted between two users in this stage to filter the unmatched sample pairs. Firstly, the timestamps are read by Alice from the text file based on the data storage format. The first 26 bits of each line in the text file are extracted and stored in a list, which is shown in the following code:

```
fmt = "%Y-%m-%d %H:%M:%S.%f"   # timestamp format
f1 = open("STA.txt", "r")
f2 = open("matchedsta.txt", "w+")

def Read_datetime(line, fmt) :
# read timestamps with date from a string
    try :
         timestamp = datetime.strptime(line, fmt)
    except ValueError as v :
          if len(v.args) > 0 and v.args[0].startswith(
               'unconverted dat a remains: ') :
             line = line[: - (len(v.args[0]) - 26)]
             timestamp = datetime.strptime(line, fmt)
          else :
             raise
    return timestamp
```

Then the timestamps are encapsulated based on the format of JSON and sent to Bob. The functions *json.dumps() and json.loads()* are used for encoding and decoding of the data. The transmission step is realized by the network programming tool *socket* as shown below:

- Alice (Station):

```
self.host = "192.168.3.13"
self.port = 12345
s = socket.socket() #creant socket object
s.connect((self.host, self.port))

while True :
      buf1 = s.recv(1024)
      if buf1 :
          s.send(le_data)
          time.sleep(2)
          s.send(se_data)
          break
```

- Bob (Access point):

```
self.host = "192.168.3.13" #IP address of host
self.port = 12345
s =socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#creat socket object
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((self.host, self.port))
s.listen(5)
conn, addr = s.accept()
print "connection address: " , addr

 conn.send(b'1')   # signal for receiving
```

Alice is set as the client and Bob is set as the server. By define the IP address of the host, the client can be connected to the server using the functions *s.bind()*, *s.connect* and *s.accept()*. When two devices are connected successfully, Bob will send a signal to Alice to indicate ready for receiving and timestamps exchange is started. After receiving the timestamps, the comparison is carried out by Alice and Bob to filter the uncorrelated RSSI pairs. Through multiple tests, the max range of the time difference for a matched packet pairs is determined to be 0.28s. Based on the time threshold, the mismatched RSSIs are filtered at both Alice and Bob sides. The main part of the code for filtering is shown as follows:

```
 t_diff = t1 - t2
if abs(t_diff.total_seconds() * 1000000) <= 280000 :
      match_times2.append(t2)
      f2.write(linef1 + '\n')
```

6.2.2 Quantization

In this stage, binary keys are obtained by comparing the measured RSSIs and the thresholds. Two types of quantization methods, mean and standard deviation-based method and differential value-based method, can be selected.

a. Mean and standard deviation-based method

In order to improve the key generation, the turning parameter α in equations (1) and (2) is set to be 0, therefore this method is actually mean value-based.

```
# mean-based threshold
average = mean(rs)
for r1 in rs :
    if r1 >= avg :
       quantized_rs.append(1)
    else :
       quantized_rs.append(0)
```

Firstly, the mean value of the measured RSSIs are calculated by the function *mean()*. Then by comparing the RSSIs and the mean value, the decimal values are converted to binary sequence. The RSSI that is larger than or equal to the *average* is appended as 1 in the list *quantized_rs*. Otherwise, it is appended as 0.

b. Differential value-based method

By testing the performance of key generation system, it is found that the RSSI is nearly constant during transmission under static condition, which indicates that the noise caused by hardware is negligible. Thus, the parameter resolution is set to be 0 in this case. The code is shown as follows:

```
#Differential-based quantization
quantized_rs=[]
drop=[]
a=rs[0]
x=0
for i in range(1,len(rs)) :
    if rs[i]>a+x:
        quantized_rs.append(1)
    elif rs[i]<a-x:
         quantized_rs.append(0)
    else:
         quantized_rs.append(2)
         drop.append(i-1)
    a=rs[i]
```

Here, *a* represents the previous RSSI and *x* is the parameter resolution. The RSSI increases compared to the previous one is appended as 1 in the list *quantized_rs*, and the measurement decreases compared to the previous one is appended as 0 in the list *quantized_rs*. Otherwise, the position of the bit that should be dropped is stored in the list *drop*. When quantization is completed, the dropped bits are shared between Alice and Bob based on JSON and the tool *socket*, and then deleted. After that, the key length is shortened to the requirement by the following code:

```
del quantized_rs[key_length:]
if int(math.sqrt(key_length))==math.sqrt(key_length):
    shaped_length=shaped_width=math.sqrt(key_length)
else:
    shaped_length=2**((math.log(key_length,2)-1)/2)
    shaped_width=2**((math.log(key_length,2)+1)/2)
shaped_array = np.array(quantized_rs).reshape((int(shaped_length),
                int(shaped_width)))
```

In order to present the generated keys more directly directly, the 2-D bitmap is designed. Therefore, only 2^n-length or n^2- length keys can be chosen to suit the presentation in bitmap. Through partial code modification, the choice of any key length can be realized, thereby improving its usability in real products. Furthermore, KDR should be calculated to analyze the performance of the key generation system, hence the quantized keys should be transmitted from Alice to Bob. Nevertheless, this process can only be applied in the development phase as transmission is carried out in the public channel.

6.2.3 Information reconciliation

In this stage, errors caused by the half-duplex mode are corrected based on BCH code which is a kind of error correcting code over Galois field GF(q). BCH code can be constructed in terms of *(n, k, t)*, where *n* refers to the codeword length, *k* is the message length, and *t* represents the error correcting capability[45]. For the program design, the python module *bchlib* is used to provide BCH code for this step. The function is realized based on [46]. Firstly, the BCH object should be created by defining the parameters *BCH_POLYNOMIAL* and *BCH_BITS* at both Alice and Bob sides, which is shown below:

```
# create a BCH object
BCH_POLYNOMIAL = 1033
BCH_BITS = int(0.25*len(key))
bch = bchlib.BCH(BCH_POLYNOMIAL, BCH_BITS)
```

The *BCH_POLYNOMIAL* is the polynomial, and *BCH_BITS* is the error correcting capacity. According to the expression of the maximum correction capacity rate of BCH code[2]:

$$\gamma = \frac{t_{max}}{n} = \frac{2^{m-2}-1}{2^m-1} \tag{7}$$

where m is the Galois filed order. The maximum correction capacity rate is approximately 0.25 of the message length. Therefore, *BCH_BITS* is set to be 0.25 of the key length. Then the codeword is generated by the following code:

```
# random data
Data = bytearray(os.urandom(int((bch.n-bch.ecc_bits)/8)))
# encode and make a packet
Ecc = bch.encode(Data)
Packet=Data+Ecc
```

The variables *bch.n* is the maximum codeword size in bits, and *bch.ecc_bits* is the number of bits an error correcting code takes up. When the codeword is obtained, the exclusive-OR operation is conducted to generate the syndrome:

```
# Exclusive OR
for i in range(0,int(len(key)/8)+1):
    if i<=int(len(key)/8)-1:
        decimal=int((key[0+i*8]*2)**7+(key[1+i*8]*2)**6+(key[2+i*8]*2)**
                    5+(key[3+i*8]*2)**4+(key[4+i*8]*2)**3+(key[5+i*8]*2)
                    **2+(key[6+i*8]*2)**1+(key[7+i*8]))
        Packet[i]^=decimal
    else:
        byte=[]
        for j in range(0,len(key)%8):
            byte.append(key[j+i*8])
        for j in range(0,8-len(key)%8):
            byte.append(0)
        decimal=int((byte[0]*2)**7+(byte[1]*2)**6+(byte[2]*2)**5+(byte[3]
                    *2)**4+(byte[4]*2)**3+(byte[5]*2)**2+(byte[6]*2)**1)
        Packet[i]^=decimal
```

The syndrome is then be transmitted from Alice to Bob through public channel. When Bob receives the syndrome, exclusive-OR operation between the syndrome and the keys of Bob is carried out to generate the new codeword. After that, the new codeword is decoded as shown below:

```
# de-packetize
Data, Ecc = Packet[:-bch.ecc_bytes], Packet[-bch.ecc_bytes:]
Bitflips, Correct_data, Correct_ecc= bch.decode(Data,Ecc)
Correct_packet=Correct_data+Correct_ecc
```

Finally, another exclusive-OR is operated between the decoded codeword and syndrome, and the corrected keys are generated. To verify the success of this process, CRC is applied by using the function *binascii.crc32()*. The CRC result is sent from Bob to Alice and compared with result obtained by Alice. If the results are the same, the identical keys are generated.

```
# cyclic redundancy check (CRC)
key_str = ''.join(str(i) for i in key)
```

```
Key = [int(key_str[i:i+8], 2) for i in range(0, len(key_str), 8)]
crc = hex(binascii.crc32(bytes(Key)) & 0xffffffff)
```

6.2.4 Privacy amplification

In this stage, the revealed keys are removed by the hash function. The secure hash algorithm SHA-256 is applied to realize this step. Before starting the hash function, the binary sequence should to be converted to hexadecimal form.

```
hex_key = binascii.a2b_hex(hex_str)
hash_hex = hashlib.sha256(hex_key).hexdigest()
```

6.2.5 Tests

a. AEC encryption test

The AES encryption test is applied to check the sameness of the generated keys. The key length for encryption can be 128, 192, or 256 bits. In addition, five operation modes are supported by AES, including Electronic Code Book(ECB), Cipher-Block Chaining(CBC), Cipher FeedBack(CFB), Output FeedBack(OFB) and CounTer Mode(CTR)[47]. CBC mode is employed in this section and the relevant codes are shown as follows:

- Alice (Station):

```
def encrypt(text):
    cipher = AES.new(self.key, AES.MODE_CBC, self.key)
    length = 16
    count = len(text)
    if(count % length != 0) :
         add = length - (count % length)
    else:
         add = 0
    text = text + ('\0' * add)
    self.ciphertext = cipher.encrypt(text)

    return b2a_hex(self.ciphertext)
```

- Bob (Access point):

```
def decrypt(text):
     cipher = AES.new(self.key, AES.MODE_CBC, self.key)
     plain_text = cipher.decrypt(a2b_hex(text))
     return plain_text.rstrip('\0')
```

Text can be input at Alice side through GUI and divided into several segments. After encryption, the ciphertext is sent to Bob and decrypted.

b. Randomness test

Seven tests of the NIST statistical test suite are applied to check the randomness of the generated keys based on the recommended key size, including monobit test, frequency test within a block, runs test, longest run of ones in a block test, cumulative sums test, serial test and approximate entropy test. The codes for the test suite are provided in [48]. All the tests return the p-value, and when the p-value is larger than 0.01, the keys are considered random.

6.2.6 GUI design

In order to demonstrate the key generation procedure and related results more intuitively, the Graphical User Interface(GUI) is designed. The code design is completed mainly based on two modules, which are *Tkinter and matplotlib*. *Tkinter* contributes the interface layout and operating system such as setting of relevant parameter and controlling of the key generation process, while *matplotlib* focuses on the display of results such as the measured RSSI and bitmaps. The GUI can basically be divided into two type of areas, which are the control areas the display areas.

a. The control areas

The control areas consist of the setup frame, button frame, and the AES test input box.

- **Setup frame and button frame:**

-- Alice (Station):

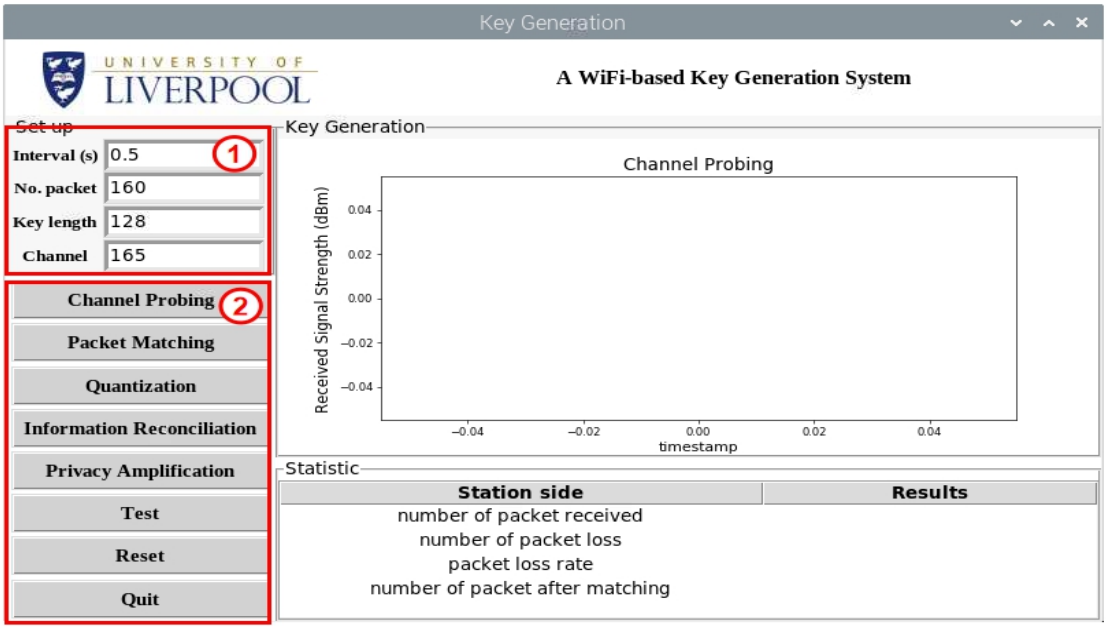


Figure 17: Homepage (Alice).

-- Bob (Access point):

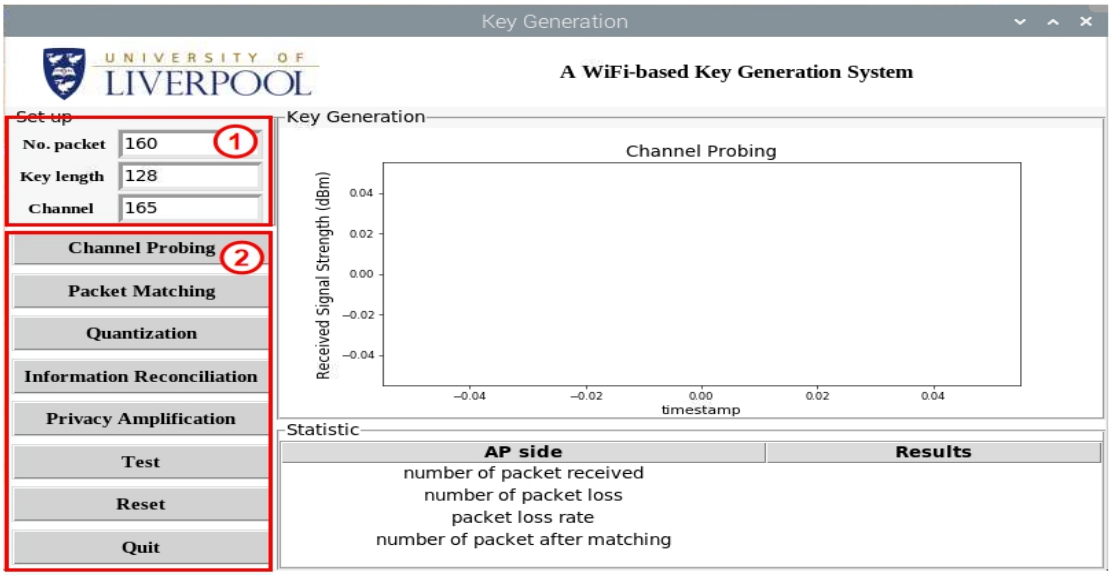


Figure 18: Homepage (Bob).

As can be seen from Figures 17 and 18, two control areas are involved in the homepage. For the setup frame in area 1, both Alice and Bob can edit the number of packets to be transmitted, the key length required and the WiFi channel. At Alice(station) side, an additional input entry is designed to set the time interval between the transmission cycles. The example code for this area is shown below:

```
self.setup_frame = tk.LabelFrame(self.button_frame, text="Set up",
                                 bg="WhiteSmoke", bd=2)
self.setup_frame.rowconfigure(0, weight=1)
self.setup_frame.columnconfigure(0, weight=1)

self.setup_interval = tk.Label(self.setup_frame, text="Interval (s)",
                           fg="black", bg="WhiteSmoke", font=setup_font)
self.setup_interval.grid(row=0, column=0)
self.interval_value = tk.DoubleVar(self.setup_frame)
self.interval_value.set(0.5)
self.interval_entry = tk.Entry(self.setup_frame, textvariable=
                               self.interval_value)
self.interval_entry.grid(row=0, column=1)
```

For the button frame in area 2, eight buttons are designed to realized diverse functions. The first five buttons are used to conduct four stages of the key generation. The *Test* button is used to start the randomness test and AES encryption. By clicking the *Reset* button, users can clear the results and back to the homepage. The *Quit* button is designed to end the system process. The example code for button construction is shown as follows:

```
self.button_frame = tk.Frame(self.master, bg="white")
```

```
2.  self.button_frame.grid(row=1, rowspan=3, column=0, sticky=tk.NSEW)
3.  self.button_frame.rowconfigure(0, weight=1)
4.
5.  self.bu_channel_pr = tk.Button(self.button_frame, text="Channel
                         Probing", font=button_font, width=23, height=1,
                         bg="LightGrey", activeforeground="red",
                         command=self.thread_cp)
6.  self.bu_channel_pr.grid(row=2, column=0)
```

In addition to the button shown in area 2, a selecting button is constructed for quantization method choosing which is shown in Figure 19.

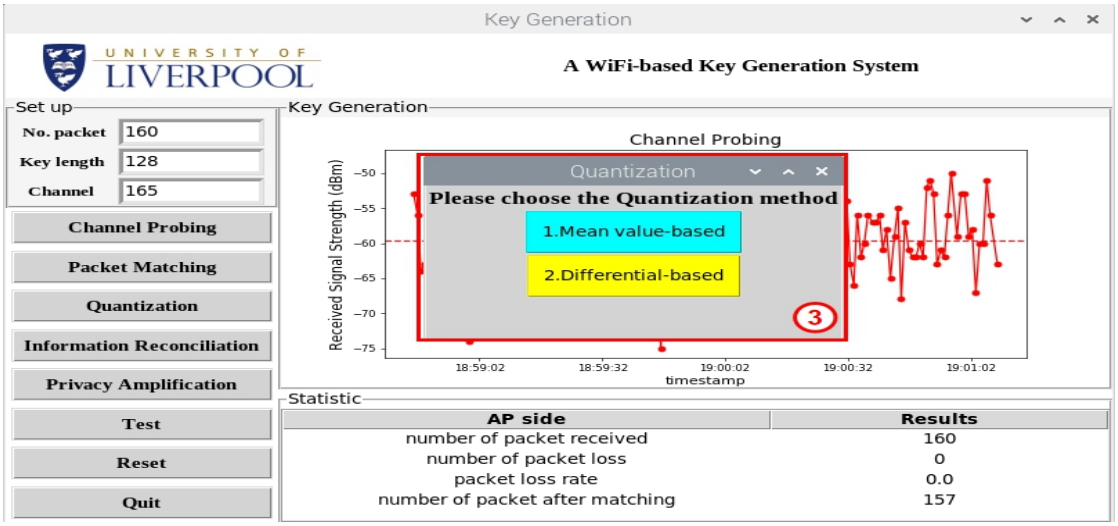
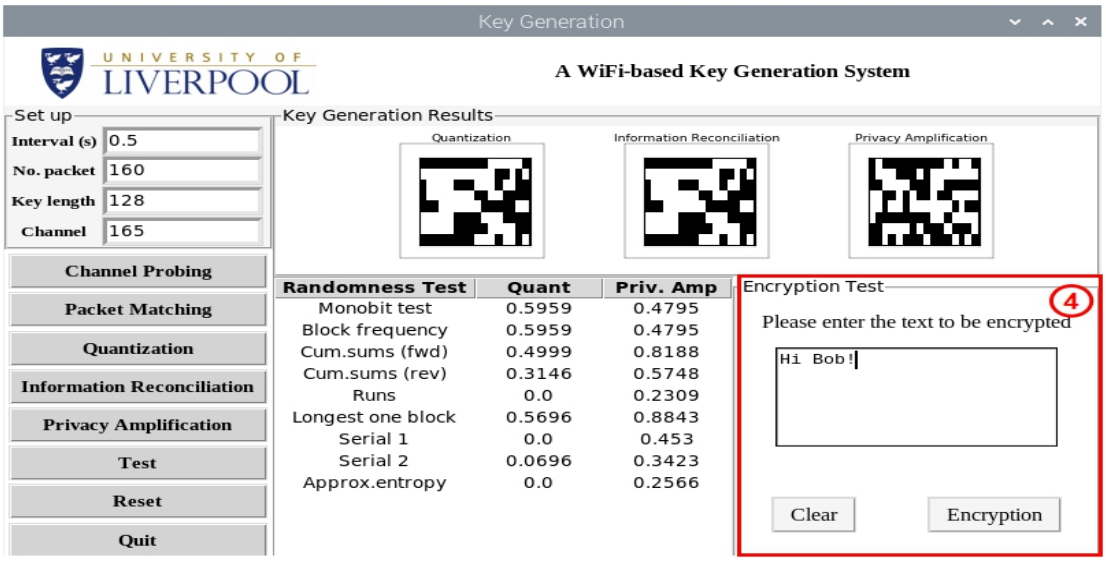


Figure 19: Quantization method selection.

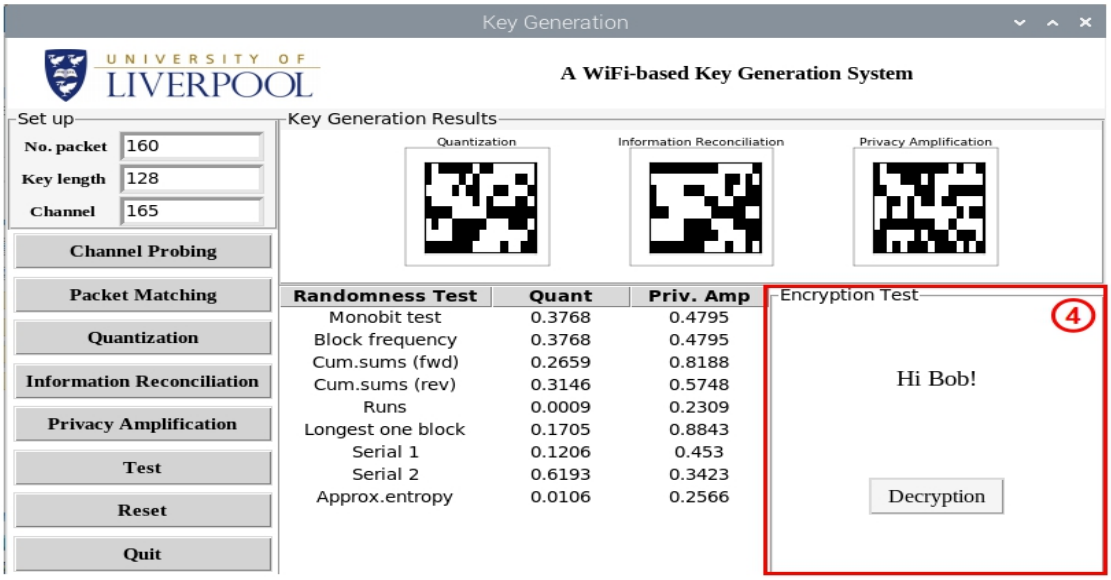
This function is completed based on the function *tk.Radiobutton()*. The relevant code is shown below:

```
1.  tk.Radiobutton(self.tl,text='1.Mean value-based',variable=self.x,
                   value=1,indicatoron=0,bg="Aqua",command=self.
                   Quantization_type).pack(ipadx=10,ipady=10)
2.  tk.Radiobutton(self.tl,text='2.Differential-based',variable=self.x,
                   value=2,indicatoron=0,bg="Yellow",command=self.
                   Quantization_type).pack(ipadx=10,ipady=10)
```

- **AES test input box:**



(a)



(b)

Figure 20: AES test input box, (a)Alice (b)Bob.

For the AES test input box in area 3, the text can be entered at Alice side, and then be encrypted and transmitted to Bob by the button *Encryption*. After that, the text can be decrypted at Bob side by the button *Decryption*. This area is constructed by combining the setup frame and the button frame as described above.

b. The display areas

The design of the display areas are composed of real-time display of measurements, the display of related parameters, as well as the bitmaps,.

- **Real-time display and parameters display:**

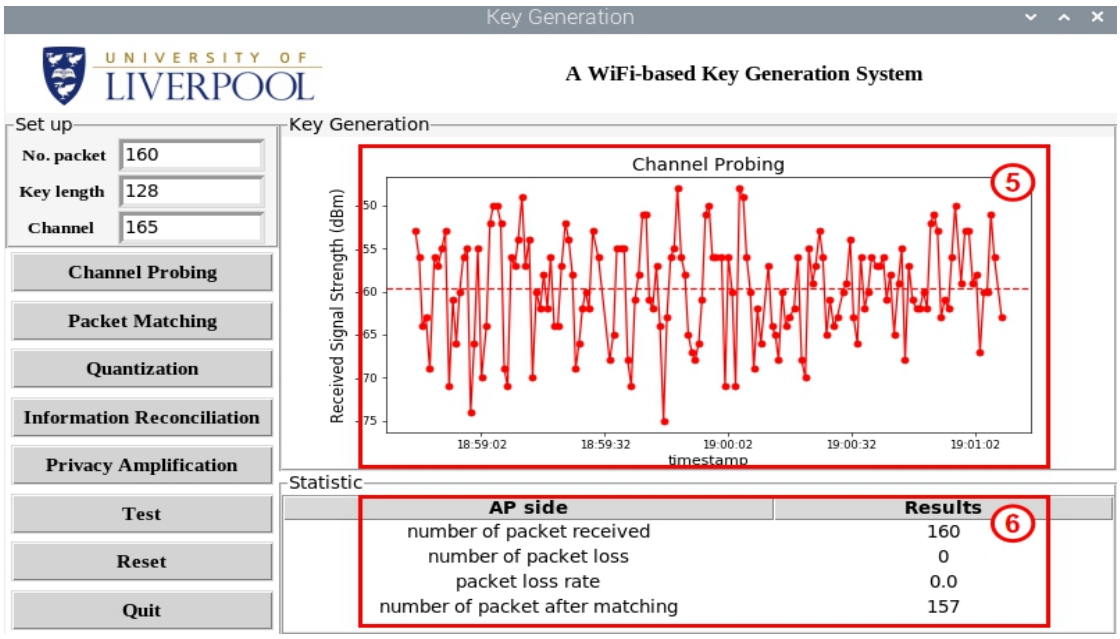


Figure 21: Real-time display and related parameters.

During the step channel measurement, the RSSI is measured and displayed in the area 5 in real-time, which is realized based on the following code:

```
self.animate = animation.FuncAnimation(self.fig, self.realtime_display,
                                       interval=100)

def realtime_display(self, i):
        self.fig_plot.clear()
        self.fig_plot.plot(self.time_array, self.rssi_array,'k',
                           color='red',marker='o')
        self.fig_plot.set_xlabel('timestamp', fontsize=13)
        self.fig_plot.set_ylabel('Received Signal Strength (dBm)',
                                 fontsize=14)
        self.fig_plot.set_title('Channel Probing', size=16)
        self.canvas.draw()

        return self.fig_plot
```

By using the function *animation.FuncAnimation()*, the measured parameter RSSI is plotted in on canvas with timestamp. The time interval between frames of the animation is set to 100ms. Another significant function applied in this design is

threading.Thread(), which is a multi-threaded tool to realize simultaneous signal transmission and RSSI plotting.

```
1.  threading.Thread(target=self.channel_probing).start()
```

The parameters related to each stage of key generation are shown in the statistic frame(area 6). In the stage channel measurement, the number of packet received, number of packet loss, and packet loss rate are involved. After packet matching, the number of packet is undated. In quantization, the bit length of keys and the key disagreement rate are calculated. After information reconciliation, the CRC result is shown. Finally, in the test section, the p-values after quantization and privacy amplification are calculated respectively and shown in the statistic frame. The example code for result presentation is provided as follows:

```
1.  self.frame_num = self.data.insert('', 0, values=
                                      ('number of packet received',))
2.  self.data.item(data_get_index[0], values=('number of packet received',
                                              self.count))
```

- **Bitmaps**

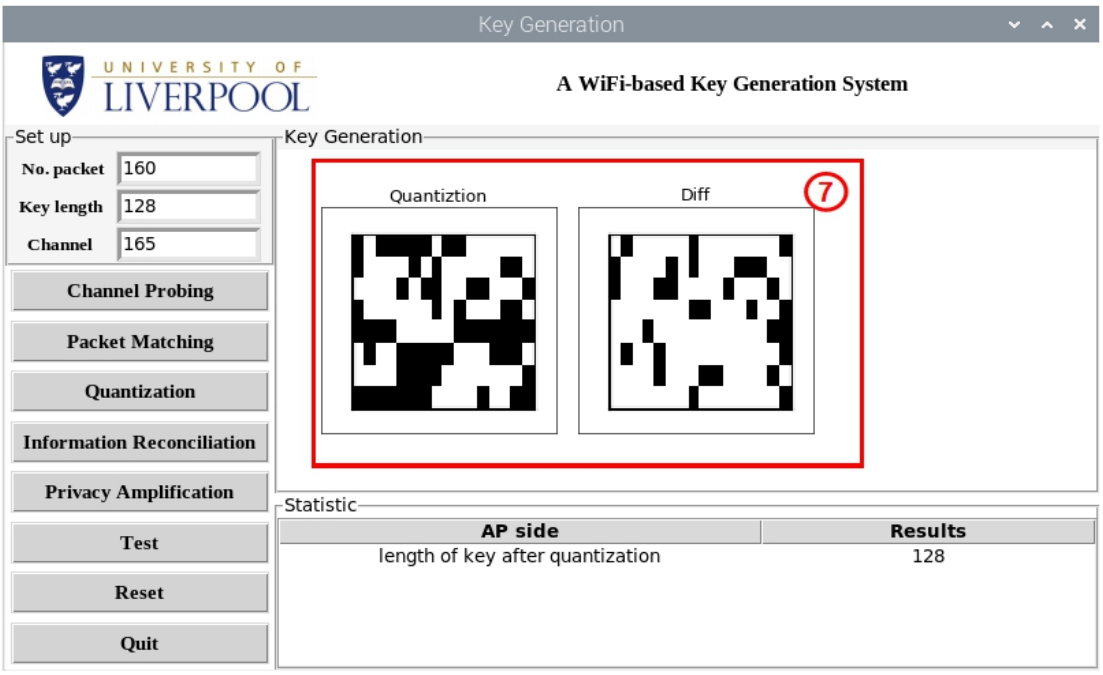


Figure 21: Bitmap of key difference plotted at the stage quantization(Bob).

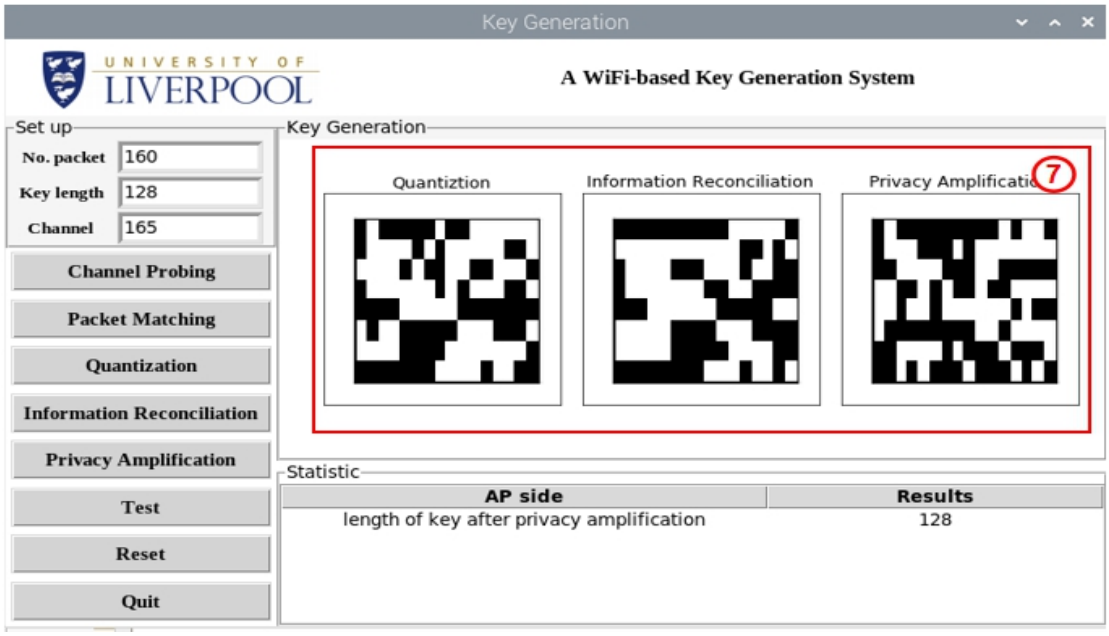


Figure 22: Bitmaps of keys after privacy amplification(Bob).

In order to achieve key visualization, the bitmaps of the keys after quantization, information reconciliation, and privacy amplification, as well as the bitmap of key difference are plotted on canvas(shown in area 7) using the module *matplotlib*. The example code of creating a bitmap is shown below:

```
1.  img = mpimg.imread("quantized.png")
2.  self.fig_plot = self.fig.add_subplot(131)
3.  self.fig_plot.imshow(img)
4.  self.fig_plot.axes.get_xaxis().set_visible(False)
5.  self.fig_plot.axes.get_yaxis().set_visible(False)
6.  self.fig_plot.set_title("Quantiztion", size=15)
7.  self.fig.tight_layout()
8.  self.canvas.draw()
```

The parameters related to each stage of key generation are shown in the statistic frame. In the stage channel measurement, the number of packet received, number of packet loss, and packet loss rate are involved. After packet matching, the number of packet is undated. In quantization, the bit length of keys and the key disagreement rate are calculated. After information reconciliation, the CRC result is shown. Finally, in the test section, the p-values after quantization and privacy amplification are calculated respectively and shown in the statistic frame. The example code for result presentation is provided as follows:

```
3.  self.frame_num = self.data.insert('', 0, values=
                                      ('number of packet received',))
4.  self.data.item(data_get_index[0], values=('number of packet received',
                                              self.count))
```

7. Experimental method

The key generation system is a combination of software and hardware. For the software part, several python modules are required, such as json, socket, scapy, bchlib, hashlib, Tkinter, and matplotlib. The functions of there modules are described as follows:

- json: encodes the parameters to be transmitted in public channel, e.g. timestamps, KDR, and CRC results.
- socket: establishes connection between two users and realize transmission of parameters.
- bchlib: provides BCH(n, k, t) code for the stage information reconciliation
- hashlib: provides hash function for the stage privacy reconciliation
- Tkinter and matplotlib: construct GUI.

For the hardware part, two Raspberry Pis are used to establish the physical layer connection for measurements and operate the designed code. A touchscreen display is used to provide hardware support for the GUI system, as well as present the key generation procedure and results.

8. Results and Calculations

In this simulated WiFi system, Alice is regarded as the station and Bob is considered as the Access point. Key generations based on two quantization methods are performed under three different scenarios, which produces six sets of results.

8.1 Scenarios description

In order to test the capability of this key generation system in various environment, three scenarios are chosen, which are static scenario, object moving scenario and mobile scenario.

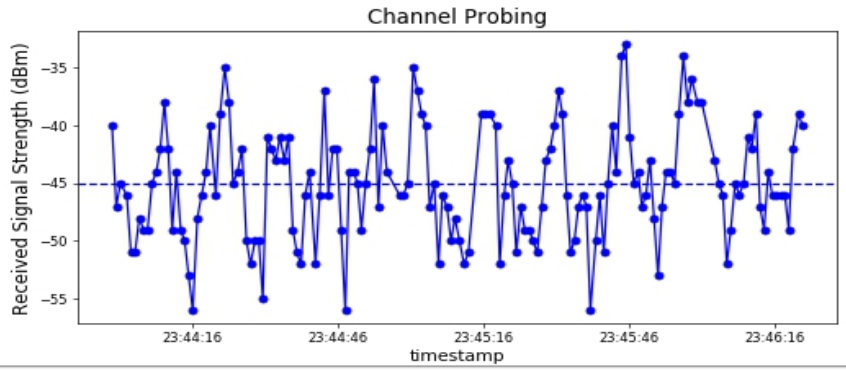
- Static scenario: All the objects in the wireless environment are static.
- Object moving scenario: One object in the wireless environment is moving between Alice and Bob.
- Mobile scenario: Alice keeps moving in the wireless environment to produce a relative displacement with Bob

In the tests under three scenarios, the time interval between transmission cycles is set to 0.5s, the key length is set to 128 bits and the packet number is set to 160.

8.2 Results

8.2.1 Channel probing

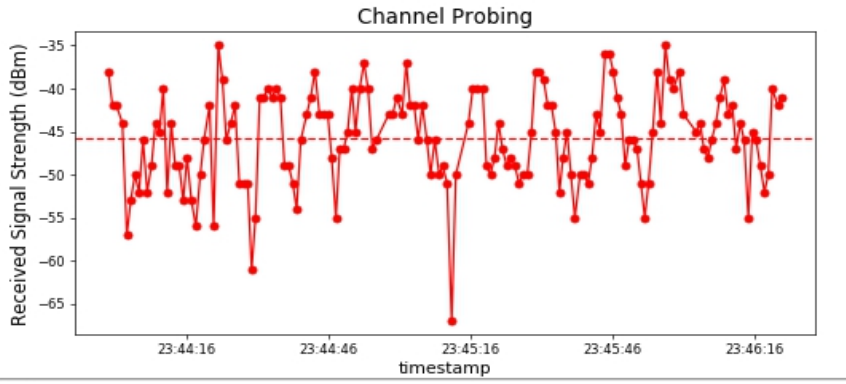
- **Mobile**



Statistic

Station side	Results
number of packet received	157
number of packet loss	3
packet loss rate	0.02
number of packet after matching	157

Figure 23: RSSI and parameters by Alice.



Statistic

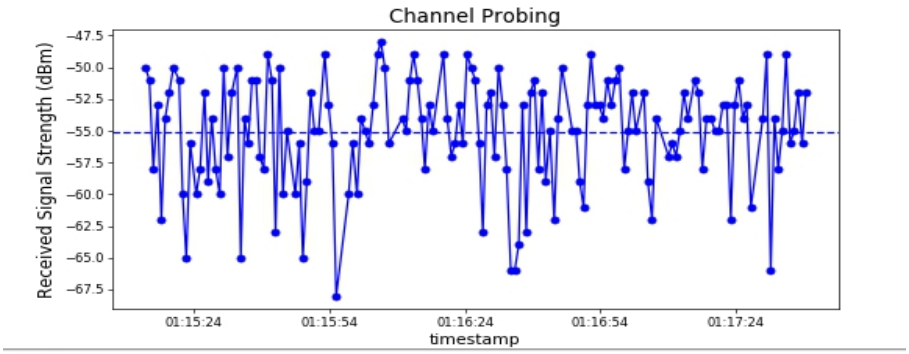
AP side	Results
number of packet received	160
number of packet loss	0
packet loss rate	0.0
number of packet after matching	157

Figure 24: RSSI and parameters by Bob.

According to Figures 23 and 24, the RSSI varied in the range of [-70, -30] under the mobile scenario. There are 157 packets received by Alice and 160 packets received by Bob, which means 3 packets are lost during the transmission from Bob to Alice. By calculation, the time period for channel measurement is approximately 142.25s, the actual time interval between receiving and sending a packet at Alice side is about 0.686s, by subtracting the assigned time interval which is 0.5s, the time to process data packet by Alice is about 0.186s. Then actual time interval at Bob side is about

0.232s. After packet matching, 3 packets are dropped at Bob side based on the maximum time difference for matched packets.

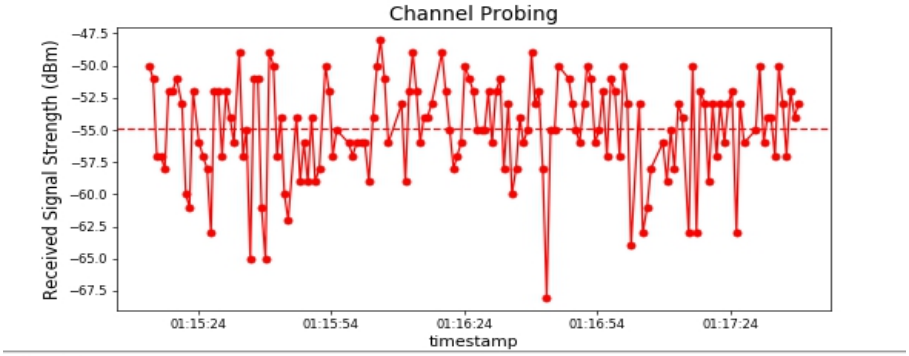
- **Objective**



Statistic

Station side	Results
number of packet received	156
number of packet loss	4
packet loss rate	0.03
number of packet after matching	152

Figure 25: RSSI and parameters by Alice.



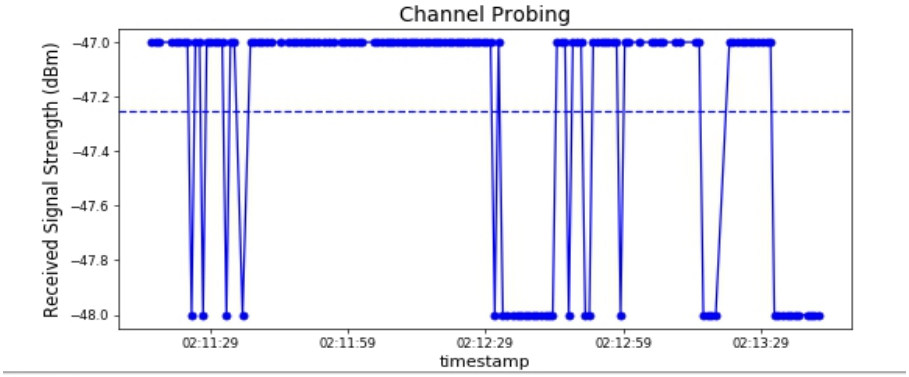
Statistic

AP side	Results
number of packet received	160
number of packet loss	0
packet loss rate	0.0
number of packet after matching	152

Figure 26: RSSI and parameters by Bob.

Based on Figures 25 and 25, the RSSI varied in the range of [-70, -47.5] under the object moving scenario. 156 packets received by Alice and 160 packets received by Bob, hence 4 packets are lost during the transmission from Bob to Alice. After packet matching, 4 pakcet are removed at Alice side and 8 packets are removed at Bob side.

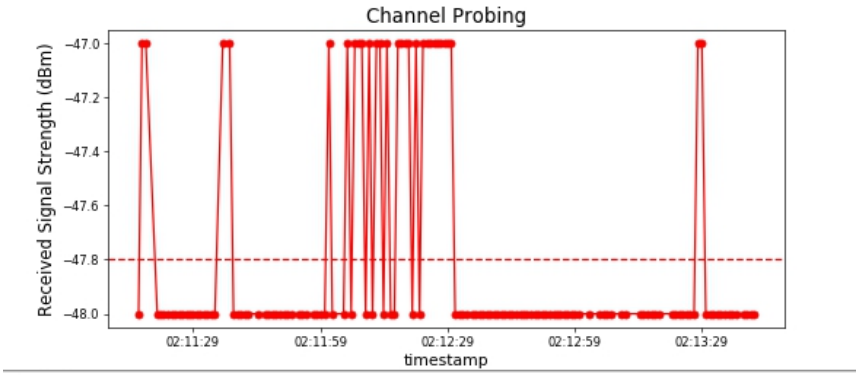
- **Static**



Statistic

Station side	Results
number of packet received	156
number of packet loss	4
packet loss rate	0.03
number of packet after matching	145

Figure 27: RSSI and parameters by Alice.



Statistic

AP side	Results
number of packet received	160
number of packet loss	0
packet loss rate	0.0
number of packet after matching	145

Figure 28: RSSI and parameters by Bob.

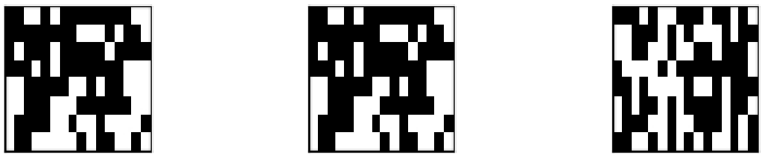
As shown in Figure 27 and 28, the RSSI varied in the range of [-48, -47], which is nearly constant.

8.2.2 Quantization, information reconciliation and privacy amplification

When the RSSI is obtained, two quantization methods are carried out. Then results of quantization, information reconciliation and privacy amplification based on two methods are shown below.

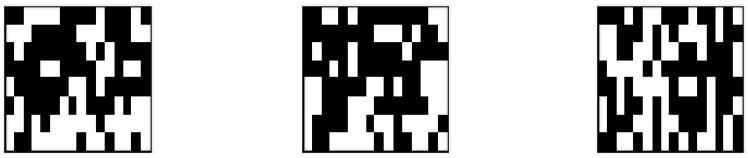
- **Mobile**

--Mean value-based



(a) (b) (c)

Figure 29: Key bitmaps of Alice (a)quantization (b)information reconciliation (c)privacy amplification.



(a) (b) (c)

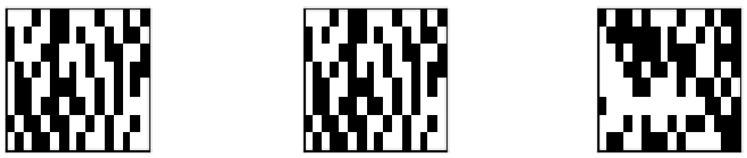
Figure 30: Key bitmaps of Bob (a)quantization (b)information reconciliation (c)privacy amplification.

Table 4: KDR and CRC result.

Device	Key length(bits)	KDR	CRC Result
Alice	128	0.219	0x45cce86L
Bob	128		0x45cce86L

According to Figures 29, 30 and Table 4, at the step quantization, the key length is shortened to 128bits and key bitmaps of two users are different. After information reconciliation, the mismatched bits are corrected. Then the revealed keys are removed through privacy amplification. In addition, the key disagreement rate is calculated as 0.219, which is within the maximum error correcting capacity. The CRC results of Alice and Bob indicate the sameness of the generated keys.

--Differential value-based



(a) (b) (c)

Figure 31: Key bitmaps of Alice (a)quantization (b)information reconciliation (c)privacy amplification.

(a) (b) (c)

Figure 32: Key bitmaps of Bob (a)quantization (b)information reconciliation (c)privacy amplification.

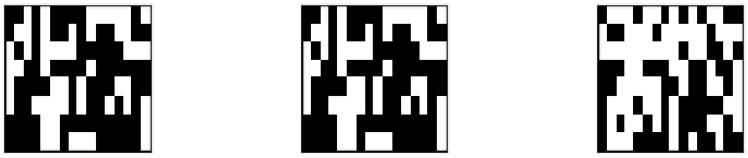
Table 5: KDR and CRC result.

Device	Key length(bits)	KDR	CRC Result
Alice	128	0.18	0x4b151993L
Bob	128		0x4b151993L

Based on Figures 31, 32 and Table 5, the KDR is 0.18 which is also within the error correcting capacity. The identical keys are generated and verified by CRC.

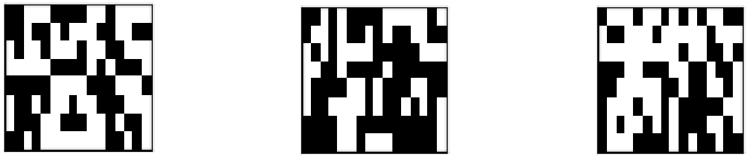
- **Objective**

--Mean value-based



(a) (b) (c)

Figure 33: Key bitmaps of Alice (a)quantization (b)information reconciliation (c)privacy amplification.



(a) (b) (c)

Figure 34: Key bitmaps of Bob (a)quantization (b)information reconciliation (c)privacy amplification.

Table 6: KDR and CRC result.

Device	Key length(bits)	KDR	CRC Result
Alice	128	0.234	0x2e5db497L
Bob	128		0x2e5db497L

According to Figures 33, 33 and Table 5, it can be seen that the same keys are also generated successfully. The KDR increases slightly compared to the KDR obtained in mobile scenario.

--Differential value-based

(a) (b) (c)

Figure 35: Key bitmaps of Alice (a)quantization (b)information reconciliation (c)privacy amplification.

(a) (b) (c)

Figure 36: Key bitmaps of Bob (a)quantization (b)information reconciliation (c)privacy amplification.

Table 7: KDR and CRC result under object.

Device	Key length(bits)	KDR	CRC Result
Alice	128	0.25	0x487c367fL
Bob	128		0x487c367fL

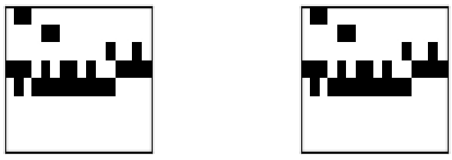
As shown in Figures 35, 36 and Table 7, the key generation system works normally with the differential value-based quantization method. Nevertheless, the KDR increases as well compared to the previous scenario, which just satisfy the error correcting capacity.

- **Static**

--Mean value-based

(a) (b)

Figure 37: Key bitmaps of Alice (a)quantization (b)information reconciliation



(a) (b)

Figure 38: Key bitmaps of Bob (a)quantization (b)information reconciliation

Table 8: KDR and CRC result.

Device	Key length(bits)	KDR	CRC Result
Alice	128	0.602	0x410f4ab1L
Bob	128		0xac618118L

As can be seen in Figure 37, 38 and Table 8, different keys are generated and therefore cannot be verified by CRC.

--Differential value-based

In this method, no binary key is generated since the constant bits are dropped and the left bits are insufficient to construct keys.

8.2.3 Randomness tests

The randomness tests on the keys after quantization and keys after privacy amplification are carried out only in the mobile and object moving scenarios, since no key is generated in the static scenario. The related results are shown in appendix B. It can be seen from the tables that after privacy amplification, all the p-values are larger than 0.01. Therefore, the generated keys can be considered random.

9. Discussion

At this stage, the designing and building of the key generation system are basically finished. The discussion is conducted based on the expected goal and related technical specifications.

9.1 Performance analysis

The performance analysis can be divided into two parts, which are the performance analysis of the mean value-based and differential value-based quantization method, as well as the performance analysis of the key generation in different scenarios. The randomness test results and key disagreement rate are regarded as the important indicators during analyzing.

9.1.1 Performance of two quantization methods

According to the tables of randomness test results, it can be seen that the p-values of the differential value-based quantization method are higher compared with the results of another method, which means the keys with higher randomness can be generated. As described in the Theory section, randomness is the basis to ensure the security of generated keys. Therefore, the key generation system with differential value-based method is securer. Nevertheless, due to the differential-based mechanism, the constant bits are all dropped, which requires more packets to be transmitted. In other words, longer time is spent to generate a fixed-length key. Thus, this method has a lower key generation rate. For the mean-value based quantization, since this method is simpler, less code space is required which may be more suitable for low-cost IoT devices. In addition, this method is more usable in the small-scale indoor

environment. Hence, the selection of quantization method should be based on the application environment and power requirement of devices.

9.1.2 Performance under different scenarios

As can be seen from the RSSI measured under different scenarios, the variation range of measurements decreases from the mobile scenario to the object moving scenario, and become nearly constant in the static scenario. On the contrary, the KDR increase as the wireless environment becomes more stable. In the mobile scenario, the wireless channel varies due to the change of transmission distance between Alice and Bob. The longer the distance between two users, the lower RSSI is received. In the object moving scenario, the public wireless channel is interfered by the object and therefore produces randomness. However, in the static scenario, the variation in the wireless channel is caused by the noise of hardware ,which is not correlated between Alice side and Bob side. Therefore, identical keys are generated under mobile scenario and object moving scenario, and cannot be generated under static scenario.

9.2 Contribution, limitations and improvements

In this project, the key generation system with two quantization methods is built which can help secure the wireless network. It is realized based on the feature of the wireless channel and therefore requires less programming compared with the classical cryptosystem. Thus, this system is more suitable for low-cost IoT devices. In addition, GUI is involved in this system, improving maneuverability.

Nevertheless, there are still limitations existing in this system. The channel parameter used for key generation, RSSI, is a coarse-grained parameter. Hence, limited information about the channel is presented, which reduces the key generating efficiency. In addition, the CSMA/CD mechanism is not designed in the tool *scapy*, which will lead to high packet loss rate when multiple devices is connected to the wireless network.

To improve the key generation system, firstly, the channel parameter RSSI can be replaced by channel state information(CSI) so that multiple keys can be generated in one transmission. Secondly, the Acknowledgment frame can be applied to shorten the time interval between receiving and sending to improve the key generating efficiency further. Furthermore, the CSMA/CD function and channel adaptive system can be designed so that this system can be applied in more scenarios.

10. Conclusions

A WiFi-based key generation system is designed by python programming on two Raspberry Pis and its performance under different scenarios is analyzed in this project. The purpose of the project is to produce secure keys for legitimate users based on the unpredictable feature of

wireless channel, and therefore provide protection for the physical layer. Four stages are involved in the key generation procedure, which are channel probing, quantization, information reconciliation, and privacy amplification. The measurements of channel information is conducted during channel probing. Then, in the step quantization, binary keys are converted from the measured RSSI. After that, mismatched bits between key pairs are corrected in information reconciliation. Finally, revealed information in the pubic channel is eliminated through privacy amplification. To check the randomness and sameness of generated keys, NIST Statistical Test Suite and AES encryption test are applied. In addition, GUI is designed to demonstrate the key generation system. Through experiments, it is found that this system can be applied under mobile and object moving scenarios. The KGR increases as the channel variation decreases. In the static scenario, the system cannot work properly. To sum up, this project is finished successfully and the knowledge about IEEE 802.11 standards and OSI model is learned. Furthermore, the programming skills with Python and Linux system are enhanced. Nevertheless, there are still several aspects required to be improved in the future, such as the key generation efficiency, high KDR under static scenario, etc.

References:

[1] Zhang, J.; Li, G.; Marshall, A.; Hu, A.; Hanzo, L. A New Frontier for IoT Security Emerging From Three Decades of Key Generation Relying on Wireless Channels. IEEE Access, 2020, vol. 8, pp.138406-138446.

[2] Zhang, J.; Duong, T. Q.; Marshall, A. Key Generation From Wireless Channels: A Review. IEEE Access, 2016, vol. 4, pp. 614-626.

[3] Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A survey on enabling technologies, protocols, and applications. IEEE Commun. Surveys Tuts., 2015, vol. 17, no. 4, pp. 2347–2376.

[4] Saxena, P. OSI Reference Model – A Seven Layered Architecture of OSI Model. International Journal of Research. 2014, 1, pp. 2348-6848.

[5] Zimmermann, H. OS1 Reference Model-The IS0 Model of Architecture for Open Systems Interconnection. IEEE , 1980, vol. 28, no. 4, pp. 425-432.

[6] Zhang, J. ; Duong, T. Q.;Marshall, A. Securing wireless communications of the internet of things from the physical layer, an overview. Entropy. 2017, vol. 19, p.420.

[7] Zou, Y.; Zhu, J.; Wang, X.; Hanzo, L. A Survey on Wireless Security: Technical Challenges, Recent Advances, and Future Trends. IEEE, 2016, vol. 104, no. 9, pp. 1727-1765.

[8] W. Stallings, Cryptography and Network Security: Principles and Practice, 5th ed. Upper Saddle River, NJ, USA: Pearson, 2017.

[9] Advanced Encryption Standard, Federal Information Processing Standards Publication Standard FIPS PUB 197, 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf. (accessed on 2nd April, 2021.)

[10] R. Ahlswede and I. Csiszar, ‘‘Common randomness in information theory and cryptography. I. Secret sharing,’’ IEEE Trans. Inf. Theory, 1993, vol. 39, no. 4, pp. 1121–1132.

[11] U. M. Maurer, ‘‘Secret key agreement by public discussion from common information,’’ IEEE Trans. 1993, Inf. Theory, vol. 39, no. 3, pp. 733–742.

[12] Y. Wei, K. Zeng, and P. Mohapatra, ‘‘Adaptive wireless channel probing for shared key generation based on PID controller,’’ IEEE Trans.Mobile Comput., 2013, vol. 12, no. 9, pp. 1842–1852.

[13] H. Liu, Y. Wang, J. Yang, and Y. Chen, ‘‘Fast and practical secret key extraction by exploiting channel response,’’ in Proc. 32nd IEEE Int. Conf. Comput. Commun. (INFOCOM), Turin, Italy, Apr. 2013, pp. 3048–3056.

[14] S. N. Premnath, P. L. Gowda, S. K. Kasera, N. Patwari, and R. Ricci, ‘‘Secret key extraction using Bluetooth wireless signal strength measurements,’’ in Proc. 11th Annu. IEEE Int. Conf. Sens., Commun., Netw. (SECON), 2014, pp. 293–301.

[15] H. Yang, W. Lee, H. Lee. IoT Smart Home Adoption: The Importance of Proper Level Automation. Hindawi., 2018, vol. 2018, DOI: 10.1155/2018/6464036.

[16] E. Park, A. P. Pobil, S. J. Kwon. The Role of Internet of Things (IoT) in Smart Cities: Technology Roadmap-oriented Approaches. Sustainability. 2018, vol. 10, no. 5, p.1388.

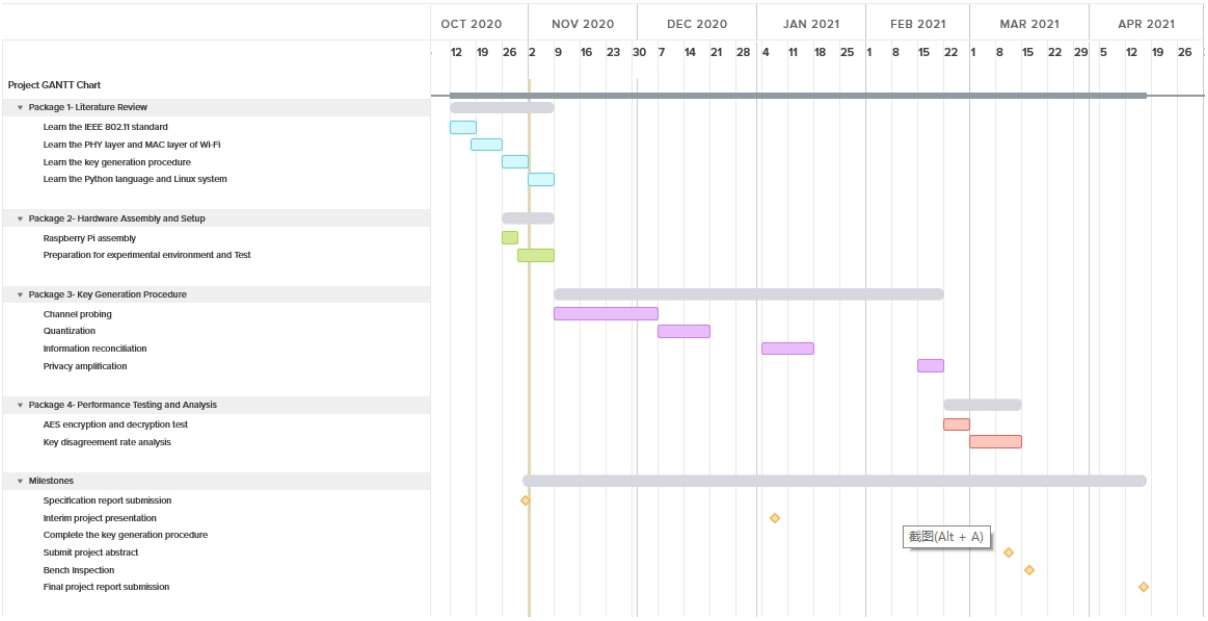
[17]]J. Manyika et al., Disruptive Technologies: Advances that Will Transform Life, Business, and the Global Economy. San Francisco, CA, USA: McKinsey Global Instit., 2013.
[18] K. Shafique, B. A. Khawaja, F. Sabir, S. Qazi, M. Mustaqim. Internet of Things (IoT) for Next-Generation Smart Systems: A Review of Current Challenges, Future Trends and Prospects for Emerging 5G-IoT Scenarios. IEEE Access, 2020, vol. 8, pp. 23022-23040.
[19] K. Lounis, M. Zulkernine. Attacks and Defenses in Short-Range Wireless Technologies for IoT. IEEE Access, 2020, vol. 8, pp. 88892-888932.
[20] Rahbari, H.; Krunz, M. Secrecy beyond encryption: Obfuscating transmission signatures in wireless communications. IEEE Commun. Mag., 2015, vol. 53, no. 12, pp. 54–60
[21] Cheng, C.; Lu, R.; Petzoldt, A.; Takagi, T. Securing the Internet of Things in a Quantum World. IEEE Commun. Mag., 2017, vol. 55, no. 2, pp. 116–120.
[22] Kaspersky(2018). New IoT-malware grew three-fold in H1 2018 [online]. Available: https://www.kaspersky.com/about/press-releases/2018_new-iot-malware-grew-three-fold-in-h1-2018. (accessed 4th April 2021).
[23] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, Y. Zhou. “Understanding the Mirai Botnet.” in Proc. 26th USENIX Secu. Symp., 2017, pp. 1093-1110.
[24] Zhang, J.; Marshall, A.; Woods, R.; Duong, T.Q. Design of an OFDM Physical Layer Encryption Scheme. IEEE Trans. Veh. Technol., 2017, vol. 66, no. 3, pp. 2114–2127.
[25] S. T. Ali, V. Sivaraman and D. Ostry. Eliminating Reconciliation Cost in Secret Key Generation for Body-Worn Health Monitoring Devices. IEEE Trans. Mob. Comp., 2014, vol. 13, no. 12, pp. 2763-2776.
[26] The University of Liverpool. Lecture 11: Mobile Radio Propagation Models[Online]. Avaliable: https://vital.liv.ac.uk/bbcswebdav/pid-2342768-dt-content-rid-15960715_1/courses/ELEC377-202021/Lecture11_PropagationModelsV2.pdf. (accessed 8th April 2021).
[27] G. R. Hiertz, D. Denteneer, L. Stibor, Y. Zang, X. P. Costa, B. Walke. The IEEE 802.11 Universe. IEEE Commu. Maga., 2010, vol. 48, no. 1, pp. 62-70.
[28] M. S. Gast. “802.11 Wireless Networks: The Definitive Guide, Second Edition”. Sebastopol, USA: O'Reilly Media, 2005, ISBN: 0-596-10052-3.
[29] B. P. Crow, I. Widjaja, J. G. Kim, P. T. Sakai. IEEE 802.11 Wireless Local Area Networks. IEEE Commu. Maga., 1997, vol. 35, no. 9, pp. 116-126.
[30] I. Tinnirello, G. Bianchi, Y. Xiao. Refinements on IEEE 802.11 Distributed Coordination Function Modeling Approaches. IEEE TRANS. VEHIC. TECH, 2010, vol. 59, no. 3, pp. 1055-1067.
[31] J. Zhang; R. Woods, T. Q. Duong, A. Marshall, Y. Ding, Y. Huang, Q. Xu. Experimental Study on Key Generation for Physical Layer Security in Wireless Communications. IEEE Access, 2016, vol. 4, pp. 4464-4477.
[32] J. Zhang; R. Woods, T. Q. Duong, A. Marshall, Y. Ding. Experimental Study on Channel Reciprocity in Wireless Key Generation. IEEE, 2016.
[33] A. Goldsmith, Wireless Communications. Cambridge, U.K.: Cambridge Univ. Press, 2005.

[34] X. He, H. Dai, W. Shen, P. Ning. Is link signature dependable for wireless security? IEEE, 2013, pp. 200-204.
[35] D. Chen, Z. Qin, X. Mao, P. Yang, Z. Qin, R. Wang. SmokeGrenade: An Efficient Key Generation Protocol With Artificial Interference. IEEE Trans. Inf. Foren. Secur., 2013, vol. 8, no. 11, pp. 1731-1745.
[36] T. S. Rappaport, Wireless Communications: Principles and Practice, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2001.
[37] B. Zan, M. Gruteser, and F. Hu. Key agreement algorithms for vehicular communication networks based on reciprocity and diversity theorems. IEEE Trans. Veh. Technol., 2013, vol. 62, no. 8, pp. 4020–4027.
[38] P. Yadav and R. Ramanathan, "Dynamic key generation using single threshold multiple level quantization scheme for secure wireless communication," in 2017 Interna. Confe. Wirele. Commun, Signal Pro. Net. (WiSPNET), 2017, pp. 34–38.
[39] C. Huth, R. Guillaume, T. Strohm, P. Duplys, I. A. Samuel, T. Güneysu. Information reconciliation schemes in physical-layer security: A survey. Comput. Netwo., 2016, vol. 109, pp. 84–104.
[40] S. Jana, S. N. Premnath, M. Clark, S. K. Kasera, N. Patwari, S. V. Krishnamurthy, "On the effectiveness of secret key extraction from wireless signal strength in real environments," in Proce. 15th Annu. Intern. Confer. Mob. Compu. Netw. (MobiCom), 2009, pp. 321–332.
[41] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, S. Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Natl. Inst. Stand. Technol. Spec. Publ. 800-22rev1a, April, 2010.
[42] Nexmon (Sep, 2017). [Online]. Available: https://github.com/seemoo-lab/nexmon#supported-devices.
[43] Scapy Documentation (Apr, 2021). [Online]. Available: https://scapy.readthedocs.io/_/downloads/en/latest/pdf/.
[44] P. Plarek (Sep, 2016). Generating WiFi communication in Scapy tool[Online]. Available: https://research.securitum.com/generating-wifi-communication-in-scapy-tool/.
[45] P. Treeviriyanupab, P. Sangwongngam, K. Sripimanwat and O. Sangaroon. "BCH-Based Slepian-Wolf Coding with Feedback Syndrome Decoding for Quantum Key Reconciliation", in 2012 9th Inter. Conf. Elec. Eng./Electr., Comp., Tele. Infor. Tech., May 2012.
[46] J. Kent (2019). python-bchlib[Online]. Available: https://github.com/jkent/python-bchlib.
[47] Legrandin (Nov, 2020). PyCryptodome Documentation[Online]. Available: https://pycryptodome.readthedocs.io/_/downloads/en/latest/pdf/.
[48] S. K. Ang (Dec, 2019). NIST Randomness Testsuit[Online]. Available: https://github.com/stevenang/randomness_testsuite.

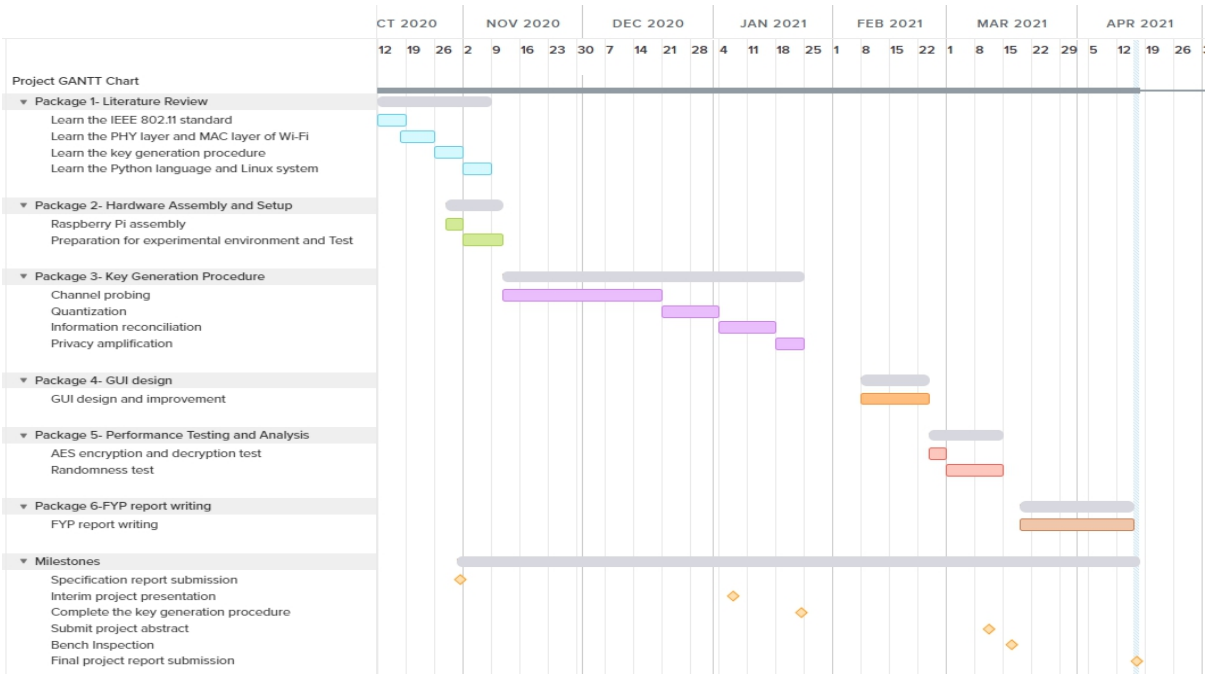
Appendices:

A. Gantt Chart

A.1 Original Gantt Chart



A.2 Revised Gantt Chart



B. Randomness test results

B.1 Mobile

- **Mean value-based**

Table 1: Randomness test results of Alice.

Randomness Test	Quantization	Privacy Amplification
Monobit test	0.0339	0.2159
Block frequency	0.0339	0.2159
Cum.sums(fwd)	1.0018	0.3146
Cum.sums(rev)	0.0678	0.4314
Runs	0.0313	0.0353
Longest one block	0.0624	0.4962
Serial 1	0.0027	0.5064
Serial 2	0.3638	0.4335
Approx. entropy	0.0405	0.2189

Table 2: Randomness test results of Bob.

Randomness Test	Quantization	Privacy Amplification
Monobit test	0.2159	0.2159
Block frequency	0.2159	0.2159
Cum.sums(fwd)	0.0094	0.3146
Cum.sums(rev)	0.2232	0.4314
Runs	0.0985	0.0351
Longest one block	0.0744	0.4962
Serial 1	0.2202	0.5064
Serial 2	0.4335	0.4335
Approx. entropy	0.0995	0.2189

- **Differential value-based**

Table 3: Randomness test results of Alice.

Randomness Test	Quantization	Privacy Amplification
Monobit test	0.4795	0.7237
Block frequency	0.4795	0.7237
Cum.sums(fwd)	0.8920	0.8188
Cum.sums(rev)	0.8920	0.4999
Runs	0.0	0.2114
Longest one block	0.0362	0.5352
Serial 1	0.0006	0.1883
Serial 2	0.0358	0.0545
Approx. entropy	0.0	0.4681

Table 4: Randomness test results of Bob.

Randomness Test	Quantization	Privacy Amplification
Monobit test	0.5959	0.7237
Block frequency	0.5959	0.7237
Cum.sums(fwd)	0.9493	0.8188
Cum.sums(rev)	0.9493	0.4999
Runs	0.0042	0.2114
Longest one block	0.0499	0.5352
Serial 1	0.0275	0.1883
Serial 2	0.4335	0.0545
Approx. entropy	0.0019	0.4681

B.2 Objective

- **Mean value-based**

Table 5: Randomness test results of Alice.

Randomness Test	Quantization	Privacy Amplification
Monobit test	0.0339	0.2888
Block frequency	0.0339	0.2888
Cum.sums(fwd)	0.0543	0.1542
Cum.sums(rev)	0.0431	0.4999
Runs	0.7485	0.5252
Longest one block	0.0835	0.5657
Serial 1	0.3696	0.3696
Serial 2	0.3638	0.1637
Approx. entropy	0.2346	0.8171

Table 6: Randomness test results of Bob.

Randomness Test	Quantization	Privacy Amplification
Monobit test	0.4795	0.2888
Block frequency	0.4795	0.2888
Cum.sums(fwd)	0.6548	0.1542
Cum.sums(rev)	0.6548	0.4999
Runs	0.5057	0.5252
Longest one block	0.3082	0.5657
Serial 1	0.5801	0.3696
Serial 2	0.2830	0.1627
Approx. entropy	0.4974	0.8171

- **Differential value-based**

Table 7: Randomness test results of Alice.

Randomness Test	Quantization	Privacy Amplification
Monobit test	0.7237	0.8597
Block frequency	0.7237	0.8587
Cum.sums(fwd)	0.7275	0.6548
Cum.sums(rev)	0.9842	0.8188
Runs	0.0001	0.3752
Longest one block	0.0402	0.9404
Serial 1	0.0093	0.2092
Serial 2	0.2163	0.1687
Approx. entropy	0.0002	0.8851

Table 8: Randomness test results of Bob.

Randomness Test	Quantization	Privacy Amplification
Monobit test	0.4795	0.8597
Block frequency	0.4795	0.8597
Cum.sums(fwd)	0.4999	0.6548
Cum.sums(rev)	0.8920	0.8188
Runs	0.0012	0.3752
Longest one block	0.2029	0.9404
Serial 1	0.0362	0.2092
Serial 2	0.0501	0.1687
Approx. entropy	0.0092	0.8851

C. Code

C.1 Code for Alice

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import logging
import os
import time
from datetime import datetime
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *
import socket
import json
from statistics import mean, StatisticsError

import bchlib
import binascii
import numpy as np
```

```
import hashlib
import math

import Tkinter as tk
import tkFont as tf
from PIL import Image
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, Navigation
Toolbar2TkAgg
from matplotlib.backend_bases import key_press_handler
from matplotlib.figure import Figure
import matplotlib.image as mpimg
import matplotlib.animation as animation
import threading
import ttk
import tkMessageBox as msg
import tkSimpleDialog as simpledialog

import FrequencyTest as FT
import CumulativeSum as CS
import RunTest as RT
import Serial
import ApproximateEntropy as AE
import Spectral as ST
import Non_overlapping as NO

import sys
from Crypto.Cipher import AES
from binascii import b2a_hex


class Key_Generation :

    def __init__(self, master) :
        self.master = master
        self.master.title("Key Generation")
        self.master.geometry("800x480+0+0")

        self.master.rowconfigure(0, weight=2)
        self.master.rowconfigure(1, weight=60)
        self.master.rowconfigure(2, weight=170)
        self.master.rowconfigure(3, weight=10)
        self.master.columnconfigure(0, weight=1)
        self.master.columnconfigure(1, weight=35)

        # logo and title frame
        self.title_frame = tk.Frame(self.master, bg="white", bd=1)
        self.title_frame.grid(row=0, column=0, columnspan=3, sticky=tk.NSEW)
        self.title_frame.rowconfigure(0, weight=1)
```

```
67.         self.title_frame.columnconfigure(0, weight=1)
68.         self.title_frame.columnconfigure(1, weight=6)
69.
70.         titlefont = tf.Font(family="Times New Roman", size=12, weight=tf.BO
    LD)
71.         self.rcv_timeitle = tk.Label(self.title_frame, fg="black", bg="whit
    e",text="A WiFi-based Key Generation System", font=titlefont)
72.         self.rcv_timeitle.grid(row=0, column=1, columnspan=2, sticky=tk.NSE
    W)
73.
74.         self.logo = tk.PhotoImage(file="logo.gif")
75.         self.sch_logo = tk.Label(self.title_frame, bg="white", image=self.l
    ogo)
76.         self.sch_logo.grid(row=0, column=0, sticky=tk.NSEW)
77.
78.         # button frame
79.         self.button_frame = tk.Frame(self.master, bg="white")
80.         self.button_frame.grid(row=1, rowspan=3, column=0, sticky=tk.NSEW)
81.         self.button_frame.rowconfigure(0, weight=1)
82.         self.button_frame.rowconfigure(1, weight=1)
83.         self.button_frame.rowconfigure(2, weight=1)
84.         self.button_frame.rowconfigure(3, weight=1)
85.         self.button_frame.rowconfigure(4, weight=1)
86.         self.button_frame.rowconfigure(5, weight=1)
87.         self.button_frame.rowconfigure(6, weight=1)
88.         self.button_frame.rowconfigure(7, weight=1)
89.         self.button_frame.rowconfigure(8, weight=1)
90.         self.button_frame.rowconfigure(9, weight=1)
91.         self.button_frame.columnconfigure(0, weight=1)
92.
93.         # set up entry
94.         self.setup_frame = tk.LabelFrame(self.button_frame, text="Set up",
    bg="WhiteSmoke", bd=2)
95.         self.setup_frame.grid(row=0, column=0, sticky=tk.NSEW)
96.         self.setup_frame.rowconfigure(0, weight=1)
97.         self.setup_frame.rowconfigure(1, weight=1)
98.         self.setup_frame.rowconfigure(2, weight=1)
99.         self.setup_frame.rowconfigure(3, weight=1)
100.        self.setup_frame.columnconfigure(0, weight=1)
101.        self.setup_frame.columnconfigure(1, weight=1)
102.        self.setup_frame.columnconfigure(2, weight=1)
103.        self.setup_frame.columnconfigure(3, weight=1)
104.
105.        setup_font = tf.Font(family="Times New Roman", size=10, weight=tf.B
    OLD)
106.        self.setup_interval = tk.Label(self.setup_frame, text="Interval (s)
    ", fg="black", bg="WhiteSmoke", font=setup_font)
107.        self.setup_interval.grid(row=0, column=0)
108.        self.interval_value = tk.DoubleVar(self.setup_frame)
109.        self.interval_value.set(0.5)
```

```
110.         self.interval_entry = tk.Entry(self.setup_frame, textvariable=self.
    interval_value, bd=3, width=12)
111.         self.interval_entry.grid(row=0, column=1)
112.
113.         self.setup_number = tk.Label(self.setup_frame, text="No. packet", f
    g="black", bg="WhiteSmoke", font=setup_font)
114.         self.setup_number.grid(row=1, column=0)
115.         self.number_value = tk.IntVar(self.setup_frame)
116.         self.number_value.set(160)
117.         self.number_entry = tk.Entry(self.setup_frame, textvariable=self.nu
    mber_value, bd=3, width=12)
118.         self.number_entry.grid(row=1, column=1)
119.
120.         self.setup_channel = tk.Label(self.setup_frame, text="Channel", fg=
    "black", bg="WhiteSmoke", font=setup_font)
121.         self.setup_channel.grid(row=3, column=0)
122.         self.channel_value = tk.IntVar(self.setup_frame)
123.         self.channel_value.set(165)
124.         self.channel_entry = tk.Entry(self.setup_frame, textvariable=self.c
    hannel_value, bd=3, width=12)
125.         self.channel_entry.grid(row=3, column=1)
126.
127.         self.setup_key = tk.Label(self.setup_frame, text="Key length", fg="
    black", bg="WhiteSmoke", font=setup_font)
128.         self.setup_key.grid(row=2, column=0)
129.         self.key_value = tk.IntVar(self.setup_frame)
130.         self.key_value.set(128)
131.         self.key_entry = tk.Entry(self.setup_frame, textvariable=self.key_v
    alue, bd=3, width=12)
132.         self.key_entry.grid(row=2, column=1)
133.
134.         # button control
135.         button_font = tf.Font(family="Times New Roman", size=11, weight=tf.
    BOLD)
136.         self.bu_channel_pr = tk.Button(self.button_frame, text="Channel Pro
    bing", font=button_font, width=23, height=1, bg="LightGrey", activeforegrou
    nd="red", command=self.thread_cp)
137.         self.bu_channel_pr.grid(row=2, column=0)
138.         self.bu_packet_ma = tk.Button(self.button_frame, text="Packet Match
    ing", font=button_font, width=23, height=1, bg="LightGrey", activeforegroun
    d="red", command=self.Packet_Matching)
139.         self.bu_packet_ma.grid(row=3, column=0)
140.         self.bu_quanti = tk.Button(self.button_frame, text="Quantization",
    font=button_font, width=23, height=1, bg="LightGrey", activeforeground="red
    ", command=self.Quantization_choose)
141.         self.bu_quanti.grid(row=4, column=0)
142.         self.bu_info_recon = tk.Button(self.button_frame, text="Information
     Reconciliation", font=button_font, width=23, height=1, bg="LightGrey", act
    iveforeground="red", command=self.Info_Recon)
143.         self.bu_info_recon.grid(row=5, column=0)
```

```
144.         self.bu_privacy_amp = tk.Button(self.button_frame, text="Privacy Am
     plification", font=button_font, width=23, height=1, bg="LightGrey", activef
     oreground="red", command=self.Privacy_Amp)
145.         self.bu_privacy_amp.grid(row=6, column=0)
146.         self.test = tk.Button(self.button_frame, text="Test", font=button_f
     ont, width=23, height=1, bg="LightGrey", activeforeground="red", command=se
     lf.Test)
147.         self.test.grid(row=7, column=0)
148.         self.bu_reset = tk.Button(self.button_frame, text="Reset", font=but
     ton_font, width=23, height=1, bg="LightGrey", activeforeground='red', comma
     nd=self.Reset)
149.         self.bu_reset.grid(row=8, column=0)
150.         self.bu_reset = tk.Button(self.button_frame, text="Quit", font=butt
     on_font, width=23, height=1, bg="LightGrey", activeforeground='red', comman
     d=self.Quit)
151.         self.bu_reset.grid(row=9, column=0)
152.
153.
154.         # main display
155.         self.main_frame = tk.LabelFrame(self.master, text='Key Generation',
      bg="WhiteSmoke")
156.         self.main_frame.grid(row=1, rowspan=2, column=1, sticky=tk.NSEW)
157.
158.         self.fig = Figure(figsize=(5, 2), dpi=60)
159.         self.fig_plot = self.fig.add_subplot(111)
160.         self.canvas = FigureCanvasTkAgg(self.fig, self.main_frame)
161.         self.canvas.draw()
162.         self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=1)
163.
164.
165.         # Data
166.         self.data_frame = tk.LabelFrame(self.master, bg="White", text="Stat
     istic")
167.         self.data_frame.grid(row=3, column=1, sticky=tk.NSEW)
168.         self.data_frame.rowconfigure(0, weight=1)
169.         self.data_frame.columnconfigure(0, weight=1)
170.
171.         self.data = ttk.Treeview(self.data_frame, show="headings")
172.         self.data.grid(row=0, column=0, sticky=tk.NSEW)
173.         self.data['columns'] = ('STA side', 'Results')
174.         self.data['height'] = 4
175.         self.data.column("STA side", width=120, anchor='center')
176.         self.data.column("Results", width=10, anchor='center')
177.         self.data.heading("STA side", text="Station side")
178.         self.data.heading("Results", text="Results")
179.
180.         self.frame_num = self.data.insert('', 0, values=('number of packet
     received',))
181.         self.frame_loss = self.data.insert('', 1, values=('number of packet
      loss',))
```

```
182.         self.frame_lossrate = self.data.insert('', 2, values=('packet loss
     rate',))
183.         self.matched_num = self.data.insert('', 3, values=('number of packe
     t after matching',))
184.
185.         self.animate = animation.FuncAnimation(self.fig, self.realtime_disp
     lay, interval=100)
186.
187.         self.timestamp=[]
188.         self.rssi=[]
189.         self.time_array=([])
190.         self.rssi_array=([])
191.         self.matched_rssi = []
192.
193.
194.     def channel_probing(self) :
195.         interface='mon0'         # wireless interface on monitor mode
196.         destination=bssid='dc:a6:32:ca:6c:da'    # destination MAC address
197.         ssid='Access Point'
198.         source='dc:a6:32:a2:7a:0c'    # source MAC address
199.         self.flag=0
200.         self.sc=-1
201.         self.bootime=time.time()
202.         self.count=0
203.
204.         # construct data frame
205.         def Dataframe(source,channel,ssid,dst,bssid, interface):
206.             verbose = 0
207.             essid = Dot11Elt(ID='SSID',info=ssid, len=len(ssid))
208.             WPS_ID = "\x00\x50\xF2\x04"
209.             WPS_Elt = Dot11Elt(ID=221,len=9,info="%s\x10\x4a\x00\x01\x10" %
      WPS_ID)
210.             dsset = Dot11Elt(ID='DSset',info=chr(channel))
211.             frame =  RadioTap()/Dot11(type=2,subtype=0,addr1=destination,ad
     dr2=source,addr3=bssid)\
212.             /Dot11()/essid/WPS_Elt/dsset
213.             # Update timestamp
214.             frame.timestamp = current_timestamp()
215.             ## Update sequence number
216.             frame.SC = next_sc()
217.             if verbose: frame.show()
218.             try:
219.                 sendp(frame,iface=interface,count=1,verbose=verbose)
220.                 rcv_time = datetime.now()
221.                 print('%s Send data packet to ESSID=[%s],BSSID=%s') % (rcv_
     time, ssid, bssid)
222.             except:
223.                 raise
224.
225.
226.         def current_timestamp():
```

```
227.             return (time.time() - self.bootime) * 1000000
228.
229.         def next_sc():
230.             self.sc = (self.sc + 1) % 4096
231.             return self.sc * 16  # Fragment number -> right 4 bits
232.
233.         def PacketSniff(frame) :
234.             if frame.haslayer(RadioTap) :
235.                 if frame.type == 2 and frame.subtype == 0:  # data packet
236.                     if frame.addr2.upper() == destination.upper() and frame.
    addr1.upper() == source.upper() :
237.                         self.count += 1
238.                         self.flag = 1
239.                         rcv_time = datetime.now()
240.                         timestamp.append(rcv_time)
241.                         print("%s Received a data packet from AP %s RSSI %s
    dBm") % (rcv_time, frame.addr2, frame[RadioTap].dBm_AntSignal)
242.                         rssi.append(int(frame[RadioTap].dBm_AntSignal))# ex
    tract RSSI
243.                         f = open("STA.txt","a")
244.                         f.write(str(rcv_time)+" "+str(frame[RadioTap].dBm_A
    ntSignal)+"\n")
245.                         f.close()
246.
247.
248.         def stopfilter(x) :
249.
250.             if self.flag == 1 :
251.                 self.flag = 0
252.                 return True
253.             else :
254.                 return False
255.
256.
257.         i = 0
258.         f_sta = open("STA.txt","w")
259.         f_sta.close()
260.         times = self.number_value.get()
261.         interval = self.interval_value.get()
262.         channel=self.channel_value.get()
263.         keylength=self.key_value.get()
264.         if keylength>times:
265.                 msg.showinfo("Information", "Please select a key length les
    s than the number of packets")
266.                 return
267.         else:
268.           timestamp=[]
269.           rssi=[]
270.           while i < times :
271.               Dataframe(source, channel, ssid, destination, bssid, interfac
    e)
```

```
272.                 sniff(iface=interface, prn = PacketSniff, stop_filter=stopfil
    ter, timeout=1)
273.                 self.time_array=np.array(timestamp)
274.                 self.rssi_array=np.array(rssi)
275.                 time.sleep(interval)
276.                 i += 1
277.
278.
279.         self.animate.event_source.stop()
280.         loss=times-self.count
281.         lossrate=round(float(loss)/times,2)
282.         data_get_index = self.data.get_children()
283.         self.data.item(data_get_index[0], values=('number of packet receive
    d', self.count))
284.         self.data.item(data_get_index[1], values=('number of packet loss',
    loss))
285.         self.data.item(data_get_index[2], values=('packet loss rate', lossr
    ate))
286.         return self.time_array, self.rssi_array
287.
288.
289.
290.     def Packet_Matching(self) :
291.
292.         fmt = "%Y-%m-%d %H:%M:%S.%f"    # timestamp format
293.         f1 = open("STA.txt", "r")
294.         f2 = open("matchedsta.txt", "w+")
295.
296.         def Read_datetime(line, fmt) :
297.         # read timestamps with date from a string
298.             try :
299.                 timestamp = datetime.strptime(line, fmt)
300.             except ValueError as v :
301.                 if len(v.args) > 0 and v.args[0].startswith('unconverted da
    ta remains: ') :
302.                     line = line[: - (len(v.args[0]) - 26)]
303.                     timestamp = datetime.strptime(line, fmt)
304.                 else :
305.                     raise
306.             return timestamp
307.
308.         timest = []
309.
310.         for linef1 in f1 :
311.             linef1 = linef1.strip('\n')
312.             timest.append(Read_datetime(linef1, fmt))
313.
314.         data = ' '.join(map(str, timest))
315.
316.         se_data = json.dumps(data).encode()
317.         le_data = json.dumps(len(se_data)).encode()
```

```
318.
319.         self.host = "192.168.3.13"
320.         self.port = 12345
321.         s = socket.socket() #creant socket object
322.         s.connect((self.host, self.port))
323.
324.         while True :
325.             buf1 = s.recv(1024)
326.             if buf1 :
327.                 s.send(le_data)
328.                 time.sleep(2)
329.                 s.send(se_data)
330.                 break
331.
332.         print("Finish sending")
333.         print("start receiving")
334.
335.         recvd = ""
336.
337.         length = json.loads(s.recv(1024).decode())
338.
339.         num_index = int(length/1024) + 2
340.
341.         for i in range(0, num_index) :
342.             recvd = recvd + s.recv(1024).decode()
343.
344.         match_data = json.loads(recvd)
345.         print("Finish receiving")
346.         s.close
347.
348.         span = 2
349.         words = match_data.split(" ")
350.
351.         fmt_ts = [" ".join(words[i : i+span]) for i in range(0, len(words),
    span)]
352.
353.         match_ts = []
354.
355.         for i in range(0, len(fmt_ts)) :
356.             match_ts.append(datetime.strptime(fmt_ts[i], fmt))
357.
358.
359.         f1.seek(0)
360.         for linef1 in f1 :
361.             linef1 = linef1.strip('\n')
362.             t1 = Read_datetime(linef1, fmt)
363.             for t2 in match_ts :
364.                 t_diff = t1 - t2
365.                 if abs(t_diff.total_seconds()) == 0 :
366.                     f2.write(linef1 +'\n')
367.
```

```
368.         print("Finish matching")
369. 
370.         f1.close
371. 
372.         f2.seek(0)
373.         matched_ts_sta=[]
374.         matched_rssi=[]
375.         for linef2 in f2 :
376.             linef2 = linef2.strip('\n')
377.             columns = linef2.split(" ")
378.             matched_ts_sta.append(Read_datetime(linef2, fmt))
379.             if len(columns) > 2 :
380.                 matched_rssi.append(int(columns[2]))
381.         self.matched_rssi=matched_rssi
382.         f2.close
383.         print(matched_ts_sta, matched_rssi)
384.         ave=np.mean(matched_rssi)
385.         matched_ts_array=np.array(matched_ts_sta)
386.         matched_rssi_array=np.array(matched_rssi)
387. 
388.         self.fig_plot.clear()
389.         self.fig_plot.plot(matched_ts_array, matched_rssi_array,'k',color='
    blue',marker='o')
390.         self.fig_plot.axhline(y=ave,color='blue',ls='--')
391.         self.fig_plot.set_xlabel('timestamp', fontsize=13)
392.         self.fig_plot.set_ylabel('Received Signal Strength (dBm)', fontsize
    =14)
393.         self.fig_plot.set_title('Channel Probing', size=16)
394.         self.canvas.draw()
395. 
396.         data_get_index = self.data.get_children()
397.         self.data.item(data_get_index[3], values=('number of packet after m
    atching', len(matched_rssi)))
398. 
399.         key_length=self.key_value.get()
400.         if len(matched_rssi)<key_length:
401.             msg.showinfo("Information", "Packet number is not enough. Pleas
    e reset and restart channel probing.")
402. 
403.         return self.fig_plot,matched_ts_array, matched_rssi_array
404. 
405.     def Quantization_choose(self):
406.         self.tl=tk.Toplevel()
407.         self.tl.geometry("300x150+300+200")
408.         self.tl.title('Quantization')
409.         self.tl["background"]="LightGrey"
410.         tlfont = tf.Font(family="Times New Roman", size=13, weight=tf.BOLD)
411.         w=tk.Label(self.tl,text="Please choose the Quantization method",bg=
    "LightGrey",font=tlfont).pack()
412. 
413.         self.x = tk.IntVar()
```

```
414.
415.         self.result=0
416.         tk.Radiobutton(self.tl,text='1.Mean value-
    based',variable=self.x,value=1,indicatoron=0,bg="Aqua",command=self.Quantiz
    ation_type).pack(ipadx=10,ipady=10)
417.         tk.Radiobutton(self.tl,text='2.Differential-
    based',variable=self.x,value=2,indicatoron=0,bg="Yellow",command=self.Quant
    ization_type).pack(ipadx=10,ipady=10)
418.
419.     def Quantization_type(self):
420.         self.result=self.x.get()
421.         self.tl.destroy()
422.         self.Quantization()
423.
424.
425.     def Quantization(self) :
426.
427.         self.fig_plot.clear()
428.         self.fig_plot.axis('off')
429.
430.         rs = []
431.         quantized_rs = []
432.         #self.result = simpledialog.askinteger(title = 'Quantization type',
    prompt='Please choose the quantization method: (1.Mean value-
    based. 2.Differential-based.)',initialvalue = '')
433.
434.         try :
435.             rs = list(map(int, self.matched_rssi))
436.         except StatisticsError, e :
437.             msg.showinfo("Information", "0 packet collected. Please reset a
    nd restart Channel Probing.")
438.         if self.result==1:
439.
440.             # mean-based threshold
441.             average = mean(rs)
442.             for r1 in rs :
443.                 if r1 >= average :
444.                     quantized_rs.append(1)
445.                 else :
446.                     quantized_rs.append(0)
447.
448.             print "length of quantized rssi is :", len(quantized_rs)
449.             print "quantized rssi is :", quantized_rs
450.         else:
451.             #Differential-based quantization
452.             quantized_rs=[]
453.             drop=[]
454.             a=rs[0]
455.             x=0
456.             for i in range(1,len(rs)) :
457.                 if rs[i]>a+x:
```

```
458.                     quantized_rs.append(1)
459.                 elif rs[i]<a-x:
460.                     quantized_rs.append(0)
461.                 else:
462.                     quantized_rs.append(2)
463.                     drop.append(i-1)
464.                 a=rs[i]
465.             print "length of quantized rssi is :", len(quantized_rs)
466.
467.             drop_str=','.join(str(v) for v in drop)
468.             print(drop_str)
469.             se_drop_str=json.dumps(drop_str).encode()
470.
471.             time.sleep(1)
472.             s = socket.socket()
473.             self.port=1234
474.
475.             s.connect((self.host, self.port))
476.
477.             while True :
478.                 buf2 = s.recv(1024)
479.
480.                 if buf2 :
481.                     s.send(se_drop_str)
482.                     break
483.
484.             print("Finish sending")
485.             print("start receiving")
486.             drop_recvd=''
487.             drop_recvd= drop_recvd+s.recv(1024).decode()
488.             print("Finish receiving")
489.
490.             s.close
491.             drop_recvd=eval(drop_recvd.encode('utf-8'))
492.             drop_recvd=drop_recvd.split(",")
493.
494.             #merge the dropped bits
495.             drop.extend(list(map(int,drop_recvd)))
496.             drop_mer=list(set(drop))
497.             drop_mer.sort()
498.             print "dropped bits: ", drop_mer
499.
500.             print(quantized_rs)
501.             #delete the dropped bits
502.             drop_mer.reverse()
503.             for d in drop_mer:
504.                 del quantized_rs[d]
505.             print(quantized_rs)
506.             print(len(quantized_rs))
507.
508.
```

```
509.         #reshape the key length
510.         key_length=self.key_value.get()
511.         if key_length>len(quantized_rs):
512.             msg.showinfo("Information", "Key length is not long enough.Plea
    se reset and restart channel probing")
513.             return
514.
515.         del quantized_rs[key_length:]
516.         if int(math.sqrt(key_length))==math.sqrt(key_length):
517.             shaped_length=shaped_width=math.sqrt(key_length)
518.         else:
519.             shaped_length=2**((math.log(key_length,2)-1)/2)
520.             shaped_width=2**((math.log(key_length,2)+1)/2)
521.
522.         shaped_array = np.array(quantized_rs).reshape((int(shaped_length),
    int(shaped_width)))
523.
524.
525.         print "shaped key length:", len(quantized_rs)
526.
527.
528.         #calculate the KDR
529.         self.quantized_rs = quantized_rs
530.         self.quantized_rs_str = ''.join(str(i) for i in quantized_rs)
531.
532.         se_data = json.dumps(self.quantized_rs_str).encode()
533.         le_data = json.dumps(len(se_data)).encode()
534.
535.         s = socket.socket()
536.         self.port=12345
537.
538.         s.connect((self.host, self.port))
539.
540.         while True :
541.             buf1 = s.recv(1024)
542.
543.             if buf1 :
544.                 s.send(le_data)
545.                 time.sleep(2)
546.                 s.send(se_data)
547.                 break
548.
549.         print("Finish sending")
550.         print("start receiving")
551.         self.kdr_value = s.recv(1024)
552.         print "KDR: ", self.kdr_value
553.         s.close
554.
555.         quan_fig = plt.figure(figsize=(1, 1))
556.         quan_fig_plot = plt.subplot(1, 1, 1)
557.         quan_fig_plot.axes.get_xaxis().set_visible(False)
```

```
558.         quan_fig_plot.axes.get_yaxis().set_visible(False)
559.         quan_fig_plot.imshow(shaped_array, extent=[0, 100, 0, 1], aspect=10
    0, cmap=plt.cm.gray_r)
560.         quan_fig.savefig("quantized.png", dpi=quan_fig.dpi)
561.
562.         img = mpimg.imread("quantized.png")
563.         self.fig_plot = self.fig.add_subplot(131)
564.         self.fig_plot.imshow(img)
565.         self.fig_plot.axes.get_xaxis().set_visible(False)
566.         self.fig_plot.axes.get_yaxis().set_visible(False)
567.         self.fig_plot.set_title("Quantiztion", size=15)
568.         self.fig.tight_layout()
569.         self.canvas.draw()
570.
571.         data_get_index = self.data.get_children()
572.         self.data.item(data_get_index[0], values=('length of key after quan
    tization', len(quantized_rs)))
573.         self.data.item(data_get_index[1], values=('key disagreement rate',
    self.kdr_value))
574.         self.data.delete(data_get_index[2])
575.         self.data.delete(data_get_index[3])
576.
577.     def Info_Recon(self) :      # information reconciliation
578.
579.         try :
580.             key = self.quantized_rs
581.             key=np.array(key)
582.         except AttributeError, e:
583.             msg.showinfo("Information", "No quantized key. Please reset and
     restart Channel Probing.")
584.
585.         # create a BCH object
586.         BCH_POLYNOMIAL = 1033
587.         BCH_BITS = int(0.25*len(key))
588.         bch = bchlib.BCH(BCH_POLYNOMIAL, BCH_BITS)
589.
590.         # random data
591.         Data = bytearray(os.urandom(int((bch.n-bch.ecc_bits)/8)))
592.         # encode and make a packet
593.         Ecc = bch.encode(Data)
594.         Packet=Data+Ecc
595.
596.         # Exclusive OR
597.         for i in range(0,int(len(key)/8)+1):
598.             if i<=int(len(key)/8)-1:
599.                 decimal=int((key[0+i*8]*2)**7+(key[1+i*8]*2)**6+(key[2+i*8]
    *2)**5+(key[3+i*8]*2)**4+(key[4+i*8]*2)**3+(key[5+i*8]*2)**2+(key[6+i*8]*2)
    **1+(key[7+i*8]))
600.                 Packet[i]^=decimal
601.             else:
602.                 byte=[]
```

```
603.                 for j in range(0,len(key)%8):
604.                     byte.append(key[j+i*8])
605.                 for j in range(0,8-len(key)%8):
606.                     byte.append(0)
607.                 decimal=int((byte[0]*2)**7+(byte[1]*2)**6+(byte[2]*2)**5+(b
    yte[3]*2)**4+(byte[4]*2)**3+(byte[5]*2)**2+(byte[6]*2)**1)
608.                 Packet[i]^=decimal
609.
610.         s=[]
611.
612.         for i in range(0,len(Packet)):
613.             s.append(Packet[i])
614.
615.         json_string=json.dumps(s).encode()
616.         length=json.dumps(len(json_string)).encode()
617.
618.         self.port=12346
619.         s = socket.socket()
620.         s.connect((self.host, self.port))
621.
622.
623.         flag=True
624.
625.         while flag:
626.             buf1=s.recv(1024)  # ready for sending
627.             if buf1:
628.                 s.send(length)
629.                 time.sleep(2)
630.                 s.send(json_string)
631.                 break;
632.
633.         print(key)
634.
635.         recvd = s.recv(1024).decode()
636.         crc_ap = json.loads(recvd)
637.         print "CRC AP: ", crc_ap
638.
639.
640.         # cyclic redundancy check (CRC)
641.         key_str = ''.join(str(i) for i in key)
642.         Key = [int(key_str[i:i+8], 2) for i in range(0, len(key_str), 8)]
643.         crc = hex(binascii.crc32(bytes(Key)) & 0xffffffff)
644.         print "CRC STA: ", crc
645.
646.         #crc_ap_int = int(crc_ap, 16)
647.         #crc_sta_int = int(crc_key, 16)
648.
649.         #if crc_ap_int == crc_sta_int :
650.         if crc == crc_ap :
651.             print "CRC are the same"
652.             s.send("y")
```

```
653.         else :
654.             s.send("n")
655.             msg.showinfo("Warning", "Cannot correct all keys. Please reset
    and restart Key Generation.")
656. 
657. 
658.         s.close
659. 
660.         self.info_recon = key
661. 
662.         qr_key = list(key)
663. 
664.         if int(math.sqrt(len(qr_key)))==math.sqrt(len(qr_key)):
665.             shaped_length=shaped_width=math.sqrt(len(qr_key))
666.         else:
667.             shaped_length=2**((math.log(len(qr_key),2)-1)/2)
668.             shaped_width=2**((math.log(len(qr_key),2)+1)/2)
669.         shaped_array = np.array(qr_key).reshape((int(shaped_length), int(sh
    aped_width)))
670. 
671. 
672.         info_fig = plt.figure(figsize=(1, 1))
673.         info_fig_plot = plt.subplot(1, 1, 1)
674.         info_fig_plot.axes.get_xaxis().set_visible(False)
675.         info_fig_plot.axes.get_yaxis().set_visible(False)
676.         info_fig_plot.imshow(shaped_array, extent=[0, 100, 0, 1], aspect=10
    0, cmap=plt.cm.gray_r)
677.         info_fig.savefig("info_recon.png", dpi=info_fig.dpi)
678. 
679.         img = mpimg.imread("info_recon.png")
680.         self.fig_plot = self.fig.add_subplot(132)
681.         self.fig_plot.imshow(img)
682.         self.fig_plot.axes.get_xaxis().set_visible(False)
683.         self.fig_plot.axes.get_yaxis().set_visible(False)
684.         self.fig_plot.set_title("Information Reconciliation", size=15)
685.         self.fig.tight_layout()
686.         self.canvas.draw()
687. 
688.         data_get_index = self.data.get_children()
689.         self.data.item(data_get_index[0], values=('length of key after info
    rmation reconciliation', len(qr_key)))
690.         self.data.item(data_get_index[1], values=('CRC code', crc))
691.         #self.data.delete(data_get_index[2])
692.         #self.data.delete(data_get_index[3])
693. 
694.         return self.fig_plot
695. 
696. 
697. 
698.     def Privacy_Amp(self) :
699. 
```

```
700.         try :
701.             bi_array = self.info_recon
702.         except AttributeError, e:
703.             msg.showinfo("Information", "Lack of steps. Please reset and re
    start Channel Probing.")
704.
705.         bi_list = list(bi_array)
706.
707.         bi_str = ''.join(str(i) for i in bi_array)
708.         print(bi_str)
709.
710.         hex_str = "{0:0>4X}".format(int(bi_str, 2))
711.         print(hex_str)
712.
713.         if len(hex_str) % 2 != 0 :
714.             hex_str = "0" + hex_str
715.
716.         hex_key = binascii.a2b_hex(hex_str)
717.
718.         hash_hex = hashlib.sha256(hex_key).hexdigest()
719.
720.         bi_hash = bin(int(hash_hex, 16))[2:]
721.
722.         hash_list = [int(i) for i in bi_hash]
723.
724.
725.         len_key = int(len(bi_list))
726.         len_hash = int(len(hash_list))
727.
728.         if len_key<= len_hash :
729.            len_hash = len_key
730.
731.         #del hash_list[int(math.pow(len_key, 2)):]
732.         del hash_list[len_key:]
733.         str_hash = ''.join([str(i) for i in hash_list])
734.
735.         print "key after privacy amplification: ", str_hash
736.
737.         self.str_hash = str_hash
738.
739.         if int(math.sqrt(len_key))==math.sqrt(len_key):
740.             shaped_length=shaped_width=math.sqrt(len_key)
741.         else:
742.             shaped_length=2**((math.log(len_key,2)-1)/2)
743.             shaped_width=2**((math.log(len_key,2)+1)/2)
744.         shaped_array = np.array(hash_list).reshape((int(shaped_length), int
    (shaped_width)))
745.
746.         hash_fig = plt.figure(figsize=(1, 1))
747.         hash_fig_plot = plt.subplot(1, 1, 1)
748.         hash_fig_plot.axes.get_xaxis().set_visible(False)
```

```
749.         hash_fig_plot.axes.get_yaxis().set_visible(False)
750.         hash_fig_plot.imshow(shaped_array, extent=[0, 100, 0, 1], aspect=10
    0, cmap=plt.cm.gray_r)
751.         hash_fig.savefig("privacy_amp.png", dpi=hash_fig.dpi)
752.
753.         img = mpimg.imread("privacy_amp.png")
754.         self.fig_plot = self.fig.add_subplot(133)
755.         self.fig_plot.imshow(img)
756.         self.fig_plot.axes.get_xaxis().set_visible(False)
757.         self.fig_plot.axes.get_yaxis().set_visible(False)
758.         self.fig_plot.set_title("Privacy Amplification", size=15)
759.         self.fig.tight_layout()
760.         self.canvas.draw()
761.
762.         data_get_index = self.data.get_children()
763.         self.data.item(data_get_index[0], values=('length of key after priv
    acy amplification', len(hash_list)))
764.         self.data.delete(data_get_index[1])
765.
766.         return self.fig_plot
767.
768.
769.     def thread_cp(self) :
770.         threading.Thread(target=self.channel_probing).start()
771.
772.     def realtime_display(self, i) :
773.         self.fig_plot.clear()
774.         self.fig_plot.plot(self.time_array, self.rssi_array,'k',color='blue
    ',marker='o')
775.         self.fig_plot.set_xlabel('timestamp', fontsize=13)
776.         self.fig_plot.set_ylabel('Received Signal Strength (dBm)', fontsize
    =14)
777.         self.fig_plot.set_title('Channel Probing', size=16)
778.         self.canvas.draw()
779.
780.         return self.fig_plot
781.
782.     def Test(self) :
783.
784.         try :
785.             monobit_test_quant = FT.FrequencyTest.monobit_test(self.quantiz
    ed_rs_str, True)
786.         except AttributeError, e :
787.             msg.showinfo("Information", "Lack of data. Please reset and res
    tart Channel Probing.")
788.
789.         # monobit test results
790.         re_monobit_test_quant = round(monobit_test_quant[0], 4)
791.         monobit_test_pri = FT.FrequencyTest.monobit_test(self.str_hash, Tru
    e)
792.         re_monobit_test_pri = round(monobit_test_pri[0], 4)
```

```
793.
794.         # block frequency
795.         block_freq_quant = FT.FrequencyTest.block_frequency(binary_data=sel
    f.quantized_rs_str, block_size=128, verbose=True)
796.         re_block_freq_quant = round(block_freq_quant[0], 4)
797.         block_freq_pri = FT.FrequencyTest.block_frequency(binary_data=self.
    str_hash, block_size=128, verbose=True)
798.         re_block_freq_pri = round(block_freq_pri[0], 4)
799.
800.         # Cum.Sum (fwd)
801.         cum_sum_fwd_quant = CS.CumulativeSums.cumulative_sums_test(self.qua
    ntized_rs_str, 0, True)
802.         re_cum_sum_fwd_quant = round(cum_sum_fwd_quant[0], 4)
803.         cum_sum_fwd_pri = CS.CumulativeSums.cumulative_sums_test(self.str_h
    ash, 0, True)
804.         re_cum_sum_fwd_pri = round(cum_sum_fwd_pri[0], 4)
805.
806.         # Cum.Sum (rev)
807.         cum_sum_rev_quant = CS.CumulativeSums.cumulative_sums_test(self.qua
    ntized_rs_str, 1, True)
808.         re_cum_sum_rev_quant = round(cum_sum_rev_quant[0], 4)
809.         cum_sum_rev_pri = CS.CumulativeSums.cumulative_sums_test(self.str_h
    ash, 1, True)
810.         re_cum_sum_rev_pri = round(cum_sum_rev_pri[0], 4)
811.
812.         # run test
813.         run_test_quant = RT.RunTest.run_test(self.quantized_rs_str, True)
814.         re_run_test_quant = round(run_test_quant[0], 4)
815.         run_test_pri = RT.RunTest.run_test(self.str_hash, True)
816.         re_run_test_pri = round(run_test_pri[0], 4)
817.
818.         # longest one block test
819.         longest_block_quant = RT.RunTest.longest_one_block_test(self.quanti
    zed_rs_str, True)
820.         re_longest_block_quant = round(longest_block_quant[0], 4)
821.         longest_block_pri = RT.RunTest.longest_one_block_test(self.str_hash,
     True)
822.         re_longest_block_pri = round(longest_block_pri[0], 4)
823.
824.         # Serial 1 2
825.         serial_quant = Serial.Serial.serial_test(self.quantized_rs_str, Tru
    e, 5)
826.         re_serial1_quant = round(serial_quant[0][0], 4)
827.         re_serial2_quant = round(serial_quant[1][0], 4)
828.         serial_pri = Serial.Serial.serial_test(self.str_hash, True, 5)
829.         re_serial1_pri = round(serial_pri[0][0], 4)
830.         re_serial2_pri = round(serial_pri[1][0], 4)
831.
832.         # ApproximateEntropy
833.         approx_entropy_quant = AE.ApproximateEntropy.approximate_entropy_te
    st(self.quantized_rs_str, True, 2)
```

```
        re_approx_entropy_quant = round(approx_entropy_quant[0], 4)
        approx_entropy_pri = AE.ApproximateEntropy.approximate_entropy_test(self.str_hash, True, 2)
        re_approx_entropy_pri = round(approx_entropy_pri[0], 4)
        '''
        # Spectral Test
        spectraltest_quant = ST.SpectralTest.sepctral_test(self.quantized_rs_str, False)
        re_spectraltest_quant = round(spectraltest_quant[0], 4)
        spectraltest_pri = ST.SpectralTest.sepctral_test(self.str_hash, False)
        re_spectraltest_pri = round(spectraltest_pri[0], 4)
        '''
        '''
        # Non overlapping Test
        non_overlapping_quant = NO.test(self.quantized_rs_str, len(self.quantized_rs_str))
        re_non_overlapping_quant = round(non_overlapping_quant[0], 4)
        non_overlapping_pri = NO.test(self.str_hash, len(self.str_hash))
        re_non_overlapping_pri = round(non_overlapping_pri[0], 4)
        '''


        self.main_frame = tk.LabelFrame(self.master, text='Key Generation Results', bg="WhiteSmoke", bd=2)
        self.main_frame.grid(row=1, column=1, sticky=tk.NSEW)

        self.fig = Figure(figsize=(7, 3), dpi=30)
        self.canvas = FigureCanvasTkAgg(self.fig, self.main_frame)
        self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=1)

        img1 = mpimg.imread("quantized.png")
        img2 = mpimg.imread("info_recon.png")
        img3 = mpimg.imread("privacy_amp.png")
        self.fig_plot = self.fig.add_subplot(131)
        self.fig_plot.imshow(img1)
        self.fig_plot.axes.get_xaxis().set_visible(False)
        self.fig_plot.axes.get_yaxis().set_visible(False)
        self.fig_plot.set_title("Quantization", size=22)

        self.fig_plot = self.fig.add_subplot(132)
        self.fig_plot.imshow(img2)
        self.fig_plot.axes.get_xaxis().set_visible(False)
        self.fig_plot.axes.get_yaxis().set_visible(False)
        self.fig_plot.set_title("Information Reconciliation", size=22)


        self.fig_plot = self.fig.add_subplot(133)
        self.fig_plot.imshow(img3)
        self.fig_plot.axes.get_xaxis().set_visible(False)
```

```
879.         self.fig_plot.axes.get_yaxis().set_visible(False)
880.         self.fig_plot.set_title("Privacy Amplification", size=22)
881.         #self.fig.tight_layout()
882.         self.canvas.draw()
883.
884.
885.         self.data_frame = tk.Frame(self.master, bg="White")
886.         self.data_frame.grid(row=2, rowspan=2, column=1, sticky=tk.NSEW)
887.
888.         self.data_frame.rowconfigure(0, weight=1)
889.         self.data_frame.columnconfigure(0, weight=7)
890.         self.data_frame.columnconfigure(1, weight=1)
891.
892.         # Randomness Test
893.         self.data = ttk.Treeview(self.data_frame, show="headings")
894.         self.data.grid(row=0, column=0, sticky=tk.NSEW)
895.         self.data['columns'] = ('Randomness Test', 'Quant', 'Priv. Amp')
896.         self.data['height'] = 5
897.         self.data.column("Randomness Test", width=70, anchor='center')
898.         self.data.column("Quant", width=15, anchor='center')
899.         self.data.column('Priv. Amp', wid=15, anchor='center')
900.         self.data.heading("Randomness Test", text="Randomness Test")
901.         self.data.heading("Quant", text="Quant")
902.         self.data.heading('Priv. Amp', text='Priv. Amp')
903.         #self.data.pack(fill=tk.BOTH, expand=1)
904.
905.         self.monobit_test = self.data.insert('', 0, values=('Monobit test',
     re_monobit_test_quant, re_monobit_test_pri), tag=('odd',))
906.         self.block_freq = self.data.insert('', 1, values=('Block frequency',
     re_block_freq_quant, re_block_freq_pri), tag=('even',))
907.         self.cum_fwd = self.data.insert('', 2, values=('Cum.sums (fwd)', re
    _cum_sum_fwd_quant, re_cum_sum_fwd_pri), tag=('odd',))
908.         self.cum_rev = self.data.insert('', 3, values=('Cum.sums (rev)', re
    _cum_sum_rev_quant, re_cum_sum_rev_pri), tag=('even',))
909.         self.runs = self.data.insert('', 4, values=('Runs', re_run_test_qua
    nt, re_run_test_pri), tag=('odd',))
910.         self.longest_block = self.data.insert('', 5, values=('Longest one b
    lock', re_longest_block_quant, re_longest_block_pri), tag=('even',))
911.         self.serial1 = self.data.insert('', 6, values=('Serial 1', re_seria
    l1_quant, re_serial1_pri), tag=('odd',))
912.         self.serial2 = self.data.insert('', 7, values=('Serial 2', re_seria
    l2_quant, re_serial2_pri), tag=('even',))
913.         self.appro_entry = self.data.insert('', 8, values=('Approx.entropy',
     re_approx_entropy_quant, re_approx_entropy_pri), tag=('odd',))
914.         #self.spectraltest = self.data.insert('', 9, values=('DFT test', re
    _spectraltest_quant, re_spectraltest_pri), tag=('even',))
915.         #self.non_overlapping = self.data.insert('', 10, values=('Non-
    overlapping', re_non_overlapping_quant, re_non_overlapping_pri), tag=('odd',
    ))
916.
917.         self.data.tag_configure('odd', background='White')
```

```
        self.data.tag_configure('even', background='WhiteSmoke')

        # Encryption Test
        self.encrypt_test = tk.LabelFrame(self.data_frame, text='Encryption Test', bg="White", bd=2)
        self.encrypt_test.grid(row=0, column=1, sticky=tk.NSEW)

        self.encrypt_test.rowconfigure(0, weight=1)
        self.encrypt_test.rowconfigure(1, weight=2)
        self.encrypt_test.rowconfigure(2, weight=1)
        self.encrypt_test.columnconfigure(0, weight=1)
        self.encrypt_test.columnconfigure(1, weight=1)

        encrypt_font = tf.Font(family="Times New Roman", size=12)
        self.info = tk.Label(self.encrypt_test, text="Please enter the text to be encrypted", fg="black", bg="White", font=encrypt_font)
        self.info.grid(row=0, column=0, columnspan=2)
        self.text = tk.Text(self.encrypt_test, width='25', height='5')
        self.text.grid(row=1, column=0, columnspan=2, sticky=tk.N)
        self.clear_button = tk.Button(self.encrypt_test, text="Clear", font=encrypt_font, bg="WhiteSmoke", activeforeground="red", command=self.clear)
        self.clear_button.grid(row=2, column=0, sticky=tk.N)
        self.encrypt_button = tk.Button(self.encrypt_test, text="Encryption", font=encrypt_font, bg="WhiteSmoke", activeforeground="red", command=self.Encryption)
        self.encrypt_button.grid(row=2, column=1, sticky=tk.N)


    def Reset(self) :

        self.animate.event_source.start()

        # main display
        self.main_frame = tk.LabelFrame(self.master, text='Key Generation', bg="WhiteSmoke")
        self.main_frame.grid(row=1, rowspan=2, column=1, sticky=tk.NSEW)

        self.fig = Figure(figsize=(5, 2), dpi=60)
        self.fig_plot = self.fig.add_subplot(111)
        self.canvas = FigureCanvasTkAgg(self.fig, self.main_frame)
        self.canvas.draw()
        self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=1)


        # Data
        self.data_frame = tk.LabelFrame(self.master, bg="White", text="Statistic")
        self.data_frame.grid(row=3, column=1, sticky=tk.NSEW)
        self.data_frame.rowconfigure(0, weight=1)
        self.data_frame.columnconfigure(0, weight=1)

```

```
962.         self.data = ttk.Treeview(self.data_frame, show="headings")
963.         self.data.grid(row=0, column=0, sticky=tk.NSEW)
964.         self.data['columns'] = ('STA side', 'Results')
965.         self.data['height'] = 4
966.         self.data.column("STA side", width=120, anchor='center')
967.         self.data.column("Results", width=10, anchor='center')
968.         self.data.heading("STA side", text="Station side")
969.         self.data.heading("Results", text="Results")
970.
971.         self.frame_num = self.data.insert('', 0, values=('number of packet
    received',))
972.         self.frame_loss = self.data.insert('', 1, values=('number of packet
     loss',))
973.         self.frame_lossrate = self.data.insert('', 2, values=('packet loss
    rate',))
974.         self.matched_num = self.data.insert('', 3, values=('number of packe
    t after matching',))
975.
976.
977.         self.timestamp=[]
978.         self.rssi=[]
979.         self.time_array=([])
980.         self.rssi_array=([])
981.         #self.matched_ts_sta = []
982.         self.matched_rssi = []
983.         #self.matched_ts_array=([])
984.         #self.matched_rssi_array=([])
985.
986.
987.     def Encryption(self) :
988.
989.         def encrypt(text):
990.             cipher = AES.new(self.key, AES.MODE_CBC, self.key)
991.             length = 16
992.             count = len(text)
993.             if(count % length != 0) :
994.                 add = length - (count % length)
995.             else:
996.                 add = 0
997.             text = text + ('\0' * add)
998.             self.ciphertext = cipher.encrypt(text)
999.
1000.                 return b2a_hex(self.ciphertext)
1001.
1002.
1003.             s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
1004.             host = "192.168.3.20"
1005.             port = 12345
1006.             s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
1007.             s.bind((host, port))
1008.             s.listen(5)
```

```
            conn, addr = s.accept()
            print "connection address: ", addr

            h = hashlib.md5(self.str_hash)
            self.key = h.hexdigest()[8:-8]

            text_str = self.text.get('0.0', 'end')
            e = encrypt(text_str)
            print "encrypted text: ", e
            se_data = json.dumps(e).encode()
            le_data = json.dumps(len(se_data)).encode()

            while True :
                buf1 = conn.recv(1024)
                if buf1 :
                    conn.send(le_data)
                    time.sleep(2)
                    conn.send(se_data)
                    break

            print "Finish sending"
            conn.close()
            s.close()


        def clear(self) :
            self.text.delete('0.0', 'end')

        def Quit(self) :
            sys.exit(0)

    if __name__ == '__main__' :
        gui = tk.Tk()
        k = Key_Generation(gui)
        gui.mainloop()
```

C.2 Code for Bob

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import logging
import os
import time
from datetime import datetime
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *
import socket
import json
from statistics import mean, StatisticsError
```

```
13.
14. import bchlib
15. import binascii
16. import numpy as np
17. import hashlib
18. import math
19.
20. import Tkinter as tk
21. import tkFont as tf
22. from PIL import Image
23. import matplotlib
24. matplotlib.use('TkAgg')
25. import matplotlib.pyplot as plt
26. from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, Navigation
    Toolbar2TkAgg
27. from matplotlib.backend_bases import key_press_handler
28. from matplotlib.figure import Figure
29. import matplotlib.image as mpimg
30. import matplotlib.animation as animation
31. import threading
32. import ttk
33. import tkMessageBox as msg
34. import tkSimpleDialog as simpledialog
35.
36. import FrequencyTest as FT
37. import CumulativeSum as CS
38. import RunTest as RT
39. import Serial
40. import ApproximateEntropy as AE
41. import Spectral as ST
42. import Non_overlapping as NO
43. #import Universal as UN
44. #import RandomExcursion as RE
45.
46. import sys
47. from Crypto.Cipher import AES
48. from binascii import a2b_hex
49. from Tkinter import StringVar
50.
51. class Key_Generation :
52.
53.     def __init__(self, master) :
54.
55.         self.master = master
56.         self.master.title("Key Generation")
57.         self.master.geometry("800x480+0+0")
58.
59.         # divide the area into grids
60.         self.master.rowconfigure(0, weight=2)
61.         self.master.rowconfigure(1, weight=60)
62.         self.master.rowconfigure(2, weight=170)
```

```
63.         self.master.rowconfigure(3, weight=10)
64.         self.master.columnconfigure(0, weight=1)
65.         self.master.columnconfigure(1, weight=35)
66.
67.         # logo and title frame
68.         self.title_frame = tk.Frame(self.master, bg="white", bd=1)
69.         self.title_frame.grid(row=0, column=0, columnspan=3, sticky=tk.NSEW)
70.         self.title_frame.rowconfigure(0, weight=1)
71.         self.title_frame.columnconfigure(0, weight=1)
72.         self.title_frame.columnconfigure(1, weight=6)
73.
74.         titlefont = tf.Font(family="Times New Roman", size=12, weight=tf.BO
    LD)
75.         self.rcv_timeitle = tk.Label(self.title_frame, fg="black", bg="whit
    e",text="A WiFi-based Key Generation System", font=titlefont)
76.         self.rcv_timeitle.grid(row=0, column=1, columnspan=2, sticky=tk.NSE
    W)
77.
78.         self.logo = tk.PhotoImage(file="logo.gif")
79.         self.sch_logo = tk.Label(self.title_frame, bg="white", image=self.l
    ogo)
80.         self.sch_logo.grid(row=0, column=0, sticky=tk.NSEW)
81.
82.
83.         # button frame
84.         self.button_frame = tk.Frame(self.master, bg="white")
85.         self.button_frame.grid(row=1, rowspan=3, column=0, sticky=tk.NSEW)
86.         self.button_frame.rowconfigure(0, weight=1)
87.         self.button_frame.rowconfigure(1, weight=1)
88.         self.button_frame.rowconfigure(2, weight=1)
89.         self.button_frame.rowconfigure(3, weight=1)
90.         self.button_frame.rowconfigure(4, weight=1)
91.         self.button_frame.rowconfigure(5, weight=1)
92.         self.button_frame.rowconfigure(6, weight=1)
93.         self.button_frame.rowconfigure(7, weight=1)
94.         self.button_frame.rowconfigure(8, weight=1)
95.         self.button_frame.columnconfigure(0, weight=1)
96.
97.         # set up entry
98.         self.setup_frame = tk.LabelFrame(self.button_frame, text="Set up",
    bg="WhiteSmoke", bd=2)
99.         self.setup_frame.grid(row=0, column=0, sticky=tk.NSEW)
100.        self.setup_frame.rowconfigure(0, weight=1)
101.        self.setup_frame.rowconfigure(1, weight=1)
102.        self.setup_frame.rowconfigure(2, weight=1)
103.        self.setup_frame.columnconfigure(0, weight=1)
104.        self.setup_frame.columnconfigure(1, weight=1)
105.        self.setup_frame.columnconfigure(2, weight=1)
106.
107.        setup_font = tf.Font(family="Times New Roman", size=10, weight=tf.B
    OLD)
```

```
108.         #self.setup_interval = tk.Label(self.setup_frame, text="Interval (s)
     ", fg="black", bg="WhiteSmoke", font=setup_font)
109.         #self.setup_interval.grid(row=0, column=0)
110.         #self.interval_value = tk.DoubleVar(self.setup_frame)
111.         #self.interval_value.set(1.0)
112.         #self.interval_entry = tk.Entry(self.setup_frame, textvariable=self.
     interval_value)
113.         #self.interval_entry.grid(row=0, column=1)
114.
115.         self.setup_number = tk.Label(self.setup_frame, text="  No. packet ",
      fg="black", bg="WhiteSmoke", font=setup_font)
116.         self.setup_number.grid(row=0, column=0)
117.         self.number_value = tk.IntVar(self.setup_frame)
118.         self.number_value.set(160)
119.         self.number_entry = tk.Entry(self.setup_frame, textvariable=self.nu
     mber_value, width=12, bd=3)
120.         self.number_entry.grid(row=0, column=1)
121.
122.         self.setup_channel = tk.Label(self.setup_frame, text="Channel", fg=
     "black", bg="WhiteSmoke", font=setup_font)
123.         self.setup_channel.grid(row=2, column=0)
124.         self.channel_value = tk.IntVar(self.setup_frame)
125.         self.channel_value.set(165)
126.         self.channel_entry = tk.Entry(self.setup_frame, textvariable=self.c
     hannel_value, bd=3, width=12)
127.         self.channel_entry.grid(row=2, column=1)
128.
129.         self.setup_key = tk.Label(self.setup_frame, text="Key length", fg="
     black", bg="WhiteSmoke", font=setup_font)
130.         self.setup_key.grid(row=1, column=0)
131.         self.key_value = tk.IntVar(self.setup_frame)
132.         self.key_value.set(128)
133.         self.key_entry = tk.Entry(self.setup_frame, textvariable=self.key_v
     alue, bd=3, width=12)
134.         self.key_entry.grid(row=1, column=1)
135.
136.
137.
138.         # button control
139.         button_font = tf.Font(family="Times New Roman", size=11, weight=tf.
     BOLD)
140.         self.bu_channel_pr = tk.Button(self.button_frame, text="Channel Pro
     bing", font=button_font, width=23, height=1, bg="LightGrey", activeforegrou
     nd="red", command=self.thread_cp)
141.         self.bu_channel_pr.grid(row=1, column=0)
142.         self.bu_packet_ma = tk.Button(self.button_frame, text="Packet Match
     ing", font=button_font, width=23, height=1, bg="LightGrey", activeforegroun
     d="red", command=self.Packet_Matching)
143.         self.bu_packet_ma.grid(row=2, column=0)
```

```
144.         self.bu_quanti = tk.Button(self.button_frame, text="Quantization", font=button_font, width=23, height=1, bg="LightGrey", activeforeground="red", command=self.Quantization_choose)
145.         self.bu_quanti.grid(row=3, column=0)
146.         self.bu_info_recon = tk.Button(self.button_frame, text="Information Reconciliation", font=button_font, width=23, height=1, bg="LightGrey", activeforeground="red", command=self.Info_Recon)
147.         self.bu_info_recon.grid(row=4, column=0)
148.         self.bu_privacy_amp = tk.Button(self.button_frame, text="Privacy Amplification", font=button_font, width=23, height=1, bg="LightGrey", activeforeground="red", command=self.Privacy_Amp)
149.         self.bu_privacy_amp.grid(row=5, column=0)
150.         self.test = tk.Button(self.button_frame, text="Test", font=button_font, width=23, height=1, bg="LightGrey", activeforeground="red", command=self.Test)
151.         self.test.grid(row=6, column=0)
152.         self.bu_reset = tk.Button(self.button_frame, text="Reset", font=button_font, width=23, height=1, bg="LightGrey", activeforeground="red", command=self.Reset)
153.         self.bu_reset.grid(row=7, column=0)
154.         self.bu_reset = tk.Button(self.button_frame, text="Quit", font=button_font, width=23, height=1, bg="LightGrey", activeforeground="red", command=self.Quit)
155.         self.bu_reset.grid(row=8, column=0)
156.
157.
158.
159.         # main display
160.         self.main_frame = tk.LabelFrame(self.master, text='Key Generation', bg="WhiteSmoke")
161.         self.main_frame.grid(row=1, rowspan=2, column=1, sticky=tk.NSEW)
162.
163.         self.fig = Figure(figsize=(5, 2), dpi=60)
164.         self.fig_plot = self.fig.add_subplot(111)
165.         self.canvas = FigureCanvasTkAgg(self.fig, self.main_frame)
166.         self.canvas.draw()
167.         self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=1)
168.         #self.toolbar = NavigationToolbar2TkAgg(self.canvas, self.main_frame)
169.         #self.toolbar.update()
170.
171.
172.         # Data
173.         self.data_frame = tk.LabelFrame(self.master, bg="White", text="Statistic")
174.         self.data_frame.grid(row=3, column=1, sticky=tk.NSEW)
175.         self.data_frame.rowconfigure(0, weight=1)
176.         self.data_frame.columnconfigure(0, weight=1)
177.
178.         self.data = ttk.Treeview(self.data_frame, show="headings")
179.         self.data.grid(row=0, column=0, sticky=tk.NSEW)
```

```
180.         self.data['columns'] = ('AP side', 'Results')
181.         self.data['height'] = 4
182.         self.data.column("AP side", width=120, anchor='center')
183.         self.data.column("Results", width=10, anchor='center')
184.         self.data.heading("AP side", text="AP side")
185.         self.data.heading("Results", text="Results")
186.
187.         self.frame_num = self.data.insert('', 0, values=('number of packet
    received',))
188.         self.matched_num = self.data.insert('', 3, values=('number of packe
    t after matching',))
189.         self.frame_loss = self.data.insert('', 1, values=('number of packet
     loss',))
190.         self.frame_lossrate = self.data.insert('', 2, values=('packet loss
    rate',))
191.
192.         self.animate = animation.FuncAnimation(self.fig, self.realtime_disp
    lay, interval=100)
193.
194.         self.timestamp=[]
195.         self.rssi=[]
196.         self.time_array=([])
197.         self.rssi_array=([])
198.         #self.matched_ts_ap = []
199.         self.matched_rssi = []
200.         #self.matched_ts_array=([])
201.         #self.matched_rssi_array=([])
202.
203.
204.
205.     def channel_probing(self) :    # channel probing
206.         interface = 'mon0'    # wireless interface on monitor mode
207.         destination = bssid = 'dc:a6:32:a2:7a:0c'      # destination MAC ad
    dress
208.         ssid = 'Access Point'
209.         source= 'dc:a6:32:ca:6c:da'    # source MAC address
210.         self.bootime = time.time()
211.         self.sc = -1
212.         self.flag = 0
213.         self.count=0
214.
215.         # construct data frame
216.         def Dataframe(source, channel, ssid, destination, bssid, interface):

217.             verbose = 0
218.             essid = Dot11Elt(ID='SSID',info=ssid, len=len(ssid))
219.             dsset = Dot11Elt(ID='DSset',info=chr(channel))
220.             frame =  RadioTap()/Dot11(type=2,subtype=0,addr1=destination,ad
    dr2=source,addr3=bssid)\
221.             /Dot11()/essid/dsset
222.             # Update timestamp
```

```
223.             frame.timestamp = current_timestamp()
224.             ## Update sequence number
225.             frame.SC = next_sc()
226.             if verbose: frame.show()
227.             try:
228.                 sendp(frame, iface=interface, verbose=verbose, count=1)
229.                 rcv_time = datetime.now()
230.                 print("%s send a data packet to STA %s") % (rcv_time, desti
    nation)
231.             except:
232.                 raise
233.
234.
235.         def current_timestamp():
236.             return (time.time() - self.bootime) * 1000000
237.
238.         def next_sc():
239.             self.sc = (self.sc + 1) % 4096
240.             return self.sc * 16  # Fragment number -> right 4 bits
241.
242.
243.         def PacketSniff(frame) :
244.             if frame.haslayer(RadioTap) :
245.                 if frame.type ==2 and frame.subtype == 0:  ## data frame
246.                         if frame.addr2.upper() == destination.upper() and f
    rame.addr1.upper() == source.upper() :#Lowercase to uppercase
247.                             self.count += 1
248.                             rcv_time = datetime.now()
249.                             timestamp.append(rcv_time)

250.                             print("%s Received a data packet from STA %s RS
    SI:%sdBm") % (rcv_time, frame.addr2, frame[RadioTap].dBm_AntSignal)
251.                             self.flag = 1
252.                             rssi.append(int(frame[RadioTap].dBm_AntSignal))
253.                             f = open("AP.txt","a")  ## open a file for writ
    ing
254.                             f.write(str(rcv_time)+" "+str(frame[RadioTap].d
    Bm_AntSignal)+"\n")
255.                             f.close()  ## close the file
256.
257.         # stop sniffing when receive the frame
258.         def stopfilter(x) :
259.             if self.flag == 1:
260.                 self.flag = 0
261.                 return True
262.
263.             else :
264.                 return False
265.
266.
267.         i = 0
```

```
268.         f_ap = open("AP.txt","w")
269.         f_ap.close()
270.         times = self.number_value.get()
271.         channel=self.channel_value.get()
272.         keylength=self.key_value.get()
273.         if keylength>times:
274.                 msg.showinfo("Information", "Please select a key length les
    s thant the number of packets")
275.                 return
276.         else:
277.           timestamp=[]
278.           rssi=[]
279.           while i < times :
280.               sniff(iface=interface, prn=PacketSniff, stop_filter=stopfilte
    r, timeout=3)
281.               Dataframe(source, channel, ssid, destination, bssid, interfac
    e)
282.               self.time_array=np.array(timestamp)
283.               self.rssi_array=np.array(rssi)
284.               i += 1
285.
286.
287.
288.         self.animate.event_source.stop()
289.         loss=times-self.count
290.         lossrate=round(float(loss)/times,2)
291.
292.         data_get_index = self.data.get_children()
293.         self.data.item(data_get_index[0], values=('number of packet receive
    d', self.count))
294.         self.data.item(data_get_index[1], values=('number of packet loss',
    loss))
295.         self.data.item(data_get_index[2], values=('packet loss rate', lossr
    ate))
296.
297.         return self.time_array, self.rssi_array
298.
299.     def Packet_Matching(self) :  # packet matching
300.
301.         f1 = open("AP.txt", "r")
302.         f2 = open("matchedAP.txt", "w+")
303.
304.         fmt = "%Y-%m-%d %H:%M:%S.%f"  # timestamp format
305.
306.         recvd=""
307.
308.         self.host = "192.168.3.13" #IP address of host
309.         self.port = 12345
310.         s =socket.socket(socket.AF_INET, socket.SOCK_STREAM) #creat socket
    object
311.         s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```
312.         s.bind((self.host, self.port))
313.         s.listen(5)
314.         conn, addr = s.accept()
315.         print "connection address: " , addr
316.
317.         conn.send(b'1')   # signal for receiving
318.
319.         length=json.loads(conn.recv(1024).decode())  # decode the JSON data
     and receive the packet length first
320.         if length >= 1024 :
321.             num_index = int(length/1024) + 1
322.         else :
323.             num_index = 1
324.
325.         #num_index=int(length/1024)+2
326.
327.         for i in range(0, num_index) :
328.             recvd = recvd + conn.recv(1024).decode()
329.
330.         data = json.loads(recvd)
331.         print "Finish receiving"
332.         print(data)
333.
334.         span = 2
335.         words = data.split(" ")    # separate data by " "
336.         #print(words)
337.
338.         ts = [" ".join(words[i:i+span]) for i in range(0, len(words), span)]
339.         # separate a string every two blank
340.         print(ts)
341.
342.         sta_ts = []
343.
344.         for i in range(0, len(ts)) :
345.             sta_ts.append(datetime.strptime(ts[i], fmt))
346.
347.         def Read_datetime(line, fmt) :
348.         # parse timestamp with date from a string
349.             try :
350.                 timestamp = datetime.strptime(line, fmt)
351.             except ValueError as v :
352.                 if len(v.args) > 0 and v.args[0].startswith('unconverted da
    ta remains: ') :
353.                     line = line[: - (len(v.args[0]) - 26)]
354.                     timestamp = datetime.strptime(line, fmt)
355.                 else :
356.                     raise
357.             return timestamp
358.
359.
360.         match_times2 = []
```

```
361.
362.         f1.seek(0)
363.         for linef1 in f1 :
364.             linef1 = linef1.strip('\n')
365.             t1 = Read_datetime(linef1, fmt)
366.
367.             for t2 in sta_ts :
368.             # compare the timestamps of Alice and Bob
369.                 t_diff = t1 - t2
370.                 if abs(t_diff.total_seconds() * 1000000) <= 280000 :
371.                     match_times2.append(t2)
372.                     f2.write(linef1 + '\n')
373.
374.         match_data = ' '.join(map(str, match_times2))
375.
376.         se_data = json.dumps(match_data).encode()
377.         le_data = json.dumps(len(match_data)).encode()
378.
379.         conn.send(le_data)
380.         time.sleep(2)
381.         conn.send(se_data)
382.         print("Finish sending")
383.
384.         conn.close
385.         s. close
386.
387.         f1.close
388.
389.         f2.seek(0)
390.
391.         # filter the timestamps
392.         matched_ts_ap=[]
393.         matched_rssi=[]
394.         for linef2 in f2 :
395.             linef2 = linef2.strip('\n')
396.             columns = linef2.split(" ")
397.             #self.matched_ts_ap.append(Read_datetime(linef2, fmt))
398.             matched_ts_ap.append(Read_datetime(linef2, fmt))
399.             if len(columns) > 2 :
400.                 matched_rssi.append(int(columns[2]))
401.         self.matched_rssi=matched_rssi
402.         f2.close
403.         print(matched_ts_ap, matched_rssi)
404.         ave=np.mean(matched_rssi)
405.         matched_ts_array=np.array(matched_ts_ap)
406.         matched_rssi_array=np.array(matched_rssi)
407.
408.         self.fig_plot.clear()
409.         self.fig_plot.plot(matched_ts_array, matched_rssi_array,'k',color='
    red',marker='o')
410.         self.fig_plot.axhline(y=ave,color='red',ls='--')
```

```
411.         self.fig_plot.set_xlabel('timestamp', fontsize=13)
412.         self.fig_plot.set_ylabel('Received Signal Strength (dBm)', fontsize
    =14)
413.         self.fig_plot.set_title('Channel Probing', size=16)
414.         self.canvas.draw()
415.
416.         data_get_index = self.data.get_children()
417.         self.data.item(data_get_index[3], values=('number of packet after m
    atching', len(matched_rssi)))
418.
419.         key_length=self.key_value.get()
420.         if len(self.matched_rssi)<key_length:
421.             msg.showinfo("Information", "Packet number is not enough. Pleas
    e reset and restart channel probing.")
422.
423.
424.         return matched_ts_array, matched_rssi_array, self.fig_plot
425.
426.     def Quantization_choose(self):
427.         self.tl=tk.Toplevel()
428.         self.tl.geometry("300x150+300+200")
429.         self.tl.title('Quantization')
430.         self.tl["background"]="LightGrey"
431.         tlfont = tf.Font(family="Times New Roman", size=13, weight=tf.BOLD)
432.         w=tk.Label(self.tl,text="Please choose the Quantization method",bg=
    "LightGrey",font=tlfont).pack()
433.
434.         self.x = tk.IntVar()
435.
436.         self.result=0
437.         tk.Radiobutton(self.tl,text='1.Mean value-
    based',variable=self.x,value=1,indicatoron=0,bg="Aqua",command=self.Quantiz
    ation_type).pack(ipadx=10,ipady=10)
438.         tk.Radiobutton(self.tl,text='2.Differential-
    based',variable=self.x,value=2,indicatoron=0,bg="Yellow",command=self.Quant
    ization_type).pack(ipadx=10,ipady=10)
439.
440.     def Quantization_type(self):
441.          self.result=self.x.get()
442.          self.tl.destroy()
443.          self.Quantization()
444.
445.
446.     def Quantization(self):
447.
448.         self.fig_plot.clear()
449.         self.fig_plot.axis('off')
450.         rs=[]
451.         quantized_rs = []
452.
```

```
453.         #self.result = simpledialog.askinteger(title = 'Quantization type',
    prompt='Please choose the quantization method: (1.Mean value-
    based. 2.Differential-based.)',initialvalue = '')
454.
455.
456.         try :
457.             rs = list(map(int, self.matched_rssi))
458.         except StatisticsError, e:
459.             msg.showinfo("Information", "0 packet collected. Please reset a
    nd restart Channel Probing.")
460.         global result
461.         if self.result==1:
462.
463.         # mean-based threshold
464.             average = mean(rs)
465.             for r1 in rs :
466.                 if r1 >= average :
467.                     quantized_rs.append(1)
468.                 else :
469.                     quantized_rs.append(0)
470.
471.             print "length of quantized rssi is : ", len(quantized_rs)
472.             print "quantized rssi is : ", quantized_rs
473.         else:
474.             #Differential-based quantization
475.             quantized_rs=[]
476.             drop=[]
477.             a=rs[0]
478.             x=0
479.             for i in range(1,len(rs)) :
480.                 if rs[i]>a+x:
481.                     quantized_rs.append(1)
482.                 elif rs[i]<a-x:
483.                     quantized_rs.append(0)
484.                 else:
485.                     quantized_rs.append(2)
486.                     drop.append(i-1)
487.                 a=rs[i]
488.             print "length of quantized rssi is : ", len(quantized_rs)
489.             print "rssi is:", quantized_rs
490.             print(drop)
491.             #send dropped bits
492.             drop_str=','.join(str(v) for v in drop)
493.             se_drop_str=json.dumps(drop_str).encode()
494.
495.             s =socket.socket(socket.AF_INET, socket.SOCK_STREAM)
496.             self.host = "192.168.3.13"
497.             self.port = 1234
498.             s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
499.             s.bind((self.host, self.port))
500.
```

```
            s.listen(5)
            conn, addr = s.accept()
            print "connection address1: " , addr

            conn.send('1')
            drop_recvd=''

            drop_recvd = drop_recvd+conn.recv(1024).decode()
            print"received:",drop_recvd

            print("Finish receiving")
            drop_recvd=eval(drop_recvd.encode('utf-8'))
            drop_recvd=drop_recvd.split(",")

            conn.send(se_drop_str)   # send dropped bits
            print("Finish sending")
            conn.close
            s.close

            #merge the dropped bits
            drop.extend(list(map(int,drop_recvd)))
            drop_mer=list(set(drop))
            drop_mer.sort()
            print(drop_mer)

            print(quantized_rs)
            #delete the dropped bits
            drop_mer.reverse()
            for d in drop_mer:
               del quantized_rs[d]
            print(quantized_rs)



        #time.sleep(2)
        # find the side length of the square
        key_length=self.key_value.get()
        if key_length>len(quantized_rs):
            msg.showinfo("Information", "Key length is not long enough.Plea
    se reset and restart channel probing")
            return

        del quantized_rs[key_length:]
        print "shaped key length:", len(quantized_rs)
        if int(math.sqrt(key_length))==math.sqrt(key_length):
            shaped_length=shaped_width=math.sqrt(key_length)
        else:
            shaped_length=2**((math.log(key_length,2)-1)/2)
            shaped_width=2**((math.log(key_length,2)+1)/2)


```

```
551.         self.quantized_rs_str = ''.join(str(i) for i in quantized_rs)
552.         self.quantized_rs = quantized_rs
553. 
554.         #creat socket object
555.         s =socket.socket(socket.AF_INET, socket.SOCK_STREAM)
556.         self.host = "192.168.3.13"
557.         self.port = 12345
558.         s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
559.         s.bind((self.host, self.port))
560. 
561.         s.listen(5)
562.         conn, addr = s.accept()
563.         print "connection address2: " , addr
564. 
565.         conn.send('1')
566. 
567.         length=json.loads(conn.recv(1024).decode())
568. 
569.         if length >= 1024 :
570.             num_index = int(length/1024) + 1
571.         else :
572.             num_index = 1
573. 
574.         recvd = ""
575.         for i in range(0, num_index) :
576.             recvd = recvd + conn.recv(1024).decode()
577. 
578.         key_sta = json.loads(recvd)
579. 
580.         kdr_value = self.KDR(key_sta, self.quantized_rs_str)
581. 
582.         conn.send(str(kdr_value))   # send kdr
583.         conn.close
584.         s.close
585. 
586.         shaped_array = np.array(quantized_rs).reshape((int(shaped_length),
    int(shaped_width)))
587. 
588.         quan_fig = plt.figure(figsize=(1, 1))
589.         quan_fig_plot = plt.subplot(1, 1, 1)
590.         quan_fig_plot.axes.get_xaxis().set_visible(False)
591.         quan_fig_plot.axes.get_yaxis().set_visible(False)
592.         quan_fig_plot.imshow(shaped_array, extent=[0, 100, 0, 1], aspect=10
    0, cmap=plt.cm.gray_r)
593.         # draw the bitmap, black represents '0', white means '1'
594.         quan_fig.savefig("quantized.png", dpi=quan_fig.dpi)
595. 
596.         shaped_array1 = np.array(self.diff).reshape((int(shaped_length), in
    t(shaped_width)))
597.         quan_fig = plt.figure(figsize=(1, 1))
598.         quan_fig_plot = plt.subplot(1, 1, 1)
```

```
599.         quan_fig_plot.axes.get_xaxis().set_visible(False)
600.         quan_fig_plot.axes.get_yaxis().set_visible(False)
601.         quan_fig_plot.imshow(shaped_array1, extent=[0, 100, 0, 1], aspect=1
    00, cmap=plt.cm.gray_r)
602.         # draw the bitmap, black represents '0', white means '1'
603.         quan_fig.savefig("diff.png", dpi=quan_fig.dpi)
604.
605.         img1 = mpimg.imread("quantized.png")
606.         img2 = mpimg.imread("diff.png")
607.         self.fig_plot = self.fig.add_subplot(131)
608.         self.fig_plot.imshow(img1)
609.         self.fig_plot.axes.get_xaxis().set_visible(False)
610.         self.fig_plot.axes.get_yaxis().set_visible(False)
611.         self.fig_plot.set_title("Quantiztion", size=15)
612.         self.fig_plot = self.fig.add_subplot(132)
613.         self.fig_plot.imshow(img2)
614.         self.fig_plot.axes.get_xaxis().set_visible(False)
615.         self.fig_plot.axes.get_yaxis().set_visible(False)
616.         self.fig_plot.set_title("Diff", size=15)
617.         self.fig.tight_layout()
618.         self.canvas.draw()
619.
620.         data_get_index = self.data.get_children()
621.         self.data.item(data_get_index[0], values=('length of key after quan
    tization', len(quantized_rs)))
622.         self.data.item(data_get_index[1], values=('key disagreement rate',
    kdr_value))
623.         self.data.delete(data_get_index[2])
624.         self.data.delete(data_get_index[3])
625.
626.         return self.fig_plot
627.
628.     def KDR(self, key_sta, key_ap):
629.
630.         key_sta_list = [int(i) for i in key_sta]
631.         key_ap_list = [int(i) for i in key_ap]
632.         key_sta_array = np.array(key_sta_list)
633.         key_ap_array = np.array(key_ap_list)
634.         KDR = round(float(sum(abs(key_sta_array -
     key_ap_array))) / float(len(key_ap_array)), 3)
635.         print "KDR: ", KDR
636.         self.diff=(key_sta_array - key_ap_array)
637.         self.diff=abs(self.diff)
638.         print (self.diff)
639.         return KDR
640.
641.
642.     def Info_Recon(self) :      # information reconciliation
643.
644.         try :
645.             key = self.quantized_rs
```

```
646.             key=np.array(key)
647.         except AttributeError, e:
648.             msg.showinfo("Information", "No quantized key. Please reset and
    restart Channel Probing.")
649.
650.         received_info=""
651.
652.         s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
653.         #host = "192.168.3.13"
654.         #port = 12345
655.         self.port=12346
656.         s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
657.         s.bind((self.host, self.port))
658.         s.listen(5)
659.
660.         conn, addr = s.accept()
661.
662.         print "connection address: ", addr
663.
664.         conn.send(b'1')
665.
666.         length=json.loads(conn.recv(1024).decode())
667.
668.         if length >= 1024 :
669.             num_index = int(length/1024) + 1
670.         else :
671.             num_index = 1
672.
673.         #num_index = int(length/1024) + 2
674.
675.         for i in range(0, num_index):
676.             received_info=received_info+conn.recv(1024).decode()
677.
678.         received_info=json.loads(received_info)
679.
680.         Packet=bytearray()
681.
682.         for i in range(0,len(received_info)):
683.             Packet.append(received_info[i])
684.
685.
686.         # Exclusive OR
687.         for i in range(0,int(len(key)/8)+1):
688.             if i<=int(len(key)/8)-1:
689.                 decimal=int((key[0+i*8]*2)**7+(key[1+i*8]*2)**6+(key[2+i*8]
    *2)**5+(key[3+i*8]*2)**4+(key[4+i*8]*2)**3+(key[5+i*8]*2)**2+(key[6+i*8]*2)
    **1+(key[7+i*8]))
690.                 Packet[i]^=decimal
691.             else:
692.                 byte=[]
693.                 for j in range(0,len(key)%8):
```

```
                    byte.append(key[j+i*8])
                for j in range(0,8-len(key)%8):
                    byte.append(0)
                decimal=int((byte[0]*2)**7+(byte[1]*2)**6+(byte[2]*2)**5+(byte[3]*2)**4+(byte[4]*2)**3+(byte[5]*2)**2+(byte[6]*2)**1)
                Packet[i]^=decimal


        # create a BCH object
        BCH_POLYNOMIAL=1033
        BCH_BITS=int(0.25*len(key))
        bch=bchlib.BCH(BCH_POLYNOMIAL,BCH_BITS)

        # de-packetize
        Data, Ecc = Packet[:-bch.ecc_bytes], Packet[-bch.ecc_bytes:]
        Bitflips, Correct_data, Correct_ecc= bch.decode(Data,Ecc)
        Correct_packet=Correct_data+Correct_ecc


        # Exclusive OR
        for i in range(0,int(len(key)/8)+1):
            Byte=list(bin(Packet[i]))
            Bit=[]
            if len(Byte)<10:
                for j in range(0,8-(len(Byte)-2)):
                    Bit.append('0')
            for j in range(2,len(Byte)):
                Bit.append(Byte[j])
            Correct_Byte=list(bin(Correct_packet[i]))
            Correct_Bit=[]
            if len(Correct_Byte)<10:
                for j in range(0,8-(len(Correct_Byte)-2)):
                    Correct_Bit.append('0')
            for j in range(2,len(Correct_Byte)):
                Correct_Bit.append(Correct_Byte[j])
            for j in range(0,8):
                if Bit[j]!=Correct_Bit[j]:
                    if key[j+i*8]==1:
                        key[j+i*8]=0
                    else:
                        key[j+i*8]=1

        print(key)

        # cyclic redundancy check (CRC)
        key_str = ''.join(str(i) for i in key)
        Key = [int(key_str[i:i+8], 2) for i in range(0, len(key_str), 8)]
        crc = hex(binascii.crc32(bytes(Key)) & 0xffffffff)

        print(crc)

```

```
744.         se_crc = json.dumps(crc).encode()
745.         conn.send(se_crc)
746.
747.         res = conn.recv(1024)
748.
749.         conn.close
750.         s.close
751.
752.         if res == "n" :
753.             msg.showinfo("Warning", "Cannot correct all keys. Please reset
    and restart Key Generation.")
754.
755.
756.         self.info_recon = key
757.
758.         qr_key = list(key)
759.
760.         if int(math.sqrt(len(qr_key)))==math.sqrt(len(qr_key)):
761.             shaped_length=shaped_width=math.sqrt(len(qr_key))
762.         else:
763.             shaped_length=2**((math.log(len(qr_key),2)-1)/2)
764.             shaped_width=2**((math.log(len(qr_key),2)+1)/2)
765.         shaped_array = np.array(qr_key).reshape((int(shaped_length), int(sh
    aped_width)))
766.
767.
768.         info_fig = plt.figure(figsize=(1, 1))
769.         info_fig_plot = plt.subplot(1, 1, 1)
770.         info_fig_plot.axes.get_xaxis().set_visible(False)
771.         info_fig_plot.axes.get_yaxis().set_visible(False)
772.         info_fig_plot.imshow(shaped_array, extent=[0, 100, 0, 1], aspect=10
    0, cmap=plt.cm.gray_r)
773.         info_fig.savefig("info_recon.png", dpi=info_fig.dpi)
774.
775.         img = mpimg.imread("info_recon.png")
776.         self.fig_plot = self.fig.add_subplot(132)
777.         self.fig_plot.imshow(img)
778.         self.fig_plot.axes.get_xaxis().set_visible(False)
779.         self.fig_plot.axes.get_yaxis().set_visible(False)
780.         self.fig_plot.set_title("Information Reconciliation", size=15)
781.         self.fig.tight_layout()
782.         self.canvas.draw()
783.
784.         data_get_index = self.data.get_children()
785.         self.data.item(data_get_index[0], values=('length of key after info
    rmation reconciliation', len(qr_key)))
786.         self.data.item(data_get_index[1], values=('CRC code', crc))
787.         #self.data.delete(data_get_index[2])
788.         #self.data.delete(data_get_index[3])
789.
790.         return self.fig_plot
```

```
791.
792.
793.     def Privacy_Amp(self) :   # privacy amplification
794.
795.         try :
796.             bi_array = self.info_recon
797.         except AttributeError, e:
798.             msg.showinfo("Information", "Lack of steps. Please reset and re
    start Channel Probing.")
799.
800.         bi_array = self.info_recon
801.
802.         bi_list = list(bi_array)
803.
804.         bi_str = ''.join(str(i) for i in bi_array)
805.
806.         hex_str = "{0:0>4X}".format(int(bi_str, 2))
807.
808.         if len(hex_str) % 2 != 0 :
809.             hex_str = "0" + hex_str
810.
811.         hex_key = binascii.a2b_hex(hex_str)
812.
813.         hash_hex = hashlib.sha256(hex_key).hexdigest()
814.
815.         bi_hash = bin(int(hash_hex, 16))[2:]
816.
817.         hash_list = [int(i) for i in bi_hash]
818.
819.         #len_key = int(math.sqrt(len(bi_list)))
820.         #len_hash = int(math.sqrt(len(hash_list)))
821.         len_key = int(len(bi_list))
822.         len_hash = int(len(hash_list))
823.
824.         if len_key <= len_hash :
825.             len_hash = len_key
826.
827.         #del hash_list[int((math.pow(len_key, 2)):]
828.         del hash_list[len_key:]
829.
830.         str_hash = ''.join([str(i) for i in hash_list])
831.
832.         print "key after privacy amplification: ", str_hash
833.
834.         self.str_hash = str_hash
835.
836.
837.         if int(math.sqrt(len_key))==math.sqrt(len_key):
838.             shaped_length=shaped_width=math.sqrt(len_key)
839.         else:
840.             shaped_length=2**((math.log(len_key,2)-1)/2)
```

```
841.             shaped_width=2**((math.log(len_key,2)+1)/2)
842.         shaped_array = np.array(hash_list).reshape((int(shaped_length), int
    (shaped_width)))
843. 
844. 
845.         hash_fig = plt.figure(figsize=(1, 1))
846.         hash_fig_plot = plt.subplot(1, 1, 1)
847.         hash_fig_plot.axes.get_xaxis().set_visible(False)
848.         hash_fig_plot.axes.get_yaxis().set_visible(False)
849.         hash_fig_plot.imshow(shaped_array, extent=[0, 100, 0, 1], aspect=10
    0, cmap=plt.cm.gray_r)
850.         hash_fig.savefig("privacy_amp.png", dpi=hash_fig.dpi)
851. 
852.         img = mpimg.imread("privacy_amp.png")
853.         self.fig_plot = self.fig.add_subplot(133)
854.         self.fig_plot.imshow(img)
855.         self.fig_plot.axes.get_xaxis().set_visible(False)
856.         self.fig_plot.axes.get_yaxis().set_visible(False)
857.         self.fig_plot.set_title("Privacy Amplification", size=15)
858.         self.fig.tight_layout()
859.         self.canvas.draw()
860. 
861.         data_get_index = self.data.get_children()
862.         self.data.item(data_get_index[0], values=('length of key after priv
    acy amplification', len(hash_list)))
863.         self.data.delete(data_get_index[1])
864. 
865. 
866.         return self.fig_plot
867. 
868.     def thread_cp(self) :
869.         threading.Thread(target=self.channel_probing).start()
870. 
871.     def realtime_display(self, i):
872.         self.fig_plot.clear()
873.         self.fig_plot.plot(self.time_array, self.rssi_array,'k',color='red',
    marker='o')
874.         self.fig_plot.set_xlabel('timestamp', fontsize=13)
875.         self.fig_plot.set_ylabel('Received Signal Strength (dBm)', fontsize
    =14)
876.         self.fig_plot.set_title('Channel Probing', size=16)
877.         self.canvas.draw()
878. 
879.         return self.fig_plot
880. 
881. 
882.     def Test(self) :
883. 
884.         try :
885.             monobit_test_quant = FT.FrequencyTest.monobit_test(self.quantiz
    ed_rs_str, True)
```

```
886.         except AttributeError, e:
887.             msg.showinfo("Information", "Lack of data. Please reset and res
    tart Channel Probing.")
888.
889.         # monobit test results
890.         re_monobit_test_quant = round(monobit_test_quant[0], 4)
891.         monobit_test_pri = FT.FrequencyTest.monobit_test(self.str_hash, Tru
    e)
892.         re_monobit_test_pri = round(monobit_test_pri[0], 4)
893.
894.         # block frequency
895.         block_freq_quant = FT.FrequencyTest.block_frequency(binary_data=sel
    f.quantized_rs_str, block_size=128, verbose=True)
896.         re_block_freq_quant = round(block_freq_quant[0], 4)
897.         block_freq_pri = FT.FrequencyTest.block_frequency(binary_data=self.
    str_hash, block_size=128, verbose=True)
898.         re_block_freq_pri = round(block_freq_pri[0], 4)
899.
900.         # Cum.Sum (fwd)
901.         cum_sum_fwd_quant = CS.CumulativeSums.cumulative_sums_test(self.qua
    ntized_rs_str, 0, True)
902.         re_cum_sum_fwd_quant = round(cum_sum_fwd_quant[0], 4)
903.         cum_sum_fwd_pri = CS.CumulativeSums.cumulative_sums_test(self.str_h
    ash, 0, True)
904.         re_cum_sum_fwd_pri = round(cum_sum_fwd_pri[0], 4)
905.
906.         # Cum.Sum (rev)
907.         cum_sum_rev_quant = CS.CumulativeSums.cumulative_sums_test(self.qua
    ntized_rs_str, 1, True)
908.         re_cum_sum_rev_quant = round(cum_sum_rev_quant[0], 4)
909.         cum_sum_rev_pri = CS.CumulativeSums.cumulative_sums_test(self.str_h
    ash, 1, True)
910.         re_cum_sum_rev_pri = round(cum_sum_rev_pri[0], 4)
911.
912.         # run test
913.         run_test_quant = RT.RunTest.run_test(self.quantized_rs_str, True)
914.         re_run_test_quant = round(run_test_quant[0], 4)
915.         run_test_pri = RT.RunTest.run_test(self.str_hash, True)
916.         re_run_test_pri = round(run_test_pri[0], 4)
917.
918.         # longest one block test
919.         longest_block_quant = RT.RunTest.longest_one_block_test(self.quanti
    zed_rs_str, True)
920.         re_longest_block_quant = round(longest_block_quant[0], 4)
921.         longest_block_pri = RT.RunTest.longest_one_block_test(self.str_hash,
     True)
922.         re_longest_block_pri = round(longest_block_pri[0], 4)
923.
924.         # Serial 1 2
925.         serial_quant = Serial.Serial.serial_test(self.quantized_rs_str, Tru
    e, 5)
```

```
926.         re_serial1_quant = round(serial_quant[0][0], 4)
927.         re_serial2_quant = round(serial_quant[1][0], 4)
928.         serial_pri = Serial.Serial.serial_test(self.str_hash, True, 5)
929.         re_serial1_pri = round(serial_pri[0][0], 4)
930.         re_serial2_pri = round(serial_pri[1][0], 4)
931.
932.         #Maurer’s Universal Statistical test
933.         #universaltest_quant = UN.Universal.statistical_test(self.quantized
    _rs_str, True)
934.         #re_universaltest_quant = round(universaltest_quant[0], 4)
935.         #universaltest_pri = UN.Universal.statistical_test(self.str_hash, T
    rue)
936.         #re_universaltest_pri = round(universaltest_pri[0], 4)
937.
938.         #Randomness Excursion
939.         #ran_excursion_quant = RE.RandomExcursions.random_excursions_test(s
    elf.quantized_rs_str, True)
940.         #re_ran_excursion_quant = round(ran_excursion_quant[0], 4)
941.         #ran_excursion_pri = RE.RandomExcursions.random_excursions_test(sel
    f.str_hash, True)
942.         #re_ran_excursion_pri = round(ran_excursion_pri[0], 4)
943.         #ran_excursion_var_quant = RE.RandomExcursions.variant_test(self.qu
    antized_rs_str, True)
944.         #re_ran_excursion_var_quant = round(ran_excursion_var_quant[0], 4)
945.         #ran_excursion_var_pri = RE.RandomExcursions.variant_test(self.str_
    hash, True)
946.         #re_ran_excursion_var_pri = round(ran_excursion_var_pri[0], 4)
947.
948.         # ApproximateEntropy
949.         approx_entropy_quant = AE.ApproximateEntropy.approximate_entropy_te
    st(self.quantized_rs_str, True, 2)
950.         re_approx_entropy_quant = round(approx_entropy_quant[0], 4)
951.         approx_entropy_pri = AE.ApproximateEntropy.approximate_entropy_test
    (self.str_hash, True, 2)
952.         re_approx_entropy_pri = round(approx_entropy_pri[0], 4)
953.
954.         '''# Spectral Test
955.         spectraltest_quant = ST.SpectralTest.sepctral_test(self.quantized_r
    s_str, False)
956.         re_spectraltest_quant = round(spectraltest_quant[0], 4)
957.         spectraltest_pri = ST.SpectralTest.sepctral_test(self.str_hash, Fal
    se)
958.         re_spectraltest_pri = round(spectraltest_pri[0], 4)'''
959.         """
960.         # Non overlapping Test
961.         non_overlapping_quant = NO.test(self.quantized_rs_str, len(self.qua
    ntized_rs_str))
962.         re_non_overlapping_quant = round(non_overlapping_quant[0], 4)
963.         non_overlapping_pri = NO.test(self.str_hash, len(self.str_hash))
964.         re_non_overlapping_pri = round(non_overlapping_pri[0], 4)
965.         """
```

```
966.         self.main_frame = tk.LabelFrame(self.master, text="Key Generation R
    esults", bg="WhiteSmoke", bd=2)
967.         self.main_frame.grid(row=1, column=1, sticky=tk.NSEW)
968.
969.         self.fig = Figure(figsize=(7, 3), dpi=30)
970.         self.canvas = FigureCanvasTkAgg(self.fig, self.main_frame)
971.         self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=1)
972.
973.         img1 = mpimg.imread("quantized.png")
974.         img2 = mpimg.imread("info_recon.png")
975.         img3 = mpimg.imread("privacy_amp.png")
976.         self.fig_plot = self.fig.add_subplot(131)
977.         self.fig_plot.imshow(img1)
978.         self.fig_plot.axes.get_xaxis().set_visible(False)
979.         self.fig_plot.axes.get_yaxis().set_visible(False)
980.         self.fig_plot.set_title("Quantization", size=22)
981.
982.         self.fig_plot = self.fig.add_subplot(132)
983.         self.fig_plot.imshow(img2)
984.         self.fig_plot.axes.get_xaxis().set_visible(False)
985.         self.fig_plot.axes.get_yaxis().set_visible(False)
986.         self.fig_plot.set_title("Information Reconciliation", size=22)

987.
988.         self.fig_plot = self.fig.add_subplot(133)
989.         self.fig_plot.imshow(img3)
990.         self.fig_plot.axes.get_xaxis().set_visible(False)
991.         self.fig_plot.axes.get_yaxis().set_visible(False)
992.         self.fig_plot.set_title("Privacy Amplification", size=22)
993.         #self.fig.tight_layout()
994.         self.canvas.draw()
995.
996.         self.data_frame = tk.Frame(self.master, bg="White")
997.         self.data_frame.grid(row=2, rowspan=2, column=1, sticky=tk.NSEW)
998.
999.         self.data_frame.rowconfigure(0, weight=1)
1000.               self.data_frame.columnconfigure(0, weight=2)
1001.               self.data_frame.columnconfigure(1, weight=1)
1002.
1003.               self.data = ttk.Treeview(self.data_frame, show="headings")
1004.               self.data.grid(row=0, column=0, sticky=tk.NSEW)
1005.               self.data['columns'] = ('Randomness Test', 'Quant', 'Priv. A
    mp')
1006.               self.data['height'] = 5
1007.               self.data.column("Randomness Test", width=70, anchor='center
    ')
1008.               self.data.column("Quant", width=15, anchor='center')
1009.               self.data.column("Priv. Amp", width=15, anchor='center')
1010.               self.data.heading("Randomness Test", text="Randomness Test")
1011.               self.data.heading("Quant", text="Quant")
1012.               self.data.heading("Priv. Amp", text="Priv. Amp")
```

```
1013.
1014.                 self.monobit_test = self.data.insert('', 0, values=('Monobit
    test', re_monobit_test_quant, re_monobit_test_pri), tag=('odd',))
1015.                 self.block_freq = self.data.insert('', 1, values=('Block fre
    quency', re_block_freq_quant, re_block_freq_pri), tag=('even',))
1016.                 self.cum_fwd = self.data.insert('', 2, values=('Cum.sums (fw
    d)', re_cum_sum_fwd_quant, re_cum_sum_fwd_pri), tag=('odd',))
1017.                 self.cum_rev = self.data.insert('', 3, values=('Cum.sums (re
    v)', re_cum_sum_rev_quant, re_cum_sum_rev_pri), tag=('even',))
1018.                 self.runs = self.data.insert('', 4, values=('Runs', re_run_t
    est_quant, re_run_test_pri), tag=('odd',))
1019.                 self.longest_block = self.data.insert('', 5, values=('Longes
    t one block', re_longest_block_quant, re_longest_block_pri), tag=('even',))
1020.                 self.serial1 = self.data.insert('', 6, values=('Serial 1', r
    e_serial1_quant, re_serial1_pri), tag=('odd',))
1021.                 self.serial2 = self.data.insert('', 7, values=('Serial 2', r
    e_serial2_quant, re_serial2_pri), tag=('even',))
1022.                 #self.universaltest = self.data.insert('', 8, values=('Maure
    r universal', re_universaltest_quant, re_universaltest_pri), tag=('odd',))
1023.                 #self.ran_excursion = self.data.insert('', 8, values=('Ran.e
    xcursion', re_ran_excursion_quant, re_ran_excursion_pri), tag=('odd',))
1024.                 #self.ran_excursion_var = self.data.insert('', 9, values=('R
    an.excursion.var', re_ran_excursion_var_quant, re_ran_excursion_var_pri), t
    ag=('even',))
1025.                 self.appro_entry = self.data.insert('', 8, values=('Approx.e
    ntropy', re_approx_entropy_quant, re_approx_entropy_pri), tag=('odd',))
1026.                 #self.spectraltest = self.data.insert('', 9, values=('DFT te
    st', re_spectraltest_quant, re_spectraltest_pri), tag=('even',))
1027.                 #self.non_overlapping = self.data.insert('', 10, values=('No
    n-
    overlapping', re_non_overlapping_quant, re_non_overlapping_pri), tag=('odd',
    ))
1028.
1029.
1030.                 self.data.tag_configure('odd', background='White')
1031.                 self.data.tag_configure('even', background='WhiteSmoke')
1032.
1033.
1034.                 # Encryption Test
1035.                 self.encrypt_test = tk.LabelFrame(self.data_frame, text='Enc
    ryption Test', bg="White", bd=2)
1036.                 self.encrypt_test.grid(row=0, column=1, sticky=tk.NSEW)
1037.
1038.                 self.encrypt_test.rowconfigure(0, weight=2)
1039.                 self.encrypt_test.rowconfigure(1, weight=1)
1040.                 self.encrypt_test.columnconfigure(0, weight=1)
1041.
1042.                 self.v = tk.StringVar()
1043.                 self.v.set("")
1044.                 self.info = tk.Label(self.encrypt_test, textvariable=self.v,
     fg = "black", bg="White", font=("Times New Roman", 14))
```

```
1045.               self.info.grid(row=0, column=0)
1046.
1047.               encrypt_font = tf.Font(family="Times New Roman", size=12)
1048.               self.decrypt_button = tk.Button(self.encrypt_test, text="Dec
    ryption", font=encrypt_font, bg="WhiteSmoke", activeforeground="red", comma
    nd=self.Decryption)
1049.               self.decrypt_button.grid(row=1, column=0, sticky=tk.N)
1050.
1051.
1052.
1053.
1054.
1055.           def Reset(self) :
1056.
1057.               self.animate.event_source.start()
1058.
1059.               self.main_frame = tk.LabelFrame(self.master, text='Key Gener
    ation', bg="WhiteSmoke", bd=2)
1060.               self.main_frame.grid(row=1, rowspan=2, column=1, sticky=tk.N
    SEW)
1061.
1062.               self.fig = Figure(figsize=(5, 2), dpi=60)
1063.               self.fig_plot = self.fig.add_subplot(111)
1064.               self.canvas = FigureCanvasTkAgg(self.fig, self.main_frame)
1065.               self.canvas.draw()
1066.               self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=1)
1067.
1068.               # Data
1069.               self.data_frame = tk.LabelFrame(self.master, bg="White", tex
    t="Statistic")
1070.               self.data_frame.grid(row=3, column=1, sticky=tk.NSEW)
1071.               self.data_frame.rowconfigure(0, weight=1)
1072.               self.data_frame.columnconfigure(0, weight=1)
1073.
1074.               self.data = ttk.Treeview(self.data_frame, show="headings")
1075.               self.data.grid(row=0, column=0, sticky=tk.NSEW)
1076.               self.data['columns'] = ('AP side', 'Results')
1077.               self.data['height'] = 4
1078.               self.data.column("AP side", width=120, anchor='center')
1079.               self.data.column("Results", width=10, anchor='center')
1080.               self.data.heading("AP side", text="AP side")
1081.               self.data.heading("Results", text="Results")
1082.
1083.               self.frame_num = self.data.insert('', 0, values=('number of
    packet received',))
1084.               self.matched_num = self.data.insert('', 3, values=('number o
    f packet after matching',))
1085.               self.frame_loss = self.data.insert('', 1, values=('number of
     packet loss',))
```

```
1086.             self.frame_lossrate = self.data.insert('', 2, values=('packe
    t loss rate',))
1087.
1088.             self.timestamp=[]
1089.             self.rssi=[]
1090.             self.time_array=([])
1091.             self.rssi_array=([])
1092.             self.matched_rssi = []
1093.
1094.
1095.
1096.         def Decryption(self) :
1097.
1098.             def decrypt(text):
1099.                 cipher = AES.new(self.key, AES.MODE_CBC, self.key)
1100.                 plain_text = cipher.decrypt(a2b_hex(text))
1101.                 return plain_text.rstrip('\0')
1102.
1103.             h = hashlib.md5(self.str_hash)
1104.             self.key = h.hexdigest()[8:-8]
1105.
1106.             recvd=""
1107.             s =socket.socket(socket.AF_INET, socket.SOCK_STREAM)
1108.             host = "192.168.3.20"
1109.             port = 12345
1110.             s.connect((host, port))
1111.
1112.             s.send(b'1')
1113.
1114.             length=json.loads(s.recv(1024).decode())
1115.
1116.             if length >= 1024 :
1117.                 num_index = int(length/1024) + 1
1118.             else :
1119.                 num_index = 1
1120.             #num_index=int(length/1024)+2
1121.
1122.             for i in range(0, num_index) :
1123.                 recvd = recvd + s.recv(1024).decode()
1124.
1125.             e = json.loads(recvd)
1126.             print "encrypted text: ", e
1127.
1128.             self.de_text = decrypt(e)
1129.             s.close()
1130.             print "Finish decryption"
1131.
1132.             self.v.set(self.de_text)
1133.
1134.         def Quit(self) :
1135.             sys.exit(0)
```

```
1136.
1137.       if __name__ == '__main__':
1138.
1139.           gui = tk.Tk()
1140.           k = Key_Generation(gui)
1141.           gui.mainloop()
```