

Implementation of three LWC Schemes in the WiFi 4-Way Handshake with Software Defined Radio

Yunjie Yi, Guang Gong and Kalikinkar Mandal

Department of Electrical and Computer Engineering
University of Waterloo,
Waterloo, Ontario N2L 3G1, Canada
{yunjie.yi, ggong, kmandal}@uwaterloo.ca

Abstract. In this paper, we present the implementation setup of the IEEE 802.1X 4-way handshake mutual authentication and IEEE 802.11i amending protected communication by using lightweight cryptography (LWC) schemes. The cryptographic functions including message integrity check (MIC) code (or equivalently message authentication code), key deviation function, and authenticated encryption are implemented by each of three LWC schemes, i.e., ACE, SPIX, and WAGE, in three different types of microcontrollers: 8-bit (Atmega128), 16-bit (MSP430f2013/MSP430f22370) and 32-bit (Cortex-m3lm3s9d96) microcontrollers. Software defined radio (SDR), contained two Universal Software Radio Peripheral (USRP) devices, is used to setup 802.11a physical layer orthogonal frequency division multiplexing (OFDM) transmission systems for the devices. We provide the experimental timing including cryptographic operations, OFDM modulation and radio transmission in the air.

Keywords: Implementation of LWC in WiFi mutual authentication by SDR · Internet-of-Things (IoT) · security and privacy · lightweight cryptography (LWC).

1 Introduction

With the rapid growth of internet-of-thing (IoT), it penetrates into our daily life deeply and poses an extraordinary effects on us. It connects wide range of devices, spanning from computers and servers through to smart devices. Most of those IoT devices are wireless connected such as sensors, actuators, radio frequency identification (RFID) tags, vehicular ad hoc networks (VANETs) and micro-controllers equipped with radio frequency (RF) transceivers capture. They communicate through various types of data regarding to, for instance, industrial and building control, e-health (e.g., medical devices embedded in our body or skin), smart home (such as lights, TV, thermostats, cameras, washing machines, dryers, and refrigerators), smart grid, self-driving cars, and other embedded systems [5].

It is known that the growth rate of IoT is approximate 20% per year, and the greatest risks for IoT are security, scalability, and reliability [10]. In 2017, IoT devices became more than the world's population and it is expected to be around 30 billion connected IoT devices by 2020 [5]. The value of users' data has given rise to new economic opportunities, such as *data markets*. Meanwhile, it generates new vulnerabilities for security and privacy caused by cyber attacks. In order to reach its portability and small size, most of IoT devices have limited computational power and limited power supply. According to current developments, the most rapid growing applications of IoT are smart cities ($\approx 26\%$), industrial IoT ($\approx 24\%$), connected health ($\approx 20\%$), smart homes ($\approx 14\%$), connected cars, wearable devices, and smart utilities [5].

IoT devices are connected by different wireless communication protocols. The major organizations for communication standardization have moved to support IoT systems. In newly amended IEEE 802.11ax [4] (i.e., WiFi systems, see the survey article in [8]), it targets at supporting established frequency bands with low power and low complexity operations, e.g., it may support the access point (AP) to talk to the client device (and vice versa) at data rates as low as 375 Kbps. Upcomming cellular 5G systems will also support IoT systems [1]. Recently, several attacks have been found against security protection mechanisms of WiFi [13] and 4G-LTE systems [16,23]. However, all those attacks employed the man-in-the-middle (MITM) attacks. In the wireless communications, in order to launch those MITM attacks, it needs to first jamming all transmitted signals. Thus, the overall timing for communications and cryptographic operations in those wireless systems are important factors to prevent those attacks.

In this paper, we aim at providing the experimental results on protected WiFi systems using lightweight cryptographic (LWC) schemes in IEEE 802.1X 4-way handshaking mutual authentication and key agreement

protocol and IEEE 802.11a physical layer orthogonal frequency-division multiplexing (OFDM) transmission systems. Those experimental results will provide some insight into actual security in practical IoT systems.

Our main contribution of this paper is as follows.

- (a) We implement the IEEE 802.1X 4-way handshake mutual authentication and key establishment by using the three LWC schemes which are ACE [11], SPIX [17] and WAGE [12]. The key derivation function (KDF) and message integrity check (MIC) generation function are implemented by using the authentication (AE) mode of the three LWC schemes. The AE mode is implemented on the three types of microcontrollers which are 8-bit, 16-bit and 32-bit. The code are written purely in assembly language in order to achieve better performance and more close to actual application environment. The running time of KDF and MIC generation on the microcontrollers has been measured by using Atmel Studio 7.0 and two IAR embedded workbenches which are for MSP430 and Cortex-M3.
- (b) The software defined radio (SDR) environment containing two Universal Software Radio Peripheral (USRP) devices has been set up to perform wireless communication for the IEEE802.1X 4-way handshake mutual authentication protocol. In detail, the standard WiFi communication protocol IEEE802.11a has been implemented in SDR, i.e., OFDM transmission. The transmission time has been captured on SDR interface which is GNU radio on Linux system.

The rest of the paper is organized as follows. In Section 2, we introduce ACE, SPIX, and WAGE, three LWC schemes, the IEEE802.1X 4-way handshake mutual authentication, IEEE802.11a physical layer communication standard, and SDR. In Section 3, we present the implementation and experiment setup for the 4-way handshake mutual authentication. In Section 4, we provide the experiment results of the 4-way handshake and data protection, and comparisons as well. We conclude this paper in Section 5.

2 Background and Preliminary

2.1 Three LWC schemes

ACE, SPIX and WAGE are three LWC schemes in the NIST LWC round 1 candidates.

- ACE is a LWC scheme [11] in the sponge construction, which provides both authenticated encryption (AE) mode and hash mode. The step function for ACE permutation is given by [11, Algorithm 1, Fig. 2.1], the AE mode is shown in [11, Algorithm 2, Fig. 2.3] and the hash mode is shown in [11, Algorithm 3, Fig. 2.4].
- SPIX is a LWC scheme [17] in the sponge mode structure and it only provides AE mode. The SPIX permutation is adopted from [17, Algorithm. 2, Fig. 2.3], and the AE mode is shown in [17, Algorithm. 1, Fig. 2.2].
- WAGE is the third LWC scheme [12] which we will consider in this paper. It only provides the AE mode. The step function for WAGE permutation is given by [12, Algorithm 1, Fig. 2.1], and the AE mode is shown in [12, Algorithm. 2, Fig. 2.2].

2.2 IEEE 802.11i: IEEE 802.1X 4-way handshake and data protection

In IEEE 802.1X, it specifies the wireless network consisting of wireless supplicants (or peers), i.e., wireless devices to be connected, and access points (AP) in IEEE 802.11 which play the role of an authenticator in IEEE 802.1X and extensible authentication protocol (EAP). A supplicant and an AP share a pairwise master key (PMK). IEEE 802.11 security solution is specified in IEEE802.11i amendment. In a summary, a device joins a network, it executes the 4-way handshake to negotiate a fresh session key. It will install this key after the fourth round of the handshake. Once the key is installed, it will be used to encrypt and authenticate traffic data frames using a protection algorithms. In other words, it has two phases as follows.

- (a) The 4-way handshake: conducts mutual entity authentication and generation of session keys, as shown in Figure 1.
- (b) Data protection: after the success of the 4-way handshake, it executes a data protection protocol, either CCMP (AES in counter mode for encryption and CBC MAC for integrity check and message authentication, i.e., MAC) or GCMP (AES in counter mode for encryption and polynomial hash for generating MAC).

WiFi data fields which carry identifiers, key information, replay counter, nonce, IV, Message Integrity Code (MIC), and transported data. Here we omit the format of EAP and only introduce the cryptographic functions employed in the 4-way handshakes. We also do not consider the case using group keys. The 4-way handshake generates a pairwise transient key (PTK) from PMK , and conducts a challenge-response protocol for mutual authentication. Figure 1 shows the messages flows, where only security related data fields are described and will be used in the implementation.

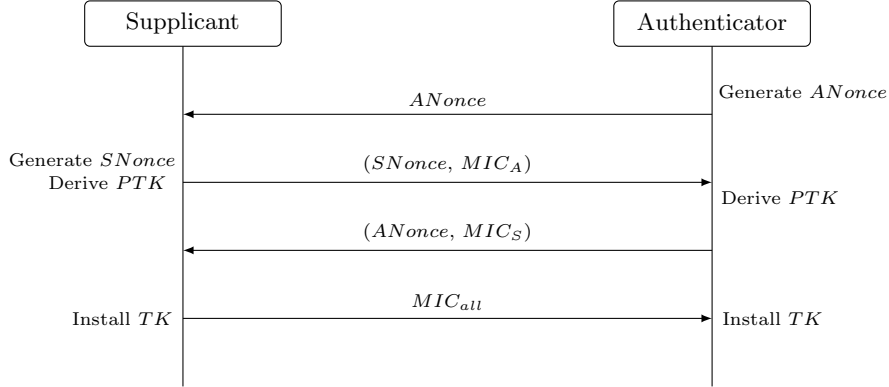


Fig. 1. IEEE 802.1X 4-way handshake

The nonces are 128 bits, the counter number r is also 128 bits. The key generated in the protocol is given as follows

$$PTK = KDF(PMK, ANonce || SNonce || AP\ MAC\ adr || STA\ MAC\ adr) = KCK || KEK || TK$$

where KDF is a key deviation function. The KCK is the first 128-bit in PTK and used to generate a MIC over the message it sends, KEK is the second 128-bit for encrypting the group key, and TK is the last segment for protecting traffic data where the length depends on a cipher suite selected. Furthermore, we use the following simplified format in our implementation.

$$\begin{aligned} MIC_A &= MIC(KCK, ANonce, r) \\ MIC_S &= MIC(KCK, SNonce, r) \\ MIC_{all} &= MIC(KCK, D, r + 1) \end{aligned} \quad (1)$$

where r is a replay counter number (see [9], Section 10.3.2 for the details), and D , 128 bits, carries the cipher suite.

2.3 IEEE 802.11a OFDM standard

For an easy to reference, we have put the definition of general OFDM systems into an Appendix. In this subsection, we introduce IEEE 802.11a physical layer OFDM system. According to [7], the channel usage for 64-IFFT OFDM system in the IEEE 802.11a is shown in Figure 2. The standard assumes that the subcarriers have been labeled from -32 to 31 . In detail, only 48 subcarriers are used to transmit data and 4 subcarriers ($-21, -7, 7, 21$) are used for pilot carriers for channel estimation as shown in Figure 2. Unused subcarriers

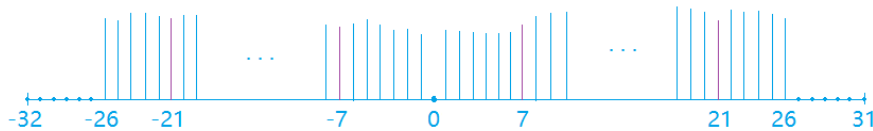


Fig. 2. 802.11a OFDM spectrum 64 subcarriers usage

from -31 to -25 and from 27 to 31 is to prevent the leakage of 52 subcarriers' sidelobes power to the outside of the total bandwidth. Finally, the DC is labeled as 0 subcarrier and IEEE 802.11a OFDM does not use the DC subcarrier to transmit the information; therefore, the DC subcarrier is inserted a complex number 0 at the carrier allocator before the IFFT.

In summary, the IEEE 802.11a OFDM has the following specifications.

- Total bandwidth is 20 MHz, and total subcarrier is 52 from -26 to 26 (not include DC at 0), and subcarriers from -32 to -27 and from 27 to 31 are not used. The DC subcarrier 0 is not used.
- Underlying modulation could be BPSK, QPSK, 16-QAM, and 64-QAM.
- There are 48 data subcarrier and 4 pilot subcarriers used for the channel estimation. The pilot subcarriers are located at $-21, -7, 7, 21$, and pilot symbols are modulated by BPSK.
- The information rate could be 6, 9, 12, 18, 24, 36, 48, 54 Mbits/s.

2.4 SDR overall

The overall connection of SDR is shown in Figure 3 [24]. SDR has two main parts which are USRP and GNU radio. USRP is the hardware consisting of ADC, DAC, low pass filter and mixer. GNU radio is a digital signal processing (DSP) software on Linux system. By setting up the IP address in a USRP block in GNU radio, GNU radio on PC can send and receive sample data from USRP hardware devices. Physically, an ethernet cable creates a connection between the USRP device and GNU radio. Since each USRP device has its own subnet, the PC needs two network cards to connect two USRP devices at the same time. Instead of using a gigabit ethernet switch [21], we use two gigabit ethernet adaptors to connect two USRP devices separately and set up their own IP addresses and subnets. After running SDR, the data transmission between USRP devices and GNU radio are recorded in real-time. As a result, we can use the QT-GUI-frequency-sink block in GNU radio to analyze the spectrum of the sampled data obtained from USRP devices in real-time.

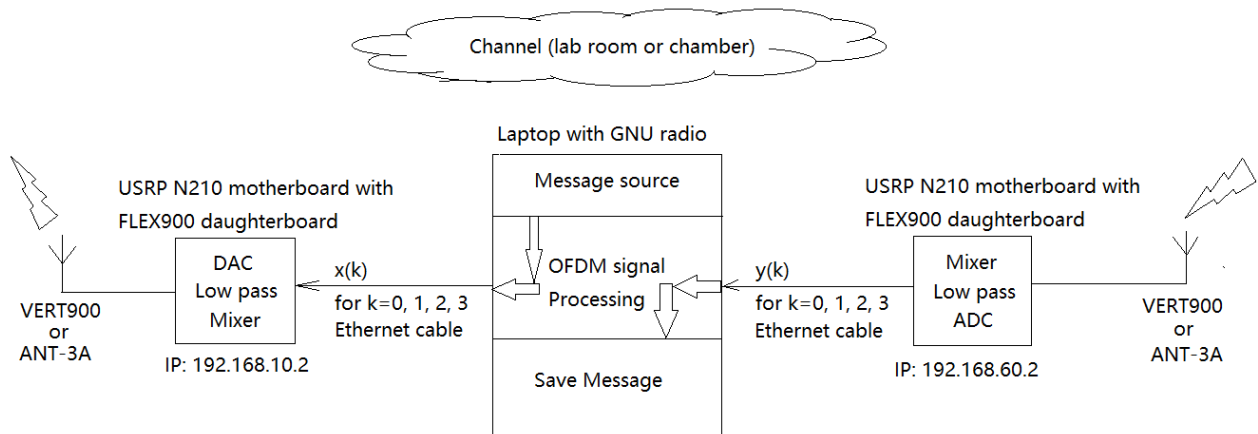


Fig. 3. Software defined radio physical connection

Basic theory of GNU radio and GNU radio companion GNU radio is a software working under GNU General Public License which is a copyleft license and free to use [24]. The GNU radio companion is a graphical software combining those functions to form graphical intergraded blocks, and It helps us focus on prototyping and signal processing graphically. In this paper, we use GNU radio companion Ver. 3.7.9 on Ubuntu 16.04 OS which is further running on visual machine on Windows 10.

In Figure 3, GNU radio on a PC is used to process digital signals in a communication system, and the connected USRP devices process the analogy signals [24]. For the sender part of SDR, the ethernet cable is used to transfer the discrete time sample data from GNU radio to the USRP device on the left side of Figure 3. Afterwards, the USRP device converts it into analogy signals and passes them through the low pass filter, the mixer and the antenna [24]. For the receiver part of SDR, the USRP device on the right side of Figure 3 receives the analogy signal from its antenna and converts it to baseband signals. Finally, the filtered signals will be converted into discrete time signal, then send to PC through the ethernet cable.

Data type in GNU radio There are five main data types used in GNU radio which are complex, float, byte, short and integer [24]. In detail,

- One integer contains 4 bytes.
- One short integer contains 2 bytes.
- One float number contains 32 bits.
- One complex number contains one 32-bit real float number and one 32-bit imaginary float number.

3 Implementation of Mutual Authentication Using LWC

3.1 SPIX, ACE and WAGE as cipher suites in IEEE 802.11i protections

We list the parameter sets for authenticated encryption with associated data (AEAD) functionalities for those three permutations. The length of each parameter is given in bits and d denotes the amount of processed data (including both associated data (AD), for authentication only, and message (M) for both encryption and authentication) before a re-keying is done. State n represents the number of internal state bits in the permutation, Key K is the key size, Rate r is data size for each call of executing a permutation in the sponge mode, and Tag t is the size of the output authentication tag.

For each executing AEAD, the algorithm processing $r\ell_{AD}$ bits AD data and $r\ell_M$ bits message data (it needs padding if AD/M data is not a multiple of r).

Algorithm	State n	Rate r	Key k	Tag t	Data $\log_2(d)$
ACE	320	64	128	128	124
SPIX	256	64	128	128	60
WAGE	259	64	128	128	60

(2)

In the IEEE 802.1X 4-way handshake, it needs two functionalities, one is an MIC generation function and the other is a KDF. Since MIC is 128-bit, in this case, it calls directly those algorithms with $\ell_M = 0$. For KDF, since all three algorithms have key size 128 bits, so, TK is 128 bits. The algorithm with input of $\ell_M = 0$ and outputs 3×128 bits for which it needs to run the permutation 4 more times for each cipher suite.

In the protected communication phase, it directly calls one of those algorithms with $\ell_M \neq 0$, but ℓ_{AD} could be either zero or nonzero.

3.2 Implementation of LWC schemes in IAR and Atmel Studio by assembly language

The LWC, SPIX, ACE and WAGE are written in assembling language. The SPIX is written in assembling language designated to 8-bit, 16-bit and 32-bit microcontrollers which are Atmega128, MSP430f2013 and Cortex-m3lm3s9d96 respectively. The ACE permutation is more complex than the SPIX permutation, so it is written in assembling language designated to only 16-bit and 32-bit microcontrollers which are MSP430f2013 and Cortex-m3lm3s9d96 respectively. Lastly, the WAGE is written in assembling language designated to 8-bit, 16-bit and 32-bit microcontrollers which are Atmega128, MSP430f2370 and Cortex-m3lm3s9d96 respectively. The reason we change the 16-bit microcontroller from MSP430f2013 in SPIX implementation to MSP430f2370 in WAGE implementation is that the latter one has the same specification as the former one, but a larger space to save the constants of WAGE. The IAR embedded workbenches for MSP430 and Cortex-m3 and the Atmel Studio 7.0 for Atmega128 have been used to import code into the three microcontrollers respectively and calculate the clock cycles for the corresponding permutations and the modes.

For the implementation of WAGE, we introduce a memory moving technology used to reduce the clock cycle cost on shifting for each permutation round. Instead of loading everything into registers, the 259 bits of the WAGE state have been continuously stored in random access memory (RAM). In order to finish the permutation, we extract each corresponding 7 bits from RAM into one register. Those extracted data are used to execute the permutation operations such as checking with look-up tables, XOR operations and so on. Once we finish one round of permutation, the new data will be stored next to the current 259 bits. The absolute locations of those extracted data in the RAM are not fixed but the relative locations to the first bit of those 259 bits are fixed. Therefore, we only need the initial memory location of the first bit of those 259 bits, which is the same for each round, denoted as *INITIAL*, the integer numbers of the relative locations, denoted as set λ , and an integer variable *INDEX* to record the current round number. Then the current locations of those extracted data will be the set $\{INITIAL + INDEX + t \mid t \in \lambda\}$. Once the permutation finished, we set *INDEX* = 0 and copy the final state to the initial state location in RAM. Then we continue to the next WAGE permutation.

The SPIX permutation with 18 rounds, SPIX AE mode with $\ell_{AD} = 0, \ell_M = 16$ and SPIX AE mode with $\ell_{AD} = 2, \ell_M = 16$ will be implemented on the three platform respectively. The memory usage, clock cycles are directly read from IAR embedded workbenches and Atmel Studio in debug mode. Throughput is calculated from CPU's frequency and total clock cycles, which is shown in Equation (3).

$$\eta = |\mathbf{m}|/(C/f), \quad (3)$$

where η is the throughput, $|\mathbf{m}|$ is the number of input message bits, C is the total clock cycles and f is the CPU frequency which is $16MHz$ for three microcontroller.

The general specifications of the three microcontrollers are summarized in Table 1. The first column is the microcontrollers and their applied range. The second column is the flash memory size which will be used to store the assembly code. The third column is about the RAM size to store the temporal running data. The last column is about the number of general-purpose registers and corresponding labels.

In order to generate PTK , the employed KDF is shown in Figure 4. The permutation blocks in Figures 4 and 5 could be the one of SPIX, ACE or WAGE permutations. The initial value is derived from the last 64 bits of $ANonce$ and $SNonce$, and PMK . By setting $l_{AD} = 0$ and $l_M = 6$, and removing the finalization phase of the authentication mode, we could have session keys, which are KCK , KEK and TK .

For the $MICs$ in the 4-way handshake, the MIC function is shown in Figure 5. It uses the AE mode with $l_{AD} = 4$. The session key KCK is used to generate the $MICs$. In Figure 1, MIC_A is generated from Figure 5 with $Nonce = ANonce$, and MIC_S is generated by setting $Nonce = SNonce$, and MIC_{all} is generated by setting $Nonce = D$. The permutation \mathcal{F} is one of the three permutations, i.e., $\mathcal{F} \in \{SPIX, ACE, WAGE\}$.

Table 1. Three types of microcontrollers

Microcontrollers	Flash memory size [kB]	RAM [kB]	Number of general-purpose register
ATmega128(SPIX, WAGE)	128	4.448	32(R0 - R31)
MSP430F2013(ACE, SPIX)	2.304	0.128	12 (R4 - R15)
MSP430F2370(WAGE)	33.024	2.048	12 (R4 - R15)
LM3S9D96(ACE, SPIX, WAGE)	524.288	131.072	13 (R0 - R12)

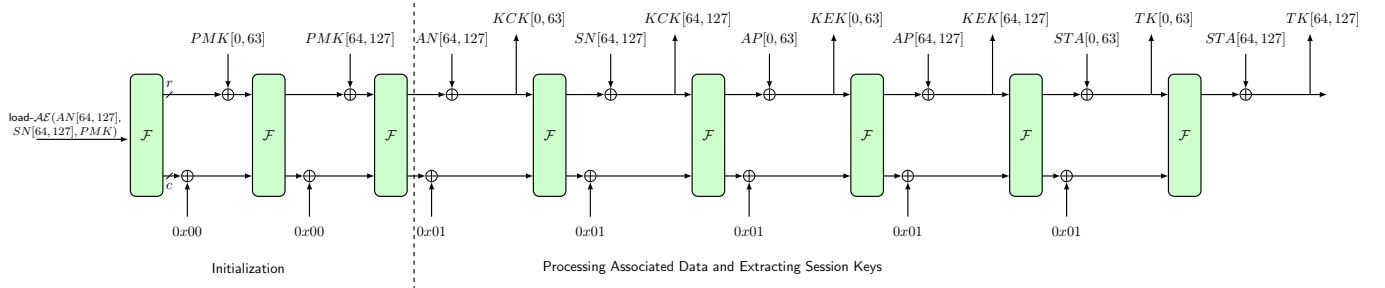


Fig. 4. A diagram of the key deviation function (KDF)

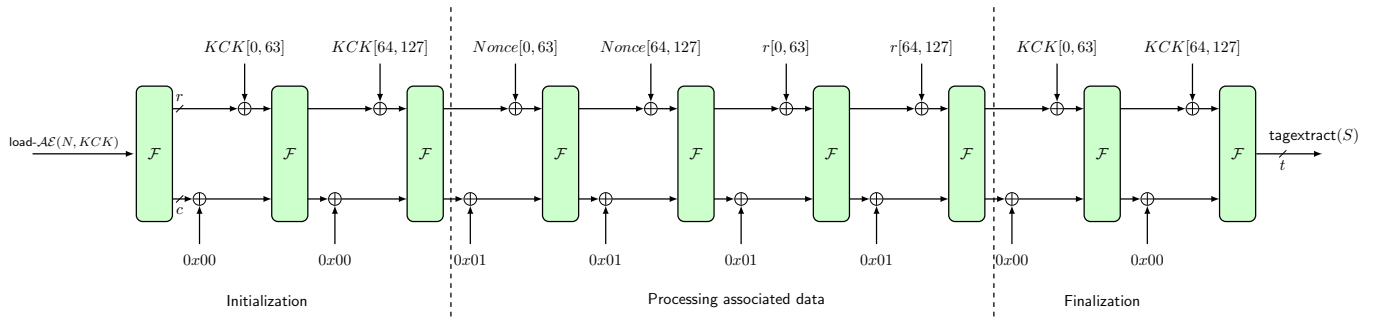


Fig. 5. MIC generation function

3.3 Implementation of OFDM system by SDR

The IEEE 802.11a OFDM sender's structure is shown in Figure 6. In detail, the message-source block in GNU radio is used to output bytes from a binary file to its next block, and it is set to repeat sending the message bit

stream automatically during the test. Each 96 bytes from the file-source block will be tagged in the stream-to-tagged-stream block. After that, the following blocks will manipulate each 96 message bytes at a time. For example, the packet-header-generator block generates 48 header bytes for each tagged message which is the tagged 96-byte. The repack-bits block in Figure 6 operates 1-byte at a time. We define 1-byte input in

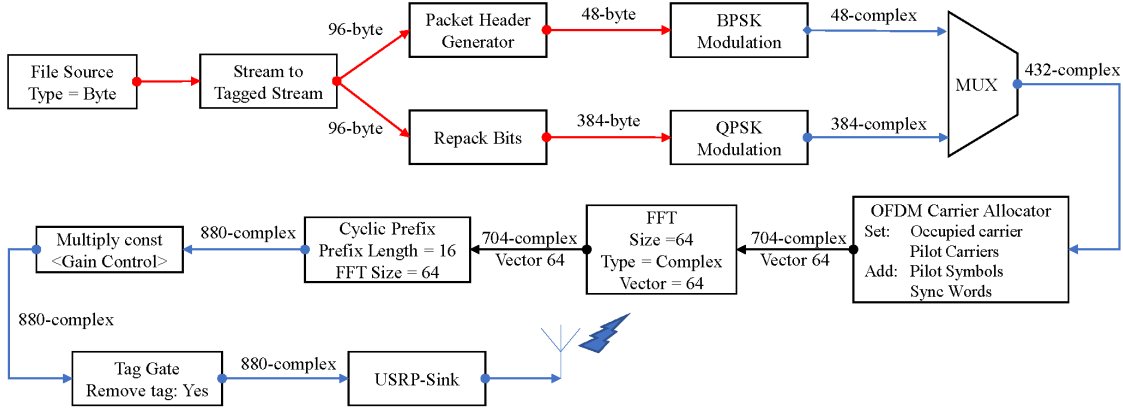


Fig. 6. OFDM sender

bit type as $\mathbf{a} = (a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$. The repack-bits block converts each 2-bit to an decimal number, $\mathbf{d} = (d_0, d_1, d_2, d_3)$. After that, each decimal number will be converted to a byte \mathbf{b} which is $\mathbf{b} = (b_0, b_1, b_2, b_3)$. The output of the repack-bits block is $\mathbf{b} = (b_3, b_2, b_1, b_0)$ which is in the endianness of LSB. Comparing the input \mathbf{a} with the output \mathbf{b} , it indicates that each input byte corresponds to four output bytes, which explains that each 96-byte input has 384-byte for the repack-bits block in Figure 6.

The BPSK-modulation block also converts each byte to a 8-byte complex number. In detail, it maps bytes $(00, 01)_{hex}$ to complex numbers $((-1, 0), (1, 0))_{decimal}$.

Similarly, a QPSK-modulation block converts each byte input into a complex number. In detail, it maps input bytes $(00, 01, 02, 03)_{hex}$ to output complex numbers $((-1/\sqrt{2}, -1/\sqrt{2}), (1/\sqrt{2}, -1/\sqrt{2}), (-1/\sqrt{2}, 1/\sqrt{2}), (1/\sqrt{2}, 1/\sqrt{2}))_{decimal}$ respectively.

The MUX block is used to combine each 48-complex header and 384-complex payload at a time. Therefore, the output of MUX block is 432 complex numbers in total, and it will be sent to the OFDM-carrier-allocator block.

The OFDM-carrier-allocator block in Figure 6 creates 11 complex vectors for each 432 complex numbers input, and those vectors are shown in Figure 7. The complex vectors are labeled as M_i for $i = 1, 2, \dots, 11$, and each vector contains 64 complex numbers as 64 subcarriers, where M_1 and M_2 are two synchronization words. Additionally, each header prime and each message prime in Figure 7 come from the header and message data after inserted 4 pilot carriers and 0 DC subcarrier. Namely, the pilot complex numbers $[1, 1, 1, -1]$ are inserted into the subcarriers $[-21, -7, 7, 21]$ respectively for each 64 subcarriers. Furthermore, the subcarriers from -32 to -27 and from 27 to 31 and subcarrier 0 are set to be complex value zeros. Thus, the format of subcarriers exactly matches the IEEE 802.11 standard in Figure 2.

The IFFT size in IFFT block is set to 64, so that IFFT block manipulates each 64 complex subcarriers at a time. In detail, it converts each 64 complex numbers which are discrete samples in frequency domain to 64 complex numbers which are discrete samples in time domain.

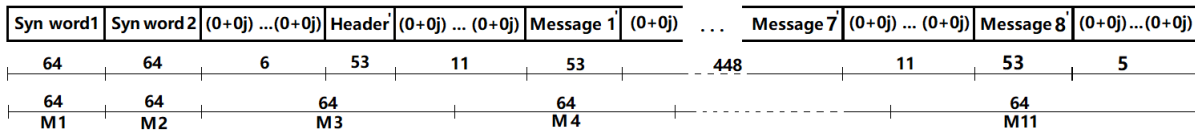


Fig. 7. Complex number stream after the OFDM carrier manipulator block

The cyclic prefix (CP) block inserts 16 complex numbers CP at the beginning of the each 64 complex numbers input. The prefix is the copy of the last 16 complex numbers of the 64 complex numbers.

The multiply-const block is used to multiply each input complex number by a constant number in order to adjust the gain of signals. The constant number is set from 0.01 to 0.03 for the implementation by using USRPs. In other words, if the constant number is lower than 0.01, then the SNR will be too small. In contrast, if constant number is higher than 0.03, the USRP will be saturated for high SNR. SDR will get high BER for both situations. Note that this constant number is the reference number for USRP devices which may not be linearly proportion to the sending signal's power, and its range is not accurate for each USRP device. Finally, the tag-gate block is used to remove the tag which is an internal variable passed by blocks.

The USRP-sink block in the GNU radio companion provides an interface to setup the parameters of the USRP device. In detail, the USRP block has parameters IP address, center frequency and sample rate. In our experiments, the USRP-sink block is used to set the parameters for the USRP sender. Its IP is set to be $addr = 192.168.10.2$, the center frequency 892MHz. Similarly, USRP-source block is the receiver which has the same parameter as the sender except that the IP is set to be $addr = 192.168.60.2$. The OFDM receiver

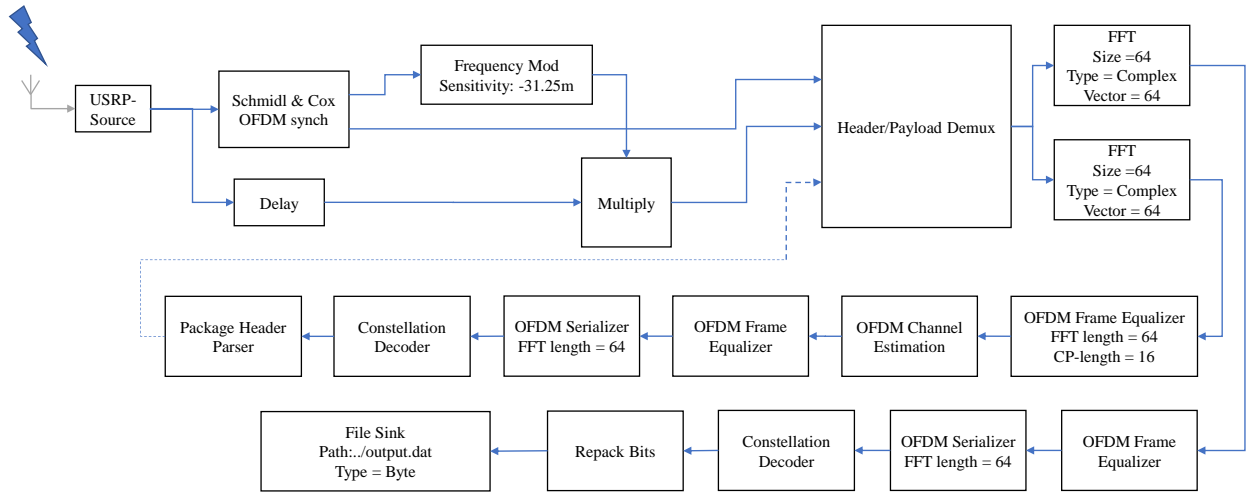


Fig. 8. OFDM receiver

is shown in Figure 8. The OFDM demodulation procedures included header detector, FFT, frame equalizer, OFDM serializer, underlying demodulation and repack are put in one block as shown in Figure 8, which used Schmidl OFDM Synchronization block given by [18] in order to increase the efficiency of frequency and timing synchronization for OFDM. The file-sink block in Figure 8 is used to save the demodulated message bytes, and used to get BER by comparing with the original sending data.

3.4 Experiment setup for the IEEE 802.1X 4-way handshake

The time of generating tags is calculated from IAR. The time for transmitting is captured in USRP by using Tag Debug block which could record the time for transmitting each 96 bytes message. Then we follow the 4-way handshake in Figure 1 using Atmel Studio, IAR and two USRP N210 devices to generate, send and receive data.

The OFDM system is a tagged system, the sending message has to be the multiple of 96 bytes or the last 96 bytes won't be sent. Therefore, we pad zeros after the $ANonce$, $SNonce||MIC_A$, $ANonce||MIC_S$ and MIC_{all} to reach 96 bytes for each of them. Therefore, the 4-way handshake needs to transmit $4 * 96 = 384$ bytes.

For the KDF and the MIC generation function, they are implemented by the three LWC schemes in authentication mode with $l_{AD} = 0$, $l_M = 6$ and $l_{AD} = 4$, $l_M = 0$ respectively. The generation time will be calculated in IAR and Atmel Studio. The 4-way transmission time will be captured in SDR. Finally, the total time for the 4-way handshake will be the sum of 4-way transmission time and the generating time for both session keys and MICs.

For the data protection phase, we set our experiments for two cases. One is to set the number of AD blocks l_{AD} to be 0 and the number of message blocks l_M to be 16 for absorbing message. We record the time for SPIX, ACE and WAGE in the AE mode in USRP interface. The second case is to set the number of AD blocks to be 2 and the number of message blocks to be 16.

4 Implementation Results and Comparisons

The implementation results of the IEEE 4-way handshake authentication are shown in Table 2. The last second column is the 4-way transmission time for the 4-way handshake mutual authentication, which is the time to transmit $6 * 128 = 768$ bits, e.g., transmitting 768 bits costs 0.7 seconds by using the SPIX in Table 2, but it does not include the time of generating session keys and *MICs*. The last column of Table 2 is the entire authentication time including the 4-way transmission time from the last second column and the running time of the *KDF* and the *MIC* generation function, which is derived from Equation (4).

$$T_{auth} = T_{4-way-tx} + 2 * T_{KDF} + 3 * T_{MIC}, \quad (4)$$

where $T_{4-way-tx}$ is the 4-way transmission time, T_{KDF} is the time of running KDF once, and T_{MIC} is the time of running MIC function once. The AE mode of the three LWC schemes are set to be $l_{AD} = 0$, $l_M = 16$ and $l_{AD} = 2$, $l_M = 16$. The generation time for tags and the one round of communication time are in the last two columns of Tables 3, 4 and 5.

Table 2. Performance of *KDF* by AE($l_{AD} = 0$, $l_M = 6$) and *MIC* by AE($l_{AD} = 4$, $l_M = 0$) on microcontrollers at a clock frequency of 16 MHz and time consumption of the IEEE 802.1X 4-way handshake mutual authentication and key establishment

Cryptographic \mathcal{F}	Platform	Function	Memory usage [Bytes]		Setup [Cycles]	Throughput [Kbps]	Gen-time [ms]	4-way-Tx-time [s]	Auth-Time [ms]
			SRAM	Flash					
SPIX	8-bits ATmega128	KDF	175	1586	705314	23.23	44.08	0.70	956.40
		MIC	175	1634	897225	18.26	56.08		
	16-bits MSP430F2013	KDF	50	1562	286679	57.15	17.92	0.69	794.09
		MIC	50	1580	363991	45.01	22.75		
	32-bits LM3S9D96	KDF	408	1230	59140	277.04	3.70	0.70	721.50
		MIC	408	1326	75132	218.07	4.70		
ACE	16-bits MSP430F2013	KDF	330	1720	550752	29.75	34.42	0.71	895.03
		MIC	330	1738	619701	26.44	38.73		
	32-bits LM3S9D96	KDF	599	1826	102762	159.44	6.42	0.73	764.50
		MIC	599	1790	115561	141.78	7.22		
WAGE	8-bits ATmega128	KDF	808	4448	139478	117.47	8.72	0.71	756.78
		MIC	808	4516	156491	104.70	9.78		
	16-bits MSP430F2013	KDF	46	4518	166993	98.11	10.44	0.72	776.01
		MIC	46	4536	187340	87.46	11.71		
	32-bits LM3S9D96	KDF	3084	6278	107071	153.02	6.69	0.69	725.91
		MIC	3084	6382	120190	136.32	7.51		

The one-way data encryption implementation results of SPIX, ACE and WAGE on authenticated encryption are shown in Tables 3, 4 and 5, respectively, where the memory usage, setup, clock cycles, throughput and generation time are imported from the microcontroller implementation we did in [11, 12, 17]. The transmission time is captured on SDR and represents that the time to transmit 1024 bits message or encrypted data and its 128 bit tag from a device, implemented by an USRP, to an access point (AP), implemented by the other USRP, i.e., AE.

Note that the generation time in Tables 2, 3, 4 and 5 is the time of running the corresponding function or AE mode one time only. On above experiments, the frame size of USRP is 1472 bytes, and the average frame rate of USRP is around 16.82 Kbps.

The AES implementation written in C on 8-bit AVR microcontrollers from [14] shows that the throughput of AES-128 permutation is $10180 * 8 * 2/1000 = 162.880Kbps$ by setting CPU frequency to be 16 MHz. Comparing with the results, the WAGE written in assembly gives higher throughput, which is $217.98Kbps$, on the same 8-bit microcontroller. Also, SPIX and ACE permutations give higher throughput on 32-bit microcontroller which is $393.76Kbps$ and $286.78Kbps$ respectively than those of the C code implementation of AES-128 permutation on 8-bit microcontroller.

When the AES is written in assembly language, the throughput of AES-128 permutation is $43671 * 8 * 2/1000 = 698.736Kbps$, which is higher than those of our implementation. However the block sizes of SPIX, ACE and WAGE are 256, 320 and 259 bits respectively which are more than twice as much as the blocksize of AES-128 in [14].

Table 3. Performance of SPIX AE mode on microcontrollers at a clock frequency of 16 MHz IEEE 802.11i data protection protocol

Cryptographic	Platform	Memory usage [Bytes]		Setup [Cycles]	Throughput [Kbps]	Gen-time [ms]	Tx-time [s]
		SRAM	Flash				
SPIX Perm-18	8-bits ATmega128	161	1262	128377	31.91	8.02	N/A
	16-bits MSP430F2013	24	1409	52294	78.33	3.27	
	32-bits LM3S9D96	352	946	10900	375.78	0.68	
SPIX-AE ($l_{AD} = 0, l_M = 16$)	8-bits ATmega128	175	1550	1667042	9.83	104.19	1.06
	16-bits MSP430F2013	50	1845	677818	24.17	42.36	1.08
	32-bits LM3S9D96	408	1210	139569	117.39	8.72	1.05
SPIX-AE ($l_{AD} = 2, l_M = 16$)	8-bits ATmega128	175	1644	1795322	9.13	112.21	1.08
	16-bits MSP430F2013	50	1891	730340	22.43	45.65	1.05
	32-bits LM3S9D96	424	1326	150313	109.00	9.39	1.07

5 Concluding Remarks

We have implemented IEEE 802.1x 4-way handshake mutual authentication and key establishment by three LWC schemes, i.e., SPIX, ACE, and WAGE, on the three types of microcontrollers, i.e., 8-bit, 16-bit and 32-bit microcontrollers. All three have highest throughput by using 32-bit microcontroller. The throughput on the three microcontrollers are almost uniform for each of them. Additionally, the throughput of WAGE written in assembly is higher than the throughput of the AES-128 written in C on the 8-bit platform. Secondly, we also implemented the IEEE 802.11a physical layer OFDM transmission systems by software defined radio to simulate the 4-way handshake modulation and communication. In current IEEE 802.11i (as well as amending), the cipher suite has only one cipher, which is AES. In the protection phase, it has two schemes, i.e., CCMP and GCMP. Our experimental results including cryptographic operation phase and radio communication phase through software defined radio provide some design choices for IoT devices connected through WiFi to the Internet. Those experiments set-up facilitates further experimental research for anti jamming, location service attack, and entry point intrusion attacks.

The WiFi has transmission rate from 50 Mbps to 320 Mbps at distance of 100m from devices to an access point. In our experiments, we have USRP transmission rate around 16.82 Kbps which is much slower than the real WiFi system. However, this can be done by transferring the 4-way transmission time which is 0.7 seconds (see Table 2) for SPIX on 8-bits microcontroller for example to the time of $0.7 * 16.82/50000 = 0.235ms$ for the real WiFi system at transmission rate of 50Mbps. Therefore, the time for executing the cryptographic operations is the dominating factor in the 4-way handshake. However, the data transmission in the 5G or satellite communications [19] is much more expensive than those in the WiFi transmission.

For cellular systems, since 5G [1] will adopt 4G-LTE's authentication and key agreement (AKA) protocol. Once the full authentication is successful, it may execute multiple times of local authentication. In the local authentication, the AKA runs between a wireless device and the mobility management entity is a 2-round authentication protocol, since it is a sequence number based authentication (see [9], Section 9.2). Our experiment set-up for LWC in microcontrollers and SDR can also be used to provide some performance evaluation of upcoming 5G security mechanisms. This will be discussed in our future work.

Table 4. Performance of ACE AE and Hash modes on microcontrollers at a clock frequency of 16 MHz IEEE 802.11i data protection protocol

Cryptographic	Platform	Memory usage [Bytes]		Setup [Cycles]	Throughput [Kbps]	Gen-time [ms]	Tx-time [s]
		SRAM	Flash				
ACE Perm	16-bits MSP430F2013	304	1456	69440	73.73	4.34	N/A
	32-bits LM3S9D96	523	1598	13003	393.76	0.81	
ACE-AE ($l_{AD} = 0, l_M = 16$)	16-bits MSP430F2013	330	1740	1445059	11.34	90.32	1.06
	32-bits LM3S9D96	559	1790	269341	60.83	16.83	1.07
ACE-AE ($l_{AD} = 2, l_M = 16$)	16-bits MSP430F2013	330	1786	1582892	10.35	98.93	1.08
	32-bits LM3S9D96	559	1858	294988	55.54	18.44	1.08
ACE-Hash ($l_M = 2, j = 4$)	16-bits MSP430F2013	330	1682	413056	4.96	25.82	N/A
	32-bits LM3S9D96	559	1822	77114	26.56	4.82	
ACE-Hash ($l_M = 16, j = 4$)	16-bits MSP430F2013	330	1684	1375672	11.91	85.98	N/A
	32-bits LM3S9D96	559	1822	256524	63.87	16.03	

Table 5. Performance of WAGE AE mode on microcontrollers at a clock frequency of 16 MHz in IEEE 802.11i data protection protocol

Cryptographic	Platform	Memory usage [Bytes]		Setup [Cycles]	Throughput [Kbps]	Gen-time [ms]	Tx-time [s]
		SRAM	Flash				
WAGE Perm	8-bits ATmega128	802	4132	19011	217.98	1.19	N/A
	16-bits MSP430F2370	4	5031	23524	176.16	1.47	
	32-bits LM3S9D96	3076	5902	14450	286.78	0.9	
WAGE-AE ($l_{AD} = 0, l_M = 16$)	8-bits ATmega128	808	4416	362888	45.15	22.68	1.08
	16-bits MSP430F2370	46	5289	433105	37.83	27.07	1.09
	32-bits LM3S9D96	3084	6230	278848	58.76	17.43	1.06
WAGE-AE ($l_{AD} = 2, l_M = 16$)	8-bits ATmega128	808	4502	397260	41.24	24.83	1.05
	16-bits MSP430F2370	46	5339	474067	34.56	29.63	1.06
	32-bits LM3S9D96	3084	6354	305284	53.67	19.08	1.06

References

1. 5G PPP. 5G PPP phase1 security landscape. In *5G PPP Security WG, European Commission*, June 2017.
2. G. Andrea. *Wireless Communications*. Cambridge University Press, 2005.
3. R. W. Chang. Synthesis of band-limited orthogonal signals for multichannel data transmission. *The Bell System Technical Journal*, 45(10):1775–1796, Dec 1966.
4. IEEE 802.11 Working Group et al. 802.11ax - ieee draft standard for information technology – telecommunications and information exchange between systems local and metropolitan area networks. *IEEE Std*, 2019.
5. G. Guang. Securing internet-of-things. In *International Symposium on Foundations and Practice of Security*, pages 3–16. Springer, 2018.
6. X. Huang. Effect of dc offset on ofdm system with zero-padded suffix. In *2006 International Symposium on Communications and Information Technologies*, pages 503–506, Oct 2006.
7. Keysight Technologies Inc. Concepts of orthogonal frequency division multiplexing (ofdm) and 802.11 wlan. http://rfmw.em.keysight.com/wireless/helpfiles/89600b/webhelp/subsystems/wlan-ofdm/content/ofdm_basicprinciplesoverview.htm. Accessed: 2018-06-09.
8. E. Khorov, A. Kiryanov, A. Lyakhov, and G. Bianchi. A tutorial on ieee 802.11ax high efficiency wlangs. *IEEE Communications Surveys Tutorials*, 21(1):197–216, 2019.
9. C. Lidong and G. Guang. *Communication system security*. Chapman and Hall/CRC, 2012.
10. C. Louis. A roundup of 2018 enterprise Internet of Things forecasts and market estimates, 2018.
11. A. Mark, A. Riham, G. Guang, M. Kalikinkar, and R. Raghvendra. ACE: An authenticated encryption and hash algorithm. *Submission to NIST-LWC*, 2019.
12. A. Mark, A. Riham, G. Guang, M. Kalikinkar, R. Raghvendra, and Z. Nusa. Wage: An authenticated cipher. 2019.
13. V. Mathy and P. Frank. Key reinstallation attacks: Forcing nonce reuse in wpa2. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1313–1328. ACM, 2017.
14. G. Meiser, T. Eisenbarth, K. Lemke-Rust, and C. Paar. Efficient implementation of estream ciphers on 8-bit avr microcontrollers. In *2008 International Symposium on Industrial Embedded Systems*, pages 58–66, June 2008.
15. S. Ohno. Preamble and pilot symbol design for channel estimation in ofdm. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, volume 3, pages III–281–III–284, April 2007.
16. B. Ravishankar, H. Lucca, P. Shinjo, and S. Altaf. New privacy threat on 3g, 4g, and upcoming 5g aka protocols. *Proceedings on Privacy Enhancing Technologies*, 2019(3):108–127, 2019.
17. A. Riham, G. Guang, H. Morgan, M. Kalikinkar, and R. Raghvendra. Spix: An authenticated cipher submission to the nist lwc competition. 2019.
18. T. M. Schmidl and D. C. Cox. Robust frequency and timing synchronization for ofdm. *IEEE Transactions on Communications*, 45(12):1613–1621, Dec 1997.
19. J. Shengming. Marine internet for internetworking in oceans: A tutorial. *Future Internet*, 11(7):146, 2019.
20. H. Steendam. How to select the pilot carrier positions in cp-ofdm? In *2013 IEEE International Conference on Communications (ICC)*, pages 3148–3153, June 2013.
21. L. C. Tran, D. T. Nguyen, F. Safaei, and P. J. Vial. An experimental study of ofdm in software defined radio systems using gnu platform and usrp2 devices. In *2014 International Conference on Advanced Technologies for Communications (ATC 2014)*, pages 657–662, Oct 2014.
22. J. Vlaović, S. Rimac-Drlje, and G. Horvat. Overview of ofdm channel estimation techniques for dvb-t2 systems. In *2016 International Conference on Smart Systems and Technologies (SST)*, pages 75–80, Oct 2016.
23. T. Wu and G. Gong. The weakness of integrity protection for LTE. In *Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'13), April 17-19, 2013, Budapest*, pages 79–88, 2013.
24. Y. Yunjie. Implementation of ofdm encryption and a new frequency hopping system, 2018.

APPENDIX

Orthogonal frequency-division multiplexing (OFDM) system

In this subsection, we introduce the basic structure of the orthogonal frequency-division multiplexing (OFDM) system. In detail, it gives basic concepts of underlying modulation, inverse discrete Fourier transform (IDFT), discrete Fourier transform (DFT), the orthogonality and cyclic prefix (CP) for a general OFDM system.

In [3], the authors demonstrated a communication scheme called OFDM to transmit multiple messages simultaneously on a linear bandlimited channel without involving intersymbol interference (ISI) and inter-channel interference. The total bandwidth W has been divided into multiple sub-channels, and those sub-channels are overlapping with each other one by one. However, they will not affect each other during the transmission in a linear bandlimited channel due to the orthogonality of the subcarriers. The basic model of OFDM system contains serial-to-parallel conversion, underlying modulation, N-inverse fast fourier transform (N-IFFT) and digital-to-analog convertor (DAC) conversion at the sender side. At the receiver side, it contains analog-to-digital convertor (ADC) conversion, N-fast fourier transform (N-FFT), parallel-to-serial conversion and underlying demodulation (see [2]). From [22] and [20], pilot symbols and pilot carriers are used for channel estimation. The number of subcarriers N is equal to the size of the IFFT and FFT, and each subcarrier is orthogonal to each other.

The underlying modulation is also called sub-carrier modulation. The process of underlying modulation is done before the serial-to-parallel conversion. The purpose of the underlying modulation is to map the input bits into constellation in complex domain. The underlying modulations include binary phase-shift keying (BPSK), quadrature phase-shift keying (QPSK), quadrature amplitude modulation (QAM) and so on. The selection of those modulations depends on the channel condition and the communication regulation. The bit error probability of M-ary phase-shift keying (MPSK) is given in Equation (5) [2].

$$P_b \approx \frac{2}{\log_2 M} Q\left(\sqrt{\frac{2E_b}{N_0} \log_2 M \sin\left(\frac{\pi}{M}\right)}\right). \quad (5)$$

Under the same signal to noise (SNR), $\frac{E_b}{N_0}$, increasing the value M will increase the bit error probability.

IFFT is a significant part in the OFDM system. The IDFT is shown in Equation (6).

$$s_i = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} S_k e^{j\frac{2\pi ik}{N}}, i, k = 0, 1, \dots, N-1. \quad (6)$$

The IDFT is used at the OFDM sender to convert frequency domain samples to time domain samples. The IFFT has a lower complexity to get the time domain samples for the realization purpose in hardware.

The DFT is shown in Equation (6).

$$S_i = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} s_k e^{-j\frac{2\pi ik}{N}}, i, k = 0, 1, \dots, N-1. \quad (7)$$

The IDFT is used at the OFDM receiver to calculate the frequency domain samples from the time domain samples. The FFT is the low complexity method to calculate the DFT in hardware.

Orthogonality is a word to demonstrate that the frequency domain signals do not affect each other and the product integral between their time domain signals is zero. In fact, the samples before the IFFT in OFDM system are viewed as discrete frequency samples, and the IFFT will convert them to discrete time samples. Cyclic Prefix is used to reduce the ISI [2]. According to [15], if the duration of CP is longer than channel delay spread, the ISI will be completely removed. The reason that uses CP instead of using padding zeros is to avoid involving DC offset which increases the BER a lot [6]. The prefix interval generally will be $N/4$ which is 16 when N is equal to 64 in our case.