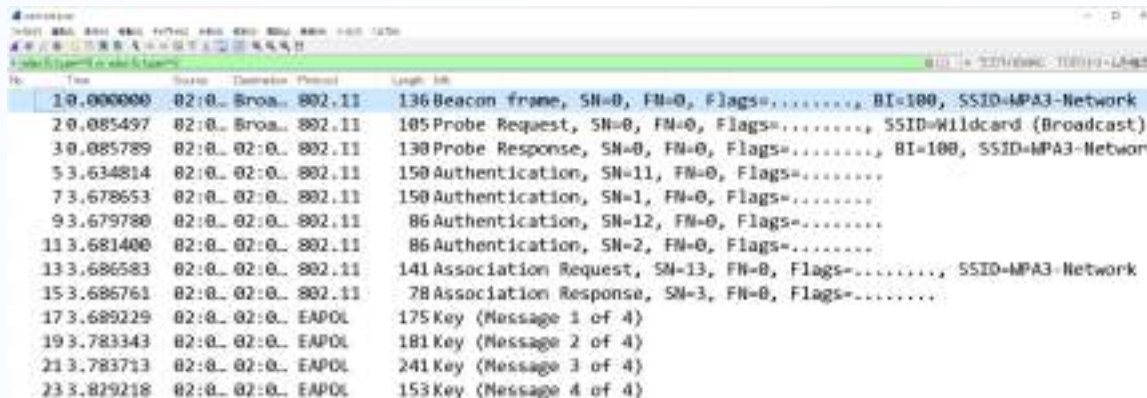


# Dissecting WPA3

#sf22us



The image shows a Wireshark packet capture of a WPA3 handshake. The packets are as follows:

| No. | Time     | Source            | Destination       | Protocol | Length | Info   |
|-----|----------|-------------------|-------------------|----------|--------|--|
| 10  | 0.000000 | 02:00:00:00:00:00 | 02:00:00:00:00:00 | 802.11   | 136    | Beacon frame, SN=0, FN=0, Flags=....., BI=100, SSID=MPA3-Network   |
| 20  | 0.085497 | 02:00:00:00:00:00 | 02:00:00:00:00:00 | 802.11   | 105    | Probe Request, SN=0, FN=0, Flags=....., SSID=Wildcard (Broadcast)  |
| 30  | 0.085789 | 02:00:00:00:00:00 | 02:00:00:00:00:00 | 802.11   | 130    | Probe Response, SN=0, FN=0, Flags=....., BI=100, SSID=MPA3-Network |
| 53  | 3.634814 | 02:00:00:00:00:00 | 02:00:00:00:00:00 | 802.11   | 150    | Authentication, SN=11, FN=0, Flags=.....                           |
| 73  | 3.678653 | 02:00:00:00:00:00 | 02:00:00:00:00:00 | 802.11   | 150    | Authentication, SN=1, FN=0, Flags=.....                            |
| 93  | 3.679780 | 02:00:00:00:00:00 | 02:00:00:00:00:00 | 802.11   | 86     | Authentication, SN=12, FN=0, Flags=.....                           |
| 113 | 3.681400 | 02:00:00:00:00:00 | 02:00:00:00:00:00 | 802.11   | 86     | Authentication, SN=2, FN=0, Flags=.....                            |
| 133 | 3.686583 | 02:00:00:00:00:00 | 02:00:00:00:00:00 | 802.11   | 141    | Association Request, SN=13, FN=0, Flags=....., SSID=MPA3-Network   |
| 153 | 3.686761 | 02:00:00:00:00:00 | 02:00:00:00:00:00 | 802.11   | 78     | Association Response, SN=3, FN=0, Flags=.....                      |
| 173 | 3.689229 | 02:00:00:00:00:00 | 02:00:00:00:00:00 | EAPOL    | 175    | Key (Message 1 of 4)   |
| 193 | 3.783343 | 02:00:00:00:00:00 | 02:00:00:00:00:00 | EAPOL    | 181    | Key (Message 2 of 4)   |
| 213 | 3.783713 | 02:00:00:00:00:00 | 02:00:00:00:00:00 | EAPOL    | 241    | Key (Message 3 of 4)   |
| 233 | 3.829218 | 02:00:00:00:00:00 | 02:00:00:00:00:00 | EAPOL    | 153    | Key (Message 4 of 4)   |

Megumi Takeshita  
Ikeriri network service



Materials: all trace files, python codes are here

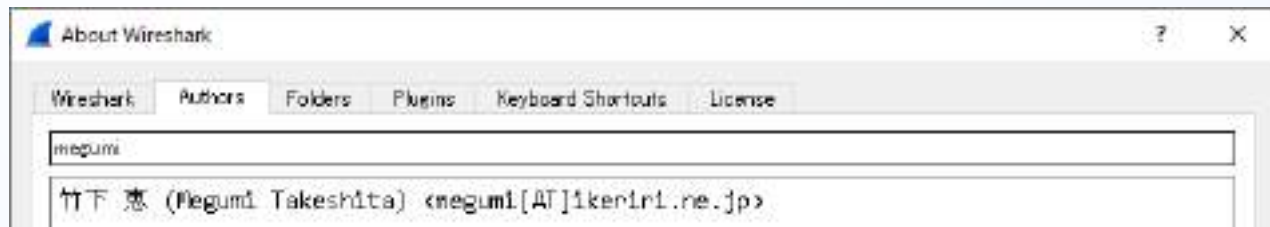
<https://www.ikeriri.ne.jp/sharkfest/03DissectingWPA3.zip>

# Megumi Takeshita, packet otaku

SharkFest'22 US  
Kansas City, MO  
July 9-14



- Founder, ikeriri network service co., ltd #sf22us
- Reseller of CACE technologies in 2008
- Worked SE/IS at BayNetwork, Nortel
- Wrote 10+ books about Wireshark
- Instruct Wireshark to JSDF and other company
- Reseller of packet capture / wireless-tools
- One of the contributors of Wireshark
- Translate Wireshark into Japanese



## Session Details

We need a new security standard in the 6/6E generation of WiFi WPA3 has an SAE ( Simultaneous Authentication of Equals ) authentication handshake and PMF ( Protected Management Frames ) mechanism. In this session, Megumi shows you WPA3 trace analysis using Wireshark. Follow the packets and understand the safe way to use the wireless network

## Materials

all trace files and Wireshark profiles are here

<https://www.ikeriri.ne.jp/sharkfest/03DissectingWPA3.zip>

# WPA2-PSK dissection

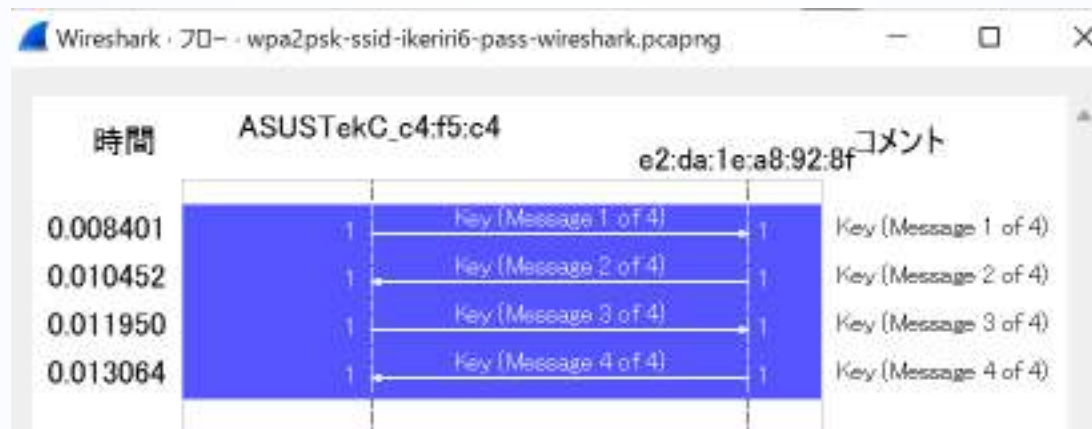


- Open wpa2psk-ssid-ikeriri6-pass-wireshark.pcapng this trace is captured and decrypted by TamoSoft CommView for WiFi in the WiFi6 environment
- It is a typical linkup process of WPA2-PSK between iphone13 (private MAC address) and ASUS AP

| Time        | PHY type       | Channel | data E | Signal (dBm) | Noise (dBm) | data M | Data rate | Retry             | Type/Subtype         |
|-------------|----------------|---------|--------|--------------|-------------|--------|-----------|-------------------|----------------------|
| 1 0.000000  | 802.11a (OFDM) | 56      |        | -39 dBm      | -93 dBm     |        |           | 6 Frame is not... | Authentication       |
| 2 0.001283  | 802.11a (OFDM) | 56      |        | -33 dBm      | -93 dBm     |        |           | 6 Frame is not... | Authentication       |
| 3 0.002746  | 802.11a (OFDM) | 56      |        | -41 dBm      | -93 dBm     |        |           | 6 Frame is not... | Association Request  |
| 4 0.005013  | 802.11a (OFDM) | 56      |        | -33 dBm      | -93 dBm     |        |           | 6 Frame is not... | Association Response |
| 5 0.008401  | 802.11a (OFDM) | 56      |        | -33 dBm      | -93 dBm     |        |           | 6 Frame is not... | QoS Data             |
| 6 0.010452  | 802.11a (OFDM) | 56      |        | -41 dBm      | -93 dBm     |        |           | 6 Frame is not... | QoS Data             |
| 7 0.011950  | 802.11a (OFDM) | 56      |        | -33 dBm      | -93 dBm     |        |           | 6 Frame is not... | QoS Data             |
| 8 0.013064  | 802.11a (OFDM) | 56      |        | -41 dBm      | -91 dBm     |        |           | 6 Frame is not... | QoS Data             |
| 9 0.072760  | 802.11ax (HE)  | 56      | 20     | -41 dBm      | -91 dBm     | 0xb    | 270.8     | 6 Frame is not... | QoS Data             |
| 10 0.072892 | 802.11ax (HE)  | 56      | 20     | -41 dBm      | -91 dBm     | 0xb    | 270.8     | 6 Frame is not... | QoS Data             |
| 11 0.073031 | 802.11ax (HE)  | 56      | 20     | -41 dBm      | -91 dBm     | 0xb    | 270.8     | 6 Frame is not... | QoS Data             |
| 12 0.073376 | 802.11ax (HE)  | 56      | 20     | -41 dBm      | -91 dBm     | 0xb    | 270.8     | 6 Frame is not... | QoS Data             |
| 13 0.073578 | 802.11a (OFDM) | 56      |        | -33 dBm      | -91 dBm     |        |           | 6 Frame is not... | Data                 |

# 4way handshake (common in WPA2 and WPA3)

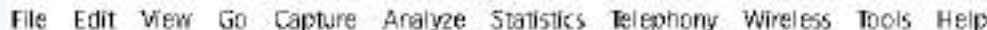
- There are 4 way handshake after link-up process,
- Enter “eapol” in Display filter and make FlowGraph



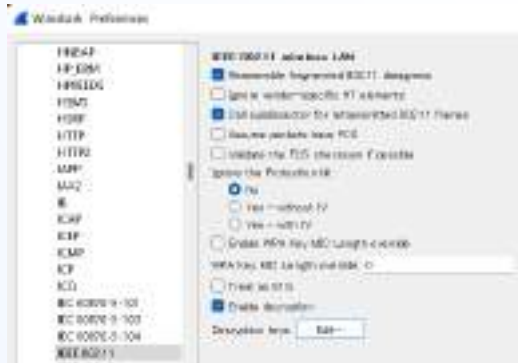
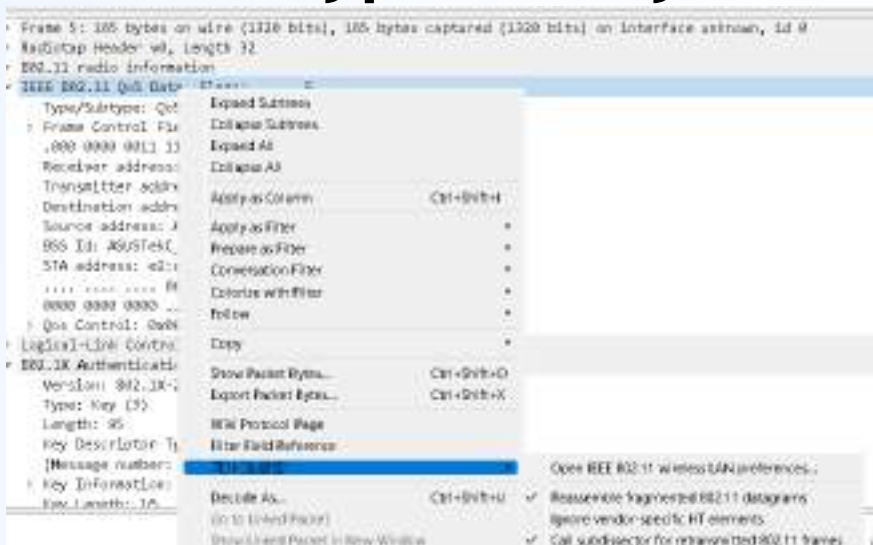
- At first, AP sends Key(Message 1 of 4) and STA reply Key (Message 2 of 4) to exchange PTK for unicast. Then AP sends Key (Message 3 of 4 ), STA replies. Key (Message 4 of 4 ) to confirm GTK for multicast.



- The passphrase is used for PSK(Pre Shared Key)
- PMK (Pairwise Master Key) is created by 4096 round times calculation of PBKDF2 hash function, with SHA1 algorithm, using PSK and SSID  
e.x.  $PMK = pbkdf2\_hmac('sha1', PSK, SSID, 4096)$
- Lets's start checking PMK using Wireshark  
Go back to the trace, wpa2psk-ssid-ikeriri6-pass-wireshark.pcapng, and check the #5 packet, the first EAPOL message 1/4

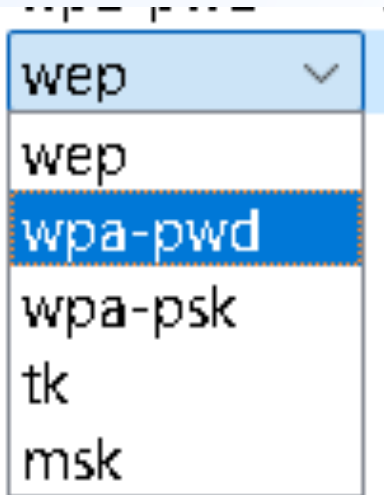
wpa2psk-ssid-ikeriri6-pass-wireshark.pcapngFile Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

# Set Decryption key in Wireshark



- Select “IEEE 802.11 QoS Data, Flags: .....F.” header, right-click and choose Protocol preference > Open IEEE802.11 wireless preferences...
- Click “Edit...” button of the Decryption keys

- We can set 5 styles of WEP and WPA decryption keys in Wireshark 3.6



| Key,    | stype                          | explanation   |
|---------|--------------------------------|---|
| wep     | WEP key (hex value)            | for WEP encryption  |
| wpa-pwd | Passphrase:SSID (ascii)        | For WPA1-PSK, WPA2-PSK  |
| wpa-psk | Raw PSK (hex value)            | 256-bit pre-shared ("raw") key<br><a href="https://www.wireshark.org/tools/wpa-psk.html">https://www.wireshark.org/tools/wpa-psk.html</a>   |
| tk      | Temporal Key (hex value)       | TK is used for actual encryption key, TK is a part of PTK (Pairwise-Transient-Key )   |
| msk     | Master Session Key (hex value) | Master Session key is derived from 802.1x EAP authentication process if you use WPA1-EAP, WPA2-EAP. You can set msk from the debug information of the AP and wireless lan controller. |



- Set key type as “wpa-pwd” and input key “wireshark:ikeriri6” in decryption key dialog

#sf22us

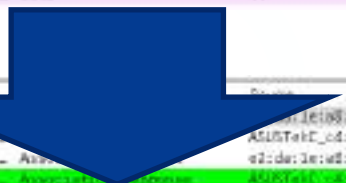


- Click “OK” to close WEP and WPA decryption Keys dialog
- “OK” again to close IEEE802.11 wireless LAN preference



# Confirm your QoS Data frames are decrypted

| No. | Time  | Pkty type      | Chn   | data E  | Signal (dBm) | Noise (dBm) | data L         | Data rate | Flags | Type/Subtype         | Source              | Destination                       | Length    | Info  |   |
|-----|-------|----------------|-------|---------|--------------|-------------|----------------|-----------|-------|----------------------|---------------------|-----------------------------------|-----------|---|---|
| 1   | 0.000 | 802.11a (OFDM) | 56    | -39 dBm | -93 dBm      |             | 6 Frame 1s     |           |       | Authentication       | e2:da:1e:a8:92:8f   | ASUSTekL_02:00:10:00:00:00        | 97        | Authentication, SM=031, FN=0, Flags=.....                     |   |
| 2   | 0.001 | 802.11a (OFDM) | 56    | -33 dBm | -93 dBm      |             | 6 Frame 1s     |           |       | Authentication       | A9:5f:8c:1c:4f:5c:4 | e2:da:1e:00:00:00:00:00           | 92        | Authentication, SM=0, FN=0, Flags=.....                       |   |
| 3   | 0.002 | 802.11a (OFDM) | 56    | -41 dBm | -93 dBm      |             | 6 Frame 1s     |           |       | Association Request  | e2:da:1e:a8:92:8f   | ASUSTekL_02:00:10:00:00:00        | 237       | Association Request, SM=030, FN=0, Flags=....., SSID=iterir16 |   |
| 4   | 0.005 | 802.11a (OFDM) | 56    | -33 dBm | -93 dBm      |             | 6 Frame 1s     |           |       | Association Response | A9:5f:8c:1c:4f:5c:4 | e2:da:1e:00:00:00:00:00           | 318       | Association Response, SM=1, FN=0, Flags=.....                 |   |
| 5   | 0.006 | 802.11a (OFDM) | 56    | -33 dBm | -93 dBm      |             | 6 Frame 1s     |           |       | QoS Data             | A9:5f:8c:1c:4f:5c:4 | e2:da:1e:00:00:00:00:00           | 168       | Key (Message 1 of 4)  |   |
| 6   | 0.016 | 802.11a (OFDM) | 56    | -41 dBm | -93 dBm      |             | 6 Frame 1s     |           |       | QoS Data             | e2:da:1e:a8:92:8f   | ASUSTekL_02:00:10:00:00:00        | 167       | Key (Message 2 of 4)  |   |
| 7   | 0.011 | 802.11a (OFDM) | 56    | -33 dBm | -93 dBm      |             | 6 Frame 1s     |           |       | QoS Data             | A9:5f:8c:1c:4f:5c:4 | e2:da:1e:00:00:00:00:00           | 237       | Key (Message 3 of 4)  |   |
| 8   | 0.011 | 802.11a (OFDM) | 56    | -41 dBm | -93 dBm      |             | 6 Frame 1s     |           |       | QoS Data             | e2:da:1e:a8:92:8f   | ASUSTekL_02:00:10:00:00:00        | 168       | Key (Message 4 of 4)  |   |
| 9   | 0.072 | 802.11ax (HE)  | 56 20 | -41 dBm | -91 dBm      | 0db         | 270.8 Frame 1s |           |       | QoS Data             | e2:da:1e:a8:92:8f   | IPV6mcast_02:00:10:00:00:00:00:00 | 170       | QoS Data, SM=0, FN=0, Flags=op.....T                          |   |
| 10  | 0.072 | 802.11ax (HE)  | 56 20 | -41 dBm | -91 dBm      | 0db         | 270.8 Frame 1s |           |       | QoS Data             | e2:da:1e:a8:92:8f   | ASUSTekL_02:00:10:00:00:00        | 126       | QoS Data, SM=1, FN=0, Flags=op.....T                          |   |
| 11  | 0.073 | 802.11ax (HE)  | 56 20 | -41 dBm | -91 dBm      | 0db         | 270.8 Frame 1s |           |       | QoS Data             | e2:da:1e:a8:92:8f   | Broadcast_02:00:10:00:00:00:00:00 | 426       | QoS Data, SM=2, FN=0, Flags=op.....T                          |   |
| 12  | 0.073 | 802.11ax (HE)  | 56 20 | -41 dBm | -91 dBm      | 0db         | 270.8 Frame 1s |           |       | QoS Data             | e2:da:1e:a8:92:8f   | IPV6mcast_02:00:10:00:00:00:00:00 | 368       | QoS Data, SM=3, FN=0, Flags=op.....T                          |   |
| 13  | 0.073 | 802.11a (OFDM) | 56    | -33 dBm | -93 dBm      |             | 6 Frame 1s     |           |       | Data                 | ::                  | ff02::1::                         | ff02::1:: | 136   | Neighbor Solicitation for fe80::1a42:c607:0001:27fa |



| No. | Time  | Pkty type      | Chn   | data E  | Signal (dBm) | Noise (dBm) | data L         | Data rate | Flags                | Type/Subtype               | Source                     | Destination                                       | Length  | Info |  |
|-----|-------|----------------|-------|---------|--------------|-------------|----------------|-----------|----------------------|----------------------------|----------------------------|---|---|------|--|
| 1   | 0.000 | 802.11a (OFDM) | 56    | -39 dBm | -93 dBm      |             | 6 Frame 1s     |           | Authentication       | ASUSTekL_02:00:10:00:00:00 | ASUSTekL_02:00:10:00:00:00 | 97 Authentication, SM=0, FN=0, Flags=.....        |   |      |  |
| 2   | 0.001 | 802.11a (OFDM) | 56    | -33 dBm | -93 dBm      |             | 6 Frame 1s     |           | Authentication       | ASUSTekL_02:00:10:00:00:00 | ASUSTekL_02:00:10:00:00:00 | 92 Authentication, SM=0, FN=0, Flags=.....        |   |      |  |
| 3   | 0.002 | 802.11a (OFDM) | 56    | -41 dBm | -93 dBm      |             | 6 Frame 1s     |           | Association Request  | ASUSTekL_02:00:10:00:00:00 | ASUSTekL_02:00:10:00:00:00 | 237 Association Request, SM=0, FN=0, Flags=.....  | SSID=iterir16   |      |  |
| 4   | 0.005 | 802.11a (OFDM) | 56    | -33 dBm | -93 dBm      |             | 6 Frame 1s     |           | Association Response | ASUSTekL_02:00:10:00:00:00 | ASUSTekL_02:00:10:00:00:00 | 318 Association Response, SM=1, FN=0, Flags=..... |   |      |  |
| 5   | 0.006 | 802.11a (OFDM) | 56    | -33 dBm | -93 dBm      |             | 6 Frame 1s     |           | QoS Data             | ASUSTekL_02:00:10:00:00:00 | ASUSTekL_02:00:10:00:00:00 | 168 Key (Message 1 of 4)                          |   |      |  |
| 6   | 0.016 | 802.11a (OFDM) | 56    | -41 dBm | -93 dBm      |             | 6 Frame 1s     |           | QoS Data             | ASUSTekL_02:00:10:00:00:00 | ASUSTekL_02:00:10:00:00:00 | 167 Key (Message 2 of 4)                          |   |      |  |
| 7   | 0.011 | 802.11a (OFDM) | 56    | -33 dBm | -93 dBm      |             | 6 Frame 1s     |           | QoS Data             | ASUSTekL_02:00:10:00:00:00 | ASUSTekL_02:00:10:00:00:00 | 237 Key (Message 3 of 4)                          |   |      |  |
| 8   | 0.011 | 802.11a (OFDM) | 56    | -41 dBm | -93 dBm      |             | 6 Frame 1s     |           | QoS Data             | ASUSTekL_02:00:10:00:00:00 | ASUSTekL_02:00:10:00:00:00 | 168 Key (Message 4 of 4)                          |   |      |  |
| 9   | 0.072 | 802.11ax (HE)  | 56 20 | -41 dBm | -91 dBm      | 0db         | 270.8 Frame 1s |           | QoS Data             | ::                         | FF02::1::                  | FF02::1::   | 170 Neighbor Solicitation for fe80::1a42:c607:0001:27fa |      |  |
| 10  | 0.072 | 802.11ax (HE)  | 56 20 | -41 dBm | -91 dBm      | 0db         | 270.8 Frame 1s |           | QoS Data             | ASUSTekL_02:00:10:00:00:00 | ASUSTekL_02:00:10:00:00:00 | 126 Who has 192.168.50.1? Tell 192.168.50.136     |   |      |  |
| 11  | 0.073 | 802.11ax (HE)  | 56 20 | -41 dBm | -91 dBm      | 0db         | 270.8 Frame 1s |           | QoS Data             | 0.0.0.0                    | 255.255.255.255            | 426 DHCP Request - Transaction ID 00ac9e7500      |   |      |  |
| 12  | 0.073 | 802.11ax (HE)  | 56 20 | -41 dBm | -91 dBm      | 0db         | 270.8 Frame 1s |           | QoS Data             | fe80::1a42:c607:0001:27fa  | FF02::1::                  | 146 Router Solicitation                           |   |      |  |
| 13  | 0.073 | 802.11a (OFDM) | 56    | -33 dBm | -93 dBm      |             | 6 Frame 1s     |           | Data                 | ::                         | FF02::1::                  | FF02::1::   | 136 Neighbor Solicitation for fe80::1a42:c607:0001:27fa |      |  |

- Choose first QoS Data frame #9 and open IEEE802.11 QoS Data Header > CCMP (Counter mode with Cipher-block chaining Message authentication code Protocol) parameters

# TK(wlan.analysis.tk) and PMK(wlan.analysis.pmk)

## ▼ CCMP parameters

CCMP Ext. Initialization Vector: 0x000000000002

Key Index: 0

[TK: 4c102fd43613c535404d0777088a6503]

[PMK: 31bb75a609a424aac01e9929b39458e87ea45b0f30204ff5642bf3067a6fd31f]

#sf22us

- Wireshark decrypt 4way handshake and add generated fields, TK(Temporal Key), actual AES key of the communication and PMK(Pairwise Master Key) 32 bytes(256bit), 4096 round times calculation of PBKDF2 function with SHA1 algorithm, using PSK and SSID
- We can also test this calculation by Python

```
from hashlib import pbkdf2_hmac
pwd="wireshark"
ssid="ikeriri6"
pmk = pbkdf2_hmac('sha1', pwd.encode('ascii'), ssid.encode('ascii'), 4096, 32)
print(pmk.hex())
```

# Check PMK generation by VisualStudio Code



#sf22us

```
psk.py > ...
1 from hashlib import pbkdf2_hmac
2 pwd="wireshark"
3 ssid="ikeriri6"
4 pmk = pbkdf2_hmac('sha1', pwd.encode('ascii'), ssid.encode('ascii'), 4096, 32)
5 print(pmk.hex())
```

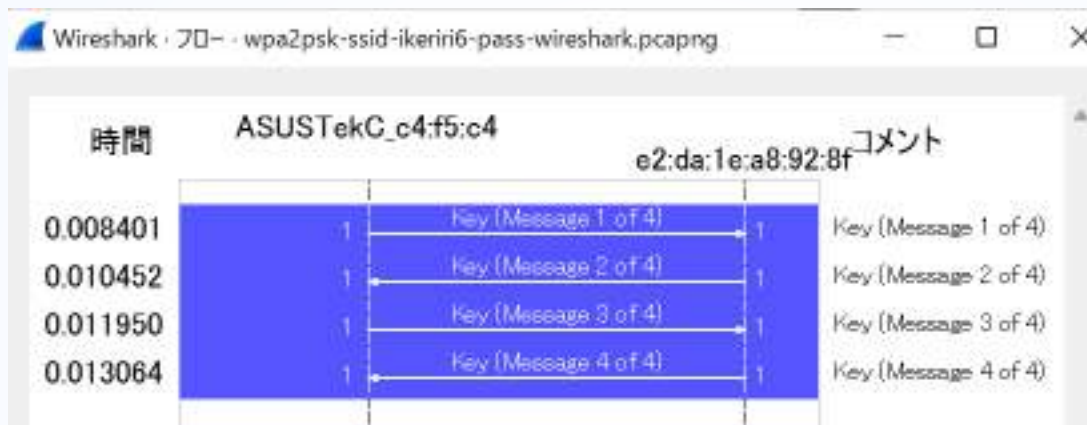
- Run pmk.py to check the PMK from the output with Wireshark [PMK:] field, it is the same value

```
gWPA3> & C:/Users/TakeshitaMegumi/AppData/Local/Microsoft/WindowsApps/python3.9.exe "c:/Users/TakeshitaMegumi/
neDrive - いけりりネットワークサービス株式会社/Sharkfest/Sharkfest2022/03DissectingWPA3/psk.py"
31bb75a609a424aac01e9929b39458e87ea45b0f30204ff5642bf3067a6fd31f
```

[PMK: 31bb75a609a424aac01e9929b39458e87ea45b0f30204ff5642bf3067a6fd31f]

- Station (iPhone13) and AP (ASUS) share this information, but never send it to the network.

# Let's start dissecting 4way handshake



- ASUS(AP) and iPhone13(STA) know the same PMK

[PMK: 31bb75a609a424aac01e9929b39458e87ea45b0f30204ff5642bf3067a6fd31f]

- Exchange and Confirm PTK in Messages 1,2  
Exchange and Confirm GTK in Messages 3,4



# Packet #5 Key(Message 1 of 4) AP->STA

|   |          |                |    |         |         |        |
|---|----------|----------------|----|---------|---------|--------|
| 4 | 0.005... | 802.11a (OFDM) | 56 | -33 dBm | -93 dBm | 6 Fram |
| 5 | 0.008... | 802.11a (OFDM) | 56 | -33 dBm | -93 dBm | 6 Fram |
| 6 | 0.010... | 802.11a (OFDM) | 56 | -41 dBm | -93 dBm | 6 Fram |
| 7 | 0.011... | 802.11a (OFDM) | 56 | -33 dBm | -93 dBm | 6 Fram |
| 8 | 0.013... | 802.11a (OFDM) | 56 | -41 dBm | -91 dBm | 6 Fram |

```
> Frame 5: 165 bytes on wire (1320 bits), 165 bytes captured (1320 bits) on interface
> Radiotap Header v0, Length 32
> 802.11 radio information
> IEEE 802.11 QoS Data, Flags: .....F.
> Logical-Link Control
> 802.1X Authentication
  Version: 802.1X-2004 (2)
  Type: Key (3)
  Length: 95
  Key Descriptor Type: EAPOL RSN Key (2)
  [Message number: 1]
> Key Information: 0x008a
  Key Length: 16
  Replay Counter: 1
  WPA Key Nonce: 812e47f04e25fe494c7d44b2f7b016e0e0e3f24865fd134f4998a8f5d0d68bc0
  Key IV: 00000000000000000000000000000000
  WPA Key RSC: 0000000000000000
  WPA Key ID: 0000000000000000
  WPA Key MIC: 00000000000000000000000000000000
  WPA Key Data Length: 8
```

- AP creates and sends Nonce (ANonce), 32bytes random value.
- Message1 packet also contains AP's MAC Address.



# PTK creation from STA(iPhone) side (1of4)



- STA starts to create PTK (Pairwise Transient Key) after STA received EAPOL Message 1of4 (Packet 5)
- STA creates its own Nonce(Snonce), 32bytes random
- PTK is created from ANonce, Snonce, AP MAC address and STA mac address

PTK = PRF-512(PMK, “Pairwise key expansion”,  
Min(AP MAC, STA MAC) || Max(AP MAC, STA MAC) ||  
Min(ANonce, SNonce) || Max(ANonce, SNonce))

\*PRF(Pseudo-Random Function) is SHA1 hash value from input parameter, PMK, “Pairwise key expansion” and Min(AP MAC, STA MAC) || Max(AP MAC, STA MAC) || Min(ANonce, SNonce) || Max(ANonce, SNonce))

PTK is a big key ring consists of many WPA2/WPA3 keys

**PMK (Primary Master Key) 256bits**

**Anonce  
32bytes random**

**SNonce  
32 bytes random**

**AMAC  
AP's MAC address**

**SMAC  
STA's MAC address**

PTK = PRF-512(PMK, "Pairwise key expansion", Min(AP MAC, STA MAC) ||  
Max(AP MAC, STA MAC) || Min(ANonce, SNonce) || Max(ANonce, SNonce))

**PTK (Pairwise Transient Key) 512bit**

**EAPOL KEK (Key Encryption Key) 128bits**

**EAPOL KCK (Key Confirmation Key) 128bits**

**Temporal Key 128bits for actual communication AES encryption/decryption key**

**Receive MIC Key (MIC Secret) 64bits for receiving packet Message Integrity Code**

**Transmit MIC Key (MIC Secret) 64bits for sending packet Message Integrity Code**

# Packet #6 Key(Message 2 of 4) STA->AP

|         |                |    |         |         |         |
|---------|----------------|----|---------|---------|---------|
| 4 0.005 | 802.11a (OFDM) | 56 | -33 dBm | -93 dBm | 6 Frame |
| 5 0.008 | 802.11a (OFDM) | 56 | -33 dBm | -93 dBm | 6 Frame |
| 6 0.010 | 802.11a (OFDM) | 56 | -41 dBm | -93 dBm | 6 Frame |
| 7 0.011 | 802.11a (OFDM) | 56 | -33 dBm | -93 dBm | 6 Frame |
| 8 0.013 | 802.11a (OFDM) | 56 | -41 dBm | -91 dBm | 6 Frame |

```
> Frame 6: 187 bytes on wire (1496 bits), 187 bytes captured (1496 bits) on interface
> Radiotap Header v0, Length 32
> 802.11 radio information
> IEEE 802.11 QoS Data, Flags: .....T
> Logical-Link Control
> 802.1X Authentication
  Version: 802.1X-2004 (2)
  Type: Key (3)
  Length: 117
  Key Descriptor Type: EAPOL RSN Key (2)
  [Message number: 2]
> Key Information: 8x010a
  Key Length: 16
  Replay Counter: 1
  NPA Key Nonce: fcf94398b971a1f20572495509733ff0000:93b142e86c9348ce23f3c287fffb
  Key IV: 00000000000000000000000000000000
  NPA Key RSC: 0000000000000000
  NPA Key ID: 0000000000000000
  NPA Key MIC: d5aa6adf088791d7cd37b8966f8a0930
  NPA Key Data Length: 22
> NPA Key Data: 3014010000fac040100000fac040100000fac028c00
```

- STA creates and sends Nonce (SNonce), 32bytes random value.
- STA calculate and add WPA Key MIC
- Message2 packet also contains STA's MAC Address

#s122us

## STA created SNonce and Add MIC Key Data



#sf22us

- Receiving 1of4 Messages, STA creates PTK
- STA sets SNonce, 32bytes random value
- STA also adds 16 bytes WPA Key MIC field, calculated SHA1 HMAC from all of the 802.1x fields

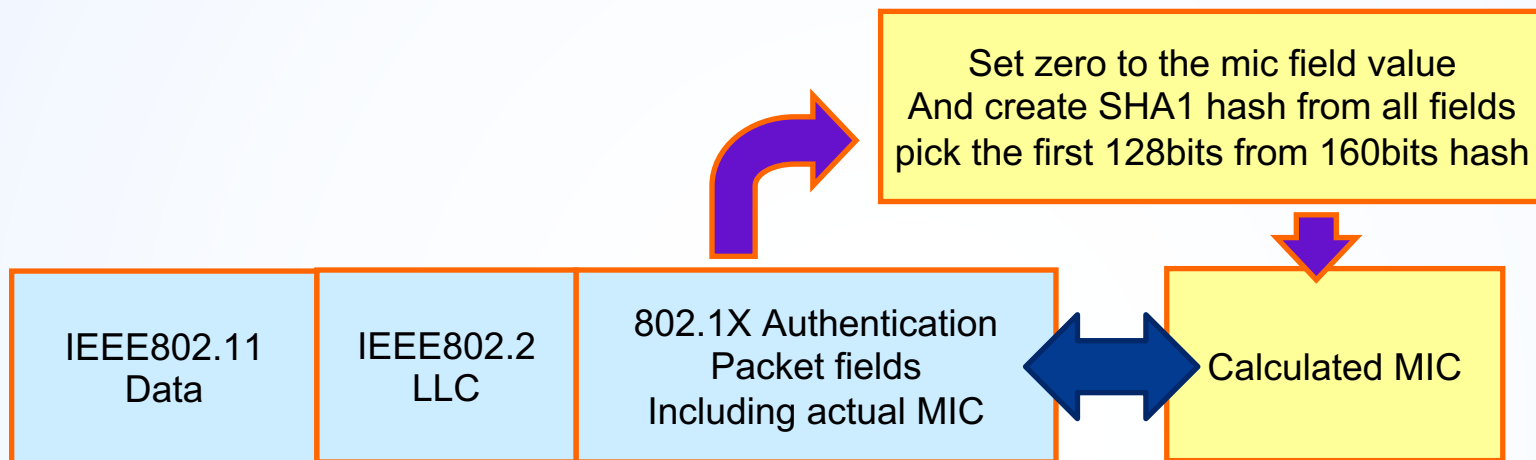
```
WPA Key MIC: d5aa6adf088791d7cd37b8866f8a0930
```

- WPA Key MIC means the confirmation that created PTK is the same with STA and AP  
(Receiving 2of4 Message, AP also creates PTK and check the WPA Key MIC is correct)

## PTK creation from AP(ASUS) side

- AP starts to create PTK (Pairwise Transcient Key) after EAPOL Message 2of4 (Packet 6)
- PTK is created from ANonce, Snonce, AP MAC address and STA mac address  
$$\text{PTK} = \text{PRF-512}(\text{PMK}, \text{"Pairwise key expansion"}, \text{Min}(\text{AP MAC}, \text{STA MAC}) || \text{Max}(\text{AP MAC}, \text{STA MAC}) || \text{Min}(\text{ANonce}, \text{SNonce}) || \text{Max}(\text{ANonce}, \text{SNonce}))$$
- AP checks the WPA Key MIC field to calculate SHA1HMAC from all of the 802.1x fields with the MIC field set to all zeros.
- If the calculated MIC is the same with the Message2of4, STA and AP shared the same PTK.

# Check Actual MIC with Calculated MIC



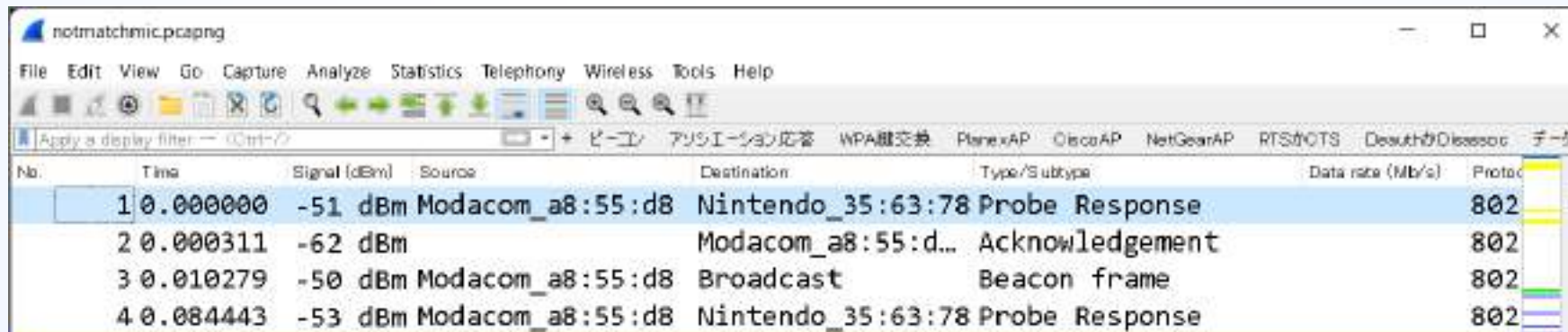
```
802.1X Authentication
Version: 802.1X-2004 (2)
Type: Key (3)
Length: 117
Key Descriptor Type: EAPOL RSN Key (2)
[Message number: 2]
Key Information: 0x010a
Key Length: 16
Replay Counter: 1
WPA Key Nonce: fcf94398b971a1f2057249569733ff0000c93b142e86c9348ce23f3c287ff8b
Key IV: 00000000000000000000000000000000
WPA Key RSC: 0000000000000000
WPA Key ID: 0000000000000000
WPA Key MIC: d5a6adfd008791d7cd37b0866f8a0930
WPA Key Data Length: 22
WPA Key Data: 3014010000fac040100000fac040100000fac028c00
```



If calculated MIC is not the same with WPA Key MIC value of Message 2 of 4 (Packet 6),  
It usually means passphrase is not the same.

#sf22us

- Open notmatchmic.pcapng and set display filter as “wlan.addr\_resolved contains Nintendo”



The image shows a Wireshark packet capture window titled 'notmatchmic.pcapng'. The display filter is set to 'wlan.addr\_resolved contains Nintendo'. The packet list shows four packets:

| No. | Time     | Signal (dBm) | Source           | Destination       | Type/Subtype    | Data rate (Mb/s) | Protocol |
|-----|----------|--------------|------------------|-------------------|-----------------|------------------|----------|
| 1   | 0.000000 | -51 dBm      | Modacom_a8:55:d8 | Nintendo_35:63:78 | Probe Response  |                  | 802.11   |
| 2   | 0.000311 | -62 dBm      |                  | Modacom_a8:55:d8  | Acknowledgement |                  | 802.11   |
| 3   | 0.010279 | -50 dBm      | Modacom_a8:55:d8 | Broadcast         | Beacon frame    |                  | 802.11   |
| 4   | 0.084443 | -53 dBm      | Modacom_a8:55:d8 | Nintendo_35:63:78 | Probe Response  |                  | 802.11   |

- This trace tested passphrase mismatch between Modacom AP and Nintendo STA.
- We can find the iteration of Message1, Message2 because AP's calculated MIC is not the same as Message 2 WPA Key MIC.

|                   |                   |              |        |     |                          |
|-------------------|-------------------|--------------|--------|-----|--------------------------|
| Modacom_a8:55:d8  | Nintendo_35:63:78 | QoS Data     | EAPOL  | 169 | Key (Message 1 of 4)     |
| Nintendo_35:63:78 | Modacom_a8:55:d8  | QoS Data     | EAPOL  | 191 | Key (Message 2 of 4)     |
| Modacom_a8:55:d8  | Nintendo_35:63:78 | QoS Data     | EAPOL  | 169 | Key (Message 1 of 4)     |
| Nintendo_35:63:78 | Modacom_a8:55:d8  | QoS Data     | EAPOL  | 191 | Key (Message 2 of 4)     |
| Modacom_a8:55:d8  | Nintendo_35:63:78 | QoS Data     | EAPOL  | 169 | Key (Message 1 of 4)     |
| Nintendo_35:63:78 | Modacom_a8:55:d8  | QoS Data     | EAPOL  | 191 | Key (Message 2 of 4)     |
| Modacom_a8:55:d8  | Nintendo_35:63:78 | QoS Data     | EAPOL  | 169 | Key (Message 1 of 4)     |
| Nintendo_35:63:78 | Modacom_a8:55:d8  | QoS Data     | EAPOL  | 191 | Key (Message 2 of 4)     |
| Modacom_a8:55:d8  | Nintendo_35:63:78 | Disassociate | 802.11 | 62  | Disassociate, SN=2, FN=6 |

# PTK exchange Demonstration



#sf22us

- I referred and created the WPA2 implementation Python code from Nicholas smith  
<https://nicholastsmith.wordpress.com/2016/11/15/wpa2-key-derivation-with-anaconda-python/>
- It is not perfect, not actual, but a Pseudo concept, I set parameters from the trace file.  
wpa2psk-ssid-ikeriri6-pass-wireshark.pcapng
- Open ptk.py using VSCode.

# Open ptk.py ( I referred and created the code from Nicholas smith)

# <https://nicholastsmith.wordpress.com/2016/11/15/wpa2-key-derivation-with-anaconda-python/>

#sf22us

```
ptk.py > ...
1 # I kept it referred and changed the code from nicholastsmith
2 # https://nicholastsmith.wordpress.com/2016/11/15/wpa2-key-derivation
3
4 import hmac
5 from binascii import a2b_hex, b2a_hex
6 from hashlib import pbkdf2_hmac, sha1, md5
7 import binascii
8
9 #Pseudo-random function for generation of
10 #the pairwise transient key (PTK)
11 #key: The PTK
12 #A: b"Pairwise key expansion"
13 #B: The apMac, cliMac, anonce, and snonce concatenated
14 # like mac1 mac2 nonce1 nonce2
15 # such that mac1 < mac2 and nonce1 < nonce2
16 #return: The ptk
17 def PRF(key, A, B):
18     #Number of bytes in the PTK
19     nByte = 64
20     i = 0
21     R = b''
22     #Each iteration produces 160-bit value and 512 bits are required
23     while(i <= ((nByte * 8 + 159) / 160)):
24         hmacsha1 = hmac.new(key, A + chr(0x00).encode() + B + chr(1).encode(), sha1)
25         R = R + hmacsha1.digest()
26         i += 1
27     return R[:nByte]
```

```
20 #Make parameters for the generation of the PTK
21 #anoncer: the anonce from the 4-way handshake
22 #snonce: the snonce from the 4-way handshake
23 #apMac: the MAC address of the access point
24 #cliMac: the MAC address of the client
25 #A,B: (A, B) where A and B are parameters
26 # for the generation of the PTK
27 def MakeAP(anonce, snonce, apMac, cliMac):
28     A = b"Pairwise key expansion"
29     B = min(apMac, cliMac) + max(apMac, cliMac) + min(anonce, snonce)
30     return (A, B)
31
32 #compute the 1st message integrity check for a WPA 4-way handshake
33 #pwd: The password to test
34 #ssid: the ssid of the AP
35 #A: b"Pairwise key expansion"
36 #B: The apMac, cliMac, anonce, and snonce concatenated
37 # like mac1 mac2 nonce1 nonce2
38 # such that mac1 < mac2 and nonce1 < nonce2
39 #data: A list of 802.11 frames with the MIC field zeroed
40 #return: (x, y, z) where x is the mic, y is the PTK, and z is the PTK
41 def MakeMIC(pwd, ssid, A, B, data, wpa = False):
42     #create the pairwise master key using 4096 iterations of hmac-sha1
43     #to generate a 32-byte value
44     pmk = pbkdf2_hmac('sha1', pwd.encode('ascii'), ssid.encode('ascii'), 4096)
45     #Make the pairwise transient key (PTK)
46     ptk = PRF(pmk, A, B)
47     #WPA uses md5 to compute the MIC while WPA2 uses sha1
48     hmacFunc = md5 if wpa else sha1
49     #Create the MICs using HMAC-SHA1 of data and return all computed MICs
50     mics = [hmac.new(ptk[0:16], i, hmacFunc).digest() for i in data]
51     return (mics, ptk, pmk)
```



Message(1of4)  
packet #5

Message(2of4  
packet #6

[illegible][illegible]

# ptk.py outputs PMK,PTK,TK and MIC value



#sf22us

```

95 A,B=FakeAP(anonce, snonce, apmac, cllmac)
96 mics,ptk,pak=FakePTC(pwd, ssid, A, B, [data1, data2, data3], wpa = False)
97
98 print ("PMK: " + pak.hex())
99 print ("PTK: " + ptk.hex())
100 kek=ptk[0:15]
101 print ("KEK: " + kek.hex())
102 kck=ptk[16:31]
103 print ("KCK: " + kck.hex())
104 tk=ptk[32:48]
105 print ("TK: " + tk.hex())
106 rmic=ptk[48:55]
107 print ("receive MIC Secret: " + rmic.hex())
108 tmic=ptk[56:63]
109 print ("transmit MIC Secret: " + tmic.hex())
110
111 mic3Str = mics.upper()
112 print("actual mic: " + mic3Str)
113 #take the first 128-bits of the 160-bit SHA1 hash
114 micStr = b2a_hex(mics[0]).decode().upper()[0:8]
115 print("calculated mic from Message2of4:" + micStr)
116 print('MATCH' if micStr == mic3Str else 'MISMATCH')
117 #Display the desired MIC2 and compare to target MIC2
118 mic2Str = mic2.upper()
119 print("actual mic: " + mic2Str)
120 #take the first 128-bits of the 160-bit SHA1 hash
121 micStr = b2a_hex(mics[1]).decode().upper()[0:8]
122 print("calculated mic from Message3of4:" + micStr)
123 print('MATCH' if micStr == mic2Str else 'MISMATCH')
124 #Display the desired MIC3 and compare to target MIC3
125 mic3Str = mics[2].upper()
126 print("packet mic: " + mic3Str)
127 #Take the first 128-bits of the 160-bit SHA1 hash
128 micStr = b2a_hex(mics[2]).decode().upper()[0:8]
129 print("packet mic from Message4of4:" + micStr)
130 print('MATCH' if micStr == mic3Str else 'MISMATCH')

```

```

PS C:\Users\TakeshitaMegumi\OneDrive - いけりネットワークサービス株式会社\Sharkfest
al\Microsoft\WindowsApps\python3.9.exe "c:\Users\TakeshitaMegumi\OneDrive - いけり
k.py"
Passphrase:wireshark
SSID:ikeriri6
AP MAC Address from 10f4:f02f74c4f5cd
AP Nonce from 10f4:812e47f04e25fe494c7d44b2f7b016e0ebe3f24865fd234f4998abf5d8d68bc0
STA MAC Address from 10f4:e2da1ea8928f
STA Nonce from 10f4:fcf94398b971a1f28572495509733ff0008c93b142e86c9348ce23f3c287ff8b
PMK: 31bb75a609a42aac01e9929b39458e87ea45b0f30204ff5642bf3067a6fd31f
PTK: 46e515c2a3677ef693f93e8368517684728fe5b36aa9e9fa60e80007a18c05574c102fd43613c535
KEK: 46e515c2a3677ef693f93e83685176
KCK: 728fe5b36aa9e9fa60e80007a18c05
TK: 4c102fd43613c535404d0777088a6503
Receive MIC Secret: 7d9de4d644bb9a
Transmit MIC Secret: de1a23f2e9f17b
actual mic:d5AA6ADF088791D7CD378866F8A093D
calculated mic from Message2of4:d5AA6ADF088791D7CD378866F8A093D
MATCH
actual mic:8FC6CCF6133542E9F8844ECB26DE5FF8
calculated mic from Message3of4:8FC6CCF6133542E9F8844ECB26DE5FF8
MATCH
packet mic:E2367DB355CCDF071008D73D0E17585C
packet mic from Message4of4:E2367DB355CCDF071008D73D0E17585C
MATCH
PS C:\Users\TakeshitaMegumi\OneDrive - いけりネットワークサービス株式会社\Sharkfest

```



# Check the MIC is correct



```

Passphrase:Wireshark
SSID:ikeiri16
AP MAC Address from 10f4:f82f74c4f5c4
AP Nonce from 10f4:812e4f0de25fe494c7d44b2f7b016e0ebe3f24865fd234f4998a0f5d8d68bc0
STA MAC Address from 10f4:e2da1ea8928f
STA Nonce from 10f4:fcf94398b971a1f2D572495509733ff0008c93b142e86c9348ce23f3c287ffab
PMK: 31bb75a009a424aac01e9929b39458e87ea45b0f30204ff5642bf3067a6fd31f
PTK: 46e515c2a3677ef693f93e8368517684728fe5b36aa9e9fa60e00007a18c05574c102fd43613c535404d0777088a65037d9de4d644bb9a67de1a23f2e9f17b8e
KEK: 46e515c2a3677ef693f93e83685176
KCK: 728fe5b36aa9e9fa60e00007a18c05
TK: 4c102fd43613c535404d0777088a6503
Receive MIC Secret: 7d9de4d644bb9a
Transmit MIC Secret: de1a23f2e9f17b
actual mic:D5AA6A0F008791D7CD37B8866F8A0930
calculated mic from Message2of4:D5AA6A0F008791D7CD37B8866F8A0930
MATCH
actual mic:8FC6CCF6133542E9F8844EC826DE5FF8
calculated mic from Message3of4:8FC6CCF6133542E9F8844EC826DE5FF8
MATCH
packet mic:E23670B355CCDF071008D73D6E17585C
packet mic from Message4of4:E23670B355CCDF071008D73D6E17585C
MATCH
PS C:\Users\Takeshita\egumi\OneDrive - いけりりネットワークサービス株式会社\sharkfest\Sharkfest2022\03DissectingMPA3>

```

- The calculated MIC value and packet are the same.
- AP and STA succeeded in sharing the same PTK ( and GTK later) without sending key data into the network.

# Check TK(Temporal Key)

- TK is used for actual encryption/decryption AES key for unicast data communication between AP and STA.
- Compare the calculated TK from ptk.py with the TK field in CCMP parameters of QoS Data packet ( for example, try packet #10) (WPA3-SAE also use this)

TK: 4c102fd43613c535404d0777088a6503

✓ CCMP parameters

CCMP Ext. Initialization Vector: 0000000000000003

Key Index: 0

[TK: 4c102fd43613c535404d0777088a6503]

[PMK: 31bb75a609a424aac01e9929b39458e87ea45b0f30204ff5642bf3067a6fd31f]

**BINGO!!**

# Packet #7 Key(Message 3 of 4) AP->STA

|            |  |                      |
|------------|--|----------------------|
| 40.005013  | -33 d..ASUSTekC_c4:f5:c4 e2:da:1e:a8:92:8f | Association Response |
| 50.008401  | -33 d..ASUSTekC_c4:f5:c4 e2:da:1e:a8:92:8f | QoS Data             |
| 60.010452  | -41 d..e2:da:1e:a8:92:8f ASUSTekC_c4:f5:c4 | QoS Data             |
| 70.011950  | -33 d..ASUSTekC_c4:f5:c4 e2:da:1e:a8:92:8f | QoS Data             |
| 80.013054  | -41 d..e2:da:1e:a8:92:8f ASUSTekC_c4:f5:c4 | QoS Data             |
| 90.072760  | -41 d...: ff02::1:ff01:27fa                | QoS Data             |
| 100.072892 | -41 d..e2:da:1e:a8:92:8f ASUSTekC_c4:f5:c5 | QoS Data             |

Frame 7: 253 bytes on wire (2024 bits), 253 bytes captured (2024 bits) on interface unknown

Radiotap Header v0, Length 32

802.11 radio information

IEEE 802.11 QoS Data, Flags: .....F.

Logical-Link Control

802.1X Authentication

Version: 802.1X-2004 (2)

Type: Key (3)

Length: 183

Key Descriptor Type: EAPOL RSN Key (2)

[Message number: 3]

Key Information: 0x13ca

Key Length: 16

Replay Counter: 2

WPA Key Nonce: 812e47f04e25fe494c7d44b2f7b016e0ebe3f24865fd234f4998a8f5d8d68bcb0

Key IV: 00000000000000000000000000000000

WPA Key RSC: 0000000000000000

WPA Key ID: 0000000000000000

WPA Key MIC: 8fc6ccf6133542e9f8844ec826de5ff8

WPA Key Data Length: 88

WPA Key Data: 8a6607c41f921c7035108bc8a60c1fab5a5ae7d0e191e1ce71fa11d8a1a65b302c4c345...

- AP creates Nonce (Gnonce), random.
- AP creates and sends GTK safely with KCK / KEK in securely. GTK is used for multicast/broadcast
- AP calculates and adds WPA Key MIC.

# GTK creation at AP side

- GTK is created from GMK, Gnonce(Group Nonce), AP's MAC address and Group Key Expansion

GTK 256bits for broadcast / multicast communication

TK(Temporal Key) 128bits AES encryption/decryption key

Receive MIC Key (MIC Secret) 64bits for receiving packet Message Integrity Code

Transmit MIC Key(MIC Secret) 64bits for sending packet Message Integrity Code

- GTK is used for broadcast and multicast, and GTK is the same key between all STA and AP, so GTK will be changed periodically (using a 2-way handshake)

# Check the GTK from Message 3of4

- Open WPA Key Data in 802.1X Authentication header of Message 3of4 (Packet #7)
- Open “Tag: Vendor Specific: Ieee 802.11: RSN GTK”

```

  WPA Key Data: 8a6607c41f921c7035108bc8a60c1fa1b5a5ae7d0e191e1ce71fa11d8a1a65b302c4c345...
    > Tag: RSN Information
    > Tag: Vendor Specific: Ieee 802.11: RSN GTK
      Tag Number: Vendor Specific (221)
      Tag length: 22
      OUI: 00:0f:ac (Ieee 802.11)
      Vendor Specific OUI Type: 1
      .... ..01 = KeyID: 1
      .... .0.. = Tx: 0
      0000 0... = Reserved: 0x00
      Reserved: 0
      GTK: eeda95a1155a28f86c030d000c1fdd9a
    > Tag: Vendor Specific: Ieee 802.11: RSN IGTK
      WPA Key Data Padding: dd000000
      [KCK: 46e515c2a3677ef693f93e8368517684]
      [KEK: 728fe5b36aa9e9fa60e80007a18c0557]
  
```


 GTK is here

# Compare KCK and KEK

- KCK(Key Confirmation Key) and KEK(Key Encryption Key) are used for secure key distribution.
- Confirm Wireshark calculated KCK, KEK is the same with ptk.py generated KCK, KEK

```
GTK: eeda95a1155a28f86c030d000c1fdd9a
```

```
> Tag: Vendor Specific: Ieee 802.11: RSN IGTK
```

```
WPA Key Data Padding: dd000000
```

```
[KCK: 46e515c2a3677ef693f93e8368517684]
```

```
[KEK: 728fe5b36aa9e9fa60e80007a18c0557]
```

```
KEK: 46e515c2a3677ef693f93e8368517684
```

```
KCK: 728fe5b36aa9e9fa60e80007a18c0557
```

- It means GTK sends/receives securely



# Packet #7 Key(Message 4of4) STA->AP

|            |   |                   |
|------------|---|-------------------|
| 40.005013  | -33 d.. ASUSTekC_c4:f5:c4 e2:da:1e:a8:92:8f | Association Respo |
| 50.008401  | -33 d.. ASUSTekC_c4:f5:c4 e2:da:1e:a8:92:8f | QoS Data          |
| 60.010452  | -41 d.. e2:da:1e:a8:92:8f ASUSTekC_c4:f5:c4 | QoS Data          |
| 70.011950  | -33 d.. ASUSTekC_c4:f5:c4 e2:da:1e:a8:92:8f | QoS Data          |
| 80.013064  | -41 d.. e2:da:1e:a8:92:8f ASUSTekC_c4:f5:c4 | QoS Data          |
| 90.072760  | -41 d.. !! ff02::1:ff01:27fa                | QoS Data          |
| 100.072092 | -41 d.. e2:da:1e:a8:92:8f ASUSTekC_c4:f5:c5 | QoS Data          |
| 110.073031 | -41 d.. 0.0.0.0 255.255.255.255             | QoS Data          |
| 120.073376 | -41 d.. fe80::1c42:c607:: ff02::2           | QoS Data          |
| 130.073578 | -33 d.. !! ff02::1:ff01:27fa                | Data              |

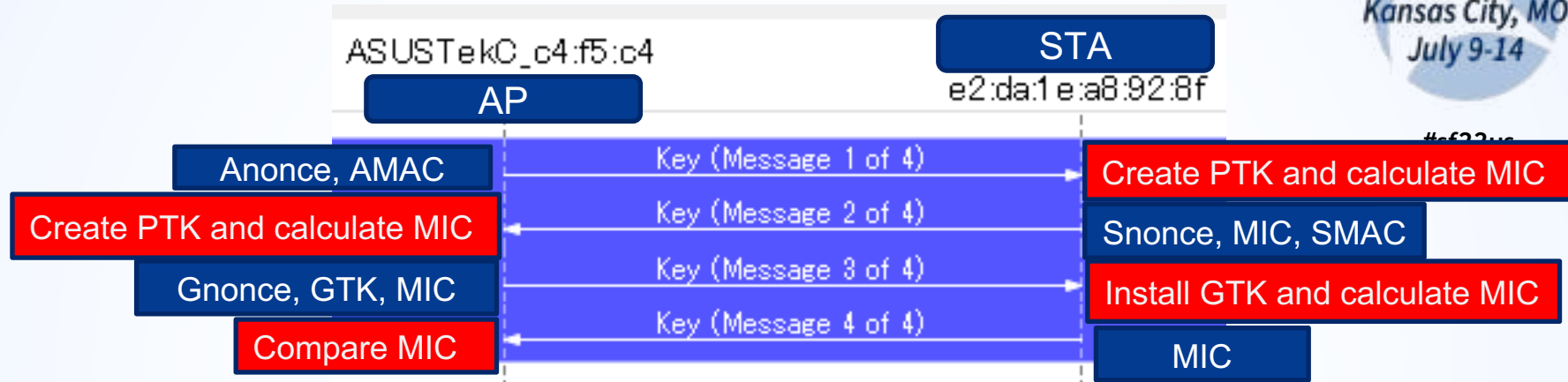
  

```

Frame 8: 165 bytes on wire (1320 bits), 165 bytes captured (1320 bits) on interface
Radiotap Header v0, Length 32
802.11 radio information
IEEE 802.11 QoS Data, Flags: .....T
Logical-Link Control
802.1X Authentication
  Version: 802.1X-2004 (2)
  Type: Key (3)
  Length: 95
  Key Descriptor Type: EAPOL RSN Key (2)
  [Message number: 4]
  Key Information: 0x030a
  Key Length: 16
  Replay Counter: 2
  WPA Key Nonce: 0000000000000000000000000000000000000000000000000000000000000000
  Key IV: 00000000000000000000000000000000
  WPA Key RSC: 0000000000000000
  WPA Key ID: 0000000000000000
  WPA Key MIC: e2367db355ccdf0710d8d73d6e175b5c
  WPA Key Data Length: 0
  
```

- STA received and installed GTK in Message 3of4 (#6)
- STA adds WPA Key MIC for confirmation
- AP receives Message 4of4 and confirms packet MIC with calculated MIC.

# Conclusion: 4 way handshake in WPA2/WPA3



```

Pasophrase:wireshark
SSID:ikeriri6
AP MAC Address from 10f4:f82f74c4f5c4
AP Nonce from 10f4:b12e47f04e25fe494c7d44b2f7b018e0e0e3f24865fd234f4998a8f5d8d68bc0
STA MAC Address from 10f4:e2da1eaa8928f
STA Nonce from 10f4:fc94398b971a1f20572495509733ff0000c93b142e06c9348ce23f3c287ff0b
PRK: 31bb75a009a424aac01e9929b39458e87ea45b0f30204ff5642bf3067a6fd31f
PTK: 46e515c2a3677ef693f93e8368517684720fe5b36aa9e9fa60e80007a18c0574c102fd43613c535404d8777088a65037d9de4d544bb9a67de1a23f2e9f17b8e
KEK: 46e515c2a3677ef693f93e8368517684
CKK: 720fe5b36aa9e9fa60e80007a18c057
TK: 4c102fd43613c535404d8777088a6503
Receive MIC Secret: 709de8d644bbsa
Transmit MIC Secret: de1a23f2e9f17b
actual mic:D5AA6ADF08879107CD37B8866F8A0930
calculated mic from Message20f4:D5AA6ADF00B79107CD37B8866F8A0930
WATCH
actual mic:8FC6CCF6133542E9F8B44EC826DE5FF8
calculated mic from Message30f4:8FC6CCF6133542E9F8B44EC826DE5FF8
WATCH
packet mic:E2367DB355CCDF071000D73D6E175B5C
packet mic from Message40f4:E2367DB355CCDF071000D73D6E175B5C
WATCH
  
```

## WPA2 is good, but...

- It has been over ten years since WPA2 was born.
- We can use a dictionary attack if we capture a complete 4-way handshake between AP and STA.
- Smartphones and tablets are positive to roam, so faked Deauthentication and Disassociation frames can lead to tons of new 4-way handshake packets.
- Some new attack method comes, for example, KRACKS blocks the original Message3of4 from AP and tries many GTK patterns to assure keys.

# Deauth attack

- faked Deauthentication and Disassociation frames can lead to tons of fresh 4-way handshake packet  
`airreplay-ng --deauth wlan0mon`
- Use dictionary attack if we capture just a set of a complete 4-way handshake between AP and STA.
- Open `deauthattack.pcap` and set Display filter as “`wlan.fc.type_subtype==12 or eapol`”
- You can find many 4-way handshake packets of Stations (Sony\_xx:yy:zz) after Deauthentication.



# Open deauthattack.pcap



#sf22us

| No.            | Time | Signal (dBm)                        | Source    | Destination | Type/Status | Data                     | Protocol | Length | Info |
|----------------|------|-------------------------------------|-----------|-------------|-------------|--------------------------|----------|--------|------|
| 442..358.904.. | -58  | d8m PlanexCo_e3:c2:79 Sony_50:73:db | Deauth... | 1802.11     | 45          | Deauthentication, SN=219 |          |        |      |
| 442..358.912.. | -58  | d8m PlanexCo_e3:c2:79 Sony_50:73:db | Deauth... | 1802.11     | 50          | Deauthentication, SN=219 |          |        |      |
| 442..358.914.. | -57  | d8m PlanexCo_e3:c2:79 Sony_50:73:db | Deauth... | 1802.11     | 50          | Deauthentication, SN=219 |          |        |      |
| 442..358.921.. | -58  | d8m PlanexCo_e3:c2:79 Sony_50:73:db | Deauth... | 1802.11     | 50          | Deauthentication, SN=219 |          |        |      |
| 443..359.850.. | -60  | d8m PlanexCo_e3:c2:79 Sony_50:73:db | Deauth... | 1802.11     | 50          | Deauthentication, SN=219 |          |        |      |
| 443..359.109.. | -62  | d8m PlanexCo_e3:c2:79 Sony_50:73:db | Deauth... | 1802.11     | 50          | Deauthentication, SN=219 |          |        |      |
| 443..359.113.. | -62  | d8m PlanexCo_e3:c2:79 Sony_50:73:db | Deauth... | 1802.11     | 50          | Deauthentication, SN=219 |          |        |      |
| 443..359.120.. | -62  | d8m PlanexCo_e3:c2:79 Sony_50:73:db | Deauth... | 1802.11     | 50          | Deauthentication, SN=219 |          |        |      |
| 443..359.124.. | -62  | d8m PlanexCo_e3:c2:79 Sony_50:73:db | Deauth... | 1802.11     | 50          | Deauthentication, SN=219 |          |        |      |
| 443..359.149.. | -63  | d8m PlanexCo_e3:c2:79 Sony_50:73:db | Deauth... | 1802.11     | 50          | Deauthentication, SN=219 |          |        |      |
| 443..359.163.. | -62  | d8m PlanexCo_e3:c2:79 Sony_50:73:db | Deauth... | 1802.11     | 50          | Deauthentication, SN=219 |          |        |      |
| 443..359.174.. | -61  | d8m PlanexCo_e3:c2:79 Sony_50:73:db | Deauth... | 1802.11     | 50          | Deauthentication, SN=219 |          |        |      |
| 443..359.177.. | -60  | d8m PlanexCo_e3:c2:79 Sony_50:73:db | Deauth... | 1802.11     | 50          | Deauthentication, SN=219 |          |        |      |
| 252..262.858.. | -58  | d8m PlanexCo_e3:c2:79 Sony_53:d6:0a | Deauth... | 1802.11     | 50          | Deauthentication, SN=217 |          |        |      |
| 252..262.374.. | -60  | d8m PlanexCo_e3:c2:79 Sony_53:d6:0a | Data      | 1EAPOL      | 177         | Key (Message 1 of 4)     |          |        |      |
| 252..262.450.. | -60  | d8m PlanexCo_e3:c2:79 Sony_53:d6:0a | Data      | 1EAPOL      | 211         | Key (Message 3 of 4)     |          |        |      |
| 313..293.295.. | -63  | d8m PlanexCo_e3:c2:79 Sony_53:d6:0a | Data      | 1EAPOL      | 177         | Key (Message 1 of 4)     |          |        |      |
| 313..293.299.. | -63  | d8m PlanexCo_e3:c2:79 Sony_53:d6:0a | Data      | 1EAPOL      | 177         | Key (Message 1 of 4)     |          |        |      |
| 313..293.571.. | -64  | d8m PlanexCo_e3:c2:79 Sony_53:d6:0a | Data      | 1EAPOL      | 177         | Key (Message 1 of 4)     |          |        |      |
| 313..293.580.. | -64  | d8m PlanexCo_e3:c2:79 Sony_53:d6:0a | Data      | 1EAPOL      | 177         | Key (Message 1 of 4)     |          |        |      |
| 313..293.587.. | -63  | d8m PlanexCo_e3:c2:79 Sony_53:d6:0a | Data      | 1EAPOL      | 177         | Key (Message 1 of 4)     |          |        |      |
| 313..293.592.. | -64  | d8m PlanexCo_e3:c2:79 Sony_53:d6:0a | Data      | 1EAPOL      | 177         | Key (Message 1 of 4)     |          |        |      |
| 314..293.644.. | -62  | d8m PlanexCo_e3:c2:79 Sony_53:d6:0a | Data      | 1EAPOL      | 89          | Key [Malformed Packet]   |          |        |      |
| 314..293.685.. | -61  | d8m PlanexCo_e3:c2:79 Sony_53:d6:0a | Data      | 1EAPOL      | 177         | Key (Message 1 of 4)     |          |        |      |

- faked Deauthentication frames (source address is faked AP) lead to tons of fresh 4-way handshake



# WPA3 Wi-Fi Protected Access

- WPA3 is the new security standard for wireless networks.  
[https://www.wi-fi.org/download.php?file=/sites/default/files/private/WPA3\\_Specification\\_v3.0.pdf](https://www.wi-fi.org/download.php?file=/sites/default/files/private/WPA3_Specification_v3.0.pdf)
- WPA3 personal mode (WPA3-SAE) uses SAE (Simultaneous Authentication of Equals)<sup>#sf22us</sup>, derived from Dragonfly Key Exchange (RFC7664), instead of open authentication.  
<https://www.rfc-editor.org/info/rfc7664>
- AP and STA exchange 4 packets (AP/STA Commit, AP/STA Confirm) and create PMK at the authentication phase, So PMK is different every time. This provides forward security, we cannot attack from an old 4-way handshake with the fresh PMK. (WPA2-PSK uses the same PMK) It means a dictionary attack is (almost) impossible!!

- Compare wpa2psk-ssid-ikeriri6-pass-wireshark.pcapng with wpa3psk-ssid-ikeriri6-pass-wireshark.pcapng
- Both WPA2 and WPA3 use the same 4-way handshake mechanism to create and share PTK, GTK

| No.        | Time | Source                | Destination       | Type/Protocol        | Data Length | Info  |
|------------|------|-----------------------|-------------------|----------------------|-------------|---|
| 10.000000  | -39  | d8e:e2:da:1e:a8:92:8f | ASUSTekC_c4:f5:c4 | Authentication       | 6802.11     | 97 Authentication, SN=833, FN=0, Flags=.....            |
| 20.001283  | -33  | d8e:ASUSTekC_c4:f5:c4 | e2:da:1e:a8:92:8f | Authentication       | 6802.11     | 62 Authentication, SN=0, FN=0, Flags=.....              |
| 30.002746  | -41  | d8e:e2:da:1e:a8:92:8f | ASUSTekC_c4:f5:c4 | Association Request  | 6802.11     | 237 Association Request, SN=834, FN=0, Flags=.....      |
| 40.005013  | -31  | d8e:ASUSTekC_c4:f5:c4 | e2:da:1e:a8:92:8f | Association Response | 6802.11     | 318 Association Response, SN=1, FN=0, Flags=.....       |
| 50.008401  | -33  | d8e:ASUSTekC_c4:f5:c4 | e2:da:1e:a8:92:8f | QoS Data             | 6EAPOL      | 165 Key (Message 1 of 4)                                |
| 60.010493  | -41  | d8e:e2:da:1e:a8:92:8f | ASUSTekC_c4:f5:c4 | QoS Data             | 6EAPOL      | 187 Key (Message 2 of 4)                                |
| 70.011950  | -33  | d8e:ASUSTekC_c4:f5:c4 | e2:da:1e:a8:92:8f | QoS Data             | 6EAPOL      | 253 Key (Message 3 of 4)                                |
| 80.013064  | -41  | d8e:e2:da:1e:a8:92:8f | ASUSTekC_c4:f5:c4 | QoS Data             | 6EAPOL      | 165 Key (Message 4 of 4)                                |
| 90.072708  | -41  | d8m:::                | ff02::1:ff01:27fa | QoS Data             | ICMPv6      | 178 Neighbor Solicitation for fe80::1c42:c607:6001:27fa |
| 100.072893 | -41  | d8e:e2:da:1e:a8:92:8f | ASUSTekC_c4:f5:c5 | QoS Data             | ARP         | 136 who has 192.168.90.1? Tell 192.168.90.236           |
| 110.073031 | -41  | d8m:0.0.0.0           | 255.255.255.255   | QoS Data             | DHCP        | 426 DHCP Request - Transaction ID 0xac9e7500            |
| 120.073576 | -41  | d8m:fe80::1c42:c607:: | ff02::2           | QoS Data             | ICMPv6      | 148 Router Solicitation                                 |
| 130.073570 | -33  | d8m:::                | ff02::1:ff01:27fa | Data                 | ICMPv6      | 136 Neighbor Solicitation for fe80::1c42:c607:6001:27fa |

WPA2



WPA3

| No.        | Time | Source                | Destination       | Type/Protocol        | Data Length | Info  |
|------------|------|-----------------------|-------------------|----------------------|-------------|---|
| 10.000000  | -36  | d8e:e2:da:1e:a8:92:8f | ASUSTekC_c4:f5:c4 | Authentication       | 6802.11     | 160 Authentication, SN=3841, FN=0, Flags=.....      |
| 20.028660  | -31  | d8m:ASUSTekC_c4:f5:c4 | e2:da:1e:a8:92:8f | Authentication       | 6802.11     | 160 Authentication, SN=0, FN=0, Flags=.....         |
| 30.058370  | -38  | d8m:e2:da:1e:a8:92:8f | ASUSTekC_c4:f5:c4 | Authentication       | 6802.11     | 96 Authentication, SN=3842, FN=0, Flags=.....       |
| 40.059667  | -31  | d8m:ASUSTekC_c4:f5:c4 | e2:da:1e:a8:92:8f | Authentication       | 6802.11     | 96 Authentication, SN=1, FN=0, Flags=.....          |
| 50.062471  | -38  | d8m:e2:da:1e:a8:92:8f | ASUSTekC_c4:f5:c4 | Association Request  | 6802.11     | 255 Association Request, SN=3843, FN=0, Flags=..... |
| 60.064576  | -31  | d8m:ASUSTekC_c4:f5:c4 | e2:da:1e:a8:92:8f | Association Response | 6802.11     | 293 Association Response, SN=3, FN=0, Flags=.....   |
| 70.067697  | -31  | d8m:ASUSTekC_c4:f5:c4 | e2:da:1e:a8:92:8f | QoS Data             | 6EAPOL      | 187 Key (Message 1 of 4)                            |
| 80.070141  | -58  | d8e:e2:da:1e:a8:92:8f | ASUSTekC_c4:f5:c4 | QoS Data             | 6EAPOL      | 205 Key (Message 2 of 4)                            |
| 90.071860  | -31  | d8m:ASUSTekC_c4:f5:c4 | e2:da:1e:a8:92:8f | QoS Data             | 6EAPOL      | 253 Key (Message 3 of 4)                            |
| 100.073031 | -38  | d8m:e2:da:1e:a8:92:8f | ASUSTekC_c4:f5:c4 | QoS Data             | 6EAPOL      | 165 Key (Message 4 of 4)                            |
| 110.075096 | -40  | d8m:e2:da:1e:a8:92:8f | ASUSTekC_c4:f5:c4 | Null function (...)  | 802.11      | 74 QoS Null function (No data), SN=3844, FN=0       |
| 120.421001 | -40  | d8m:e2:da:1e:a8:92:8f | IP6mcast:ff:01::  | QoS Data             | 802.11      | 170 QoS Data, SN=0, FN=0, Flags=op.....T            |
| 130.421005 | -40  | d8m:e2:da:1e:a8:92:8f | Broadcast         | QoS Data             | 802.11      | 426 QoS Data, SN=1, FN=0, Flags=op.....T            |

# Difference between WPA2 and WPA3

| Explanation                          | WPA2 Personal<br>WPA2-PSK                              | WPA3 Personal<br>WPA3-SAE  |
|--------------------------------------|--|--|
| PTK, GTK exchange                    | Both WPA2 and WPA3 use 4 way handshake                 |  |
| Passphrase length                    | From 8 to 63 characters                                | From 8 to 128 characters   |
| Temporal Key (encryption key)        | AES(128bits)   | AES(128bits)   |
| Authentication method (PMK creation) | Open System/Shared key authentication<br>PSK+SSID->PMK | SAE (Simultaneous Authentication of Equals)                            |
| Encryption of Management frame       | Not nessary  | PMF (Protected Management Frames) (Optionally)                         |
| Brute force prevention               | Not nessary  | Lock out a device after a number of unsuccessful attempts (Optionally) |

- Open wpa2 trace file and set display filter as “wlan.fc.type\_subtype == 0x000b”
- Extract IEEE802.11 Wireless Management

```
> Frame 1: 97 bytes on wire (776 bits), 97 bytes captured (776 bits) on interface 0  
> Radiotap Header v0, Length 32  
> 802.11 radio information  
> IEEE 802.11 Authentication, Flags: .....  
v IEEE 802.11 Wireless Management  
  v Fixed parameters (6 bytes)  
    Authentication Algorithm: Open System (0)  
    Authentication SEQ: 0x0001  
    Status code: Successful (0x0000)  
> Tagged parameters (35 bytes)
```

```
> Frame 2: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0  
> Radiotap Header v0, Length 32  
> 802.11 radio information  
> IEEE 802.11 Authentication, Flags: .....  
v IEEE 802.11 Wireless Management  
  v Fixed parameters (6 bytes)  
    Authentication Algorithm: Open System (0)  
    Authentication SEQ: 0x0002  
    Status code: Successful (0x0000)
```

- WPA2 Open System Authentication checks the match of SSID name (ikeriri6)

- Open wpa3 trace file and set display filter as “wlan.fc.type\_subtype == 0x000b”
- Extract IEEE802.11 Wireless Management
- There are 4 Authentication packets with SAE Message type as follows STA Commit(1), AP Commit(1), STA Confirm(2), AP Confirm(2)

✓ IEEE 802.11 Wireless Management

✓ Fixed parameters (104 bytes)

Authentication Algorithm: Simultaneous Authentication of Equals (SAE) (3)

Authentication SEQ: 0x0001

Status code: Successful (0x0000)

SAE Message Type: Commit (1)

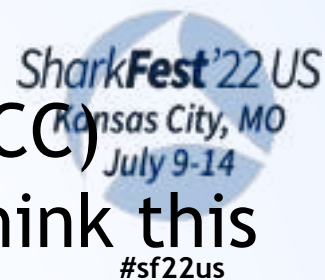
Group Id: 256-bit random ECP group (19)

Scalar: c67801ac5941d1e0fad412b255567e53c885a0d12a22439a3e021c7d633f37e7

Finite Field Element: f4b7c34e9f0d5444381e1dde353e54dcc838435b372a3933b7cc0



## Understand Dragonfly key exchange with simple example



- Dragonfly use Elliptic-Curve Cryptography(ECC)  
ECC is a difficult mathematical theory, so think this in programming words easily.
- The finite field is a mathematical term, in other words, the calculatable mod value collection.  
if we set mod value  $p=11$  ( $p$ :prime field  $GF(p)$ )  
and the mod collection is  $\{0,1,2,...,9,10\}$   
we can calculate the mod collection, for example  
 $3 * 5 = 15$  is a element of the mod collection  
 $3*5=15 \equiv 4 \pmod{11}$ . We use this for Elliptic-Curve Discrete Logarithm Problem (seems difficult...)

## Understand Dragonfly key exchange with simple example



#sf22us

- We can translate Elliptic-Curve Discrete Logarithm Problem in a computer program like below,  
int a=2,n=5,p=11, b=a^n mod p (^:exponentiation)  
we can calculate b easily from a,n,p  
$$b = 2 * 2 * 2 * 2 * 2 \bmod 11 = 32 \bmod 11 = 10$$
- So how do we find n (Logarithm) from a,b and p,  
we need to test incrementally, n=1, n=2, n=3, ...  
if the parameters are such a vast number, finding Logarithm n is almost impossible in today's PC
- ECC use this ECDLP for encryption

## Understand Dragonfly key exchange with simple example



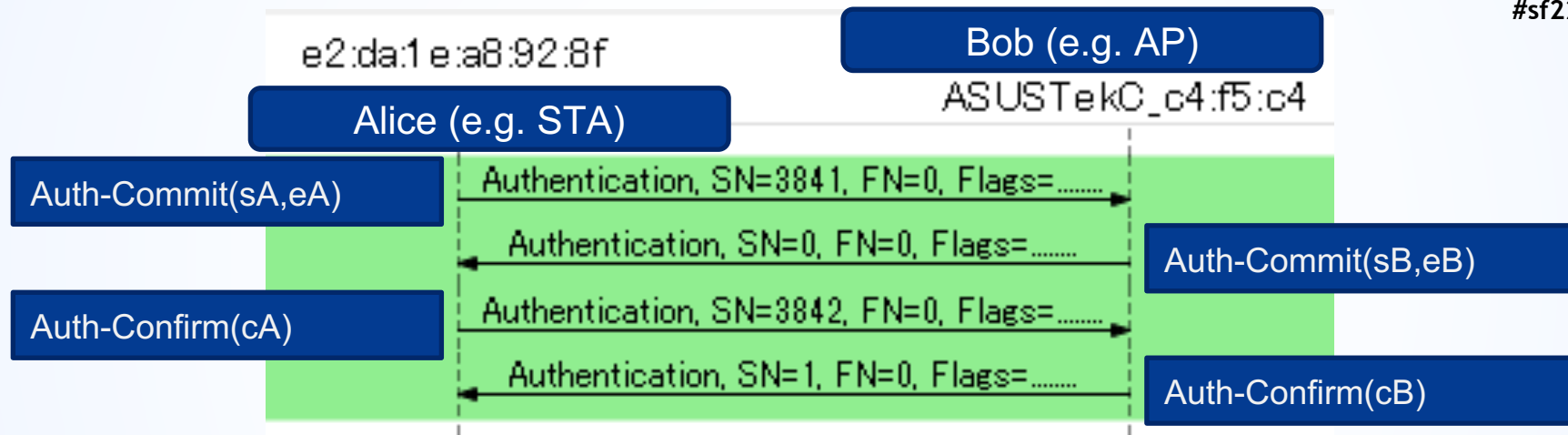
- We can translate the Elliptic-Curve Discrete Logarithm Problem in a computer program like,  
int a=2,n=5,p=11, b=a^n mod p (^:exponentiation)  
we can calculate b easily from a,n,p  
$$b = 2 * 2 * 2 * 2 * 2 \bmod 11 = 32 \bmod 11 = 10$$
- So how do we find n (Logarithm) from a,b and p,  
we need to test incrementally, n=1, n=2, n=3, ...  
if the parameters are such a vast number, finding  
Logarithm n is almost impossible to calculate.
- ECC use this ECDLP for encryption.

# Dragonfly handshake of WPA3-PSK



#sf22us

## ○ Create Statistics>Flow Graph



- Both AP and STA can initiate the handshake, send Auth-Commit and Auth-Confirm each other with scholar and (finite field) element value.

```

- IEEE 802.11 Wireless Management
  - Fixed parameters (104 bytes)
    Authentication Algorithm: Simultaneous Authentication of Equals (SAE) (3)
    Authentication SEQ: 0x0001
    Status code: Successful (0x0000)
    SAE Message Type: Commit (1)
    Group Id: 256-bit random ECP group (19)
    Scalar: c67801ac5941d1e0fad412b255567e53c885a0d12a22439a3e021c7d633f37e7
    Finite Field Element: f4b7c34e9f0d5444381e1dde353e54dcc838435b372a3933b7ccc

```

```

- IEEE 802.11 Wireless Management
  - Fixed parameters (104 bytes)
    Authentication Algorithm: Simultaneous Authentication of Equals (SAE) (3)
    Authentication SEQ: 0x0001
    Status code: Successful (0x0000)
    SAE Message Type: Commit (1)
    Group Id: 256-bit random ECP group (19)
    Scalar: 3fed4910393e5fa8fa5ac12ab8fa9bdfcf8094ded96acfa887620f801c0ee564
    Finite Field Element: 94a0809ac7b9759a54dc8a9e408e7566f053d79673f2f5a650ed

```

```

- IEEE 802.11 Wireless Management
  - Fixed parameters (40 bytes)
    Authentication Algorithm: Simultaneous Authentication of Equals (SAE) (3)
    Authentication SEQ: 0x0002
    Status code: Successful (0x0000)
    SAE Message Type: Confirm (2)
    Send-Confirm: 1
    Confirm: a05a007d7ffce2004c85d7d7d32296c5b8ffa87e5777d2dfa3f7a8e76fce2343

```

```

- IEEE 802.11 Wireless Management
  - Fixed parameters (40 bytes)
    Authentication Algorithm: Simultaneous Authentication of Equals (SAE) (3)
    Authentication SEQ: 0x0002
    Status code: Successful (0x0000)
    SAE Message Type: Confirm (2)
    Send-Confirm: 0
    Confirm: 6dc3f845c7772c6fa7ec01b95802b850ceb848e9dd13019c6513c3311c05cc4f

```

- SAE handshake has 2 Auth-Commit and 2 Auth-Confirm message
- Auth-Commit has Scalar( $s_A, s_B$ ) and Finite Field Element ( $e_A, e_B$ )
- Auth-Confirm has a Confirm value
- They create and share PMK during these 4 packets



# #1:Auth-Commit from Alice(STA)

## ✓ IEEE 802.11 Wireless Management

### ✓ Fixed parameters (104 bytes)

Authentication Algorithm: Simultaneous Authentication of Equals (SAE) (3)

Authentication SEQ: 0x0001

Status code: Successful (0x0000)

SAE Message Type: Commit (1)

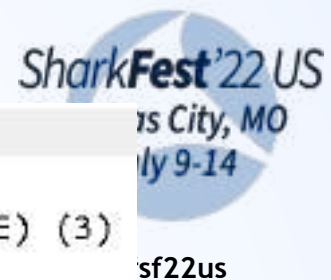
Group Id: 256-bit random ECP group (19)

Scalar: c67801ac5941d1e0fad412b255567e53c885a0d12a22439a3e021c7d633f37e7

Finite Field Element: f4b7c34e9f0d5444381e1dde353e54dcc838435b372a3933b7cc

- Alice(STA) picks random  $r_A$  and  $m_A$  and calculates  $s_A = (r_A + m_A) \bmod q$   
 $e_A = -m_A \cdot GF(p)$  (  $\cdot$  means inner products of vector)
- Then send Auth-Commit with  $s_A$  (256bits Scalar) and  $GF(p)$  (512bits Finite Field Element)

## #2:Auth-Commit from Bob(AP) at same time



### IEEE 802.11 Wireless Management

#### Fixed parameters (104 bytes)

Authentication Algorithm: Simultaneous Authentication of Equals (SAE) (3)

Authentication SEQ: 0x0001

Status code: Successful (0x0000)

SAE Message Type: Commit (1)

Group Id: 256-bit random ECP group (19)

Scalar: 3fed4910393e5fa8fa5ac12ab8fa9bdfcf8094ded96acfa887620f801c0ee564

Finite Field Element: 94a0809ac7b9759a54dc8a9e408e7566f053d79673f2f5a650ed

- Bob(AP) picks random  $r_B$  and  $m_B$  and calculates  $s_B = (r_B + m_B) \bmod q$   
 $e_B = -m_B \cdot GF(p)$  (  $\cdot$  means inner products of vector)
- Then send Auth-Commit with  $s_B$  (256bits Scalar) and  $GF(p)$  (512bits Finite Field Element)

# Auth-Commit (Scalar, Finite Field Element)

Random:  $a, A$

$$sA = (a + A) \bmod q$$

$$\text{element}A = PE^{-A}$$

Random:  $b, B$

$$sB = (b + B) \bmod q$$

$$\text{element}B = PE^{-B}$$

$$sA, PE^{-A}$$

$$sB, PE^{-B}$$

$$\begin{aligned} ss &= (PE^{sB} \times PE^{-B})^a \\ &= (PE^{b+B-B})^a \\ &= PE^{ab} \end{aligned}$$

$$\begin{aligned} ss &= (PE^{sA} \times PE^{-A})^b \\ &= (PE^{a+A-A})^b \\ &= PE^{ab} \end{aligned}$$

- Each Alice(STA) and Bob(AP) calculate their own and the other side Scalar and Finite field element to create and share PE(Password Equivalent) value.

### 3.2.1. Hunting and Pecking with ECC

- Each Alice(STA) and Bob(AP) determine random values and  $GF(p)$  Finite Field Element, but How? <sup>#sf22us</sup>
- RFC7664 Dragonfly key exchange defines a “Hunting and Pecking” algorithm to determine PE(Password Equivalent), try to find the point in the Elliptic Curve from Alice(STA) and Bob(AP) MAC addresses.  
<https://www.rfc-editor.org/info/rfc7664>
- We need over 40 times iterations of hunting and pecking loop against side-channel attack.  
(first implementation of Dragonfly)

### 3.2.1. Hunting and Pecking with ECC

- We calculate the base value, the hash from the counter, mac addresses of Alice and Bob and the passphrase (counter=1)<sup>#sf22us</sup>  
$$\text{base} = H(\text{max}(\text{Alice}, \text{Bob}) \mid \text{min}(\text{Alice}, \text{Bob}) \mid \text{password} \mid \text{counter})$$
- We use KDF(key derivation function) to create bitstream temp value(length is prime number) and the seed  
$$n = \text{len}(p) + 64$$
$$\text{temp} = \text{KDF-}n(\text{base}, \text{"Dragonfly Hunting and Pecking"})$$
$$\text{seed} = (\text{temp} \bmod (p - 1)) + 1$$
- Start loop to find the valid point of Elliptic Curve, use seed as x-axis parameter to check  $x^3 + a*x + b$  is a quadratic residue modulo  $p$ . if not, the counter increase, create new seed and set x-axis, current base value.



# Hunting and Pecking with ECC Groups 3.2.1 RFC7664



#sf22us

```

found = 0
  counter = 1
  n = len(p) + 64
  do {
    base = H(max(Alice,Bob) | min(Alice,Bob) | password |
counter)
    temp = KDF-n(base, "Dragonfly Hunting And Pecking")
    seed = (temp mod (p - 1)) + 1
    if ( (seed^3 + a*seed + b) is a quadratic residue mod p)
    then
      if ( found == 0 )
      then
        x = seed
        save = base
        found = 1
      fi
    fi
    counter = counter + 1
  } while ((found == 0) || (counter <= k))
    y = sqrt(x^3 + ax + b)
    if ( lsb(y) == lsb(save) )
    then
      PE = (x,y)
    else
      PE = (x,p-y)
    fi

```

# #3:Auth-Confirm from Alice(STA)

```

v IEEE 802.11 Wireless Management
  v Fixed parameters (40 bytes)
    Authentication Algorithm: Simultaneous Authentication of Equals (SAE) (3) 22us
    Authentication SEQ: 0x0002
    Status code: Successful (0x0000)
    SAE Message Type: Confirm (2)
    Send-Confirm: 1
    Confirm: e05e00747ffce2d04a55d7d7d32296c5b8ffa07e5777d2dfa3f7a8e74fce2343
  
```

- Alice(STA) verifies  $s_B$  and  $e_B$ , calculates  
 $K = r_A \cdot (s_B \cdot P + e_B)$  (  $\cdot$  means inner products)  
 $tr = (s_A, e_A, s_B, e_B)$  (Alice and Bob know these values)  
 $c_A = \text{HMAC}(\text{Hash}(K), tr)$
- Then send Auth-Confirm with  $c_A$  (256bits Confirm)

## #4:Auth-Confirm from Bob(AP)



```

v IEEE 802.11 Wireless Management
  v Fixed parameters (40 bytes)
    Authentication Algorithm: Simultaneous Authentication of Equals (SAE) (3)
    Authentication SEQ: 0x0002
    Status code: Successful (0x0000)
    SAE Message Type: Confirm (2)
    Send-Confirm: 0
    Confirm: 6dc3f845c7772c6fa7ec01b95802b850ceb840e9dd13019c6515c3311c05cc4f

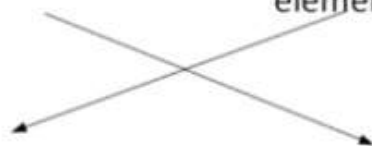
```

- Bob(AP) verifies  $sA$  and  $eA$ , calculates  $K = rB \cdot (sA \cdot P + eA)$  ( $\cdot$  means inner products)  
 $tr = (sB, eB, sA, eA)$  (Alice and Bob know these values)  
 $cB = \text{HMAC}(\text{Hash}(K), tr)$
- Then send Auth-Confirm with  $cB$  (256bits Confirm)

# Auth-Confirm (Confirm value)

Confirm-A = Hash(KCK | scalarA | a |  
elementA | elementB)

Confirm-B = Hash(KCK | scalarB | b,  
elementB | elementA)



#sf22us

- Each Alice(STA) and Bob(AP) can verify the packet's Confirm value with the calculated Confirm value
- $K = r_B \cdot (s_A \cdot P + e_A)$        $K = r_A \cdot (s_B \cdot P + e_B)$   
 $tr = (s_B, e_B, s_A, e_A)$        $tr = (s_A, e_A, s_B, e_B)$   
 $c_B = \text{HMAC}(\text{Hash}(K), tr)$        $c_A = \text{HMAC}(\text{Hash}(K), tr)$
- If the calculated Confirm value is the same as the packet, we can share PE(Password Equivalent) value without sending passphrase information to each other.

# PMK creation from PE value



#sf22us

- Then Alice(STA) and Bob(AP) create PMK from PE(Password Equivalent) value.
- Random values make PE, so PMK is different every time during Dragonfly key exchange
- Let's check this, dragonfly\_implementation.py is the sample Python code for the Dragonfly (SAE) handshake implementation by NikolaiT.  
[https://github.com/NikolaiT/Dragonfly-SAE/blob/master/dragonfly\\_implementation.py](https://github.com/NikolaiT/Dragonfly-SAE/blob/master/dragonfly_implementation.py)
- Open dragonfly\_implementation.py in VSCode



- There are many parameters, such as ECC curve value, I also set the parameters of the passphrase, STA mac Address and AP mac Address.
- Note this code is not actual, just a demonstration example.
- Please run this code more than 2 times and check the outputs.

**#sf22us**

[illegible]

- Please check Commit Value
  - [STA]Scalar and [STA] Element(Finite Field Element)
  - [AP]Scalar and [AP] Element (Finite Field Element)
- Please check Confirm Value
  - [STA] Received Token from Peer
  - [AP] Received Token from Peer

# Check Confirm value

```
2022-07-03 23:34:02,968 - dragonfly - INFO - Confirm Exchange...
```

```
2022-07-03 23:34:02,968 - dragonfly - INFO - [STA] Computed Token from Peer=13e297cb9a9dcea5d13be06237fa2878dadf4d994287bf0f1de343bcbac51cfd
2022-07-03 23:34:02,968 - dragonfly - INFO - [STA] Received Token from Peer=13e297cb9a9dcea5d13be06237fa2878dadf4d994287bf0f1de343bcbac51cfd
2022-07-03 23:34:02,968 - dragonfly - INFO - [STA] Pairwise Master Key(PMK)=e0c89767cd00f2f6193bb9a39a7df0f7e37a853ff4e6e612b84b4967ca0849b9
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Computed Token from Peer=c16f119f89b4b0c39b3450c0466baacacaa1824b8a0ab2dd1fb46ffb33bef4c8
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Received Token from Peer=c16f119f89b4b0c39b3450c0466baacacaa1824b8a0ab2dd1fb46ffb33bef4c8
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Pairwise Master Key(PMK)=e0c89767cd00f2f6193bb9a39a7df0f7e37a853ff4e6e612b84b4967ca0849b9
PS C:\Users\takeshitaseguri\OneDrive - いけり ネットワークサービス株式会社\sharkfest\sharkfest2022\03DissectingMPA1>
```

- Also, check Packet's Confirm Value is the same with calculated Confirm Value

[STA] Computed Token from Peer is the same with

[STA] Received Token from Peer

[AP] Computed Token from Peer is the same with

[AP] Received Token from Peer

# Check shared secret(PE), PMK

- Look for each shared secret(Password Equivalent) value between Alice(STA) and Bob(AP).  
[STA] Shared Secret and [AP] Shared Secret
- Also, check PMK values, [STA] Pairwise Master Key(PMK) and [AP] Pairwise Master Key(PMK)  
Yes, we can share PE, PMK sending passphrase information to each other.
- Let's try over 2 times. You can find these values are different at every try.

# PMK, shared key is different!!



## 1<sup>st</sup> try

```

2022-07-03 23:34:02,858 - dragonfly - INFO - [STA] Shared Secret ss=69807381929808938225177021188985319505788035726401238610784021527476497801166
2022-07-03 23:34:02,968 - dragonfly - INFO - [AP] Shared Secret ss=69807381929808938225177021188985319505788035726401238610784021527476497801166
2022-07-03 23:34:02,968 - dragonfly - INFO - Confirm Exchange...
2022-07-03 23:34:02,968 - dragonfly - INFO - [STA] Computed Token from Peer=13e297cb9a9dcea5d33be06237fa2878dadfd994287bf0f1de343bcbac51cfd
2022-07-03 23:34:02,968 - dragonfly - INFO - [STA] Received Token from Peer=13e297cb9a9dcea5d33be06237fa2878dadfd994287bf0f1de343bcbac51cfd
2022-07-03 23:34:02,968 - dragonfly - INFO - [STA] Pairwise Master Key(PMK)=e0c89767cd0bf2f6193bb9439a7d0f7e37a853ff4e6e612b84b4967ca0849b9
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Computed Token from Peer=c16f119f89b4b0e39b3450c0466baacacaa1824b8a0ab2dd11b4611b33bef4c8
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Received Token from Peer=c16f119f89b4b0e39b3450c0466baacacaa1824b8a0ab2dd11b4611b33bef4c8
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Pairwise Master Key(PMK)=e0c89767cd0bf2f6193bb9439a7d0f7e37a853ff4e6e612b84b4967ca0849b9
PS C:\Users\takeshita\OneDrive - いけり ネットワークサービス株式会社\Sharkfest\Sharkfest2022\03DissectingMPA3>

```



## 2<sup>nd</sup> try

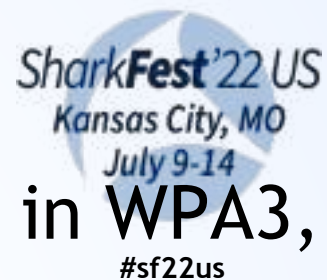
```

2022-07-03 23:51:33,556 - dragonfly - INFO - [STA] Shared Secret ss=1189115326235807803685727071259681376814918908581134690318478988316425160022
2022-07-03 23:51:33,665 - dragonfly - INFO - [AP] Shared Secret ss=1189115326235807803685727071259681376814918908581134690318478988316425160022
2022-07-03 23:51:33,666 - dragonfly - INFO - Confirm Exchange...
2022-07-03 23:51:33,666 - dragonfly - INFO - [STA] Computed Token from Peer=ae98151668650840769ccabb0951d16ddbabb736f4d6500272b0ec7c12d64ffb1
2022-07-03 23:51:33,666 - dragonfly - INFO - [STA] Received Token from Peer=ae98151668650840769ccabb0951d16ddbabb736f4d6500272b0ec7c12d64ffb1
2022-07-03 23:51:33,667 - dragonfly - INFO - [STA] Pairwise Master Key(PMK)=d29fb108095ad3ec6b0cf4da267eaad8605c6b45ef7a5dbf5d5632689e4190f0
2022-07-03 23:51:33,667 - dragonfly - INFO - [AP] Computed Token from Peer=e6d936182c952456c74cc26b05636965c531d0f1d71ed4d6c9a2759f70fe31d7
2022-07-03 23:51:33,667 - dragonfly - INFO - [AP] Received Token from Peer=e6d936182c952456c74cc26b05636965c531d0f1d71ed4d6c9a2759f70fe31d7
2022-07-03 23:51:33,667 - dragonfly - INFO - [AP] Pairwise Master Key(PMK)=d29fb108095ad3ec6b0cf4da267eaad8605c6b45ef7a5dbf5d5632689e4190f0
PS C:\Users\takeshita\OneDrive - いけり ネットワークサービス株式会社\Sharkfest\Sharkfest2022\03DissectingMPA3>

```



# Forward Security, PMF and lockout



- We understand PMK is different every time in WPA3, it provides Forward Security.
- We cannot use a offline dictionary attack
- Deauth attack is impossible with PMF (Protected Management Frames). (optionally)
- Wrong passphrase lockout function prevents brute force attack. (optionally)
- WPA3-SAE is (almost) impossible for cracking now.

# Vulnerabilities of WPA3 dragon blood



#sf22us

- Downgrade WPA3-SAE to WPA2-PSK
  - DoS attack with over 70 connection requests (Hunting and Pecking calculation DoS) may stop AP.
  - Hunting and Pecking use 40 round time to find random values, so the old implementation may be weak with a side-channel attack.
  - Chosen random value attack: set rB to zero.
  - Enable Brute force using a faked mac address to avoid lockout, and so on...
- > these vulnerabilities are (almost) fixed now!!

## Appendix: WPA3-EAP



- WPA3 Enterprise mode is called WPA3-EAP
- WPA3-EAP use CSNA (Commercial National Security Algorithm) 192bit encryption instead of AES.
- WPA3 needs a RADIUS (802.1x authentication) server  
We can use TLS, LEAP, PEAP and other authentication methods, the authentication server provides each connection's PMK.
- WPA3-EAP is the best choice if your network has many users and APs. ( if your company has a budget)

# USE WIRESHARK



# Thank you for watching !!

#sf22us

**Please complete app-based survey**

trace files and python codes are here:

<https://www.ikeriri.ne.jp/sharkfest/03DissectingWPA3.zip>



ikeriri network service

<http://www.ikeriri.ne.jp>