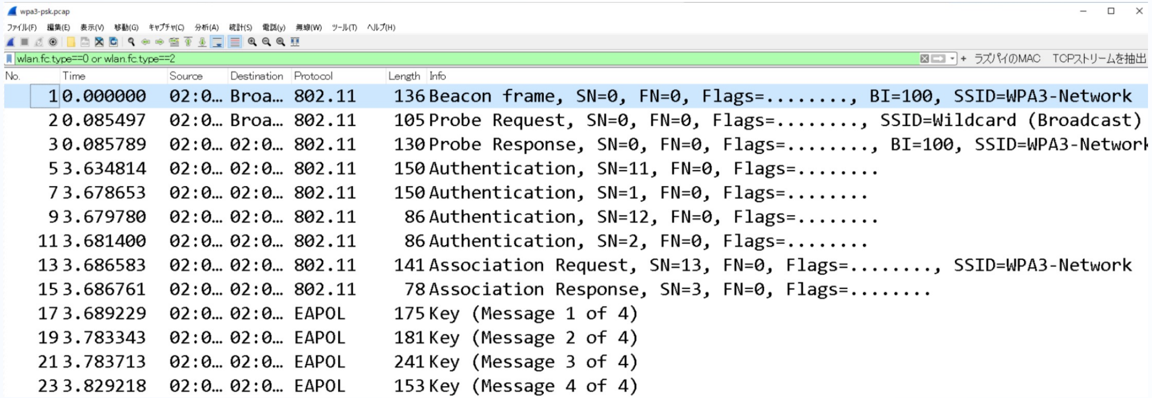
Dissecting WPA3



Megumi Takeshita
Ikeriri network service

Materials: all trace files, python codes are here

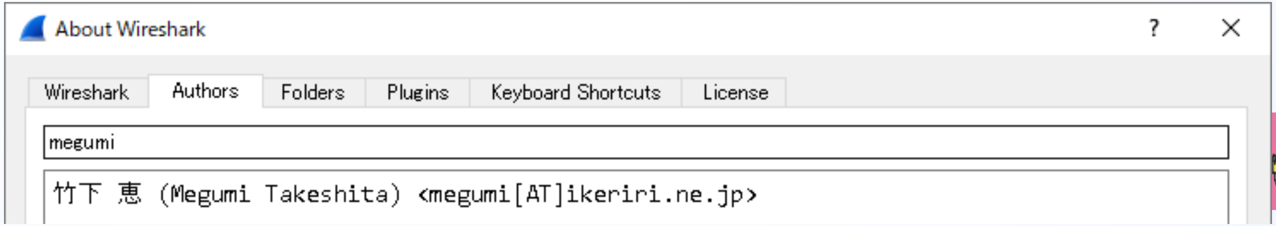
https://www.ikeriri.ne.jp/sharkfest/03DissectingWPA3.zip

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

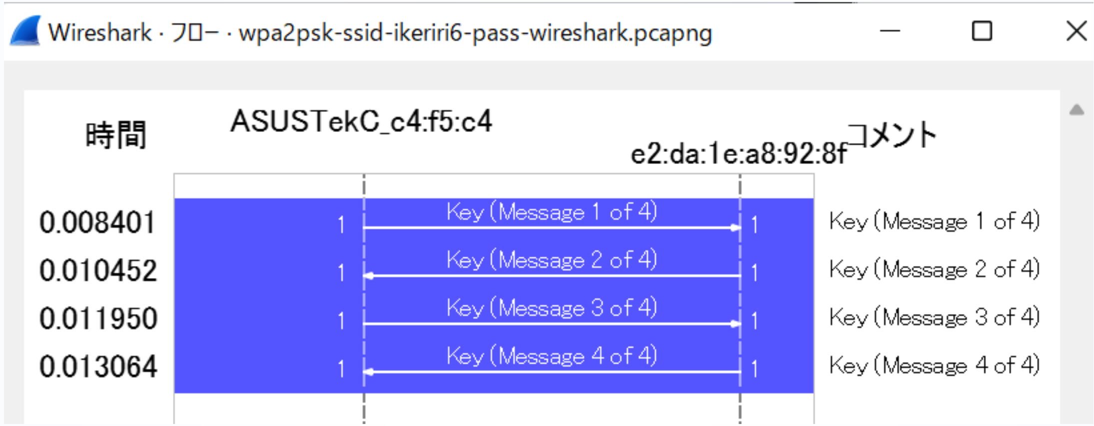
WPA2-PSK dissection

- 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	Frame is not…	QoS Data
10	0.072892	802.11ax (HE)	56	20	-41 dBm	-91 dBm	0xb	270.8	Frame is not…	QoS Data
11	0.073031	802.11ax (HE)	56	20	-41 dBm	-91 dBm	0xb	270.8	Frame is not…	QoS Data
12	0.073376	802.11ax (HE)	56	20	-41 dBm	-91 dBm	0xb	270.8	Frame is not…	QoS Data
13	0.073578	802.11a (OFDM)	56		-33 dBm	-91 dBm		6	Frame is not…	Data

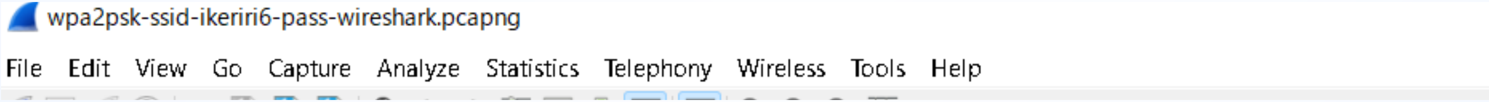
4way handshake (common in WPA2 and WPA3)

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

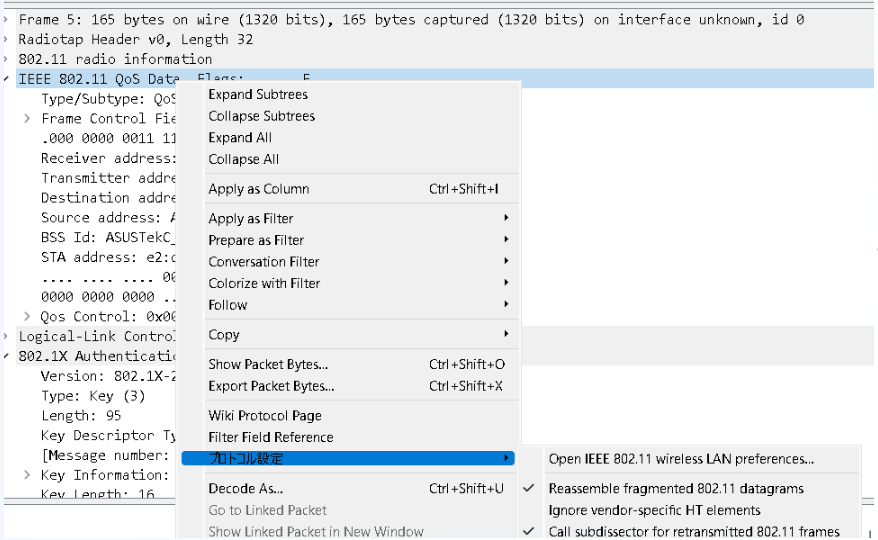


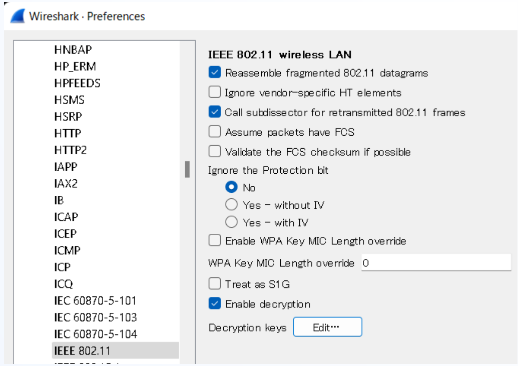
- At first, AP sends Key(Message 1 of 4) and STA reply Key (Message 2 of 4) to exchange PTK for unicast. Then AP sends Key (Message 3 of 4), STA replies. Key (Message 4 of 4) to confirm GTK for multicast.

- 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



Set Decryption key in Wireshark



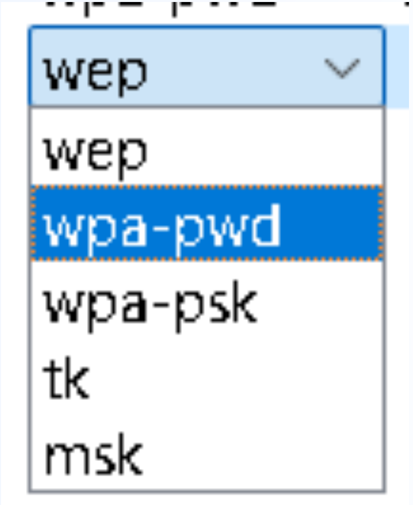


#sf22us

- 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

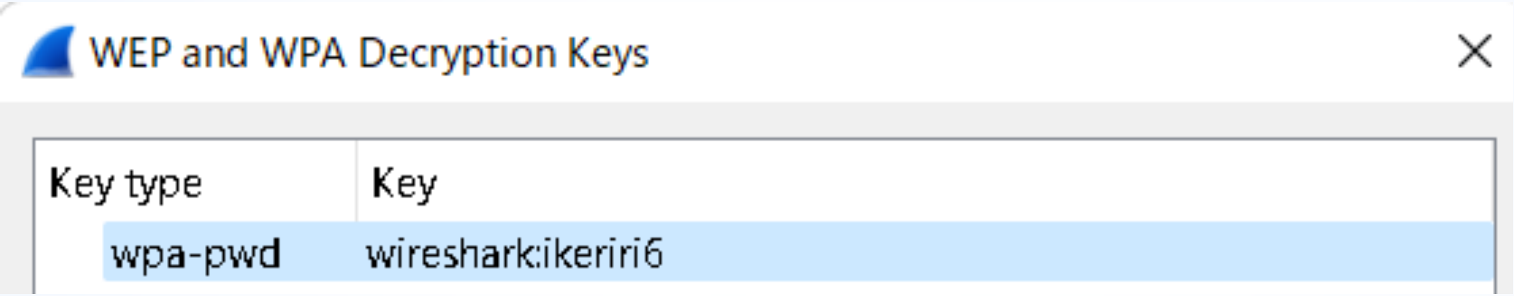
SharkFest'22 US
Kansas City, MO
July 9-14

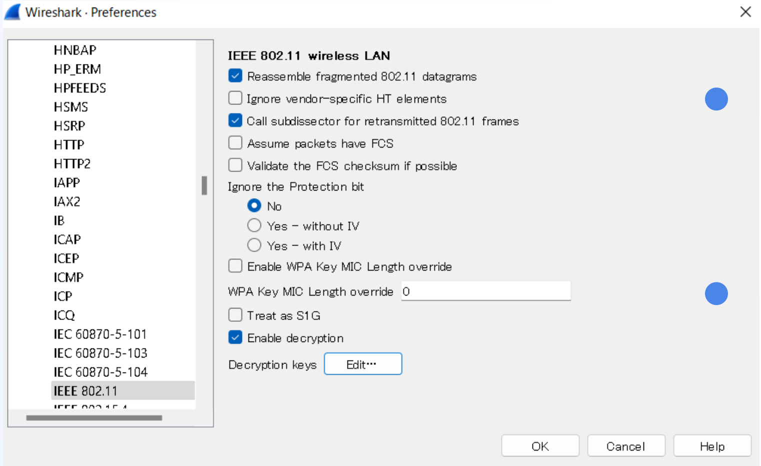
- We can set 5 styles of WEP and WPA decryption keys in Wireshark 3.6



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 https://www.wireshark.org/tools/wpa-psk.html
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





- 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

SharkFest'22 US Kansas City, MO

No.	Time	PHY type	Chanr	data E	Signal (dBm	Noise (dBn	data N	Data rat	Retry	Type/Subtype	Source	Destination	Lengtl	Info
1	0.000…	802.11a (OFDM)	56		-39 dBm	-93 dBm		6	Frame is …	Authentication	e2:da:1e:a8:92:8f	ASUSTekC…	97	Authentication, SN=833, FN=0, Flags=........
2	0.001…	802.11a (OFDM)	56		-33 dBm	-93 dBm		6	Frame is …	Authentication	ASUSTekC_c4:f5:c4	e2:da:1e…	62	Authentication, SN=0, FN=0, Flags=........
3	0.002…	802.11a (OFDM)	56		-41 dBm	-93 dBm		6	Frame is …	Association Request	e2:da:1e:a8:92:8f	ASUSTekC…	237	Association Request, SN=834, FN=0, Flags=........, SSID=ikeriri6
4	0.005…	802.11a (OFDM)	56		-33 dBm	-93 dBm		6	Frame is …	Association Response	ASUSTekC_c4:f5:c4	e2:da:1e…	318	Association Response, SN=1, FN=0, Flags=........
5	0.008…	802.11a (OFDM)	56		-33 dBm	-93 dBm		6	Frame is …	QoS Data	ASUSTekC_c4:f5:c4	e2:da:1e…	165	Key (Message 1 of 4)
6	0.010…	802.11a (OFDM)	56		-41 dBm	-93 dBm		6	Frame is …	QoS Data	e2:da:1e:a8:92:8f	ASUSTekC…	187	Key (Message 2 of 4)
7	0.011…	802.11a (OFDM)	56		-33 dBm	-93 dBm		6	Frame is …	QoS Data	ASUSTekC_c4:f5:c4	e2:da:1e…	253	Key (Message 3 of 4)
8	0.013…	802.11a (OFDM)	56		-41 dBm	-91 dBm		6	Frame is …	QoS Data	e2:da:1e:a8:92:8f	ASUSTekC…	165	Key (Message 4 of 4)
9	0.072…	802.11ax (HE)	56	20	-41 dBm	-91 dBm	0xb	270.8	Frame is …	QoS Data	e2:da:1e:a8:92:8f	IPv6mcas…	170	QoS Data, SN=0, FN=0, Flags=op.....T
10	0.072…	802.11ax (HE)	56	20	-41 dBm	-91 dBm	0xb	270.8	Frame is …	QoS Data	e2:da:1e:a8:92:8f	ASUSTekC…	126	QoS Data, SN=1, FN=0, Flags=op.....T
11	0.073…	802.11ax (HE)	56	20	-41 dBm	-91 dBm	0xb	270.8	Frame is …	QoS Data	e2:da:1e:a8:92:8f	Broadcast	426	QoS Data, SN=2, FN=0, Flags=op.....T
12	0.073…	802.11ax (HE)	56	20	-41 dBm	-91 dBm	0xb	270.8	Frame is …	QoS Data	e2:da:1e:a8:92:8f	IPv6mcas…	146	QoS Data, SN=3, FN=0, Flags=op.....T
13	0.073…	802.11a (OFDM)	56		-33 dBm	-91 dBm		6	Frame is …	Data	::	ff02::1:…	136	Neighbor Solicitation for fe80::1c42:c607:6801:27fa

No.	Time	PHY type	Chanr	data E	Signal (dBm	Noise (dBn	data N	Data rat	Retry	Type/Subtype	Source	Destination	Lengtl	Info
1	0.000…	802.11a (OFDM)	56		-39 dBm	-93 dBm		6	Fra		:1e:a8:92:8f	ASUSTekC…	97	Authentication, SN=833, FN=0, Flags=........
2	0.001…	802.11a (OFDM)	56		-33 dBm	-93 dBm		6	Frame is		ASUSTekC_c4:f5:c4	e2:da:1e…	62	Authentication, SN=0, FN=0, Flags=........
3	0.002…	802.11a (OFDM)	56		-41 dBm	-93 dBm		6	Frame is …	Ass	e2:da:1e:a8:92:8f	ASUSTekC…	237	Association Request, SN=834, FN=0, Flags=........, SSID=ikeriri6
4	0.005…	802.11a (OFDM)	56		-33 dBm	-93 dBm		6	Frame is …	Associati esponse	ASUSTekC_c4:f5:c4	e2:da:1e…	318	Association Response, SN=1, FN=0, Flags=........
5	0.008…	802.11a (OFDM)	56		-33 dBm	-93 dBm		6	Frame is …	QoS Data	ASUSTekC_c4:f5:c4	e2:da:1e…	165	Key (Message 1 of 4)
6	0.010…	802.11a (OFDM)	56		-41 dBm	-93 dBm		6	Frame is …	QoS Data	e2:da:1e:a8:92:8f	ASUSTekC…	187	Key (Message 2 of 4)
7	0.011…	802.11a (OFDM)	56		-33 dBm	-93 dBm		6	Frame is …	QoS Data	ASUSTekC_c4:f5:c4	e2:da:1e…	253	Key (Message 3 of 4)
8	0.013…	802.11a (OFDM)	56		-41 dBm	-91 dBm		6	Frame is …	QoS Data	e2:da:1e:a8:92:8f	ASUSTekC…	165	Key (Message 4 of 4)
9	0.072…	802.11ax (HE)	56	20	-41 dBm	-91 dBm	0xb	270.8	Frame is …	QoS Data	::	ff02::1:…	170	Neighbor Solicitation for fe80::1c42:c607:6801:27fa
10	0.072…	802.11ax (HE)	56	20	-41 dBm	-91 dBm	0xb	270.8	Frame is …	QoS Data	e2:da:1e:a8:92:8f	ASUSTekC…	126	Who has 192.168.50.1? Tell 192.168.50.236
11	0.073…	802.11ax (HE)	56	20	-41 dBm	-91 dBm	0xb	270.8	Frame is …	QoS Data	0.0.0.0	255.255.…	426	DHCP Request - Transaction ID 0xac9e7500
12	0.073…	802.11ax (HE)	56	20	-41 dBm	-91 dBm	0xb	270.8	Frame is …	QoS Data	fe80::1c42:c607:68…	ff02::2	146	Router Solicitation
13	0.073…	802.11a (OFDM)	56		-33 dBm	-91 dBm		6	Frame is …	Data	::	ff02::1:…	136	Neighbor Solicitation for fe80::1c42:c607:6801:27fa

- 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)


```
v CCMP parameters
    CCMP Ext. Initialization Vector: 0x000000000002
    Key Index: 0
    [TK: 4c102fd43613c535404d0777088a6503]
    [PMK: 31bb75a609a424aac01e9929b39458e87ea45b0f30204ff5642bf3067a6fd31f]
```

- 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

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

#sf22us

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

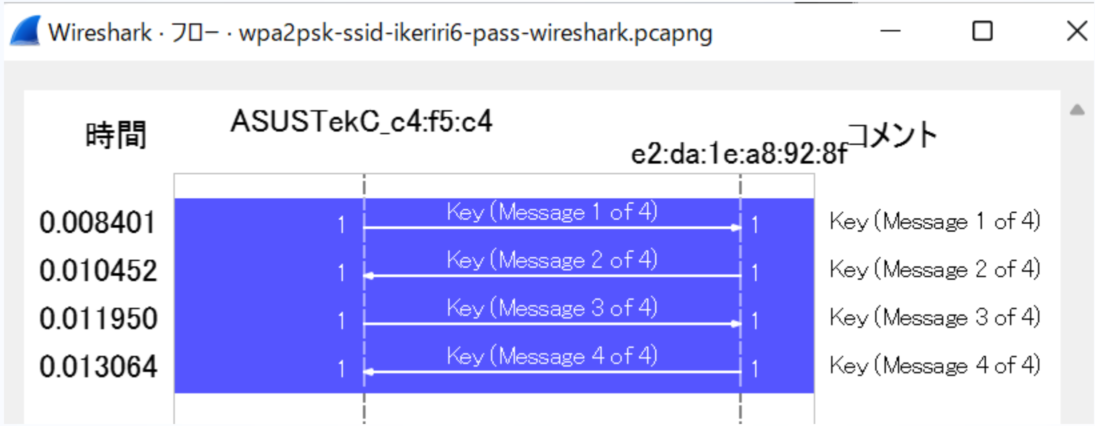
```
[PMK: 31bb75a609a424aac01e9929b39458e87ea45b0f30204ff5642bf3067a6fd31f]
```

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

Let's start dissecting 4way handshake

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

#sf22us

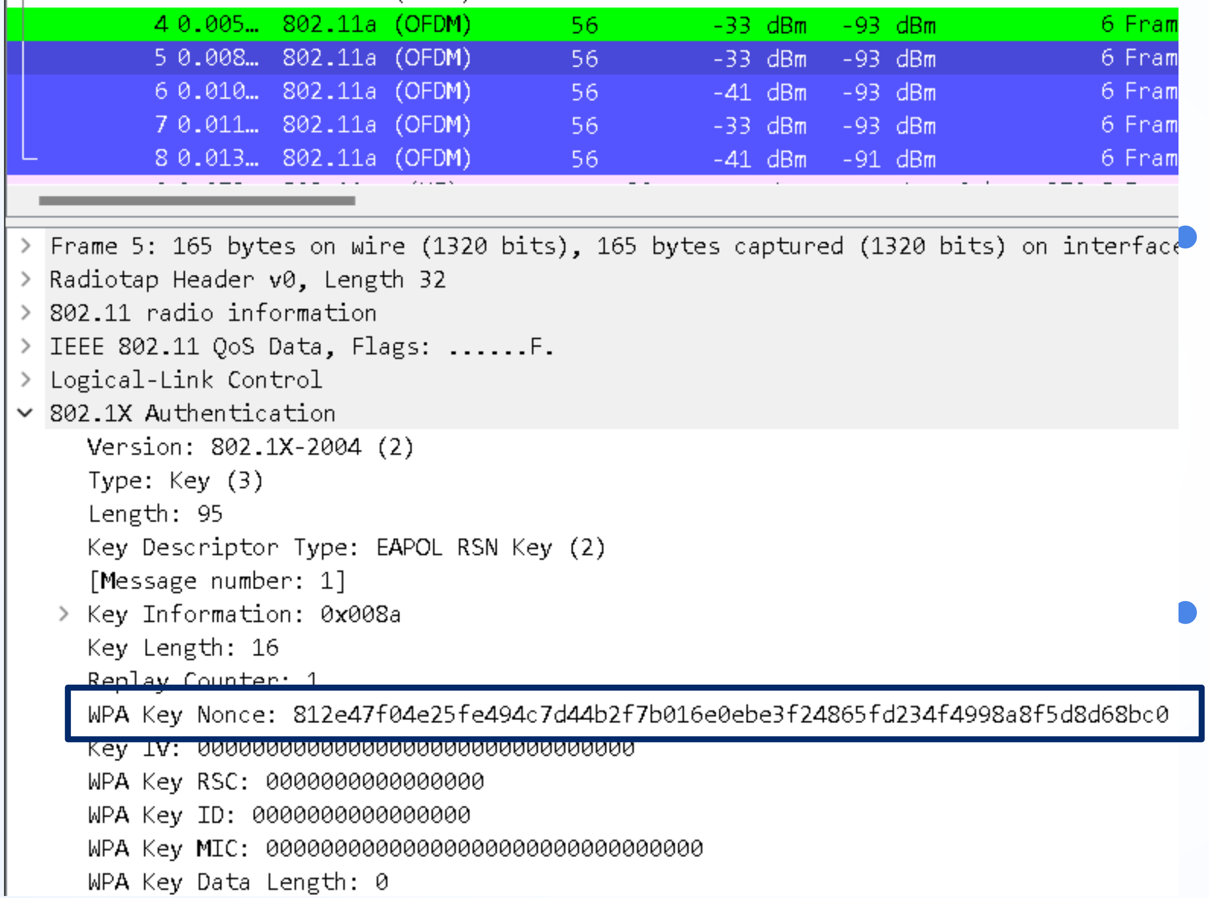


- ASUS(AP) and iPhone13(STA) know the same PMK

  [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

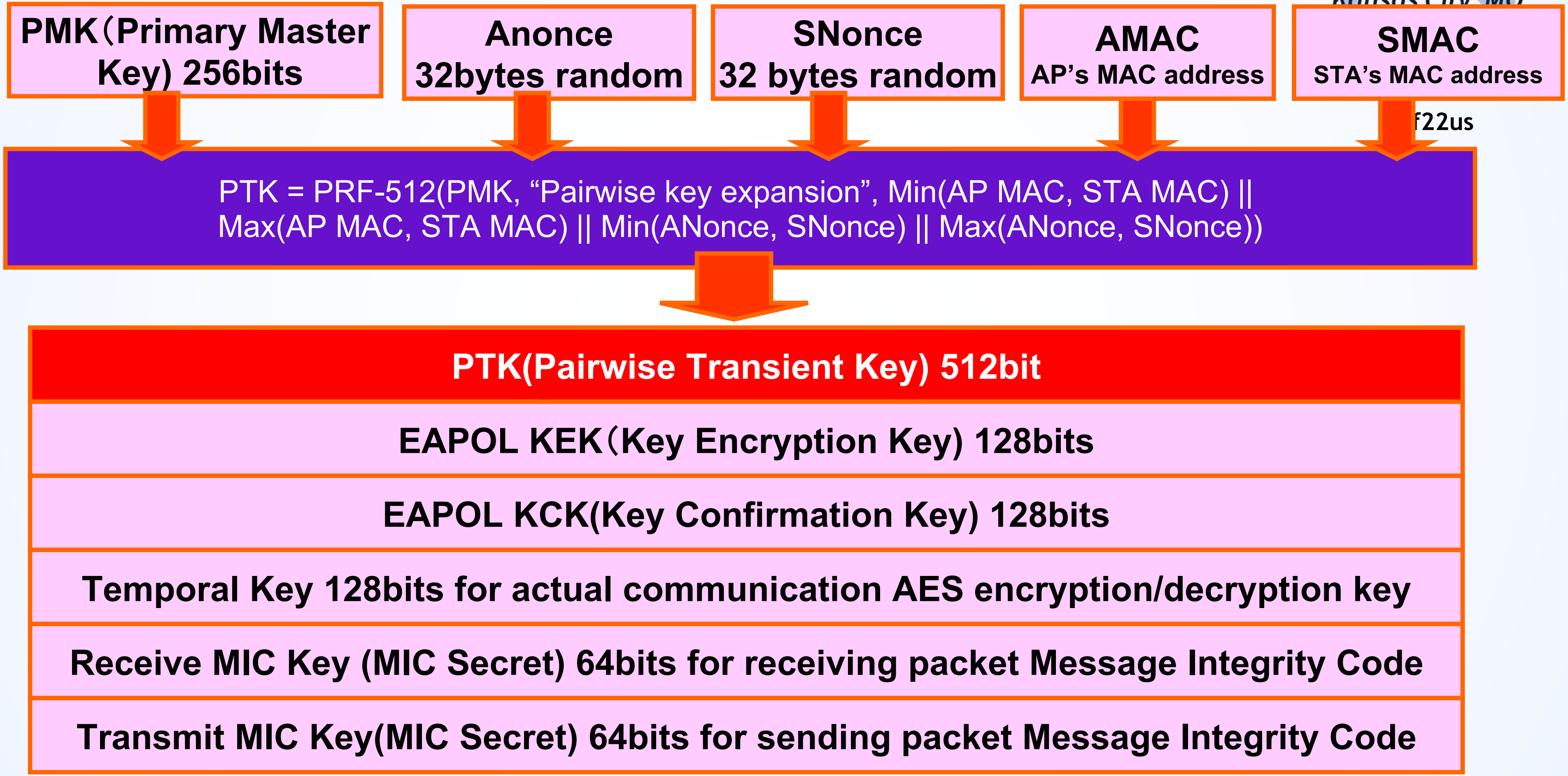


- AP creates and sends Nonce (ANonce),32bytes random value.
- Message1 packet also contains AP's MAC Address.

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

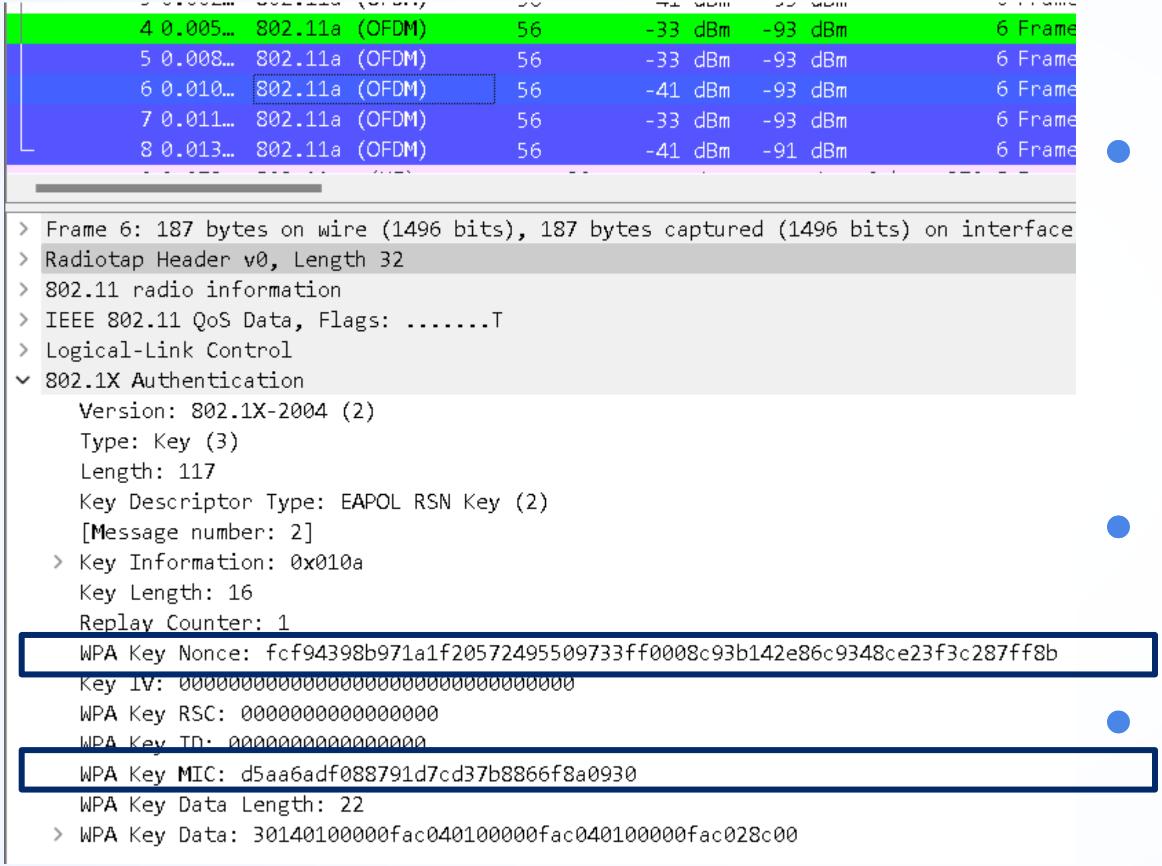
- STA starts to create PTK (Pairwise Transient Key) after STA received EAPOL Message 1of4 (Packet 5)
- STA creates its own Nonce(Snonce), 32bytes random
- PTK is created from ANonce, Snonce, AP MAC address and STA mac address
  PTK = PRF-512(PMK, “Pairwise key expansion”, Min(AP MAC, STA MAC) || Max(AP MAC, STA MAC) || Min(ANonce, SNonce) || Max(ANonce, SNonce))
  *PRF(Pseudo-Random Function) is SHA1 hash value from input paratmeter, 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



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

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



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

STA created SNonce and Add MIC Key Data

- 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
```

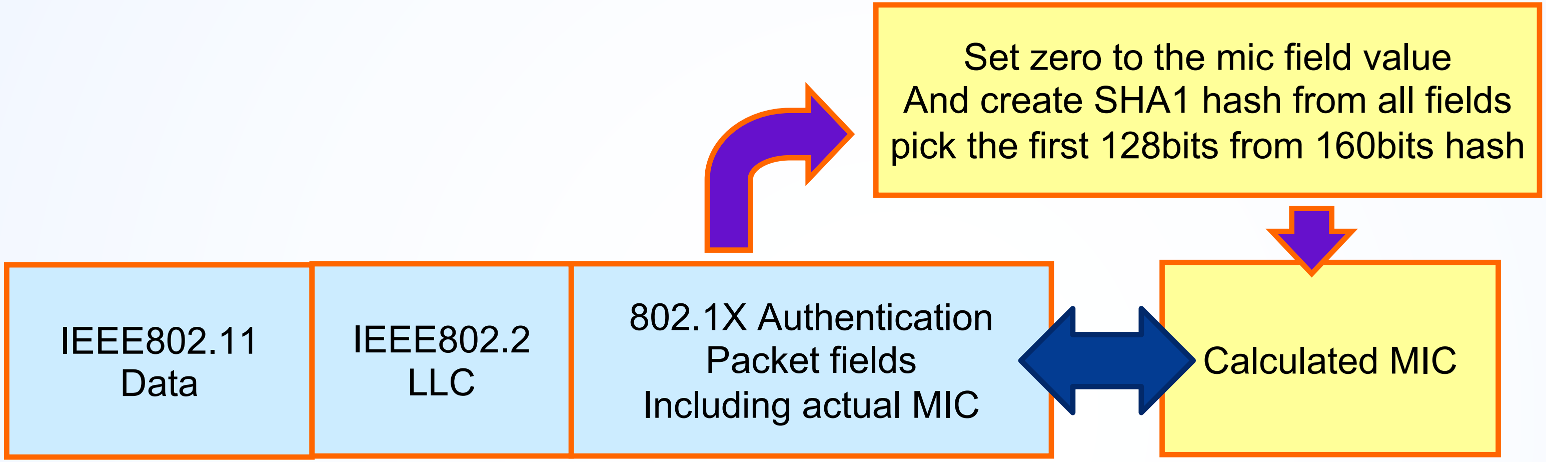
- WPA Key MIC means the confirmation that created PTK is the same with STA and AP (Receiving 2of4 Message, AP also creates PTK and check the WPA Key MIC is correct)

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

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

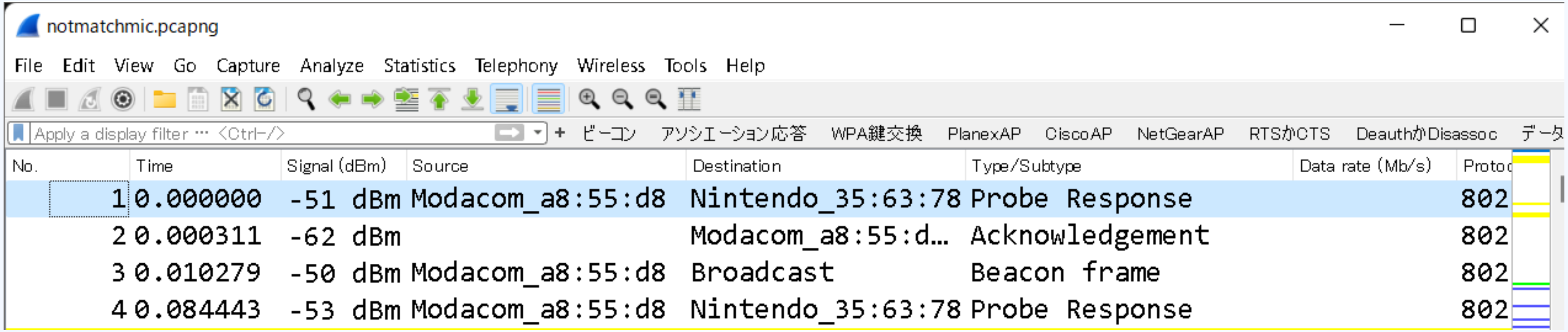
Check Actual MIC with Calculated MIC




```
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: fcf94398b971a1f20572495509733ff0008c93b142e86c9348ce23f3c287ff8b
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: d5aa6adf088791d7cd37b8866f8a0930
    WPA Key Data Length: 22
  > WPA Key Data: 30140100000fac040100000fac040100000fac028c00
```

If calculated MIC is not the same with WPA Key MIC value of Message 2of4 (Packet6), It usually means passphrase is not the same.

- Open notmatchmic.pcapng and set display filter as “wlan.addr_resolved contains Nintendo”



- 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	Disasso…	802.11	62	Disassociate, SN=2, FN=0

PTK exchange Demonstration

- 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)

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

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

#sf22us

```
ptk.py > ...
# ikeriri referred and changed the code from nicholastsmith
# https://nicholastsmith.wordpress.com/2016/11/15/wpa2-key-derivation-

import hmac
from binascii import a2b_hex, b2a_hex
from hashlib import pbkdf2_hmac, sha1, md5
import binascii

#Pseudo-random function for generation of
#the pairwise transient key (PTK)
#key:       The PMK
#A:         b'Pairwise key expansion'
#B:         The apMac, cliMac, aNonce, and sNonce concatenated
#           like mac1 mac2 nonce1 nonce2
#           such that mac1 < mac2 and nonce1 < nonce2
#return:    The ptk
def PRF(key, A, B):
    #Number of bytes in the PTK
    nByte = 64
    i = 0
    R = b''
    #Each iteration produces 160-bit value and 512 bits are required
    while(i <= ((nByte * 8 + 159) / 160)):
        hmacsha1 = hmac.new(key, A + chr(0x00).encode() + B + chr(i).e
        R = R + hmacsha1.digest()
        i += 1
    return R[0:nByte]
```

```
#Make parameters for the generation of the PTK
#aNonce:        The aNonce from the 4-way handshake
#sNonce:        The sNonce from the 4-way handshake
#apMac:         The MAC address of the access point
#cliMac:        The MAC address of the client
#return:        (A, B) where A and B are parameters
#               for the generation of the PTK
def MakeAB(aNonce, sNonce, apMac, cliMac):
    A = b"Pairwise key expansion"
    B = min(apMac, cliMac) + max(apMac, cliMac) + min(aNonce, sNonce)
    return (A, B)

#Compute the 1st message integrity check for a WPA 4-way handshake
#pwd:       The password to test
#ssid:      The ssid of the AP
#A:         b'Pairwise key expansion'
#B:         The apMac, cliMac, aNonce, and sNonce concatenated
#           like mac1 mac2 nonce1 nonce2
#           such that mac1 < mac2 and nonce1 < nonce2
#data:      A list of 802.1x frames with the MIC field zeroed
#return:    (x, y, z) where x is the mic, y is the PTK, and z is the P
def MakeMIC(pwd, ssid, A, B, data, wpa = False):
    #Create the pairwise master key using 4096 iterations of hmac-sha1
    #to generate a 32 byte value
    pmk = pbkdf2_hmac('sha1', pwd.encode('ascii'), ssid.encode('ascii'
    #Make the pairwise transient key (PTK)
    ptk = PRF(pmk, A, B)
    #WPA uses md5 to compute the MIC while WPA2 uses sha1
    hmacFunc = md5 if wpa else sha1
    #Create the MICs using HMAC-SHA1 of data and return all computed v
    mics = [hmac.new(ptk[0:16], i, hmacFunc).digest() for i in data]
    return (mics, ptk, pmk)
```

Run ptk.py with this trace file parameters

naconda-python/


```python
pwd = "wireshark"
print("Passphrase:"+pwd)
ssid = "ikeriri6"
print("SSID:"+ssid)

AMAC="f02f74c4f5c4"
print("AP MAC Address from 1of4:"+AMAC)
apMac = binascii.a2b_hex(AMAC)
ANONCE="812e47f04e25fe494c7d44b2f7b016e0ebe3f24865fd234f4998a8f5d8d68bc0"
print("AP Nonce from 1of4:"+ANONCE)
aNonce = binascii.a2b_hex(ANONCE)

SMAC="e2da1ea8928f"
print("STA MAC Address from 1of4:"+SMAC)
cliMac = binascii.a2b_hex(SMAC)
SNONCE="fcf94398b971a1f20572495509733ff0008c93b142e86c9348ce23f3c287ff8b"
print("STA Nonce from 1of4:"+SNONCE)
sNonce = binascii.a2b_hex(SNONCE)

#The first MIC
mic1 = "d5aa6adf088791d7cd37b8866f8a0930"
#The entire 802.1x frame of the second handshake message with the MIC field set to all zeros
data1 = a2b_hex("0203007502010a00100000000000000001fcf94398b971a1f20572495509733ff0008c93b142e
#The second MIC
mic2 = "8fc6ccf6133542e9f8844ec826de5ff8"
#The entire 802.1x frame of the third handshake message with the MIC field set to all zeros
data2 = a2b_hex("020300b70213ca00100000000000000002812e47f04e25fe494c7d44b2f7b016e0ebe3f24865
#The third MIC
mic3 = "e2367db355ccdf0710d8d73d6e175b5c"
#The entire 802.1x frame of the forth handshake message with the MIC field set to all zeros
data3 = a2b_hex("0203005f02030a0010000000000000000200000000000000000000000000000000000000000000
```

Message(1of4)
packet #5

```
IEEE 802.11 QoS Data, Flags: ......F.
    Type/Subtype: QoS Data (0x0028)
  > Frame Control Field: 0x8802
    .000 0000 0011 1100 = Duration: 60 microseconds
    Receiver address: e2:da:1e:a8:92:8f (e2:da:1e:a8:92:8f)
    Transmitter address: ASUSTekC_c4:f5:c4 (f0:2f:74:c4:f5:c4)
    Destination address: e2:da:1e:a8:92:8f (e2:da:1e:a8:92:8f)
    Source address: ASUSTekC_c4:f5:c4 (f0:2f:74:c4:f5:c4)
    BSS Id: ASUSTekC_c4:f5:c4 (f0:2f:74:c4:f5:c4)
    STA address: e2:da:1e:a8:92:8f (e2:da:1e:a8:92:8f)
    .... .... .... 0000 = Fragment number: 0
    0000 0000 0000 .... = Sequence number: 0
  > Qos Control: 0x0006
> 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: 812e47f04e25fe494c7d44b2f7b016e0ebe3f24865fd234f4998a8f5d8d68bc0
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: 00000000000000000000000000000000
    WPA Key Data Length: 0
```

Message(2of4)
packet #6

```
IEEE 802.11 QoS Data, Flags: .......T
    Type/Subtype: QoS Data (0x0028)
  > Frame Control Field: 0x8801
    .000 0000 0011 1100 = Duration: 60 microseconds
    Receiver address: ASUSTekC_c4:f5:c4 (f0:2f:74:c4:f5:c4)
    Transmitter address: e2:da:1e:a8:92:8f (e2:da:1e:a8:92:8f)
    Destination address: ASUSTekC_c4:f5:c4 (f0:2f:74:c4:f5:c4)
    Source address: e2:da:1e:a8:92:8f (e2:da:1e:a8:92:8f)
    BSS Id: ASUSTekC_c4:f5:c4 (f0:2f:74:c4:f5:c4)
    STA address: e2:da:1e:a8:92:8f (e2:da:1e:a8:92:8f)
    .... .... .... 0000 = Fragment number: 0
    0000 0000 0000 .... = Sequence number: 0
  > Qos Control: 0x1210
> 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: 0x010a
    Key Length: 16
    Replay Counter: 1
    WPA Key Nonce: fcf94398b971a1f20572495509733ff0008c93b142e86c9348ce23f3c287ff8b
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: d5aa6adf088791d7cd37b8866f8a0930
    WPA Key Data Length: 22
  > WPA Key Data: 30140100000fac040100000fac040100000fac028c00
```

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

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

#sf22us

```python
A,B=MakeAB(aNonce, sNonce, apMac, cliMac)
mics,ptk,pmk=MakeMIC(pwd, ssid, A, B, [data1, data2, data3], wpa = False)

print ("PMK: " + pmk.hex())
print ("PTK: " + ptk.hex())
kek=ptk[0:15]
print ("KEK: " + kek.hex())
kck=ptk[16:31]
print ("KCK: " + kck.hex())
tk=ptk[32:48]
print ("TK: " + tk.hex())
rmic=ptk[48:55]
print ("Receive MIC Secret: " + rmic.hex())
tmic=ptk[56:63]
print ("Transmiit MIC Secret: " + tmic.hex())

mic1Str = mic1.upper()
print("dactual mic:" + mic1Str)
#Take the first 128-bits of the 160-bit SHA1 hash
micStr = b2a_hex(mics[0]).decode().upper()[:-8]
print("calculated mic from Message2of4:" + micStr)
print('MATCH' if micStr == mic1Str else 'MISMATCH')
#Display the desired MIC2 and compare to target MIC2
mic2Str = mic2.upper()
print("actual mic:" + mic2Str)
#Take the first 128-bits of the 160-bit SHA1 hash
micStr = b2a_hex(mics[1]).decode().upper()[:-8]
print("calculated mic from Message3of4:" + micStr)
print('MATCH' if micStr == mic2Str else 'MISMATCH')
#Display the desired MIC3 and compare to target MIC3
mic3Str = mic3.upper()
print("packet mic:" + mic3Str)
#Take the first 128-bits of the 160-bit SHA1 hash
micStr = b2a_hex(mics[2]).decode().upper()[:-8]
print("packet mic from Message4of4:" + micStr)
print('MATCH' if micStr == mic3Str else 'MISMATCH')
```

```
PS C:\Users\TakeshitaMegumi\OneDrive - いけり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 - いけりりネ
k.py"
Passphrase:wireshark
SSID:ikeriri6
AP MAC Address from 1of4:f02f74c4f5c4
AP Nonce from 1of4:812e47f04e25fe494c7d44b2f7b016e0ebe3f24865fd234f4998a8f5d8d68bc0
STA MAC Address from 1of4:e2da1ea8928f
STA Nonce from 1of4:fcf94398b971a1f20572495509733ff0008c93b142e86c9348ce23f3c287ff8b
PMK: 31bb75a609a424aac01e9929b39458e87ea45b0f30204ff5642bf3067a6fd31f
PTK: 46e515c2a3677ef693f93e8368517684728fe5b36aa9e9fa60e80007a18c05574c102fd43613c535
KEK: 46e515c2a3677ef693f93e83685176
KCK: 728fe5b36aa9e9fa60e80007a18c05
TK: 4c102fd43613c535404d0777088a6503
Receive MIC Secret: 7d9de4d644bb9a
Transmiit MIC Secret: de1a23f2e9f17b
dactual mic:D5AA6ADF088791D7CD37B8866F8A0930
calculated mic from Message2of4:D5AA6ADF088791D7CD37B8866F8A0930
MATCH
actual mic:8FC6CCF6133542E9F8844EC826DE5FF8
calculated mic from Message3of4:8FC6CCF6133542E9F8844EC826DE5FF8
MATCH
packet mic:E2367DB355CCDF0710D8D73D6E175B5C
packet mic from Message4of4:E2367DB355CCDF0710D8D73D6E175B5C
MATCH
PS C:\Users\TakeshitaMegumi\OneDrive - いけりりネットワークサービス株式会社\Sharkfest
```

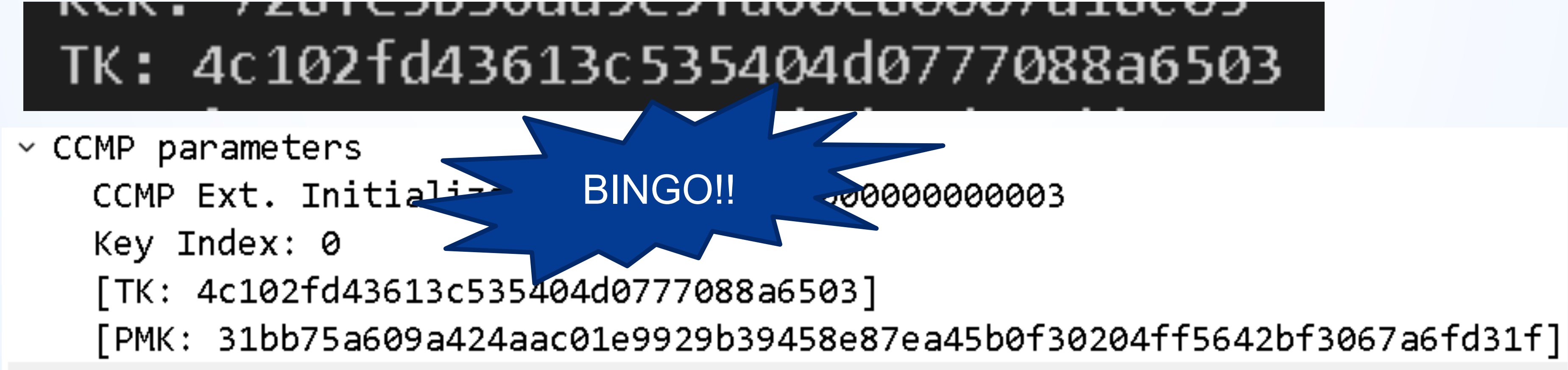
Check the MIC is correct


```
Passphrase:wireshark
SSID:ikeriri6
AP MAC Address from 1of4:f02f74c4f5c4
AP Nonce from 1of4:812e47f04e25fe494c7d44b2f7b016e0ebe3f24865fd234f4998a8f5d8d68bc0
STA MAC Address from 1of4:e2da1ea8928f
STA Nonce from 1of4:fcf94398b971a1f20572495509733ff0008c93b142e86c9348ce23f3c287ff8b
PMK: 31bb75a609a424aac01e9929b39458e87ea45b0f30204ff5642bf3067a6fd31f
PTK: 46e515c2a3677ef693f93e8368517684728fe5b36aa9e9fa60e80007a18c05574c102fd43613c535404d0777088a65037d9de4d644bb9a67de1a23f2e9f17b8e
KEK: 46e515c2a3677ef693f93e83685176
KCK: 728fe5b36aa9e9fa60e80007a18c05
TK: 4c102fd43613c535404d0777088a6503
Receive MIC Secret: 7d9de4d644bb9a
Transmiit MIC Secret: de1a23f2e9f17b
dactual mic:D5AA6ADF088791D7CD37B8866F8A0930
calculated mic from Message2of4:D5AA6ADF088791D7CD37B8866F8A0930
MATCH
actual mic:8FC6CCF6133542E9F8844EC826DE5FF8
calculated mic from Message3of4:8FC6CCF6133542E9F8844EC826DE5FF8
MATCH
packet mic:E2367DB355CCDF0710D8D73D6E175B5C
packet mic from Message4of4:E2367DB355CCDF0710D8D73D6E175B5C
MATCH
PS C:\Users\TakeshitaMegumi\OneDrive - いけりりネットワークサービス株式会社\Sharkfest\Sharkfest2022\03DissectingWPA3>
```

- 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)



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

4	0.005013	-33	d…	ASUSTekC_c4:f5:c4	e2:da:1e:a8:92:8f	Association Response
5	0.008401	-33	d…	ASUSTekC_c4:f5:c4	e2:da:1e:a8:92:8f	QoS Data
6	0.010452	-41	d…	e2:da:1e:a8:92:8f	ASUSTekC_c4:f5:c4	QoS Data
7	0.011950	-33	d…	ASUSTekC_c4:f5:c4	e2:da:1e:a8:92:8f	QoS Data
8	0.013064	-41	d…	e2:da:1e:a8:92:8f	ASUSTekC_c4:f5:c4	QoS Data
9	0.072760	-41	d…	::	ff02::1:ff01:27fa	QoS Data
10	0.072892	-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 unknowr
> Radiotap Header v0, Length 32
> 802.11 radio information
> IEEE 802.11 QoS Data, Flags: ......F.
> Logical-Link Control
v 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: 812e47f04e25fe494c7d44b2f7b016e0ebe3f24865fd234f4998a8f5d8d68bc0
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: 8fc6ccf6133542e9f8844ec826de5ff8
    WPA Key Data Length: 88
  > WPA Key Data: 8a6607c41f921c7035108bc8a60c1fa1b5a5ae7d0e191e1ce71fa11d8a1a65b302c4c345…
```

AP creates Nonce (Gnonce), random.

AP crates 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

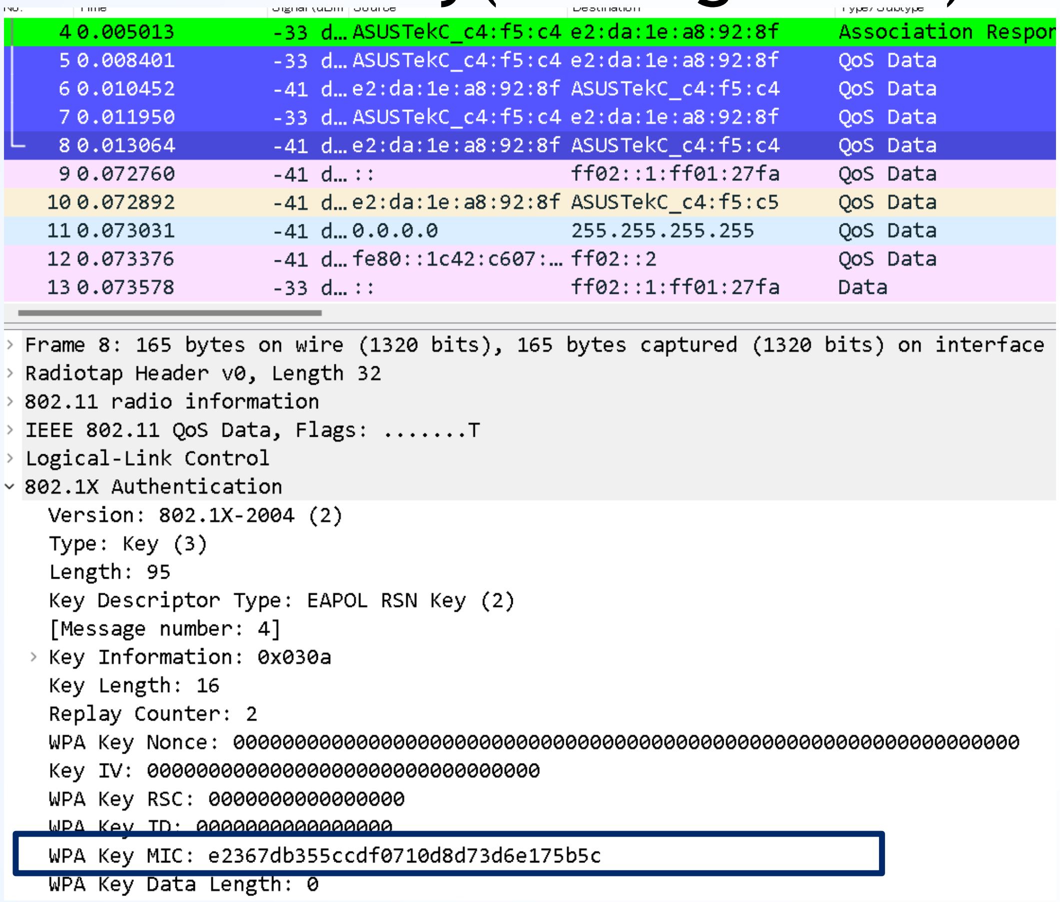
- KCK(Key Confirmation Key) and KEK(Key Encryption Key) are used for secure key distribution.
- Confirm Wireshark calculated KCK, KEK is the same with ptk.py generated KCK, KEK

```
      GTK: eeda95a1155a28f86c030d000c1fdd9a
   > Tag: Vendor Specific: Ieee 802.11: RSN IGTK
     WPA Key Data Padding: dd000000
     [KCK: 46e515c2a3677ef693f93e8368517684]
     [KEK: 728fe5b36aa9e9fa60e80007a18c0557]
```

```
KEK: 46e515c2a3677ef693f93e8368517684
KCK: 728fe5b36aa9e9fa60e80007a18c0557
```

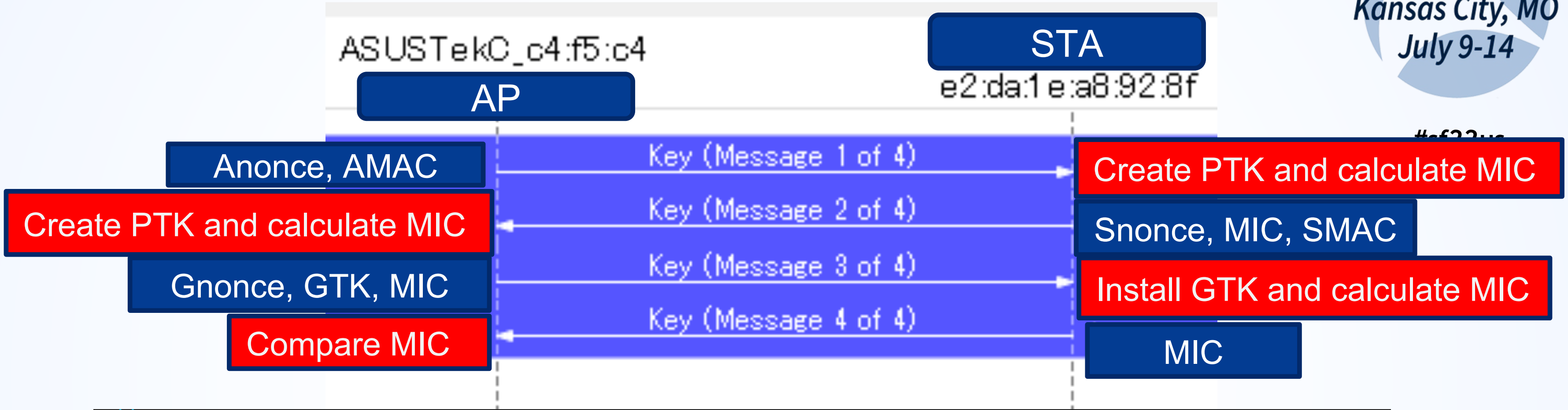
- It means GTK sends/receives securely

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



- 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




```
Passphrase:wireshark
SSID:ikeriri6
AP MAC Address from 1of4:f02f74c4f5c4
AP Nonce from 1of4:812e47f04e25fe494c7d44b2f7b016e0ebe3f24865fd234f4998a8f5d8d68bc0
STA MAC Address from 1of4:e2da1ea8928f
STA Nonce from 1of4:fcf94398b971a1f20572495509733ff0008c93b142e86c9348ce23f3c287ff8b
PMK: 31bb75a609a424aac01e9929b39458e87ea45b0f30204ff5642bf3067a6fd31f
PTK: 46e515c2a3677ef693f93e8368517684728fe5b36aa9e9fa60e80007a18c05574c102fd43613c535404d0777088a65037d9de4d644bb9a67de1a23f2e9f17b8e
KEK: 46e515c2a3677ef693f93e8368517684
KCK: 728fe5b36aa9e9fa60e80007a18c0557
TK: 4c102fd43613c535404d0777088a6503
Receive MIC Secret: 7d9de4d644bb9a
Transmit MIC Secret: de1a23f2e9f17b
dactual mic:D5AA6ADF088791D7CD37B8866F8A0930
calculated mic from Message2of4:D5AA6ADF088791D7CD37B8866F8A0930
MATCH
actual mic:8FC6CCF6133542E9F8844EC826DE5FF8
calculated mic from Message3of4:8FC6CCF6133542E9F8844EC826DE5FF8
MATCH
packet mic:E2367DB355CCDF0710D8D73D6E175B5C
packet mic from Message4of4:E2367DB355CCDF0710D8D73D6E175B5C
MATCH
```

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

wlan.fc.type_subtype==12 or eapol + ビーコン アソシエーション応答 WPA鍵交換 PlanexAP CiscoAP NetGearAP RTSかCTS DeauthかDisassoc

No.	Time	Signal (dBm)	Source	Destination	Type/Subtype	Data	Protocol	Length	Info
442…	358.904…	-58 dBm	PlanexCo_e3:c2:79	Sony_50:73:db	Deauthe…	1	802.11	46	Deauthentication, SN=219
442…	358.912…	-58 dBm	PlanexCo_e3:c2:79	Sony_50:73:db	Deauthe…	1	802.11	50	Deauthentication, SN=219
442…	358.914…	-57 dBm	PlanexCo_e3:c2:79	Sony_50:73:db	Deauthe…	1	802.11	50	Deauthentication, SN=219
442…	358.921…	-58 dBm	PlanexCo_e3:c2:79	Sony_50:73:db	Deauthe…	1	802.11	50	Deauthentication, SN=219
443…	359.050…	-60 dBm	PlanexCo_e3:c2:79	Sony_50:73:db	Deauthe…	1	802.11	50	Deauthentication, SN=219
443…	359.109…	-62 dBm	PlanexCo_e3:c2:79	Sony_50:73:db	Deauthe…	1	802.11	50	Deauthentication, SN=219
443…	359.113…	-62 dBm	PlanexCo_e3:c2:79	Sony_50:73:db	Deauthe…	1	802.11	50	Deauthentication, SN=219
443…	359.120…	-62 dBm	PlanexCo_e3:c2:79	Sony_50:73:db	Deauthe…	1	802.11	50	Deauthentication, SN=219
443…	359.124…	-62 dBm	PlanexCo_e3:c2:79	Sony_50:73:db	Deauthe…	1	802.11	50	Deauthentication, SN=219
443…	359.149…	-63 dBm	PlanexCo_e3:c2:79	Sony_50:73:db	Deauthe…	1	802.11	50	Deauthentication, SN=219
443…	359.163…	-62 dBm	PlanexCo_e3:c2:79	Sony_50:73:db	Deauthe…	1	802.11	50	Deauthentication, SN=219
443…	359.174…	-61 dBm	PlanexCo_e3:c2:79	Sony_50:73:db	Deauthe…	1	802.11	50	Deauthentication, SN=219
443…	359.177…	-60 dBm	PlanexCo_e3:c2:79	Sony_50:73:db	Deauthe…	1	802.11	50	Deauthentication, SN=219
252…	262.058…	-58 dBm	PlanexCo_e3:c2:79	Sony_53:d6:0a	Deauthe…	1	802.11	50	Deauthentication, SN=217
252…	262.374…	-60 dBm	PlanexCo_e3:c2:79	Sony_53:d6:0a	Data	1	EAPOL	177	Key (Message 1 of 4)
252…	262.450…	-60 dBm	PlanexCo_e3:c2:79	Sony_53:d6:0a	Data	1	EAPOL	211	Key (Message 3 of 4)
313…	293.295…	-63 dBm	PlanexCo_e3:c2:79	Sony_53:d6:0a	Data	1	EAPOL	177	Key (Message 1 of 4)
313…	293.299…	-63 dBm	PlanexCo_e3:c2:79	Sony_53:d6:0a	Data	1	EAPOL	177	Key (Message 1 of 4)
313…	293.571…	-64 dBm	PlanexCo_e3:c2:79	Sony_53:d6:0a	Data	1	EAPOL	177	Key (Message 1 of 4)
313…	293.580…	-64 dBm	PlanexCo_e3:c2:79	Sony_53:d6:0a	Data	1	EAPOL	177	Key (Message 1 of 4)
313…	293.587…	-63 dBm	PlanexCo_e3:c2:79	Sony_53:d6:0a	Data	1	EAPOL	177	Key (Message 1 of 4)
313…	293.592…	-64 dBm	PlanexCo_e3:c2:79	Sony_53:d6:0a	Data	1	EAPOL	177	Key (Message 1 of 4)
314…	293.644…	-62 dBm	PlanexCo_e3:c2:79	Sony_53:d6:0a	Data	1	EAPOL	89	Key[Malformed Packet]
314…	293.685…	-61 dBm	PlanexCo_e3:c2:79	Sony_53:d6:0a	Data	1	EAPOL	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
- WPA3 personal mode (WPA3-SAE) uses SAE(Simultaneous Authentication of Equals), 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	Signal (dBm)	Source	Destination	Type/Subtype	Data	Protocol	Length	Info
1	0.000000	-39 dBm	e2:da:1e:a8:92:8f	ASUSTekC_c4:f5:c4	Authentication	6	802.11	97	Authentication, SN=833, FN=0, Flags=........
2	0.001283	-33 dBm	ASUSTekC_c4:f5:c4	e2:da:1e:a8:92:8f	Authentication	6	802.11	62	Authentication, SN=0, FN=0, Flags=........
3	0.002746	-41 dBm	e2:da:1e:a8:92:8f	ASUSTekC_c4:f5:c4	Association Request	6	802.11	237	Association Request, SN=834, FN=0, Flags=........,
4	0.005013	-33 dBm	ASUSTekC_c4:f5:c4	e2:da:1e:a8:92:8f	Association Response	6	802.11	318	Association Response, SN=1, FN=0, Flags=........
5	0.008401	-33 dBm	ASUSTekC_c4:f5:c4	e2:da:1e:a8:92:8f	QoS Data	6	EAPOL	165	Key (Message 1 of 4)
6	0.010452	-41 dBm	e2:da:1e:a8:92:8f	ASUSTekC_c4:f5:c4	QoS Data	6	EAPOL	187	Key (Message 2 of 4)
7	0.011950	-33 dBm	ASUSTekC_c4:f5:c4	e2:da:1e:a8:92:8f	QoS Data	6	EAPOL	253	Key (Message 3 of 4)
8	0.013064	-41 dBm	e2:da:1e:a8:92:8f	ASUSTekC_c4:f5:c4	QoS Data	6	EAPOL	165	Key (Message 4 of 4)
9	0.072760	-41 dBm	::	ff02::1:ff01:27fa	QoS Data		ICMPv6	170	Neighbor Solicitation for fe80::1c42:c607:6801:27fa
10	0.072892	-41 dBm	e2:da:1e:a8:92:8f	ASUSTekC_c4:f5:c5	QoS Data		ARP	126	Who has 192.168.50.1? Tell 192.168.50.236
11	0.073031	-41 dBm	0.0.0.0	255.255.255.255	QoS Data		DHCP	426	DHCP Request - Transaction ID 0xac9e7500
12	0.073376	-41 dBm	fe80::1c42:c607…	ff02::2	QoS Data		ICMPv6	146	Router Solicitation
13	0.073578	-33 dBm	::	ff02::1:ff01:27fa	Data	6	ICMPv6	136	Neighbor Solicitation for fe80::1c42:c607:6801:27fa

No.	Time	Signal (dBm)	Source	Destination	Type/Subtype	Data	Protocol	Length	Info
1	0.000000	-36 dBm	e2:da:1e:a8:92:8f	ASUSTekC_c4:f5:c4	Authentication	6	802.11	160	Authentication, SN=3841, FN=0, Flags=.....
2	0.028660	-31 dBm	ASUSTekC_c4:f5:c4	e2:da:1e:a8:92:8f	Authentication	6	802.11	160	Authentication, SN=0, FN=0, Flags=........
3	0.058370	-38 dBm	e2:da:1e:a8:92:8f	ASUSTekC_c4:f5:c4	Authentication	6	802.11	96	Authentication, SN=3842, FN=0, Flags=.....
4	0.059667	-31 dBm	ASUSTekC_c4:f5:c4	e2:da:1e:a8:92:8f	Authentication	6	802.11	96	Authentication, SN=1, FN=0, Flags=........
5	0.062471	-38 dBm	e2:da:1e:a8:92:8f	ASUSTekC_c4:f5:c4	Association Request	6	802.11	255	Association Request, SN=3843, FN=0, Flags=
6	0.064576	-31 dBm	ASUSTekC_c4:f5:c4	e2:da:1e:a8:92:8f	Association Response	6	802.11	292	Association Response, SN=2, FN=0, Flags=..
7	0.067697	-31 dBm	ASUSTekC_c4:f5:c4	e2:da:1e:a8:92:8f	QoS Data	6	EAPOL	187	Key (Message 1 of 4)
8	0.070141	-38 dBm	e2:da:1e:a8:92:8f	ASUSTekC_c4:f5:c4	QoS Data	6	EAPOL	205	Key (Message 2 of 4)
9	0.071860	-31 dBm	ASUSTekC_c4:f5:c4	e2:da:1e:a8:92:8f	QoS Data	6	EAPOL	253	Key (Message 3 of 4)
10	0.073031	-38 dBm	e2:da:1e:a8:92:8f	ASUSTekC_c4:f5:c4	QoS Data	6	EAPOL	165	Key (Message 4 of 4)
11	0.075096	-40 dBm	e2:da:1e:a8:92:8f	ASUSTekC_c4:f5:c4	QoS Null function (…		802.11	74	QoS Null function (No data), SN=3844, FN=0
12	0.421081	-40 dBm	e2:da:1e:a8:92:8f	IPv6mcast_ff:01…	QoS Data		802.11	170	QoS Data, SN=0, FN=0, Flags=op.....T
13	0.421085	-40 dBm	e2:da:1e:a8:92:8f	Broadcast	QoS Data		802.11	426	QoS Data, SN=1, FN=0, Flags=op.....T

WPA2

WPA3

Difference between WPA2 and WPA3

SharkFest'22 US
Kansas City, MO

Explanation	WPA2 Personal WPA2-PSK	WPA3 Personal 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 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)

https://www.wi-fi.org/download.php?file=/sites/default/files/private/WPA3_Specification_v3.0.pdf

- 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 byte
> 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 byte
> 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: f4b7c34e9f0d5444381e1dde353e54dcc838435b372a3933b7cc
```

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 4
3*5=15≡4 (mod 11). We use this for Elliptic-Curve Discrete Logarithm Problem (seems difficult...)

Understand Dragonfly key exchange with simple example

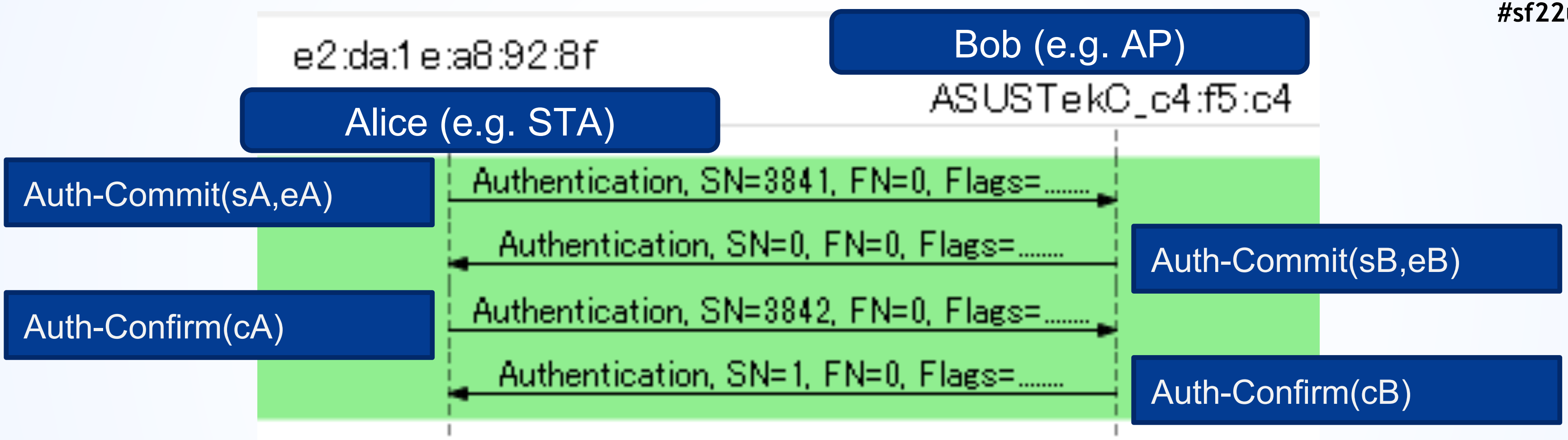
- We can translate Elliptic-Curve Discrete Logarithm Problem in a computer program like below,
  int a=2,n=5,p=11, b=a^n mod p (^:exponentiation)
  we can calculate b easily from a,n,p
  b=2*2*2*2*2 mod 11=32 mod 11 =10
- So how do we find n (Logarithm) from a,b and p,
  we need to test incrementally, 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 mod 11=32 mod 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

- 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: f4b7c34e9f0d5444381e1dde353e54dcc838435b372a3933b7cc
```

```
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: e05e00747ffce2d04a55d7d7d32296c5b8ffa07e5777d2dfa3f7a8e74fce2343
```

```
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: 6dc3f845c7772c6fa7ec01b95802b850ceb840e9dd13019c6515c3311c05cc4f
```

- SAE handshake has 2 Auth-Commit and 2 Auth-Confirm message
- Auth-Commit has Scalar(sA,sB) and Finite Field Element (eA,eB)
- 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
```

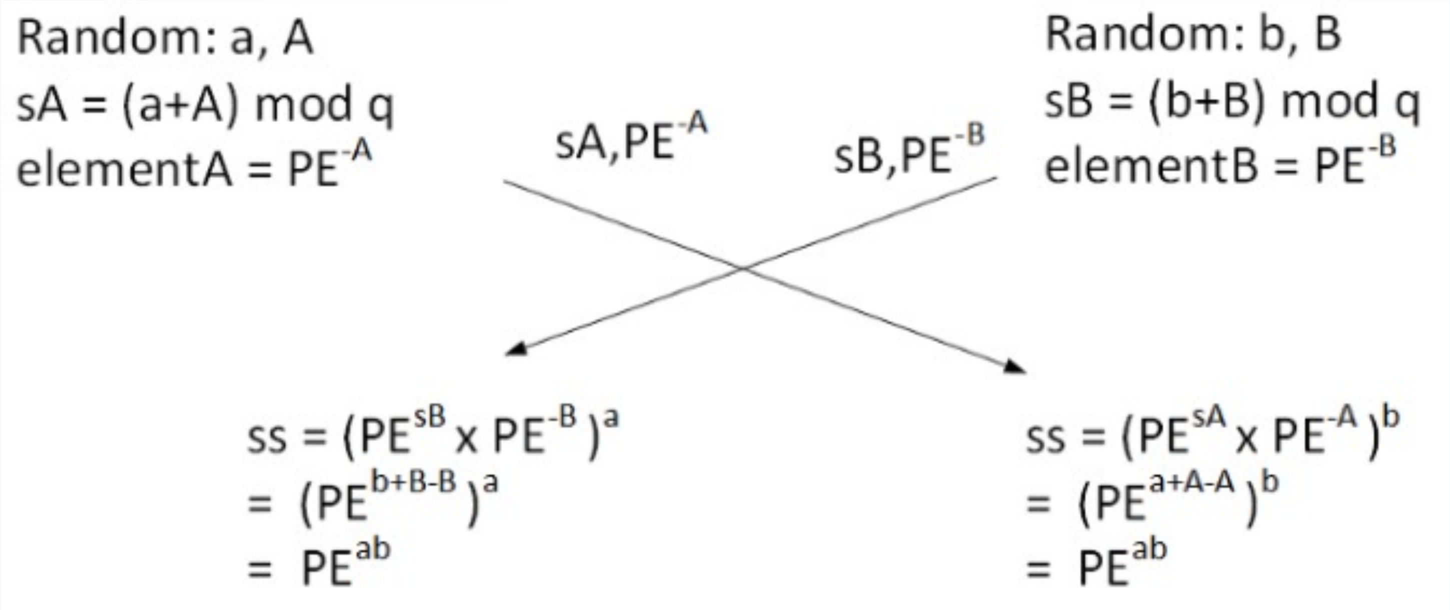
- Alice(STA) picks random rA and mA and calculates
  sA=(rA+mA) mod q
  eA=-mA・GF(p) (・means inner products of vector)
- Then send Auth-Commit with sA (256bits Scalar) and GF(p) (512bits Finite Field Element)

#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 rB and mB and calculates
  sB=(rB+mB) mod q
  eB=-mB・GF(p) (・means inner products of vector)
- Then send Auth-Commit with sB (256bits Scalar) and GF(p) (512bits Finite Field Element)

Auth-Commit (Scalar, Finite Field Element)



- 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?
- 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)
  base = H(max(Alice,Bob) | min(Alice,Bob) | password | counter)
- We use KDF(key derivation function) to create bitstream temp value(length is prime number) and the seed
  n = len(p) + 64
  temp = KDF-n（base、 "Dragonfly Hunting and Pecking"）
  seed = (temp mod (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


```
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)


```
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: e05e00747ffce2d04a55d7d7d32296c5b8ffa07e5777d2dfa3f7a8e74fce2343
```

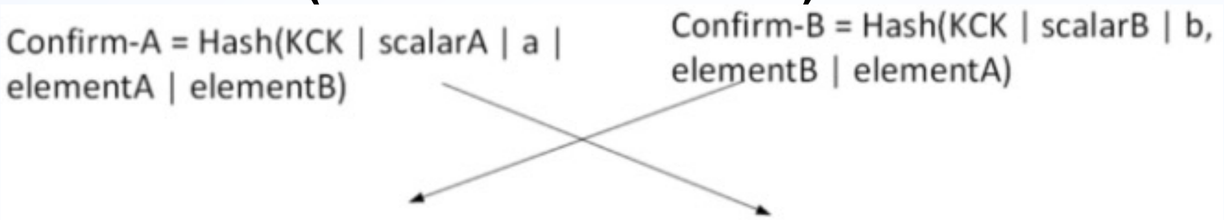
- Alice(STA) verifies sB and eB, calculates
  K=rA・(sB・P＋eB) (・means inner products)
  tr=(sA,eA,sB,eB) (Alice and Bob know these values)
  cA=HMAC(Hash(K),tr)
- Then send Auth-Confirm with cA (256bits Confirm)

#4:Auth-Confirm from Bob(AP)


```
⌄ 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: 6dc3f845c7772c6fa7ec01b95802b850ceb840e9dd13019c6515c3311c05cc4f
```

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

Auth-Confirm (Confirm value)



- Each Alice(STA) and Bob(AP) can verify the packet's Confirm value with the calculated Confirm value
- $K = rB \cdot (sA \cdot P + eA)$
  $tr = (sB, eB, sA, eA)$
  $cB = HMAC(Hash(K), tr)$
- $K = rA \cdot (sB \cdot P + eB)$
  $tr = (sA, eA, sB, eB)$
  $cA = HMAC(Hash(K), tr)$
- 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

- 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
- Open dragonfly_implementation.py in VSCode

dragonfly_implementation.py

dragonfly_implementation.py > Peer > commit_exchange

```python
#!/usr/bin/env python3

"""
Implements the Dragonfly (SAE) handshake.
Instead of using a client (STA) and a access point (AP), we
just programmatically create a peer to peer network of two participiants.
Either party may initiate the SAE protocol, either party can be the client and server.
In a mesh scenario, where two APs (two equals) are trying to establish a connection
between each other and each one could have the role of supplicant or authenticator.
SAE is build upon the Dragonfly Key Exchange, which is described in https://tools.ietf.org/html/rfc7664.
https://stackoverflow.com/questions/31074172/elliptic-curve-point-addition-over-a-finite-field-in-python
"""
import time
import hashlib
import random
import logging
from collections import namedtuple

logger = logging.getLogger('dragonfly')
logger.setLevel(logging.INFO)
# create file handler which logs even debug messages
fh = logging.FileHandler('dragonfly.log')
fh.setLevel(logging.DEBUG)
# create console handler with a higher log level
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG)
# create formatter and add it to the handlers
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
ch.setFormatter(formatter)
fh.setFormatter(formatter)
# add the handlers to logger
logger.addHandler(ch)
logger.addHandler(fh)


Point = namedtuple("Point", "x y")
# The point at infinity (origin for the group law).
O = 'Origin'

def lsb(x):
    binary = bin(x).lstrip('0b')
    return binary[0]

def legendre(a, p):
    return pow(a, (p - 1) // 2, p)

def tonelli_shanks(n, p):
    """
    # https://rosettacode.org/wiki/Tonelli-Shanks_algorithm#Python
    """
    assert legendre(n, p) == 1, "not a square (mod p)"
    q = p - 1
    s = 0
    while q % 2 == 0:
        q //= 2
        s += 1
    if s == 1:
        return pow(n, (p + 1) // 4, p)
    for z in range(2, p):
        if p - 1 == legendre(z, p):
            break
    c = pow(z, q, p)
    r = pow(n, (q + 1) // 2, p)
    t = pow(n, q, p)
    m = s
    t2 = 0
    while (t - 1) % p != 0:
        t2 = (t * t) % p
        for i in range(1, m):
            if (t2 - 1) % p == 0:
                break
            t2 = (t2 * t2) % p
        b = pow(c, 1 << (m - i - 1), p)
        r = (r * b) % p
        c = (b * b) % p
        t = (t * c) % p
        m = i
    return r
```

- 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.

Check Commit Values

```
2022-07-03 23:34:02,692 - dragonfly - INFO - [STA] Sending scalar and element to the Peer!
2022-07-03 23:34:02,692 - dragonfly - INFO - [STA] Scalar=37665713968707636108463552758153804253866020869123773870082307673943289574929
2022-07-03 23:34:02,692 - dragonfly - INFO - [STA] Element=Point(x=5370985677817642780906761615458726723608646202400217542479733492405994134 0937, y=24560607
031508637145355645308281569136437199158698003538515219094487652065590)
2022-07-03 23:34:02,749 - dragonfly - INFO - [AP] Sending scalar and element to the Peer!
2022-07-03 23:34:02,750 - dragonfly - INFO - [AP] Scalar=18769519039188788754471066505026034067427887351975226134301023022936721879221
2022-07-03 23:34:02,750 - dragonfly - INFO - [AP] Element=Point(x=28143476516887803504064428630728579495169134189770038374073020424238475455413, y=245807196
77194949606219360213182272371291201029985376767539415701335918085913)

2022-07-03 23:34:02,750 - dragonfly - INFO - Computing shared secret...

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=13e297cb9a9dcea5d33be06237fa2878dadf4d994287bf0f1de343bcbac51cfd
2022-07-03 23:34:02,968 - dragonfly - INFO - [STA] Received Token from Peer=13e297cb9a9dcea5d33be06237fa2878dadf4d994287bf0f1de343bcbac51cfd
2022-07-03 23:34:02,968 - dragonfly - INFO - [STA] Pairwise Master Key(PMK)=e0c89767cd0bf2f6193bb9439a7df0f7e37a853ff4e6e612b84b4967ca0849b9
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Computed Token from Peer=cf6f119f89b4b0e39b3450c0466baacacaa1824b8a0ab2dd1fb46ffb33bef4c8
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Received Token from Peer=cf6f119f89b4b0e39b3450c0466baacacaa1824b8a0ab2dd1fb46ffb33bef4c8
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Pairwise Master Key(PMK)=e0c89767cd0bf2f6193bb9439a7df0f7e37a853ff4e6e612b84b4967ca0849b9
PS C:\Users\TakeshitaMegumi\OneDrive - いけりりネットワークサービス株式会社\Sharkfest\Sharkfest2022\03DissectingWPA3>
```

- 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=13e297cb9a9dcea5d33be06237fa2878dadf4d994287bf0f1de343bcbac51cfd
2022-07-03 23:34:02,968 - dragonfly - INFO - [STA] Received Token from Peer=13e297cb9a9dcea5d33be06237fa2878dadf4d994287bf0f1de343bcbac51cfd
2022-07-03 23:34:02,968 - dragonfly - INFO - [STA] Pairwise Master Key(PMK)=e0c89767cd0bf2f6193bb9439a7df0f7e37a853ff4e6e612b84b4967ca0849b9
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Computed Token from Peer=cf6f119f89b4b0e39b3450c0466baacacaa1824b8a0ab2dd1fb46ffb33bef4c8
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Received Token from Peer=cf6f119f89b4b0e39b3450c0466baacacaa1824b8a0ab2dd1fb46ffb33bef4c8
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Pairwise Master Key(PMK)=e0c89767cd0bf2f6193bb9439a7df0f7e37a853ff4e6e612b84b4967ca0849b9
PS C:\Users\TakeshitaMegumi\OneDrive - いけりりネットワークサービス株式会社\Sharkfest\Sharkfest2022\03DissectingWPA3>
```

- 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 defferent!!

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

1st 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=13e297cb9a9dcea5d33be06237fa2878dadf4d994287bf0f1de343bcbac51cfd
2022-07-03 23:34:02,968 - dragonfly - INFO - [STA] Received Token from Peer=13e297cb9a9dcea5d33be06237fa2878dadf4d994287bf0f1de343bcbac51cfd
2022-07-03 23:34:02,968 - dragonfly - INFO - [STA] Pairwise Master Key(PMK)=e0c89767cd0bf2f6193bb9439a7df0f7e37a853ff4e6e612b84b4967ca0849b9
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Computed Token from Peer=cf6f119f89b4b0e39b3450c0466baacacaa1824b8a0ab2dd1fb46ffb33bef4c8
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Received Token from Peer=cf6f119f89b4b0e39b3450c0466baacacaa1824b8a0ab2dd1fb46ffb33bef4c8
2022-07-03 23:34:02,969 - dragonfly - INFO - [AP] Pairwise Master Key(PMK)=e0c89767cd0bf2f6193bb9439a7df0f7e37a853ff4e6e612b84b4967ca0849b9
PS C:\Users\TakeshitaMegumi\OneDrive - いけりりネットワークサービス株式会社\Sharkfest\Sharkfest2022\03DissectingWPA3>
```

2nd 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=ae98151668650840769ccabb0951d16ddbab736f4d6500272b0ec7c12d64ffbf
2022-07-03 23:51:33,666 - dragonfly - INFO - [STA] Received Token from Peer=ae98151668650840769ccabb0951d16ddbab736f4d6500272b0ec7c12d64ffbf
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=e6d936182c952456c74cc26b05636965c53fd0f1d71ed4dfc9a2759f70fe31d7
2022-07-03 23:51:33,667 - dragonfly - INFO - [AP] Received Token from Peer=e6d936182c952456c74cc26b05636965c53fd0f1d71ed4dfc9a2759f70fe31d7
2022-07-03 23:51:33,667 - dragonfly - INFO - [AP] Pairwise Master Key(PMK)=d29fb108095ad3ec6b0cf4da267eaad8605c6b45ef7a5dbf5d5632689e4190f0
PS C:\Users\TakeshitaMegumi\OneDrive - いけりりネットワークサービス株式会社\Sharkfest\Sharkfest2022\03DissectingWPA3>
```

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

- 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

#sf22us

Thank you for watching !!

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