

Cryptography Lab 1

Name: Albert Ferguson **SID:** 13611165

Part 1. AES

1.1 AES Encryption

The following uses <https://www.cryptool.org/en/cto/aes-step-by-step>.

First off, generate a 32 hex-key, 9dfc79e4 89684c8c 909df0ac ed2de066. Then some simple input, 1234abcd 1234abcd 1234abcd 1234abcd. The resultant 1st round appears as,

Permutation

Chaining:NoneCBCECB

Initial Vector (CBC only)

Key

9dfc79e4 89684c8c 909df0ac ed2de066

Expanded Key

Input

1234abcd 1234abcd 1234abcd 1234abcd

Encoding Rounds

Round 1

input to Round 1

8fc8d229 9b5ce741 82a95b61 ff194bab

after S-Box:ON

73e8b5a5 144a9483 13d339ef 16d4b362

after permutation:ON

734a3962 14d3b3a5 13d4b583 16e894ef

after mult:ON

63ceed22 50c24e0d 77e72849 7495e783

used subkey:

441d4ab1 cd75063d 5de8f691 b0c516f7

after mix with key:ON

27d3a793 9db74830 2a0fded8 c450f174

The difference between the first and last round, **r1** and **rn**, is that the 10th round skips the column mixing step. This is visible as the "*after mult*" step in the below screenshot,

The column mixing step applies a matrix multiplication to the previous permutation step. Hence, the naming.

Round 4

Round 5

Round 6

Round 7

Round 8

Round 9

Round 10

input to Round 10

e88945f8 ca8a4635 5e92e70a 50ccba56

after S-Box:

ON

9ba76e41 747e5a96 584f9467 534bf4b1

after permutation:

ON

9b7e94b1 744ff441 584b6e96 53a75a67

used subkey:

a96d6dd4 ee6eb0c5 e8a899f8 31a08603

after mix with key:

ON

3213f965 9a214484 b0e3f76e 6207dc64

Encoded

3213f965 9a214484 b0e3f76e 6207dc64

Decoding Rounds

Round 10

The resultant encoded output is, 3213f9659a214484b0e3f76e6207dc64. The decoding uses the same atomic steps as the encode but in a different order. Where encode would take input > S-box > permutate > multiply > apply subkey. Then decode would take input > permutate > S-box > apply subkey > multiply. Likewise with encoding, the nth round does not execute the multiply step.

Decoding Rounds

Round 10

input to Round 10

9b7e94b1 744ff441 584b6e96 53a75a67

after permutation:

ON

9ba76e41 747e5a96 584f9467 534bf4b1

after S-Box:

ON

e88945f8 ca8a4635 5e92e70a 50ccba56

used subkey:

afad62e1 4703dd11 06c6293d d9081ffb

after mix with key:

ON

47242719 8d899b24 5854ce37 89c4a5ad

after mult:

ON

995543d2 c93bb2fb 32f05562 f8e25e01

1.2 AES Encryption (Visualisation)

The following uses <https://www.cryptool.org/en/cto/aes-animation>.

As in #1, the same 32 hex-key is used,, 9dfc79e4 89684c8c 909df0ac ed2de066. This time, using a valid UTF-8 plaintext string, "How long is a piece of string?". The resultant config and ciphertext is then,

and the key. For an in-depth look at the mathematics behind the algorithm please consult the according literature.

AES Animation Data

The values in the animation change when updating the data below. Try it out!

Enter message in ASCII ☒ or in hex ☐

Plaintext (input in ASCII)

How long is a piece of string?

Key (input in hex)

9dfc79e489684c8c909df0aced2de066

Plaintext (in hex)

486F77206C6F6E672069732061207069 656365206F6620737472696E673F

Ciphertext (output in hex)

264318BE7C2C01C7E49309B1A1B7CFAD 1B677DC7086C487FF0E7E29DF9A05226

Encrypt

Print page

Share link

The animation pauses on the 12th slide to show the various outputs of each round-step,

3 / 7

	Start of round				After SubBytes				After ShiftRows				After MixColumns				Round key			
Round 6	29	3A	FA	26	A5	80	2D	F7	A5	80	2D	F7	D8	F0	39	C4	04	43	7E	37
	65	61	BB	80	4D	EF	EA	CD	EF	EA	CD	4D	2E	40	37	F9	2A	8B	6D	60
	46	5B	63	53	5A	39	FB	ED	FB	ED	5A	39	4F	CE	CB	B2	54	38	22	89
	32	3F	EF	5E	23	75	DF	58	58	23	75	DF	50	DA	0A	D3	EE	FB	53	36
Round 7	DC	B3	47	F3	86	6D	A0	0D	86	6D	A0	0D	F1	4F	20	9E	94	D7	A9	9E
	04	CB	5A	99	F2	1F	BE	EE	1F	BE	EE	F2	43	99	70	FF	8D	06	6B	0B
	1B	F6	E9	3B	AF	42	1E	E2	1E	E2	AF	42	D5	E5	17	3D	51	69	4B	C2
	BE	21	59	E5	AE	FD	CB	D9	D9	AE	FD	CB	39	AC	5B	2A	74	8F	DC	EA
Round 8	65	98	89	00	4D	46	A7	63	4D	46	A7	63	1C	93	F6	33	3F	E8	41	DF
	CE	9F	1B	F4	8B	DB	AF	BF	DB	AF	BF	8B	84	DA	05	D5	A8	AE	C5	CE
	84	8C	5C	FF	5F	64	4A	16	4A	16	5F	64	D7	FB	CC	19	D6	BF	F4	36
	4D	23	87	C0	E3	26	17	BA	BA	E3	26	17	29	AE	5E	64	7F	F0	2C	C6
Round 9	23	7B	B7	EC	26	21	A9	CE	26	21	A9	CE	DC	33	87	4F	AF	47	06	D9
	2C	74	C0	1B	71	92	BA	AF	92	BA	AF	71	2A	C0	30	41	AD	03	C6	08
	01	44	38	2F	7C	1B	07	15	07	15	7C	1B	F4	79	16	49	62	DD	29	1F
	56	5E	72	A2	B1	58	40	3A	3A	B1	58	40	8B	B5	83	A3	E1	11	3D	FB
Round 10	73	74	81	96	8F	92	0C	90	8F	92	0C	90					A9	EE	E8	31
	87	C3	F6	49	17	2E	42	3B	2E	42	3B	17					6D	6E	A8	A0
	96	A4	3F	56	90	49	75	B1	75	B1	90	49					6D	B0	99	86
	6A	A4	BE	58	02	49	AE	6A	6A	02	49	AE					D4	C5	F8	03
Output	26	7C	E4	A1																
	43	2C	93	B7																
	18	01	09	CF																
	BE	C7	B1	AD																
Ciphertext																				

⏮

⏪

⏩

⏭

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

⚙

Example one only allows manual hex inputs and limits the input to the length of the key. I am no bothering with typing it out manually for comparison (tempting though). However, if I did then I would expect each round tab to contain matching 32 hex values with the above screenshot. Where each row from the screenshot is a segment in the round table.

Part 2. Wi-Fi Cracking

2.1 Setup

Installing the dependencies is as simple as,

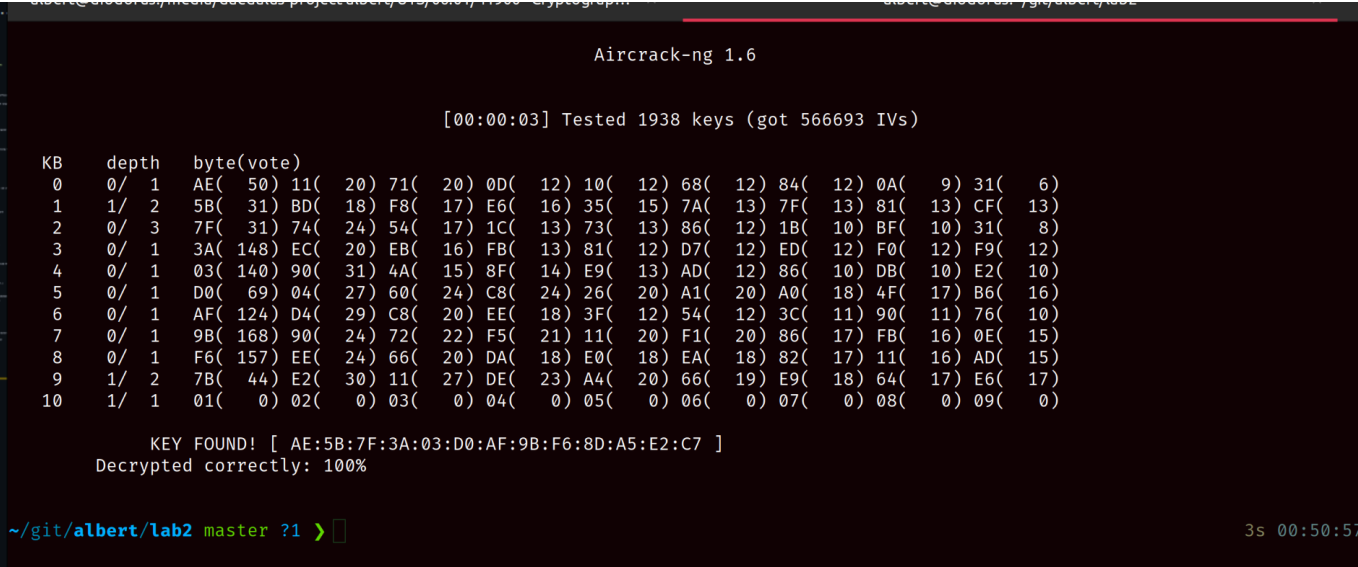
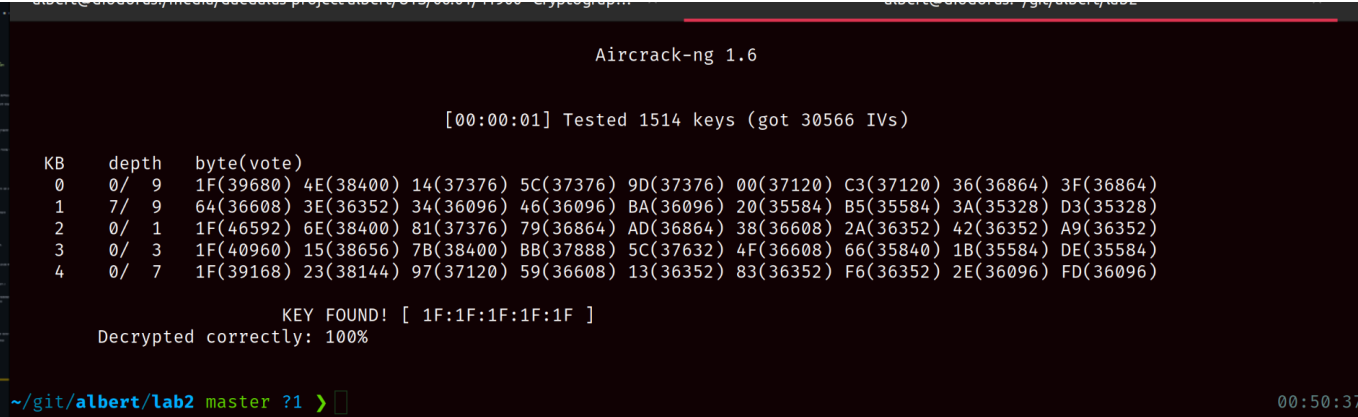
```
sudo apt-get install aircrack-ng
```

```
~/git/albert master ?1 > sudo apt-get install aircrack-ng
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
aircrack-ng is already the newest version (1:1.6+git20210130.91820bc-2).
0 to upgrade, 0 to newly install, 0 to remove and 8 not to upgrade.
~/git/albert master ?1 > man aircrack-ng
~/git/albert master ?1 > whatis aircrack-ng
aircrack-ng (1) - a 802.11 WEP / WPA-PSK key cracker
~/git/albert master ?1 >
```

2.2 Cracking pre-prepared WEP network dumps

Running the following I can crack the provided dumps using PTW and Korek modes,

```
aircrack-ng wep_64_ptw.cap
# then later
aircrack-ng -K wep_KoreK.ivs
```



2.3 Cracking pre-prepared WPA2 network dumps

Using the pre-prepared WPA2 handshake capture, I ran the following to execute a dictionary attack,

```
aircrack-ng -w password.lst wpa2.cap
```

The result was instant, and resolved the expected password (12345678),

```

Aircrack-ng 1.6

[00:00:00] 21/2294 keys tested (666.67 k/s)

Time left: 3 seconds                                0.92%

KEY FOUND! [ 12345678 ]

Master Key      : EE 51 88 37 93 A6 F6 8E 96 15 FE 73 C8 0A 3A A6
                  F2 DD 0E A5 37 BC E6 27 B9 29 18 3C C6 E5 79 25

Transient Key   : EA 0E 40 46 33 C8 02 45 03 02 86 8C CA A7 49 DE
                  5C BA 5A BC B2 67 E2 DE 1D 5E 21 E5 7A CC D5 07
                  9B 31 E9 FF 22 0E 13 2A E4 F6 ED 9E F1 AC C8 85
                  45 82 5F C3 2E E5 59 61 39 5A E4 37 34 D6 C1 07

EAPOL HMAC     : D5 35 53 82 B8 A9 B8 06 DC AF 99 CD AF 56 4E B6

~/git/albert/lab2 master ?1 >

```

Part 3. WPA2/3 Questions

Q1: What is the vulnerability of WPA2 Personal?

There are several vulnerabilities, primarily the potential for brute-force attacks. Secondary is the "castle" defence, as a single bad-actor can snoop other clients' traffic once the network is breached.

Q2: How does WPA3 solve WPA2 shortcomings?

In two ways primarily,

1. PSK is dropped in favour of Simultaneous Authentication Exchange (SAE) forces realtime-handshakes and creates forward secrecy
2. personalized encryption between clients, avoids bad-actors snooping other clients' traffic.

Q3: Is there any possible attacks against WPA3?

Yes, although either non-trivial or novel. The "Dragonblood" analysis provide details various vulnerabilities. These revolve around the SAE "dragonfly" handshake and often rely on misimplementations of the underlying protocol. These include,

- various side-channel attacks (timing and cache based),
- DOS attacks during the SAE handshake
- and downgrade attacks during the handshake

These are revised as recently as [2020 based on this source](#).

Part 3. Further Questions

Q1: How to capture the four-way handshake?

As described in the related reading. By using Wireshark in monitor mode, the packets can be passively scanned once connected to the target AP. They can be filtered from by searching for the EAPOL protocol.

Q2: What is the vulnerability of WEP used in this cracking?

WEP utilises an IV prefix to create "randomness". This is available in plaintext and easily gathered in significant quantities (40 - 85K packets). This lab provided the packet captures for both the PTW and FMS/Koreks methods. By XOR'ng packets with the same IV prefix, the original packet can be recovered and then the key is easily deduced.

Q3: What is the vulnerability of WPA2 used in this cracking?

WPA2 utilises a 4-way handshake during it's setup. This can be captured passively and attacked offline to apply a brute-force dictionary attack. This lab provided the handshake capture and an example (password) dictionary to demonstrate.

Q4: What have you learnt in this lab?

I learnt about the WEP, WPA2, and WPA3 specifications in depth leading up to this lab. This lab encouraged me to read further into the various vulnerabilities and explore the KRACKs, Dragonblood, and PTW papers in-depth. Overall, I have learnt both that wireless security is still a "new" domain with several exploits, and that I should immediately upgrade my personal network's security.

Summary

In summary, this lab had me revise my notes on AES and the various Wi-Fi standards. The initial AES part was interesting, as it provided a visual example for the algorithm. Ironically, I think this is the first and most concise example of the AES algorithm from all the examples in this subject so far. Although, I personally found the original Rijndael paper far more useful. The remainder of the lab was a quick practical on the usage of aircrack to demonstrate the vulnerabilities of WEP and WPA2. This was an excellent intro to the tool, and thankfully skipped the packet sniffing setup step that so many other labs would've encouraged.