COMP3210 Report
Decryption of WPA2-PSK Packets

Dan Harris
Electronics and Computer Science
Faculty of Engineering and Physical Sciences
University of Southampton
Email: dmh2g16@soton.ac.uk

***Abstract*—WPA2 is the de-facto standard for encryption on WiFi connections today. The majority of home based networks make use of WPA2-PSK, whereby a single pre shared key is used on all devices in the network, which relies on all devices being trustworthy. This report investigates how a compromised WiFi network preshared key can be used to decrypt and analyse packets sent over the network. The research shows that through deliberate de-authentication of devices and monitoring of device handshakes, arbitrary packets can be decrypted with minimal complexity. In addition it does not require the malicious device to be active on the network.**

I. Introduction

One of the key requirements of any WiFi enabled network is that it is secure and impenetrable to eavesdropping, due to the use of a shared medium whereby signals are broadcast. This is opposed to a traditional wired network where communications are direct. There have been multiple implementations of this, where each progression aims to make this more difficult, along with increasing levels of encryption. Currently, the standard deployed on the majority of devices and access points is WiFi Protected Access (WPA) 2. This makes the use of the Advanced Encryption Standard (AES) along with Counter Cipher Mode with Block Chaining Message Authentication Code Protocol (CCMP) to verify the integrity and authenticity of messages sent over a shared medium.

The most widespread implementation uses a pre shared key (PSK): a single password that is used on all clients wishing to connect to the network. A 4-way handshake takes place to then produce keys unique for that connection, to prevent eavesdropping. This handshake makes the assumption that the preshared key is kept secret as it is used for the basis of the key derivation.

The goal of this project is to understand how WPA2-PSK authentication works, such that it would be possible for a malicious device to decrypt packets sent over the air (when the PSK is known), even when that device is not itself connected to the network. This derivation and decryption process will be implemented, with the resulting data being analysed and published to a client application.

II. Research

The main source of literature used to understand the authentication process was the official IEEE 802.11i standard [1], which contains details of the algorithms and standards used. This has been split into two distinct sections, which will now be discussed.

A. Key Derivation

There are two keys used in WPA2 encryption: The temporal key (TK) which is used for unicast based traffic, and the group temporal key (GTK) which is used for broadcast traffic. This report focusses on unicast traffic, as this is likely to contain the most information of interest. However, both require the use of a pseudorandom function (PRF). This function takes four arguments:

1) A key K
2) A purpose A, to be given as a string literal
3) Data B
4) Output length in bits

The output is then determined by the following algorithm (Listing 1), where `H-SHA-1` is already defined.

```
PRF(K, A, B, Len) {
    R=""
    for i = 0 to (Len+159)/60 do {
        R += H-SHA-1(K, A, B, i)
    }
    return substring(R, 0, Len)
}
```

Listing 1. PRF Pseudocode

To derive the temporal key, the data to the PRF consists of the hardware address of the access point (AA) and that of the supplicant (SPA). In addition, the first two messages of the handshake transmit a nonce from each device (ANonce in message 1 and SNonce in message 2). These values are all concatenated together to produce a single value. This is outlined in Table I. If AES CCMP encryption is to be used, the TK is defined as bytes 128-384 from the output of the function.

From this it is evident that if the 4-way handshake could be intercepted and monitored, it would be possible to derive the temporal key used for that connection. It is important to note that this key will be unique for every connection, as the nonce will change.

Parameter	Value
K	Preshared Master Key (derived from the SSID and passphrase)
A	"Pairwise key expansion"
B	$Min(AA, SPA)\|\|Max(AA, SPA)\|\|$ $Min(ANonce, SNonce)\|\|Max(ANonce, SNonce)$

TABLE I
ARGUMENTS TO PRF

B. Decryption

The next phase involves decrypting 802.11 packets as they are received by any device monitoring the network. Figure 1 shows the steps used to retrieve the plaintext. This appears to take five inputs, however the AAD and MIC are not required as is only used for integrity verification which is not desirable in this project. The TK has already been derived previously, meaning the only other parameter aside from the data itself is the nonce. The nonce is 13 bytes and maps as shown in Table II. All this data is sent in the 802.11 header, therefore presenting the possibility of decrypting the packet to get the plaintext data.

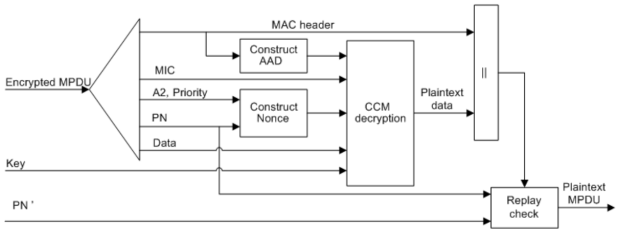

Fig. 1. CCMP decapsulation block diagram (reproduced from [1])

Octet	Value
1	Priority (fixed at 0)
2-7	Second address
8-13	Packet number

TABLE II
NONCE STRUCTURE

III. DESIGN AND IMPLEMENTATION

The implementation consists of a back-end and front-end application, with one consuming data the other produces. A brief outline is displayed as a flowchart in Figure 2 and each section is discussed in more detail below.

A. Connections and Decryption

The main library used in the back-end application is pcap4j[1], a Java library for capturing packets off a network card and allowing the packet layers to be traversed easily. Using this, a handler runs which peeks at every packet received. The first two bytes of the packet are inspected to determine

[1]https://www.pcap4j.org/

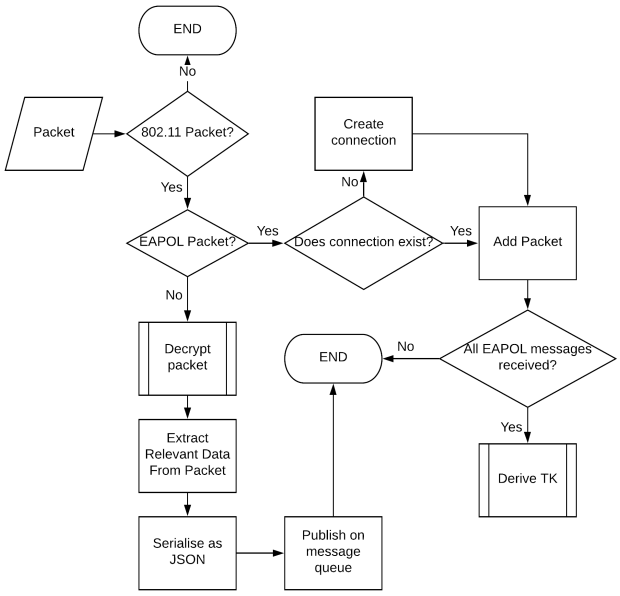

Fig. 2. System flowchart

whether it is an 802.11 packet or an EAPOL packet. In the case of the latter, this causes a new `Connection` object to be instantiated and is provided with the parameters identifying that connection (a pair of MAC addresses). Upon receipt of the final EAPOL message, a task is started to derive the temporal key. It is possible to force a handshake between a pair of devices by intentionally de-authenticating the device off the network, using an application such as `aireplay-ng`.

Once a connection has been established successfully with a derived temporal key, further packets intercepted can be analysed. When the main listener thread detects an incoming packet it retrieves the relevant connection object via the MAC addresses and calls its decrypt method with the entire packet. The decrypt method's role is to extract the packet number and second address from the header and pass these as the arguments to an AES-CCM decrypter. The integrity checks are intentionally ignored here as this requires information locally stored on the pair of devices involved in the connection. The encrypted payload of the packet is extracted and decrypted with the AES-CCM instance.

B. Analysis

Upon successful decryption, a `pcap4j.packet.LlcPacket` is constructed from the raw byte data. This is so that any further traversal of the packet's structure can be done via library method calls, rather than inspecting the contents of the byte array manually. Within the `LlcPacket` is an `Ipv4Packet`, where the source and destination IP addresses can be extracted. Inside this is either a TCP or UDP packet where the ports used for the connection can be extracted. This is a significant amount of information to analyse traffic, as things such as the underlying application layer protocol can be determined.

However, for some protocols additional information of use can be obtained, particularly with DNS queries. If the core packet is a DNS request, the questions asked to the server (record type and domain) can easily be viewed. With this information, user activity can be monitored with habits analysed.

To be used by clients, the data is published onto an MQTT broker using the `eclipse-paho`[2] library under the topic `root/packets/` and then the packet type (in this case either `generic` or `dns`). The data is serialised into a JSON object using the `Gson`[3] library, as this puts it in a standard format which is readable by a variety of applications.

C. Client

The job of the client is to subscribe to the MQTT topics published to by the back-end application, and display the data to the user. The core of this is a time-series database which will allow for efficient querying of data over time intervals. This is ideal as these are the type of operations to be performed to produce graphs and analysis of the data. InfluxDB[4] was selected for this, due to its simple Java API and support for multiple fields to be associated with a given measurement.

A Java application was constructed to connect the data on the MQTT broker with the database. This is accomplished by subscribing to the relevant topics published to by the back-end application and then deserialise the JSON format into an object, so that the different fields can easily be retrieved. Then, using the InfluxDB API, a new measurement is recorded at that timestamp with the various fields published.

To visualise the data stored in the database an existing service called Grafana[5] was deployed. This provides intuitive and simple creation of graphs based on TSQL queries which are executed directly on a provided data source. From this a few graphs were created to show various metrics of all decrypted packets, as shown below. In addition, for DNS packets, the questions asked in the query are represented as another measurement.

- The source address
- The destination address
- The source port
- The destination port
- The protocol used

From this, queries such as which clients are most active over time and what different requests are made can be shown. This is extremely flexible and can be modified in future to include additional packets if they are implemented on a separate topic.

IV. Example Analysis

This section outlines the process the back-end application takes to produce a decrypted packet from a device. This starts when each of EAPOL messages are received during the handshake, causing a new `Connection` object to be created with the addresses and these packets. Once the fourth and final EAPOL message is added, the decoding process is initiated. Table III shows the values used for the connection.

[2]https://www.eclipse.org/paho/
[3]https://github.com/google/gson
[4]https://influxdata.com
[5]https://grafana.com

Parameter	Value
PMK	`8c36c8f2e805fea9e153ff1ed457b 3c1cf87f428de5432566b77e7e91a 8ab5aa`
AA	e4956e4400e6
SPA	448500dc39ee

TABLE III
Sample Parameters

The decoding function takes the two addresses of the connection (see Figure I) and the nonces given by both devices (highlighted in Figures 3 and 4) to construct the data argument used in the PRF. The constructed data argument is shown in Listing 2.

```
448500DC39EEE4956E4400E6BD2735CA00654390
EF452863D853D2D2760C36C85997AF77C05CA33A
272EC55CCBC4F0A9F9879A00EF6317C7D67300C2
0DB915717C8180991D2A99A054679DEE
```

Listing 2. Data argument to PRF function

```
0000  00 00 20 00 ae 40 00 a0  20 08 00 a0 20 08 00 00
0010  10 02 71 09 a0 00 dc 00  64 00 00 00 00 00 00 01
0020  88 02 3a 01 44 85 00 dc  39 ee e4 95 6e 44 00 e6
0030  e4 95 6e 44 00 e6 00 00  07 00 aa aa 03 00 00 00
0040  88 8e 02 03 00 5f 02 00  8a 00 10 00 00 00 00 00
0050  00 00 01 cb c4 f0 a9 f9  87 9a 00 ef 63 17 c7 d6
0060  73 00 c2 0d b9 15 71 7c  81 80 99 1d 2a 99 a0 54
0070  67 9d ee 00 00 00 00 00  00 00 00 00 00 00 00 00
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00a0  00 00 00 00 00 b4 26 1a  22
```

Fig. 3. EAPOL Message 1 with ANonce

```
0010  10 02 71 09 a0 00 de 00  64 00 00 00 00 00 00 01
0020  88 01 3a 01 e4 95 6e 44  00 e6 44 85 00 dc 39 ee
0030  e4 95 6e 44 00 e6 00 00  07 00 aa aa 03 00 00 00
0040  88 8e 01 03 00 75 02 01  0a 00 00 00 00 00 00 00
0050  00 00 01 bd 27 35 ca 00  65 43 90 ef 45 28 63 d8
0060  53 d2 d2 76 0c 36 c8 59  97 af 77 c0 5c a3 3a 27
0070  2e c5 5c 00 00 00 00 00  00 00 00 00 00 00 00 00
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0090  00 00 00 5e f5 97 65 1f  fb ab 7c 38 d3 02 f3 aa
00a0  9a be fe 00 16 30 14 01  00 00 0f ac 04 01 00 00
00b0  0f ac 04 01 00 00 0f ac  02 00 00 c0 35 78 e7
```

Fig. 4. EAPOL Message 2 with SNonce

Now the `generateTK` method is called, as all the data required to compute it is present. This returns the value shown in Listing 3. Production of this value means that future packets can be decrypted and analysed when they arrive.

```
5CED6B863FCCFC3E0E51837CD5FEC81D
```

Listing 3. Outputted TK

When a 802.11 packet is detected, such as that in Figure 5, it is passed to the decrypt method of the connection identified by the two MAC addresses. This produces the bytestring shown

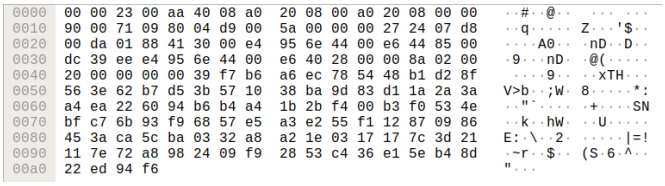

Fig. 5. Encrypted Payload

in Figure 6. An ASCII dump of this shows that the decryption was successful and shows a DNS request packet with the query of `www.googleapis.com`. This is picked up by pcap4j and allows it to be represented as a Java object, as shown in Listing 4.

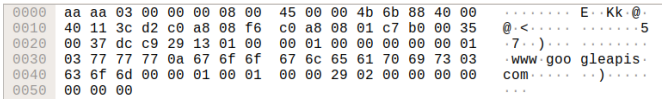

Fig. 6. Decrypted Payload

```
[Logical Link Control header (3 bytes)]
    ...
[Subnetwork Access Protocol header (5
   bytes)]
    ...
[IPv4 Header (20 bytes)]
    ...
[UDP Header (8 bytes)]
    ...
[DNS Header (47 bytes)]
    ...
```

Listing 4. Decrypted packet represented as pcap object (header values omitted)

Finally relevant data is extracted from this to produce a single JSON object, shown in Listing 5, which is then published to an MQTT topic.

```
{
    "questions":["www.googleapis.com"],
    "srcAddress":"192.168.8.246",
    "destAddress":"192.168.8.1",
    "srcPort":51120,
    "destPort":53,
    "packetType":"DNS"
}
```

Listing 5. Serialised JSON

A. Runtime

As a high rate of packets can be interepted by a monitoring device, performance is an important consideration. While Java may have not been a perfect choice in this regard, when using primitive types such as byte arrays to represent data with minimal object orientation, the overhead is reduced. The entire process outlined above takes approximately 500ms-1s.

V. Evaluation

The implementation succeeds in its goal it set out to achieve and is able to visualise the data produced in an intuitive way. However, there are some aspects where it can be improved. The most notable of these is reliably detecting a packet is 802.11 encrypted, as during development there were a few cases where the application had some false positives and negatives. In addition, only a minimal number of properties are extracted and DNS is the only protocol that has specialised behaviour. As a result future work could involve expanding this into other protocols such as HTTP, SMTP, DHCP etc.

VI. Conclusion

This proof of concept shows that minimal implementation is required to reverse the encryption process and shows that while WPA2 solves a significant number of issues compared to its predecessors such as WEP and WPA with regards to privacy and integrity of messages, there are still cases where it can be circumvented to be used maliciously when the preshared key is known to the attacker. This is worsened by the fact that the attacker does not need to be present on the network, making them harder to detect.

This poses concerns for some home users as while they do not have a large range of devices or a potentially malicious user, many consumer routers do have pseudorandom passwords that can be derived. In enterprises this project highlights that using a single key for an entire network with a large number of users is not secure, as if a single user determines the key the entire network is compromised. An alternative solution such as WPA2-EAP, where each user has their own credentials rather than a single shared credential, is much more appropriate as a compromised password only affects that user's connections.

VII. Contributions

The majority of the functionality (particularly the decryption and connection management code) was completed by both team members by adopting a pair programming style, where both were sat around a single computer. Aside from this, Dan was the main lead on the front-end code (MQTT, Database and Grafana integration) whereas Elliot was the main lead on the threading and packet detection of the back-end code.

References

[1] Ieee standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements-part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications: Amendment 6: Medium access control (mac) security enhancements. *IEEE Std 802.11i-2004*, pages 1–190, July 2004.