WPA2 is a networking protocol which allows user to discover and connect to Wi-Fi networks. We are going to focus on the authentication capabilities of the protocol in our attack.

Before learning about the attack, you need to be familiar with a few important concepts. If you aren't familiar with hash functions or neworking and packets, take a look at the additional materials provided to you first :)

- MAC Addresses:
  Every networking card (the physical device responsible for interacting with the network) has a unique address called a MAC address, which consists of 6 bytes, usually written as hexadecimal. For example- 60:e9:8e:4f:aa:6f. A special mac address is ff:ff:ff:ff:ff:ff, or `Broadcast`. Using this address allows data packets to be sent to all devices on a local area network (LAN) simultaneously, rather than to a single target address. We would use this address, for example, to 'announce' the ssid, the unique Wi-Fi network identifier, or 'name'.
- HMAC -
  HMAC is a cryptographic technique that combines a cryptographic hash function with a secret key to provide data integrity and authentication for messages. Any cryptographic hash function can be used, but the WPA2 protocol uses sha1-based HMAC.

Now we can start diving into the protocol!

Our attack focuses on WPA2 key generation, which requires an authentication process called the four way handshake. The server, or AP (access point), uses this process to verify that the client indeed known the Wi-Fi's passphrase. You might see the passphrase referred to as a **PSK**, or Pre-Shared-Key, in the protocol.

Your goal should be to be able to calculate the **client key MIC**, sent in the second of the four handshake packets, from any password. If the MIC matches the one in the packet- then you know the right password!

This will allow you to perform brute force and dictionary attacks on WPA2 really soon :)

This is how the MIC is calculated:

1. First, an association process between the client, or supplicant, as referred to in the protocol, AP happens. This process is irrelevant to our attack. All you need to know is that before the four-way handshake begins, the client should know the **SSID**, the Wi-Fi network's 'name'. This 'name' can be found in a Beacon packet, which we provided for you in all captures, sent by the AP.
2. Before starting the handshake, the client and AP both calculate something called a **PMK** (Pair Master Key). The **PMK** is 256-bit (32 byte) key, calculated from the The passphrase (or PSK) and SSID (service set identifier, the unique "name" of the Wi-Fi network), by PBKDF2 (password based key derivation function 2). Note: We implemented PBKDF2 for you! The calculation is performed with the following parameters: password = passphrase, salt = ssid, length = 32 (the resulting length of pmk, in bytes), and iterations = 32.
3. In addition to the PMK, they client and AP each produce 256-bit (or 32-byte) random nonces, called **Anonce** (AP's nonce) and **Snonce** (client's nonce).

4. After the client and AP both have the PMK and salts, the handshake begins!Four EAPOL (Extensible Authentication Protocol over LAN) packets are sent between the AP and client, that should prove that the client knows the correct passphrase, by proving they know the correct PMK (and relying on the strength of the hashing algorithm). The first handshake packet is sent from the AP to the client; in it, the server nonce, or ANonce, is sent.
5. After receiving the first packet, the client calculates the **PTK** (Pair Temporary Key) by a PRF (pseudo-random function).
   The PRF that is used is in fact:
   sha1_HMAC(key=pmk, password=A_salt || 0 || B_salt || i) for i = 0...3.
   This means that the ptk is: first_hmac || second_hmac || third_hmac || fourth_hmac.
   Note that we also implemented sha1-HMAC for you!
   The two salts used for the PTK calculations are: the constant string "Pairwise key expansion", called the "A" salt, and another dynamic salt, calculated from the client and AP MAC addresses, along with the two nonces, called the "B" salt. The B salt is calculated from the client's and AP's MAC addresses and nonces, by: min(AP_MAC, S_MAC) || max(AP_MAC, S_MAC) || min(APNonce,SNonce) || max(APNonce,SNonce), where S_MAC is the client's MAC address.
6. The client then splits the PTK is then split into three different keys. Only one of them, the **KCK** (Key Confirmation Key) is relevant to our attack. The **KCK** is just the first 128 bits, or 16 bytes, of the PTK.
7. Finally, the client key MIC (Message Integrity code) is calculated by the client. The client takes the authentication layer of the second EAPOL packet (which he has prepared but not sent yet), with a BLANK (all zeroes) client key MIC, and performs: sha1_HMAC(key=kck, password=authentication_layer). Note that the client key MIC is at bytes 81-96 of the EAPOL authentication layer.

If you can follow these steps, you will be able to calculate the MIC by yourselves! Don't worry, we guide you through the process step-by-step, and you can ask for the implementation of any step as a hint!

Good luck!