Part 1

1) What is salt in cryptography and what is its purpose? (10 pt)
   Salting refers to adding random data to a hash function to obtain a unique output which refers to the hash. The main reason this is done is to increase security and protect against various attacks like brute-force. They can be of variable length and are unique. Salts help us mitigate hash table attacks by forcing attackers to re-compute them using the salts for each user.

   References:
   i) About the authorDan AriasStaff Developer AdvocateHowdy! ? I do technology research at Auth0 with a focus on security and identity and develop apps to showcase the advantages or pitfalls of such technology. I also contribute to the development of our SDKs. “Add Salt to Hashing: A Better Way to Store Passwords.” *Auth0*, auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/. Accessed 11 Apr. 2025.

   ii) Devi, Navanita. “What Is a Salt and How It Boosts Security?” *Loginradius*, LoginRadius, www.loginradius.com/blog/identity/what-is-salt/. Accessed 11 Apr. 2025.

2) What types of attacks can be done if salt is not used in hash functions? Please explain one attack and how it would recover confidential information. (10 pt)

   Attacks that involve using a database can be performed if salt is not used. Salting protects against rainbow table attacks. It is a password cracking method that uses a special table to crack password hashes in the database. The rainbow table itself refers to the precomputed table that contains the password hash value for each plaintext character used during authentication. So basically, if the attackers gain access to the list of password hashes they can crack all passwords using rainbow table attack. Attackers usually gain access to these tables using phishing techniques. As long as the password hashes don't include salt, the attackers can crack the passwords.

   References:
   i) “Rainbow Table Attack.” *Beyond Identity*, www.beyondidentity.com/glossary/rainbow-table-attack. Accessed 12 Apr. 2025.

Part 2.1

1) The salt size is 16 bytes.

2) The algorithm uses SHA256.

3) The size of the nonce is 12 bytes.

4)

```
cpre3310@cpre3310:~/homework/Lab10$ python3 template.py
Encrypted: b"\xa8\xda\x04\xa8y{\xe4'\x87Q6\xc9\xd1\xac\xd2\x98'=\x8f/\x9e\x
ddM,\xe0{\r!\xe1\x9e\x8569.t"
Decrypted: This is the second part for AES-GCM
cpre3310@cpre3310:~/homework/Lab10$
```

Part 2.2

1) The LLM chosen is ChatGPT-4. I chose this as it has built in knowledge of permutation based AEAD like Ascon and KMAC. The prompt used was:
" Create a working Python example of AEAD using permutation-based construction like KMAC or sponge functions with pycryptodome. " Number of prompts tried were two. Prompt 1 used pyascon and the second one switched to KMAC..

2) The generated code uses the KMAC128 (Keccak-based MAC), which uses the sponge permutation function behind SHA-3.

3) The generated code uses nonce and hash function. It does not explicitly use salt and IV.

4) Uploaded.

Part 2.3

1) AES GCM: 0.214504 sec
KMAC128 AEAD: 0.000107 sec

   AES GCM was faster mostly due to the hardware acceleration and optimized libraries. AES -GCM uses optimized CPU instructions.

2) I used the timeit.default_timer() as its cross-platform, avoids pitfalls, and measures wall-clock time.

3) Increasing plaintext:
AES-GCM : 0.171179 sec
KMAC: 0.000089 sec

   KMAC is slower than AES-GCM.

   Increasing password size:
AES-GCM: 0.171977sec
KMAC: No change

   AES-GCM scales better with password changes.

4) Uploaded.