Data Confidentiality

What is data confidentiality?

- Confidentiality means ensuring that data is only accessible to those authorised to view it.
- Examples of sensitive data you might want kept confidential include addresses, phone numbers, and bank account details.
- As the maintainer of a web service, it’s important that you keep sensitive user data confidential.
- If you suffer a confidentiality breach, and did not follow proper practices, you may be liable for damages!

https://haveibeenpwned.com/

How can we keep data confidential?

- First line of defence is to sanitize external inputs, preventing unauthorized database access via SQL injection (as you have covered in week 2).
- Second line of defence is to encrypt user data, making it unreadable without knowing the user’s password.

How do we encrypt user data?

- By using established software libraries such as OpenSSL.
- There are two important algorithms you'll need from this library to secure the user data in your projects…
    - **Advanced Encryption Standard (AES)** to perform the actual encryption and decryption.
    - **Password-Based Key Derivation Function 2 (PBKDF2)** to convert a user's password into a suitable *key*, i.e., a fixed-length string of bits.

Advanced Encryption Standard (AES)

- A *symmetric encryption algorithm*, i.e., encrypts and decrypts data using the same key. The key **must be kept secret.**
- Requires an *initialization vector* (IV), which sets the initial state of the algorithm. This does not have to be kept secret but should be random.
- You should use a different IV for every user. This means that should two users have the same key and the same data, their encrypted data will be different (while this is unlikely it’s still good practice).

Password-Based Key Derivation Function 2 (PBKDF2)

- Converts a user's password into a key for AES. AES requires a fixed-length key (e.g. 16 bytes), so using a password directly is usually not possible.
- Requires a *salt*, i.e., a string of random bits. This **must be different for every user**. PBKDF2 combines the password and salt to ensure that users with the same password get different keys (mitigating against *rainbow table attacks*).
- PBKDF2 does not make a password harder to guess. It should not be considered a substitute for a strong password.

Procedure for encrypting user data

1. Generate a random IV and salt for the user.
2. Use PBKDF2 with the user's password and salt to produce the key.
3. Use AES with the IV and key to encrypt the data.
4. Store the user's IV, salt, and encrypted data in the database.

```
aes = OpenSSL::Cipher.new( 'AES-128-CBC').encrypt                          # Set up AES for encryption
self.iv = aes.random_iv                                                    # Generate a random IV
self.salt = OpenSSL::Random.random_bytes( 16)                              # Generate a random salt
aes.key = OpenSSL::PKCS5.pbkdf2_hmac_sha1(password, salt,  20000, 16)      # Get key from pass. and
salt
self.data_crypt = aes.update(data) + aes.final                             # Encrypt the data
```

Procedure for decrypting user data

1. Read the user's IV, salt, and encrypted data from the database.
2. Use PBKDF2 with the user's password and salt to produce the key.
3. Use AES with the IV and key to decrypt the data.

```
aes = OpenSSL::Cipher.new( 'AES-128-CBC' ).decrypt                          # Set up AES for decryption
aes.iv = iv                                                                 # Set the IV
aes.key = OpenSSL::PKCS5.pbkdf2_hmac_sha1(password, salt,  20000, 16)   # Get key from pass. and
salt
begin                                                                    # Try to decrypt the data
  return aes.update(data_crypt) + aes.final
rescue OpenSSL::Cipher::CipherError
  return nil
end
```

Some things to note

- You can only decrypt a user’s data if you know their password. This makes it inaccessible to anyone else (including admins).
- There is no way to recover a user’s data the if they forget their password.
- For these reasons, this type of encryption sacrifices accessibility for confidentiality and should only be applied when necessary.