Report

1. Introduction

We had to implement a simple program that reads in a folder path and a password from the user and ecryptes the folder.

2. Implementation

We decided it's easiest to make a simple cli tool. Also we think it's the easiest to first compress the folder into one file and then encrypt that file.

3. Encryption

To generate a key with a password we first used the `hashlib` module to hash the password and then used `os.urandom` to generate a key and IV.

```python
def __generate_key(self, pwd: str) -> tuple[bytes, bytes]:
        """Generates a key, IV, and salt from a password using sha."""
        hashed_pwd = hashlib.sha256(pwd.encode()).digest()
        key = os.urandom(hashed_pwd[16:])
        iv = os.urandom(16)

        return key, iv
```

Then we used this to encrypt the file.

But after some googeling and asking Claude we found that this will not be secure enough. Because if someone finds out we are using sha they can attack with us precomputed tables (rainbow tables). Especiall if the password is weak.

So then we used the `PBKDF2HMAC` function from the `cryptography` module to generate the key, IV and **salt**.

Salt

A salt is a random value that is generated for each encryption operation. Here's what it does:

1. Makes Each Encryption Unique: Even if two users encrypt files with the same password, the resulting encrypted files will be completely different because they use different salts.
2. Prevents Rainbow Table Attacks: Rainbow tables become ineffective because the attacker would need a separate rainbow table for each possible salt value. With a 16-byte salt, that's 2ˆ128 possible values - far too many to precompute.

3. Ensures Unique Keys: The salt is combined with the password before hashing, ensuring that even identical passwords produce different encryption keys.