```python
#Discussed with Sean, Louis, Min Htet, Min Yue

import os
import base64
import getpass
import sys
from cryptography.hazmat.primitives import hashes, hmac
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms
from cryptography.hazmat.primitives.ciphers.modes import CBC
from cryptography.hazmat.primitives.padding import PKCS7
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC

def encryption():
    # First, we grab the contents of stdin and make sure it's a single string
    plaintext = "".join( sys.stdin.readlines() ).encode('utf-8')

    # Use getpass to prompt the user for a password
    password = getpass.getpass()
    password2 = getpass.getpass("Enter password again:")

    # Do a quick check to make sure that the password is the same!
    if password != password2:
        sys.stderr.write("Passwords did not match")
        sys.exit()

    #randomize salt and IV
    salt = b'0' #in order for key to be same on encode and decode
    iv = os.urandom(16)

    # derive key with pbkdf2
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000
    )

    key = kdf.derive(bytes(password,'utf-8'))
    signingkey = key[:16]
    encryptionkey = key[16:]

    # Define the Ciphertext
    mode = CBC(iv) # Prepare CBC mode with random IV
    aes = algorithms.AES128(encryptionkey) #prepare AES128
    cipher = Cipher(aes,mode)
    encryptor  = cipher.encryptor()

    #pad text
    pkcs7 = PKCS7(128).padder()
    pkcs7.update(plaintext)
    paddedtext= pkcs7.finalize()

    # Actually do the encryption
    ciphertext = encryptor.update(paddedtext) + encryptor.finalize()

    #sign text
    signingtext = iv + ciphertext
    h2 = hmac.HMAC(signingkey,hashes.SHA256())
    h2.update(signingtext)
    hmactext = h2.finalize()

    #concatanate IV, ciphertext and HMAC together
    token = signingtext+hmactext

    # encode to url safe base 64
    encoded_token = base64.urlsafe_b64encode(token)

    # Return the ciphertext to standard out
    sys.stdout.write(encoded_token.decode('utf-8'))



def decryption():
    # Grab stdin.
    stdin_contents = "".join( sys.stdin.readlines() )

    #use getpass to prompt user for password
    password = getpass.getpass()

    # Cinvert to bytes for the ciphertext,decode url_safe base 64
    encoded_token = stdin_contents.encode('utf-8')
    token = base64.urlsafe_b64decode(encoded_token)

    #extract the different components of message
    iv = token[0:16]
    ciphertext = token[16:len(token)-32]
    hmactext = token[len(token)-32:]
    signingtext = token[0:len(token)-32]

    #retrive key
    salt = b'0' #in order for key to be same on encode and decode

    # derive key with pbkdf2
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000
    )

    key = kdf.derive(bytes(password,'utf-8'))
    signingkey = key[:16]
    encryptionkey = key[16:]

    # Define the Ciphertext
    mode = CBC(iv) # Prepare CBC mode with derived IV
    aes = algorithms.AES128(encryptionkey) #prepare AES128
    cipher = Cipher(aes,mode)
    decryptor  = cipher.decryptor()

    try:
        #attempt to verify integrity
        h2 = hmac.HMAC(signingkey,hashes.SHA256())
        h2.update(signingtext)
        h2.verify(hmactext)
        #attempt decryption
        paddedtext = decryptor.update(ciphertext) + decryptor.finalize()

        #attempt depadding
        pkcs7 = PKCS7(128).unpadder()
        pkcs7.update(paddedtext)
        plaintext= pkcs7.finalize()


    except:
        sys.stderr.write("Decryption failed. Check your password or the
file.\n")
        sys.exit()

    # Return the plaintext to stdout
    sys.stdout.write(plaintext.decode('utf-8'))

try:
    mode = sys.argv[1]
    assert( mode in ['-e', '-d'] )
except:
    sys.stderr.write("Unrecognized mode. Usage:\n")
    sys.stderr.write("'python3 fernet.py -e' encrypts stdin and returns the
ciphertext to stdout\n")
    sys.stderr.write("'python3 fernet.py -d' decrypts stdin and returns the
plaintext to stdout\n")

if mode == '-e':
    encryption()
elif mode == '-d':
    decryption()
```