This is the working pseudocode solution to the program.

```
# Source Code File:  Part 4: DECRYPTION: Based on the parameters below.
# Name:              part4.py
# Author:            <students name>

IMPORT argparse
IMPORT getpass
IMPORT hashlib
IMPORT nacl.utils
IMPORT SecretBox FROM nacl.secret
IMPORT scrypt FROM nacl.pwhash

# Parameters
SET SALT_SIZE TO 32
SET SCRYPT_OPSLIMIT TO scrypt.OPSLIMIT_INTERACTIVE
SET SCRYPT_MEMLIMIT TO scrypt.MEMLIMIT_INTERACTIVE

DEFINE FUNCTION get_password():
    """Prompt the user for a password."""
    RETURN getpass.getpass("Enter password: ").encode()

DEFINE FUNCTION calculate_sha256(file_path):
    """Calculate the SHA-256 hash of a file."""
    INITIALIZE sha256_hash AS hashlib.sha256()
    TRY:
        OPEN file_path AS binary file f
        FOR byte_block IN ITERATE f.read(4096) UNTIL byte_block IS EMPTY:
            UPDATE sha256_hash WITH byte_block
        RETURN sha256_hash.hexdigest()
    EXCEPT FileNotFoundError:
        PRINT "Error: File not found for hash calculation."
        EXIT PROGRAM

DEFINE FUNCTION decrypt_file(password, input_file, output_file):
    """Decrypt a file."""
    TRY:
        OPEN input_file AS binary file f
        READ contents INTO encrypted_data
    EXCEPT FileNotFoundError:
        PRINT "Error: Encrypted file not found."
        EXIT PROGRAM

    # Extract components
    EXTRACT encrypted_data[:SALT_SIZE] INTO salt
    EXTRACT encrypted_data[SALT_SIZE:SALT_SIZE + 72] INTO outer_encrypted
    EXTRACT encrypted_data[SALT_SIZE + 72:] INTO inner_encrypted

    # Derive the password key
    password_key = scrypt.kdf(
        SecretBox.KEY_SIZE,
        password,
        salt,
        opslimit=SCRYPT_OPSLIMIT,
        memlimit=SCRYPT_MEMLIMIT
    )

    # Decrypt the outer box to retrieve the inner key
    TRY:
        INITIALIZE outer_box AS SecretBox(password_key)
        SET inner_key TO RESULT OF outer_box.decrypt(outer_encrypted)
    EXCEPT Exception:
        PRINT "Error: Invalid password or corrupted outer encryption."
        EXIT PROGRAM
```

```
    # Decrypt the inner box to retrieve the payload
    INITIALIZE inner_box AS SecretBox(inner_key)
    TRY:
        SET payload TO RESULT OF inner_box.decrypt(inner_encrypted)
        OPEN output_file IN binary write mode AS f
        WRITE payload TO f
        PRINT f"File decrypted and saved to {output_file}"
    EXCEPT Exception:
        PRINT "Error: Corrupted inner encryption."
        EXIT PROGRAM

    # Calculate and display the SHA-256 hash of the decrypted file
    SET sha256_hash TO RESULT OF calculate_sha256(output_file)
    PRINT f"SHA-256 hash of the decrypted file: {sha256_hash}"

# Main execution
IF SCRIPT IS RUN DIRECTLY:
    INITIALIZE parser AS argparse.ArgumentParser WITH description "Decrypt an encrypted file."
    ADD argument "input" TO parser WITH help "Path to the encrypted input file"
    ADD argument "output" TO parser WITH help "Path to the output decrypted file"
    PARSE arguments INTO args

    SET password TO RESULT OF get_password()
    CALL decrypt_file(password, args.input, args.output)
```

This is the working pseudocode solution to the program.

```
# Source Code File:     Part 4: ENCRYPTION/DECRYPTION: Based on the parameters below.
# Name:                 part4_pw_change.py
# Author:               <students name>

IMPORT argparse, os, getpass, tempfile
IMPORT nacl.utils
IMPORT SecretBox FROM nacl.secret
IMPORT scrypt FROM nacl.pwhash

# Parameters
SET SALT_SIZE TO 32
SET SCRYPT_OPSLIMIT TO scrypt.OPSLIMIT_INTERACTIVE
SET SCRYPT_MEMLIMIT TO scrypt.MEMLIMIT_INTERACTIVE

DEFINE FUNCTION get_password(prompt="Enter password: ", confirm=False):
    """Prompt user for a password."""
    WHILE True:
        SET password TO getpass.getpass(prompt)
        IF confirm IS True:
            SET confirm_password TO getpass.getpass("Confirm password: ")
            IF password EQUALS confirm_password:
                RETURN password AS BYTES
            ELSE:
                PRINT "Passwords do not match. Please try again."
        ELSE:
            RETURN password AS BYTES

DEFINE FUNCTION encrypt_file(password, input_file, output_file):
    """Encrypt a file."""
    # Generate a random salt
    SET salt TO os.urandom(SALT_SIZE)

    # Derive a password key
    SET password_key TO scrypt.kdf(
        SecretBox.KEY_SIZE,
        password,
        salt,
        opslimit=SCRYPT_OPSLIMIT,
        memlimit=SCRYPT_MEMLIMIT
    )

    # Generate a random key for the inner box
    SET inner_key TO os.urandom(SecretBox.KEY_SIZE)

    # Encrypt the payload using the inner key
    INITIALIZE inner_box AS SecretBox(inner_key)

    TRY:
        OPEN input_file AS binary file f
        READ f INTO payload
        SET inner_encrypted TO inner_box.encrypt(payload)
    EXCEPT FileNotFoundError:
        PRINT "Error: Input file not found."
        EXIT PROGRAM

    # Encrypt the inner key with the password-derived key
    INITIALIZE outer_box AS SecretBox(password_key)
    SET outer_encrypted TO outer_box.encrypt(inner_key)

    # Save salt, outer box, and inner box to file
    OPEN output_file AS binary file f
    WRITE (salt + outer_encrypted + inner_encrypted) TO f

    PRINT "File encrypted and saved to {output_file}"

DEFINE FUNCTION decrypt_file(password, input_file, output_file):
    """Decrypt a file."""
    TRY:
        OPEN input_file AS binary file f
```

```
        READ f INTO encrypted_data
    EXCEPT FileNotFoundError:
        PRINT "Error: Encrypted file not found."
        EXIT PROGRAM

    # Extract salt, outer box, and inner box
    EXTRACT encrypted_data[:SALT_SIZE] INTO salt
    EXTRACT encrypted_data[SALT_SIZE:SALT_SIZE + 72] INTO outer_encrypted
    EXTRACT encrypted_data[SALT_SIZE + 72:] INTO inner_encrypted

    # Derive the password key
    SET password_key TO scrypt.kdf(
        SecretBox.KEY_SIZE,
        password,
        salt,
        opslimit=SCRYPT_OPSLIMIT,
        memlimit=SCRYPT_MEMLIMIT
    )

    # Decrypt the outer box to get the inner key
    TRY:
        INITIALIZE outer_box AS SecretBox(password_key)
        SET inner_key TO outer_box.decrypt(outer_encrypted)
    EXCEPT Exception:
        PRINT "Error: Invalid password or corrupted outer encryption."
        EXIT PROGRAM

    # Decrypt the inner box to retrieve the payload
    INITIALIZE inner_box AS SecretBox(inner_key)

    TRY:
        SET payload TO inner_box.decrypt(inner_encrypted)
        OPEN output_file AS binary file f
        WRITE payload TO f
        PRINT "File decrypted and saved to {output_file}"
    EXCEPT Exception:
        PRINT "Error: Corrupted inner encryption."
        EXIT PROGRAM

DEFINE FUNCTION change_password(input_file):
    """Change the password of an encrypted file."""
    # Prompt user for old and new passwords
    SET old_password TO get_password("Enter old password: ")
    SET new_password TO get_password("Enter new password: ", confirm=True)

    TRY:
        OPEN input_file AS binary file f
        READ f INTO encrypted_data
    EXCEPT FileNotFoundError:
        PRINT "Error: Encrypted file not found."
        EXIT PROGRAM

    # Extract salt, outer box, and inner box
    EXTRACT encrypted_data[:SALT_SIZE] INTO salt
    EXTRACT encrypted_data[SALT_SIZE:SALT_SIZE + 72] INTO outer_encrypted
    EXTRACT encrypted_data[SALT_SIZE + 72:] INTO inner_encrypted

    # Derive the old password key
    SET old_password_key TO scrypt.kdf(
        SecretBox.KEY_SIZE,
        old_password,
        salt,
        opslimit=SCRYPT_OPSLIMIT,
        memlimit=SCRYPT_MEMLIMIT
    )

    # Decrypt the outer box to get the inner key
    TRY:
        INITIALIZE outer_box AS SecretBox(old_password_key)
        SET inner_key TO outer_box.decrypt(outer_encrypted)
    EXCEPT Exception:
```

```
        PRINT "Error: Invalid old password or corrupted outer encryption."
        EXIT PROGRAM

    # Derive the new password key
    SET new_salt TO os.urandom(SALT_SIZE)
    SET new_password_key TO scrypt.kdf(
        SecretBox.KEY_SIZE,
        new_password,
        new_salt,
        opslimit=SCRYPT_OPSLIMIT,
        memlimit=SCRYPT_MEMLIMIT
    )

    # Encrypt the inner key with the new password key
    INITIALIZE new_outer_box AS SecretBox(new_password_key)
    SET new_outer_encrypted TO new_outer_box.encrypt(inner_key)

    # Save new encrypted data to a temporary file
    OPEN TEMP FILE AS binary file temp_file
    WRITE (new_salt + new_outer_encrypted + inner_encrypted) TO temp_file
    REPLACE input_file WITH TEMP FILE
    PRINT "Password changed successfully."

# Main execution
IF SCRIPT IS RUN DIRECTLY:
    INITIALIZE parser AS argparse.ArgumentParser
    ADD subcommands: encrypt, decrypt, change-password TO parser
    PARSE arguments INTO args

    IF args.command IS "encrypt":
        SET password TO get_password(confirm=True)
        CALL encrypt_file(password, args.input, args.output)
    ELIF args.command IS "decrypt":
        SET password TO get_password()
        CALL decrypt_file(password, args.input, args.output)
    ELIF args.command IS "change-password":
        CALL change_password(args.input)
```