```python
#!/usr/bin/env python3

import sys, getopt, os, secrets
from constants import MODES


def main() -> None:
    try:
        opts, args = getopt.gnu_getopt(sys.argv[1:], "hdli:o:k:m:", ["help"])
    except getopt.GetoptError as error:
        print(error)
        usage()
        sys.exit(2)
    if any(i in sys.argv for i in ["-h", "--help"]):
        usage()
        sys.exit()
    if "-l" in sys.argv:
        print("Supported cipher block modes: " + " ".join(MODES))
        sys.exit()
    if "-k" not in sys.argv:
        print("Error: Need to specify a key with -k [0-255]")
        usage()
        sys.exit(2)
    if "-i" not in sys.argv:
        print("Error: Need to specify an input file with -i FILE")
        usage()
        sys.exit(2)
    mode = "ECB"
    for opt, arg in opts:
        if opt == "-i":
            inFile = arg
            if not os.path.isfile(inFile):
                print("Error: input file not found")
                sys.exit(2)
        elif opt == "-o":
            outFile = arg
        elif opt == "-m":
            mode = arg
            if mode not in MODES:
                print("Unsupported cipher block mode")
                print("Supported cipher block modes: " + " ".join(MODES))
                sys.exit(2)
        elif opt == "-k":
            try:
                key = int(arg)
```

```
                if key < 0 or key > 255:
                    raise ValueError
            except ValueError:
                print(
                    "Error: Could not read key value. Please use an integer
between 0 and 255 inclusive."
                )
                usage()
                sys.exit(2)
    keys = keySchedule(key)
    with open(inFile, "rb") as file:
        inData = [i for i in file.read()]
    if ("-d", "") in opts:
        decData = decrypt(inData, keys, mode)
        if "-o" in sys.argv:
            with open(outFile, "wb") as file:
                file.write(bytes(decData))
        else:
            sys.stdout.buffer.write(bytes(decData))
    else:
        encData = encrypt(inData, keys, mode)
        if "-o" in sys.argv:
            with open(outFile, "wb") as file:
                file.write(bytes(encData))
        else:
            sys.stdout.buffer.write(bytes(encData))


def encrypt(data: list[int], keys: list[int], mode: str) -> list[int]:
    computed = {}
    match mode:
        case "ECB":
            encrypted = []
            for i in data:
                temp = computed.get(i)
                if temp:
                    encrypted.append(temp)
                    continue
                enc = encryptData(i, keys)
                encrypted.append(enc)
                computed[i] = enc
        case "CBC":
            iv = secrets.randbelow(256)
            encrypted = [encryptData(iv ^ data[0], keys)]
            computed[iv ^ data[0]] = encrypted[0]
```

```
            for i in range(1, len(data)):
                temp = computed.get(encrypted[i - 1] ^ data[i])
                if temp:
                    encrypted.append(temp)
                    continue
                enc = encryptData(encrypted[i - 1] ^ data[i], keys)
                encrypted.append(enc)
                computed[encrypted[i - 1] ^ data[i]] = enc
            encrypted.append(iv)
        case "OFB":
            encrypted = []
            iv = secrets.randbelow(256)
            result = encryptData(iv, keys)
            computed[iv] = result
            for i in data:
                encrypted.append(result ^ i)
                temp = computed.get(result)
                if temp:
                    result = temp
                else:
                    enc = encryptData(result, keys)
                    computed[result] = enc
                    result = enc
            encrypted.append(iv)
        case "CFB":
            iv = secrets.randbelow(256)
            encrypted = [data[0] ^ encryptData(iv, keys)]
            computed[iv] = data[0] ^ encrypted[0]
            for i in range(1, len(data)):
                temp = computed.get(encrypted[i - 1])
                if temp:
                    encrypted.append(temp ^ data[i])
                    continue
                enc = data[i] ^ encryptData(encrypted[i - 1], keys)
                encrypted.append(enc)
                computed[encrypted[i - 1]] = data[i] ^ enc
            encrypted.append(iv)
    return encrypted


def decrypt(data: list[int], keys: list[int], mode: str) -> list[int]:
    computed = {}
    match mode:
        case "ECB":
            decrypted = []
```

```
            for i in data:
                temp = computed.get(i)
                if temp:
                    decrypted.append(temp)
                    continue
                dec = decryptData(i, keys)
                decrypted.append(dec)
                computed[i] = dec
        case "CBC":
            iv = data[-1]
            decrypted = [iv ^ decryptData(data[0], keys)]
            computed[data[0]] = iv ^ decrypted[0]
            for i in range(1, len(data) - 1):
                temp = computed.get(data[i])
                if temp:
                    decrypted.append(data[i - 1] ^ temp)
                    continue
                dec = data[i - 1] ^ decryptData(data[i], keys)
                decrypted.append(dec)
                computed[data[i]] = dec ^ data[i - 1]
        case "OFB":
            decrypted = []
            iv = data[-1]
            result = encryptData(iv, keys)
            computed[iv] = result
            for i in data[:-1]:
                decrypted.append(result ^ i)
                temp = computed.get(result)
                if temp:
                    result = temp
                else:
                    enc = encryptData(result, keys)
                    computed[result] = enc
                    result = enc
        case "CFB":
            iv = data[-1]
            decrypted = [data[0] ^ encryptData(iv, keys)]
            computed[iv] = data[0] ^ decrypted[0]
            for i in range(1, len(data) - 1):
                temp = computed.get(data[i - 1])
                if temp:
                    decrypted.append(temp ^ data[i])
                    continue
                dec = data[i] ^ encryptData(data[i - 1], keys)
                decrypted.append(dec)
```

```
                computed[data[i - 1]] = dec ^ data[i]
    return decrypted


def keySchedule(key: int) -> list[int]:
    keys = [key]
    for i in range(5):
        temp = intToCrumbs(keys[i])
        temp[0] ^= g(temp[3], i % 2)
        temp[1] ^= temp[0]
        temp[2] ^= temp[1]
        temp[3] ^= temp[2]
        keys.append(intFromCrumbs(temp))
    return keys


def g(crumb: int, round: int) -> int:
    return (crumb % 2 ^ round) * 2 + crumb // 2


def intToCrumbs(num: int) -> list[int]:
    return [num // 64, num % 64 // 16, num % 16 // 4, num % 4]


def intFromCrumbs(crumbs: list[int]) -> int:
    return crumbs[0] * 64 + crumbs[1] * 16 + crumbs[2] * 4 + crumbs[3]


def encryptData(data: int, keys: list[int]) -> int:
    temp = data ^ keys[0]
    for j in range(5):
        temp = intToCrumbs(temp)
        temp = sbox(temp)
        temp = shiftRows(temp)
        temp = mixColumns(temp) if j < 4 else temp
        temp = intFromCrumbs(temp)
        temp ^= keys[j + 1]
    return temp


def sbox(byte: list[int]) -> list[int]:
    SBOX = [2, 0, 3, 1]
    return [SBOX[byte[0]], SBOX[byte[1]], SBOX[byte[2]], SBOX[byte[3]]]


def shiftRows(byte: list[int]) -> list[int]:
```

```
    return [byte[0], byte[1], byte[3], byte[2]]


def mixColumns(byte: list[int]) -> list[int]:
    return [
        (byte[0] // 2) * 2 + byte[2] % 2,
        byte[1] % 2 + (byte[3] // 2) * 2,
        (byte[2] // 2) * 2 + byte[0] % 2,
        byte[3] % 2 + (byte[1] // 2) * 2,
    ]


def decryptData(data: int, keys: list[int]) -> int:
    temp = data
    for j in range(5):
        temp ^= keys[5 - j]
        temp = intToCrumbs(temp)
        temp = mixColumns(temp) if j > 0 else temp
        temp = shiftRows(temp)
        temp = isbox(temp)
        temp = intFromCrumbs(temp)
    temp ^= keys[0]
    return temp


def isbox(byte: list[int]) -> list[int]:
    ISBOX = [1, 3, 0, 2]
    return [ISBOX[byte[0]], ISBOX[byte[1]], ISBOX[byte[2]], ISBOX[byte[3]]]


def usage() -> None:
    print("""Usage: miniAES.py [option(s)] -k [0-255] -i FILE
Encrypts data in a minified version of AES with 8-bit block size and an 8-bit key

Mandatory options:
  -k [0-255]\tSpecify key value as an integer between 0 and 255 inclusive
  -i FILE\tSpecify file to encrypt

Options:
  -h, --help\tDisplays this information
  -d\t\tSwitch mode to decryption
  -m MODE\tSets cipher block mode, ECB by default
  -l\t\tLists supported cipher block modes
  -o FILE\tWrite output to FILE instead of stdout""")
```

```python
if __name__ == "__main__":
    main()
```