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

import miniAES
from constants import *
import sys, getopt, os


def main() -> None:
    try:
        opts, args = getopt.gnu_getopt(sys.argv[1:], "hlo:s:p:", ["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 file types for non-UTF-8 text:", SUPPORTED)
        sys.exit()
    if len(args) == 0:
        print("Error: Need to specify an input file as an argument")
        usage()
        sys.exit(2)
    elif len(args) > 1:
        print("Error: Too many arguments")
        usage()
        sys.exit(2)
    else:
        inFile = args[0]
        if not os.path.isfile(inFile):
            print("Error: input file not found")
            sys.exit(2)
    skip = 0
    pattern = None
    for opt, arg in opts:
        if opt == "-o":
            outFile = arg
        elif opt == "-p":
            pattern = arg
        elif opt == "-s":
            try:
                skip = int(arg)
                if skip < 0:
                    raise ValueError
```

```
            except ValueError:
                print("Error: Bad skip value. Please use a positive integer.")
                usage()
                sys.exit(2)

    with open(inFile, "rb") as file:
        inData = [i for i in file.read()]
    sample = inData[:65536]
    bestEntropy = (65536, 256)
    bestLetter = (65536, 256)
    bestPrintable = (0, 256)
    for mode in MODES:
        if mode == "CBC" and len(inData) > 65536:
            sample.append(inData[-1])
        for i in range(256):
            decData = miniAES.decrypt(sample, KEYS[i], mode)
            if pattern:
                if known(pattern, decData):
                    if "-o" in sys.argv:
                        with open(outFile, "wb") as file:
                            file.write(bytes(miniAES.decrypt(inData, KEYS[i],
mode)))
                        sys.exit(0)
                    else:
                        sys.stdout.buffer.write(
                            bytes(miniAES.decrypt(inData, KEYS[i], mode))
                        )
                        sys.exit(0)
                elif mode != "CFB" or i < 255:
                    continue
                else:
                    print("Pattern not found in decryption attempts")
                    sys.exit(1)
            sig = fileSigs(decData, skip)
            if sig:
                if "-o" in sys.argv:
                    print(
                        "Decrypted data matches file signature for '"
                        + sig
                        + "' file type"
                    )
                    print("Writing to", outFile + ".cracked." + sig)
                    with open(outFile + ".cracked." + sig, "wb") as file:
                        file.write(bytes(miniAES.decrypt(inData, KEYS[i], mode)))
                    sys.exit(0)
```

```
                else:
                    sys.stdout.buffer.write(
                        bytes(miniAES.decrypt(inData, KEYS[i], mode))
                    )
                    sys.exit(0)
            entScore = entropy(decData)
            if entScore < bestEntropy[0]:
                bestEntropy = (entScore, i, mode)
            if not validUTF(decData):
                continue
            count = [0] * 256
            for j in decData:
                count[j] += 1
            freqScore = letterFrequency(count)
            if freqScore < bestLetter[0]:
                bestLetter = (freqScore, i, mode)
            printableScore = spaceOrLetters(count, len(decData))
            if printableScore > bestPrintable[0]:
                bestPrintable = (printableScore, i, mode)

    if bestEntropy[1] == bestLetter[1] or bestEntropy[1] == bestPrintable[1]:
        if "-o" in sys.argv:
            with open(outFile, "wb") as file:
                file.write(
                    bytes(miniAES.decrypt(inData, KEYS[bestEntropy[1]],
bestEntropy[2]))
                )
        else:
            sys.stdout.buffer.write(
                bytes(miniAES.decrypt(inData, KEYS[bestEntropy[1]],
bestEntropy[2]))
            )
    elif bestLetter[1] == bestPrintable[1]:
        if "-o" in sys.argv:
            with open(outFile, "wb") as file:
                file.write(
                    bytes(miniAES.decrypt(inData, KEYS[bestLetter[1]],
bestLetter[2]))
                )
        else:
            sys.stdout.buffer.write(
                bytes(miniAES.decrypt(inData, KEYS[bestLetter[1]],
bestLetter[2]))
            )
    else:
```

```
        print("Unclear, try these key mode combinations with miniAES.py:")
        print(f"Key {bestPrintable[1]}, Mode: {bestPrintable[2]}")
        print(f"Key {bestLetter[1]}, Mode: {bestLetter[2]}")
        print(f"Key {bestEntropy[1]}, Mode: {bestEntropy[2]}")


def spaceOrLetters(counts: list[int], length: int) -> float:
    return sum([counts[32]] + counts[65:91] + counts[97:123]) / length


def entropy(data: list[int]) -> float:
    count = [0] * 256
    for i in set(data):
        count[i] = 1
    earliest = count.index(1)
    latest = 255 - count[::-1].index(1)
    return (latest - earliest) / 256


def fileSigs(data: list[int], skip: int) -> str:
    for b, t in FILESIGS:
        if data[: len(b)] == b:
            if skip > 0:
                skip -= 1
            else:
                return t
    if data[257:263] == [117, 115, 116, 97, 114, 0] or data[257:263] == [
        117,
        115,
        116,
        97,
        114,
        32,
    ]:
        if skip > 0:
            skip -= 1
        else:
            return "tar"
    if data[4:12] == [102, 116, 121, 112, 77, 83, 78, 86] or data[4:12] == [
        102,
        116,
        121,
        112,
        105,
        115,
```

```
        111,
        109,
    ]:
        if skip > 0:
            skip -= 1
        else:
            return "mp4"
    return


def known(clue: str, data: list[int]) -> bool:
    plain = list(clue.encode())
    length = len(plain)
    for i in range(len(data) - length + 1):
        if plain == data[i : i + length]:
            return True
    return False


def letterFrequency(counts: list[int]) -> int:
    letters = []
    for i in range(26):
        letters.append((counts[65 + i] + counts[97 + i], chr(97 + i)))
    letters.sort(reverse=True)
    score = 0
    for j in range(26):
        score += abs(RANKINGS.index(letters[j][1]) - j)
    return score


def validUTF(data: list[int]) -> bool:
    i = 0
    while i < len(data):
        try:
            if data[i] > 247:
                return False
            elif data[i] > 239:
                if (
                    data[i + 1] < 128
                    or data[i + 1] > 191
                    or data[i + 2] < 128
                    or data[i + 2] > 191
                    or data[i + 3] < 128
                    or data[i + 3] > 191
                ):
```

```
                    return False
                i += 4
            elif data[i] > 223:
                if (
                    data[i + 1] < 128
                    or data[i + 1] > 191
                    or data[i + 2] < 128
                    or data[i + 2] > 191
                ):
                    return False
                i += 3
            elif data[i] > 191:
                if data[i + 1] < 128 or data[i + 1] > 191:
                    return False
                i += 2
            elif data[i] > 127:
                return False
            else:
                i += 1
        except IndexError:
            return False
    return True


def usage() -> None:
    print("""Usage: crackMini.py [option(s)] FILE
Attempts to decrypt data encrypted with miniAES without knowing the key

Options:
  -h, --help\tDisplays this information
  -o FILE\tWrite output to FILE instead of stdout
  -l\t\tLists supported file types for non-text
  -s NUM\tSkips NUM incorrect file type matches
  -p PATTERN\tMatches known pattern in decrypted text""")


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