pythonese

First i decompiled the file using pylingual.
The entry point is `f16()` which:

1. Reads input string `i`
2. Builds a VM that creates a `run()` function
3. Calls `run()` and prints its return value

The key is in `f15()` which creates the `run()` function:

```
def f15(j):
    if f12(''.join([i[:5], i[5:]])):  # Check if input passes f12
        r = f13(*[int(i[:4]), int(i[4:6])])  # Extract k1, k2 from input
        return (marshal.dumps(compile('def run(x):\n return \'', r() +
'\'\n', '<s2>', 'exec')), True)
```

The `f13` function is where the flag is constructed. It takes two parameters
`k1 = int(i[:4])` (first 4 digits of input)
`k2 = int(i[4:6])` (next 2 digits of input)
Inside `f13`, there's a nested function `fvdy` that decodes integer arrays:

```
def fvdy(arr): return ''.join(chr((((v >> 1) + k2) ^ (k1 & 0xFF)) & 0xFF)
for v in arr)
```

After decoding, `f13` assembles the final string:

```
L = [b, c, d, e, f, g, h, i]
L = [s[::(-1)] for s in L]  # Reverse each string
P = [3, 6, 1, 7, 0, 5, 2, 4]  # Permutation order
R = [''] * 8
for k, v in enumerate(P):
    R[v] = L[k]  # Reorder according to permutation
u = ''.join((s[::(-1)] for s in R))  # Reverse again and join
return a + u + j  # prefix + reordered_middle + suffix
```

We know CTF flags typically start with "CTF{", so we need: `a = fvdy([382, 356, 392, 430]) = "CTF{"`

```
    'C' (67): (191 + k2) ^ (k1 & 0xFF) = 67
    'T' (84): (178 + k2) ^ (k1 & 0xFF) = 84
    'F' (70): (196 + k2) ^ (k1 & 0xFF) = 70
    '{' (123): (215 + k2) ^ (k1 & 0xFF) = 123
```

Solving the first two:

```
XORing these: 191 ^ 178 = 67 ^ 84, which gives us k2 = 83
```

Substituting back: `(191 + 83) ^ (k1 & 0xFF) = 67` This gives us `k1 & 0xFF = 81`, so `k1 ≡ 81 (mod 256) We need a 4-digit k1, so` k1 = 1105` works (since 1105 % 256 = 81)
So the final solution:

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

def fvdy(arr, k1, k2):
    """Decode array using the transformation from f13"""
    return ''.join(chr((((v >> 1) + k2) ^ (k1 & 0xFF)) & 0xFF) for v in arr)

def solve_flag():
    """Solve for k1 and k2 to get the flag"""

    # Arrays from the decompiled code
    arrays = {
        'a': [382, 356, 392, 430],      # prefix
        'b': [442, 544, 456, 552, 544, 538, 540, 446],
        'g': [546, 548, 556, 446, 442, 556, 442, 448],
        'e': [540, 542, 452, 544, 456, 546, 448, 456],
        'd': [446, 540, 456, 548, 540, 538, 442, 546],
        'f': [544, 554, 548, 548, 446, 450, 446, 540],
        'j': [434],                     # suffix
        'c': [554, 538, 544, 448, 548, 554, 542, 452],
        'h': [456, 442, 546, 542, 556, 554, 542, 542],
        'i': [544, 550, 554, 456, 448, 456, 446, 556]
    }

    # We want 'a' to decode to "CTF{"
    target_prefix = "CTF{"
    prefix_array = arrays['a']

    k1_mod = 81
    k2 = 83

    # Find a 4-digit k1 that satisfies k1 ≡ 81 (mod 256)
    k1 = 1105

    print(f"Using k1={k1}, k2={k2}")

    decoded_parts = {}
    for name, arr in arrays.items():
        decoded_parts[name] = fvdy(arr, k1, k2)
        print(f"{name}: '{decoded_parts[name]}'")
```

```
    # Assemble the flag following the logic from f13
    L = [decoded_parts[name] for name in ['b', 'c', 'd', 'e', 'f', 'g', 'h',
'i']]
    L = [s[::-1] for s in L]  # Reverse each string

    P = [3, 6, 1, 7, 0, 5, 2, 4]  # Permutation
    R = [''] * 8
    for k, v in enumerate(P):
        R[v] = L[k]

    u = ''.join(s[::-1] for s in R)  # Reverse again and join

    flag = decoded_parts['a'] + u + decoded_parts['j']

    print(f"\nFinal flag: {flag}")
    return flag

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