STEP 1 (KEY EXTRACTION)
Upon opening the website and reading the info written we can safely assume we need to extract the key from the oracle provided. Upon reviewing the code we see that the encryption logic uses Affine cipher + AES-CBC where each letter is treated like a block. So we can extract the key by inputting values like "AAAAAAAAAAAAAAAA" or "BBBBBBBBBBBBBBBB" and then creating quadratics based on the results and solving for variables. This can be done manually or via scripting like this:

```
import requests
from string import ascii_uppercase

BASE_URL = "https://oracle-web-crypto.vercel.app/"
A = 11
B = 17
ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
ALPHA_IDX = {c: i for i, c in enumerate(ALPHABET)}
IDX_ALPHA = {i: c for i, c in enumerate(ALPHABET)}
BLOCK_SIZE = 16

def to_nums(s):
    s = ''.join(c for c in s.upper() if c in ALPHABET)
    s = (s + 'H' * BLOCK_SIZE)[:BLOCK_SIZE]
    return [ALPHA_IDX[c] for c in s]

def nums_to_text(nums):
    return ''.join(IDX_ALPHA[n % 26] for n in nums)

session = requests.Session()

plaintext = "ABCDEFGHIJKLMNOP" # any plaintext of your choice it
will be used to generate quadratics
r = session.post(BASE_URL + "/lv1_enc", data={"plaintext":
plaintext})
r.raise_for_status()
ct = r.json()["ciphertext"].strip()
print("ciphertext:", ct)

pt_nums = to_nums(plaintext)
ct_nums = to_nums(ct)

# compute k[i] for i >= 1: from c[i] = A*p[i] + B*c[i-1] + k[i]
# => k[i] = c[i] - A*p[i] - B*c[i-1] (mod 26)
```

```
key_nums = [None] * BLOCK_SIZE
for i in range(1, BLOCK_SIZE):
    k = (ct_nums[i] - (A * pt_nums[i]) - (B * ct_nums[i-1])) % 26
    key_nums[i] = k

print("Recovered key indices for k[1..15] (as letters):",
nums_to_text(key_nums[1:]))


for guess in range(26):
    key_try = key_nums.copy()
    key_try[0] = guess
    candidate = nums_to_text(key_try)
    resp = session.post(BASE_URL + "/lv1_keycheck", data={"key":
candidate})
    if resp.status_code == 200:
        print("Key found:", candidate)
        break
else:
    print("Brute-force failed")
```

This gives us:

```
ciphertext: ETKBCQNTEEUACCYG
Recovered key indices for k[1..15] (as letters): SDGTFNHJTYHAHSF
Key found: WSDGTFNHJTYHAHSF
```

STEP 2 (BIT-FLIPPING)
Now we can use this key to move on to stage 2. We have another info section for this stage that basically tells us we are “STRANDEDPIRATE” and we need to become “CAPTAINJACK”. We are also given the Ciphertext for “STRANDEDPIRATE” which is “QWHUCKNFISFHICSPUKBMNLJIIYJYXKFY” where the first 16 digits are the IV and the next 16 are the actual ciphertext which is the standard notation in AES. So AES-CBC has a vulnerability in which changes to the ciphertext result in changes to the plaintext, this is known as bit-flipping and we can use it here to become “CAPTAINJACK”. This can also be scripted:

```
import requests
import re
from string import ascii_uppercase
```

```python
BASE = "https://oracle-web-crypto.vercel.app"

A = 11
B = 17
BLOCK = 16
ALPH = ascii_uppercase
IDX = {c: i for i, c in enumerate(ALPH)}
INV_A = pow(A, -1, 26)
PAD_CHAR = 'H'

def url(path):
    return BASE.rstrip('/') + path

def letters_to_indices(s):
    return [IDX[c] for c in s]

def indices_to_letters(lst):
    return ''.join(ALPH[i % 26] for i in lst)

def idx(x):
    return x % 26

def letter(i):
    return ALPH[i % 26]

def find_first_upper_run(text, length=11):
    m = re.search(r'\b([A-Z]{' + str(length) + r'})\b', text)
    if m:
        return m.group(1)
    all_matches = re.findall(r'[A-Z]{' + str(length) + r',}',
text)
    if all_matches:
        return all_matches[0][:length]
    return None

def extract_flag_key(text):
    m = re.search(r"reward[:''\"` ]*\s*([A-Za-z0-9_\-:]+)", text,
re.IGNORECASE)
    if m:
        return m.group(1)
    m =
re.search(r"(?:reward|Pirate)[^A-Za-z0-9\n\r]{0,10}([A-Za-z0-9_\-:
```

```
]{6,})", text, re.IGNORECASE)
    if m:
        return m.group(1)
    return None

def main():
    s = requests.Session()

    M_test = "A" * 16
    U_test = "B" * 16
    test_input = M_test + U_test
    print("Querying /lv2_test with test input to leak key bytes
...")
    r = s.post(url("/lv2_test"), data={"user_input": test_input})
    text = r.text

    decrypted_id = find_first_upper_run(text, length=11)
    if not decrypted_id:
        print("Could not find identity in response. Full response
below:")
        print(text)
        return


    # 3) compute k_i for i=0..10 using k = u_i - B*c_prev - A*p_i
(mod26)
    M_idx = letters_to_indices(M_test)
    U_idx = letters_to_indices(U_test)
    P_idx = letters_to_indices(decrypted_id)[:11]

    K = {}
    for i in range(0, 11):
        c_prev = M_idx[i] if i == 0 else U_idx[i-1]
        k_i = idx(U_idx[i] - B * c_prev - A * P_idx[i])
        K[i] = k_i
    recovered = ''.join(letter(K[i]) for i in range(11))
    print("Recovered KEY_2[first 11 letters]:", recovered)

    target = "CAPTAINJACK"
    P_target_idx = letters_to_indices(target)
    M_final = M_test
    M_final_idx = letters_to_indices(M_final)
    U_final_idx = [0] * BLOCK

    for i in range(0, 11):
        c_prev = M_final_idx[i] if i == 0 else U_final_idx[i-1]
        # u_i = A * p_i + B * c_prev + k_i  (mod 26)
        u_i = idx(A * P_target_idx[i] + B * c_prev + K[i])
        U_final_idx[i] = u_i
    for i in range(11, BLOCK):
        U_final_idx[i] = 0

    M_final_str = indices_to_letters(M_final_idx)
    U_final_str = indices_to_letters(U_final_idx)
    final_input = M_final_str + U_final_str
    print("Final input prepared (32):", final_input)

    r2 = s.post(url("/lv2_test"), data={"user_input":
final_input})

    FLAG_key = extract_flag_key(r2.text)
    if not FLAG_key:
        print("Could not extract FLAG_key from response. Full
response:")
        print(r2.text)
        return
    print("FLAG_key:", FLAG_key)

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

This gives us the output:

```
Querying /lv2_test with test input to leak key bytes ...
Recovered KEY_2[first 11 letters]: SHIVERMETIM
Final input prepared (32): AAAAAAAAAAAAAAAAOLWGCJWJQQEAAAAA
FLAG_key: PLOKUHGSTRFDFBYU
```

Now that we have the FLAG_KEY we can input it and get the flag which is: “DJSISACA{1sNt_C13C_4M4z1nG_LsZWLKymfYp}”.