The core idea of the solution comes from realizing we can remove the polinom function altogether, since it is called with values ranging from 00 to 99, which we can calculate beforehand:

```python
def polinom(n, m):
    i = 0
    z = []
    s = 0
    while n > 0:
        if n % 2 != 0:
            z.append(2 - (n % 4))
        else:
            z.append(0)
        n = (n - z[i]) / 2
        i = i + 1
    z = z[::-1]
    l = len(z)
    for i in range(0, l):
        s += z[i] * m ** (l - 1 - i)
    return s

with open("polinom_results.txt", "w") as file:
    for number in range(1, 100):
        result = int(polinom(number, 3))
        formatted_number = f"{number:02}"
        file.write(f"{formatted_number}:{result}\n")
```

Now that this is done, we get to the hard part. We need to generate all possible iflags. At first sight it might seem impossible, but looking through polinom_results.txt we see that it might just be possible. We do this by mapping the result to the number, and make a recursive search by guessing how many digits from encrypted_flag we take at a time, ranging from 1 to 4 (since the biggest number in polinom_results is 4 digits long). Meaning, for 4672511, we might split 46 72 511 or 4 672 5 1 1 but not 46725 11.

Then, we need the lookup the chosen slice and see if it matches a polinom result. If yes, we assume we are correct and look further (recursively). If at any point we do not find a slice of length 1-4 from the current position, we just continue as normal.

Afterwards, we need to convert each iflag back to flag.

```python
polinom_lookup = {}
def is_valid_integer(s):
    if s == "0":
        return True
    return s.isdigit() and s[0] != '0'
for i in range(10000):
    polinom_lookup[str(i)] = -1
```

```python
with open('polinom_results.txt', 'r') as f:
    for line in f:
        number, result = line.strip().split(':')
        polinom_lookup[result] = number # Mapping results to numbers

def recover_inputs(encrypted_text, current_position=0, current_input='',
results=None):
    if results is None:
        results = []
    if current_position == len(encrypted_text): # If we got to the end of
encrypted_text we found a valid iflag
        results.append(current_input)
        return
    for length in range(1, 5): # Lengths 1-4
        if current_position + length <= len(encrypted_text):
            prefix = encrypted_text[current_position:current_position + length] #
Select the slice
            if is_valid_integer(prefix) == 0: # Since 01 can't be a valid result
we skip such prefixes. This is to prevent an error.
                continue
            if polinom_lookup[prefix] != -1: # If prefix exists in our lookup
table
                new_input = current_input + polinom_lookup[prefix]
                recover_inputs(encrypted_text, current_position + length,
new_input, results)
    return results

encrypted_text = "242712673639869973827786401934639193473972235217215301"
possible_inputs = recover_inputs(encrypted_text)
for decimal_str in possible_inputs:
   decimal_value = int(decimal_str) // 100 * 10 + 1 # This is done as I noticed
the program mistakenly places a 01 at the end instead of 1.
   hex_value = hex(decimal_value)[2:]
   try:
      original_text = bytes.fromhex(hex_value).decode('utf-8')
      print(f"{original_text}")
      break
   except Exception as e:
      pass
```

After running this we get the secret message! All that’s left is to sha256 it and wrap it in CTF{}

Made with love by: AndreiCat